BLUE TECHNIX

BLACKSheep

Software Manual

www.tinyboards.com

Maximum Power at Minimum Size

Contact

Bluetechnix Mechatronische Systeme GmbH
Waidhausenstr. 3/19
A-1140 Vienna
AUSTRIA/EUROPE
office@bluetechnix.at
http://www.bluetechnix.at
http://www.tinyboards.com

Version 1.1

2005-05-03

Document No. 099-0001-01

Table of Contents

The information herein is given to describe certain components and shall not be considered as a guarantee of characteristics.
Terms of delivery and rights of technical change reserved.

We hereby disclaim any warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Bluetechnix makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. Bluetechnix specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Bluetechnix takes no liability for any damages and errors causing of the usage of this board. The user of this board is responsible by himself for the functionality of his application. He is allowed to use the board only if he has the qualification. More information is found in the General Terms and Conditions (AGB).

Information
For further information on technology, delivery terms and conditions and prices please contact Bluetechnix (http://www.bluetechnix.at).

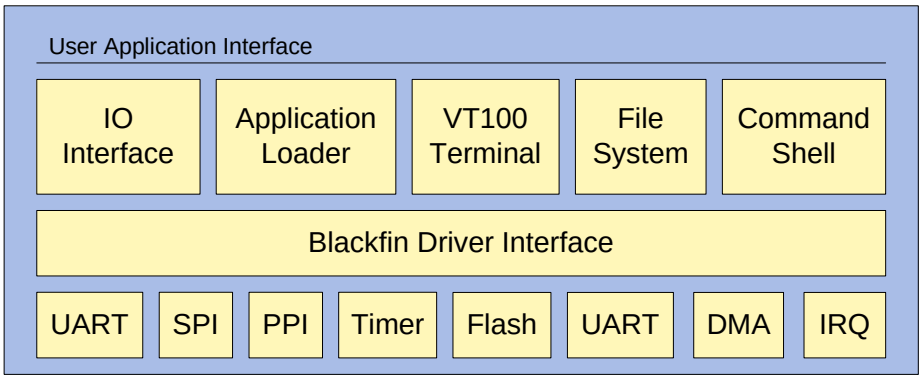
Warnings
Due to technical requirements components may contain dangerous substances

The Core Boards and Development systems contain ESD (electrostatic discharge) sensitive devices. Electro-static charges readily accumulate on the human body and equipment and can discharge without detection. Permanent damage may occur on devices subjected to high-energy discharges. Proper ESD precautions are recommended to avoid performance degradation or loss of functionality. Unused core boards and development boards should be stored in the protective shipping package.

1 Introduction

The BLACKSheep is a single threaded simple input output system, including the most important low level driver to evaluate and use the DEV- and EVAL kits from Bluetechnix. It also provides an interface for downloading, executing and or flashing VDSP++ projects if you don't have an Analog Devices JTAG interface. It is in continuous development, and you can found the newest version on the "Bluetechnix" website (http://www.tinyboards.com).

1.1 Overview



1.2 BLACKSheep Components

Low level driver for:

- SPI
- UART
- PPI (The parallel peripheral interface)
- Core and general purpose Timer
- Programming and erasing functions for the flash on the CM-BF5xx
- Memory and peripheral DMA transfers.
- Interrupt handling
- General purpose flags.
- Camera interface for the Omnivision OV7645/7648 and the OV7660
- SD-card
- Compact flash card
- SMSC 91c1111 Ethernet driver chip

It also includes:

- Command shell interface
- Commands to load and execute VDSP++ projects

2 Target Environment

The BLACKSheep was developed for the Blackfin core modules CM-BF533 amd CM-BF561 provided by Bluetechnix. Some parts of the BLACKSheep code uses peripheral interfaces located on the DEV- and EVALboards also provided by Bluetechnix.

2.1 Overview

The core modules are shipped with a basic version of the BLACKSheep software to evaluate the core modules and EVAL/DEV boards. A simple bootloader is located in the flash that loads and executes the BLACKSheep code after power on or hardware reset. Than the BLACKSheep changes the coreclock and systemclock frequencies and loads the most important driver. After the welcome message the BLACKSheep is ready for a command input.

2.2 Boot Strategy

For booting the Blacksheep the Boot option on the core modules has to be set to BM1=0 and BM0=1. That means Boot from external flash. The Blackfin core boots the first application in flash using the Blackfin internal bootloader. In our case this is the BLACKSheep boot loader. The BLACKSheep bootloader executes from internal instruction memory from the Blackfin. Ones started, the loader is searching for the second application in flash. (The first one is the loader itself. The second one is the BLACKSheep code.) After initializing the SD-RAM the BLACKSheep code is loaded into the SD-RAM and then the loader jumps to the start address of the BLACKSheep 0x0, the first address from SD-RAM. Refer to the Analog Devices loader manual to get more information about the Blackfin boot strategy and the structure of *.dxe, the VDSP++ executables.

2.3 Development Environment

The entire BLACKSheep was developed with the VDSP++ development environment provided by Analog Devices. It is recommended to use the VDSP++ to include BLACKSheep code into your own applications, but the BLACKSheep is written in ANSI C, so it is possible to use this code also on other platforms.

