bc630AT
Software Developer's Kit

PN
8500 - 0093

User's Guide
May, 1999

bc630AT
DEVELOPER'S KIT

TABLE OF CONTENTS

SECTION **PAGE**

CHAPTER ONE
INTRODUCTION

CHAPTER TWO
INSTALLATION

CHAPTER THREE
LIBRARY DEFINITIONS

This Page Intentionally Left Blank.

CHAPTER ONE
INTRODUCTION

1.0 GENERAL

The bc630AT Developer's Kit is designed to provide a suite of tools useful in the development of applications which access features of the Datum bc630AT Real Time Clock Module. This kit has been designed to provide an interface between the bc630AT and applications developed for Windows 95™, and Windows NT™ environments. In addition to the interface DLL, two example programs are provided, complete with source code, in order to provide a better understanding of the kit features and benefits.

1.1 FEATURES

The salient features of the Developer's Kit include:

- Interface library with access to all features of the bc630AT.
- Hardware driver for Windows NT™ and VxD for Windows 95™
- Example programs, with source, utilizing the interface library.
- Console application to configure registry keys.
- User's Guide providing a library definition.

1.2 OVERVIEW

The Developer's Kit was designed to provide an interface to the Real Time Clock Module in the 32-bit environments of Windows 95™ and Windows NT™ . The example programs were developed under Microsoft Visual C ++ 5.0. The example programs provides sample code which exercise the interface DLL as well as examples of converting many of the ASCII format data objects passed to and from the device into a binary format suitable for operation and conversion. The example programs were developed using discrete functions for each operation which allows the developer to clip any useful code and use it in their own applications. A resource file is included with interface dialogs to allow the operator of a program to set any configurable parameters for operating the bc630AT hardware. A discrete 32-bit console application is provided in the Developer's Kit which can be distributed to end users to configure registry keys to access the hardware interface.

This Page Intentionally Left Blank.

CHAPTER TWO

INSTALLATION

2.0 GENERAL

Installation of the Developer's Kit is handled by the installer program. Following the installation, the user must set up the appropriate hardware driver and registry key information for the operating system. The following steps are required for a full system installation.

- Use the setup.exe program on the Developer's Kit to install the kit.
- Copy the appropriate hardware driver to the system location.
- Use the supplied registry utility to configure the registry keys.
- Use the compiled example programs to test the system.

Note: A reboot is necessary after configuring the registry entries for the first time.

2.1 CONFIGURATION

Directory structures are created in the specified location. These structures contains all required files to develop 32-bit user applications. In addition, copies of the hardware driver files and configuration utilities are provided for redistribution with user-developed 32-bit applications.

Directory of dist\…\Example Programs\bc630at
This directory contains all the files for rebuilding the example program.

Directory of dist\…\Example Programs\bc630atTrayTime
This directory contains all files for rebuilding the example program.

Directory of dist\…\Example Programs\Hardware Libraries
This directory contains compiled dll and lib files.

Directory of dist\…\Hardware Drivers
This directory contains Windows 95 and WinNT Drivers.

Directory of dist\…\Utility Programs
This directory contains three .EXE programs

Directory of dist\…\Documentation
This directory contains the manual for the software developer's kit.

2.2 HARDWARE DRIVER INSTALLATION

A hardware driver handles the underlying I/O space access in the Developer's Kit routines. A service is used for Windows NT™ and a virtual device driver for Windows 95™.

Copy the appropriate file for the host platform from the Developer's Kit util subdirectory into the defined location.

Platform	File	Location
Windows NT™	WINRT.SYS	\windir\SYSTEM32\DRIVERS
Windows 95™	WRTDEV0.VXD	\windir\SYSTEM\VMM32

2.3 BOARD ADDRESS CONFIGRATION

Use the supplied registry utility bc630Reg.exe to configure the registry keys. The keys differ with the host OS. The utility will determine the correct operating system and create and/or modify the appropriate register keys.

The registry utility needs to know the base address set on the bc630AT hardware and an interrupt level, if any interrupt jumpers were set. The command syntax can be queried by executing the program with no parameters.

```
bc630Reg  0x300 0
```

In this example, the base address is set to hex 300 and the interrupt is ignored. A sample of the output from the command is shown below.

```
C:> bc630Reg 0x300 0

Using Windows 95
Using base address 0x300
Interrupt disabled
Registry info set-up
```

If this key were being set up for the first time, a message would be displayed indicating that the system must be rebooted before the changes will take effect.

2.4 TEST INSTALLATION

Use the compiled version of the example program supplied in the Developer's Kit located in the utility directory to test the installation.

If a device open error is received, the hardware interface was not installed or configured properly. Verify that the correct driver was installed according to the guidelines above.

If the device opens but “00000” are displayed instead of valid time values in the main window, make sure you have a valid TimeCode connected to the bc630AT. If you do have TimeCode coming in to the module and the time is "00000", then the hardware interface was not configured correctly. Verify the base address of the installed bc630AT and use the registry utility in the utils subdirectory to reconfigure the driver. If the error persists, an address conflict may exist with some other piece of hardware in the system. Try changing the hardware address of the bc630AT and reconfiguring the driver before executing the example program again.

2.5 PROJECT CREATION

You can easily rebuild bc630at.exe and bc630atTrayTime.exe by opening the corresponding project file with Visual C++ 5.0.
If you want to use bc_io.dll in your own MFC project, you may follow the instructions below:

1) Insert bc_io.lib into your project.
2) If building a new project similar to bc630AT, you don't need to change the default settings of the project.
3) If building a new project similar to bc630atTrayTime, you may need to change the project settings:

a) For both debug version and release version, go to "C/C++" tab; select "Precompiled Headers" category and then check "Not using precompiled headers" button. Next, go to the Link tab, select "General category" and add "bc_io.lib" to "Object/Library Module" edit box.

b) For release version, Link tab, select "Customize" category and then check "Force File Output" box.

CHAPTER THREE
LIBRARY DEFINITIONS

3.0 GENERAL

The interface library provides functions for each of the software commands supported by the bc630AT Real Time Clock Module. In addition, functions are provided to both read and write individual registers on the card. To understand the usage and effects of each of these functions, please refer to the User's Guides provided with the hardware.

3.1 FUNCTIONS

Note: Library functions bcOpen and bcClose are not applicable for 16-bit applications.

bcOpen	
Prototype	int bcOpen (int devno);
SW Command	N/A
Input Parameter	Device Number ***Note***: This value must be set to 0.
Returns	RC_OK on Success RC_ERROR on Failure
Description: This opens the underlying hardware layer. The developer's kit currently only supports one hardware device per application.	

bcClose	
Prototype	int bcClose (void);
SW Command	N/A
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Closes the underlying hardware layer.	

bcGetByte	
Prototype	int bcGetByte (int offset, unsigned char *value);
SW Command	N/A

Input Parameter	offset = Base Offset of Requested Register
	value = Pointer to Unsigned Char to Return Value Requested

Returns	RC_OK on Success RC_ERROR on Failure
Description: Returns the contents of the requested register.	

bcSetByte	
Prototype	int bcSetByte (int offset, unsigned char value);
SW Command	N/A

Input Parameter	offset = Base Offset of Requested Register
	value = Unsigned Char Value to be Set

Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets the contents of the specified register.	

bcReadTime	
Prototype	int bcReadTime (unsigned char *maj, unsigned long *min, unsigned char *status);
SW Command	<Request Time>
Input Parameter	**maj** = unsigned char pointer to output string. This string will be filled with five bytes corresponding to major time in <Request Time> software command. This array is NOT null terminated. **min** = unsigned long pointer to minor time. Binary minor time in <Request Time> SW Command was combined to output min. ***Note***: *See Bctime.c for example* **status** = pointer to unsigned char status ***Note***: Use the following return values for status 0x00 = time code present 0x01 = flywheeling to the internal crystal 0x02 = flywheeling to an external 1PPS 0x03 = flywheeling to an external 1, 5, 10 MHz frequency reference.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Latches and returns time captured from the time registers.	

bcSetTime	
Prototype	int bcSetTime (char *day, char *hour, char *min, char *sec);
SW Command	<Set major time>
Input Parameter	char *day = Julian day number (Jan 1 = 001) [3 characters] char *hour = hour [2 characters] char *min = minute [2 characters] char *sec = second [2 characters] ***Note***: These are fixed length fields passed exactly as given to the bc630AT. It is not necessary to null terminate the arrays.

Returns	RC_OK on Success RC_ERROR on Failure
Description: Set the major time buffer.	

bcSetRTC	
Prototype	int bcSetRTC (char *dayw, char *year, char *month, char *mday, char *hour, char *min, char *sec);
SW Command	<Set RTC Chip IC Time>
Input Parameter	char *dayw = day of week (Sun=0, ……, Sat = 6) [1 character] char *year = year (00-99)[2 characters] char *month = month (Jan = 01) [2 characters] char *mday = day (e.g. 1 = 01) [2 characters] (01 to 31) char *hour = hour [2 characters] char *min = minute [2 characters] char *sec = second [2 characters] ***Note***: These are fixed length fields passed exactly as given to the bc630AT. It is not necessary to null terminate the arrays.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Set the time in the Real Time Clock chip.	

bcReqRTC	
Prototype	int bcReqRTC (unsigned char *RTC);
SW Command	<Request RTC IC Time>
Input Parameter	unsigned char RTC = unsigned char pointer to output string. This string will be filled with seven bytes corresponding to time in <Request RTC IC Time> software command. This array is NOT null terminated.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Requests the Real Time Clock IC Time	

bcReqPowOffTime	
Prototype	int bcReqPowOffTime (unsigned char *time);
SW Command	<Request Power Off Time>
Input Parameter	unsigned char time = unsigned char pointer to output string. This string will be filled with seven bytes corresponding to time in <Request Power Off Time> software command. This array is NOT null terminated.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Returns the time when computer was last turned off	

bcReqEvntTime	
Prototype	int bcReqEvntTime (unsigned char *major, unsigned long *minor);
SW Command	<Request Event Time>
Input Parameter	unsigned char major = unsigned char pointer to output string. This string will be filled with five bytes corresponding to major time in <Request Time> software command. This array is NOT null terminated. unsigned long minor = unsigned long pointer to minor time. Binary minor time in <Request Event Time> SW Command was combined to output minor.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Request the External Event capture time	

Note: You need to enable the Event capture control once only using (bcSetEventCap( )) before you issue this command.

bcReqAuxData	
Prototype	int bcReqAuxData (unsigned char *aux);
SW Command	<Output auxiliary data>
Input Parameter	unsigned char aux = unsigned char pointer to output string. This string will be filled with seven bytes corresponding to the data in <Output auxiliary data> software command. This array is NOT null terminated.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Requests auxiliary data	

bcReqStatus	
Prototype	int bcReqStatus (unsigned char *stat);
SW Command	N/A
Input Parameter	unsigned char stat = unsigned char pointer to status ***Note***: Use the following return values for status 0x00 = time code present 0x01 = flywheeling to the internal crystal 0x02 = flywheeling to an external 1PPS 0x03 = flywheeling to an external 1, 5, 10 MHz frequency reference.
Returns	RC_OK on Success RC_ERROR on Failure
Description: Returns the status of the bc630AT.	

bcSetHrtBt	
Prototype	int bcSetHrtBt (int frequency);
SW Command	N/A
Input Parameter	int frequency = (1 to 2000) Hz
Returns	RC_OK on Success RC_ERROR on Failure
Description: Program a periodic output in Hz	

bcSetPropDelay	
Prototype	int bcSetPropDelay (int delay);
SW Command	N/A
Input Parameter	int delay = propagation delay (-1022 to +1021) microseconds
Returns	RC_OK on Success RC_ERROR on Failure
Description: Program a propagation delay into the timing engine to account for delays introduced by long cable runs.	

bcSetTcMod	
Prototype	int bcSetTcMod (unsigned char mode);
SW Command	N/A
Input Parameter	mode = TimeCode mode settings ***Note***: The following are defined in bc630at.h #define TC_AUTO 0x00 #define TC_IRIG_A 0x01 #define TC_IRIG_B 0x02 #define TC_2137 0x03 #define TC_RTC 0x04 #define TC_MASTER 0x05 #define TC_NASA36 0x06 #define TC_EXT_1PPS 0x0A #define TC_EXT_MHZ 0x0B
Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets the mode of the bc630AT	

bcSetTcFormat	
Prototype	int bcSetTcFormat (unsigned char type);
SW Command	N/A
Input Parameter	unsigned char type = modulation type of time code ***Note***: The following are defined in bc630at.h #define TC_DCLS 0x00 #define TC_MOD 0x40
Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets time code type	

bcSetInitMode	
Prototype	int bcSetInitMode (unsigned char initial);
SW Command	N/A
Input Parameter	initial = sets the initialization mode upon power on ***Note***: The following are defined in bc630at.h #define NORM_INITIAL 0x00 #define RTC_INITIAL 0x10 #define BATT_INITIAL 0x20 #define USER_INITIAL 0x30
Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets the initialization mode upon power on of the bc630AT	

bcSetEventCap	
Prototype	int bcSetEventCap (unsigned char event);
SW Command	N/A
Input Parameter	unsigned char event = External Event Capture Control ***Note***: The following are defined in bc630at.h #define DIS_EVNT 0x00 #define FAL_EVNT 0x01 #define RIS_EVNT 0x02 #define BTH_EVNT 0x03
Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets the External Event Capture Control for the bc630AT.	

bcSetXFW	
Prototype	int bcSetXFW (unsigned char xfw);
SW Command	N/A
Input Parameter	unsigned char xfw = Enable/Disable External Flywheel Synchronization ***Note***: The following are defined in bc630at.h #define CLR_XFW 0x00 #define SET_XFW 0x04
Returns	RC_OK on Success RC_ERROR on Failure
Description: Enable/Disable External Flywheel Synchronization of the bc630AT	

bcSetFilter	
Prototype	int bcSetFilter (unsigned char filter);
SW Command	N/A
Input Parameter	unsigned char filter = Enable/Disable Digital Filtering of the time source signal ***Note***: The following are defined in bc630at.h #define CLR_FLTR 0x00 #define SET_FLTR 0x08
Returns	RC_OK on Success RC_ERROR on Failure
Description: Enable/Disable Digital Filtering of the time source signal	

bcDelay	
Prototype	int bcDelay (unsigned long delay);
SW Command	N/A
Input Parameter	unsigned long delay = delay in increments of 1 millisecond
Returns	RC_OK on Success RC_ERROR on Failure
Description: Sets a delay	

bcProcMask	
Prototype	int bcProcMask (void);
SW Command	<Process Masks Register Only>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Process Masks Register Only	

bcSynchRTC

Prototype	int bcSynchRTC (void);
SW Command	<Synchronize RTC IC>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Synchronizes the battery backed RTC to the external time source being decoded	

bcRstDef	
Prototype	int bcRstDef (void);
SW Command	<Reset to Default Values>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Reset to Default Values	

bcRstBat	
Prototype	int bcRstBat (void);
SW Command	<Reset to Battery Backed Values>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Reset to Battery Backed Values	

bcInitialize	
Prototype	int bcInitialize (void);
SW Command	<Initialization command>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Initialization command	

bcSyncHeart	
Prototype	int bcSyncHeart (void);
SW Command	<Synchronize Heartbeat Pulses/Interrupts>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: Loads and synchronizes the heartbeat frequency to the current time source	

bcClrEvntCap	
Prototype	int bcClrEvntCap (void);

SW Command	<Clear Event Capture>
Input Parameter	None
Returns	RC_OK on Success RC_ERROR on Failure
Description: The event capture is disabled after each event until this command is executed.	

bcStartInt	
Prototype	int bcStartInt (HWND hWnd, INT dev_no, INT int_mode);
Packet	N/A
Input Parameter	HWND hWnd = Window handle to receive interrupt messages. INT dev_no = 0 INT int_mode = type of interrupt. ***Note***: The following are defined in bc_int.h #define BC_INT_ONE_SHOT 1 #define BC_INT_RECURRING 2
Returns	RC_OK On Success RC_ERROR On Failure
Description: Start the interrupt thread. This thread will send a message to the program using the window handle passed in. The two allowed messages are; #define WM_INT_DYING 0x7026 #define WM_INT_DETECTED 0x7025	

bcStopInt	
Prototype	int bcStopInt (void);
Packet	N/A
Input Parameter	None
Returns	RC_OK On Success RC_ERROR On Failure
Description: Stop the interrupt thread. This thread will send a message to the program using the window handle passed in. The two allowed messages are; #define WM_INT_DYING 0x7026 #define WM_INT_DETECTED 0x7025	

bcSetInts	
Prototype	int bcSetInts (UCHAR *mask);
Packet	N/A
Input Parameter	UCHAR *mask = pointer to mask to load into INTERRUPT MASK register. ***Note***: The following are defined in bc630at.h #define INT_NONE 0x00 #define INT_RDYB 0x20 #define INT_EVNT 0x40 #define INT_HRTB 0x60 #define INT_1PPS 0x80
Returns	RC_OK On Success RC_ERROR On Failure
Description: Only one Interrupt source can be selected. This thread will send a message to the program using the window handle passed in. The two allowed messages are; #define WM_INT_DYING 0x7026 #define WM_INT_DETECTED 0x7025	

bcReqInts	
Prototype	int bcReqInts (UCHAR *mask);
Packet	None
Input Parameter	UCHAR *mask = pointer to mask to load from INTERRUPT MASK register.
Returns	RC_OK On Success RC_ERROR On Failure
Description: Query the currently enabled interrupt.	

Note: Refer to the bc630AT User’s Guide for more information regarding allowed values for the INTERRUPT MASK.