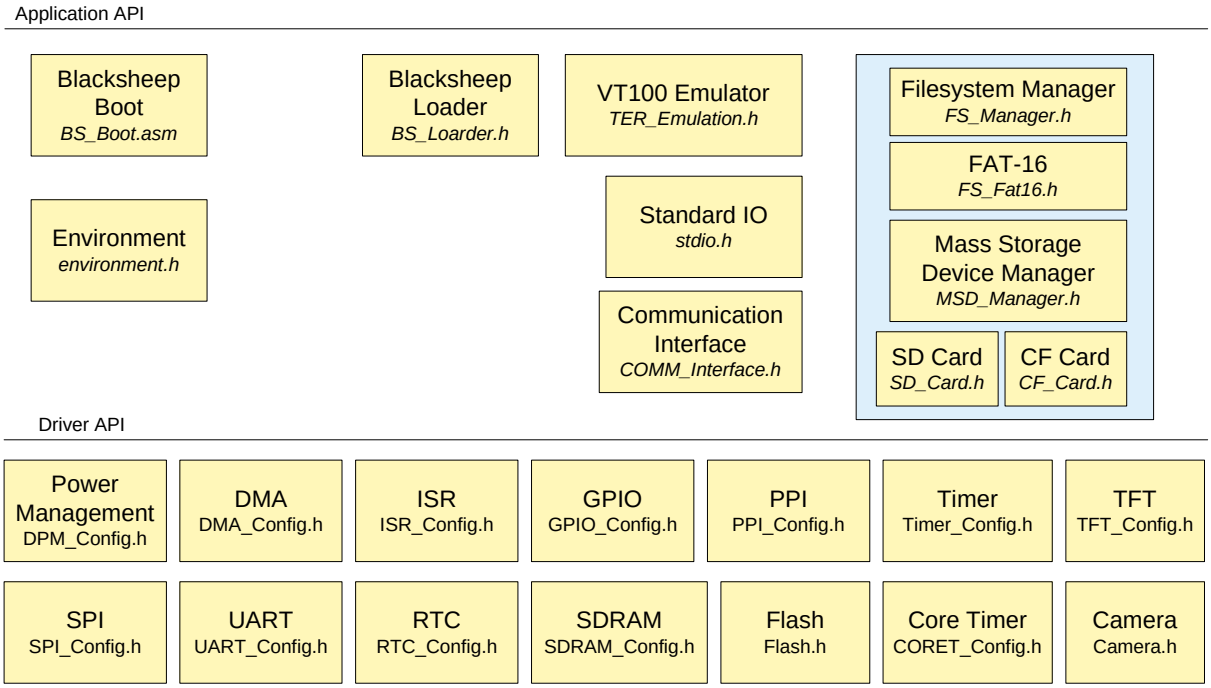
2.4 VDSP Flasher

The VDSP Flasher executable is needed for flashing the BLACKSheep, if you have overwritten the actual version in the flash or to flash your own standalone application on the core modules CM-BF5xx. The flasher was developed to work with the VDSP++ flash programmer tool. You need an Analog Devices JTAG interface to use this flasher tool.

Be aware to use the right flash driver for the appropriate module. There are two different executables, one for the CM-BF561 and one for the CM-BF533.

3 Module Specification

3.1 Overview



3.2 IO Functions

3.2.1 COMM_Interface

The COMM_interface.c provides the connection between the lower level I/O interfaces like the SPI and the UART as well as the ‘stdin’ and ‘stdout’ interfaces used by the terminal emulation or the ‘printf’ function from stdio.c.

3.2.2 Conio

The conio.c provides the functions to read characters from ‘standard in’. An incoming character is handled by the terminal emulation (TER_Emulation.c). If a command is already in execution the terminal emulation passes the character to the standard in buffer where they can be read with a getchar() call. The function kbhit() tells you if there are characters to be read in the stdin buffer. These functions can be used within a project linked with the BALCKSheep linker file. Refer to the BLACKSheep_DeveloperGuide manual to see how to develop projects that uses BLACKSheep functions.

3.2.3 Stdio

In the stdio.c are located the ANSI C file IO and the printf() function used by the BLACKSheep code. You can use this function also within your own projects if you use the special linker file provided on the Support-CD and you use the BLACKSheep as loader for your application. With this special loader file the printf() and standard IO file functions are statically linked into your project.

3.3 File System

3.3.1 File System Manager

The file system manager manages different file systems. Currently only FAT16 is supported. The user interface of the file system manager provides a basic set of file manipulating functions. The stdio.h links to this basic functions and provides the standart ANSI C file I/O functions.

3.3.2 FAT 16

The FAT16 driver is a basic version only for single threaded machines. Only one file per time can be open. It provides all basic file manipulation functions like open file, close file, write a character, read a character and delete file. It also provides functions to navigate in the directory tree like change directory and find files in directory. The FAT16 driver currently not supports writes in subdirectories.

3.3.3 Mass Storage Device Manager (MSD)

The mass storage device manager manages the low level driver of all devices where file systems are located like SD-card, CF-card, RAM-drive and FLASH-drive.

The interface to the lower level hardware driver can be the SPI or standard memory I/O writes and reads or other parallel or serial interfaces. The interface to the file system consists of a set of function pointers. You have to call the MSD_RegisterDevice function to be able to use a certain mass storage device within the BLACKSheep. This function registers the I/O-functions of the low level driver so that they can be used by the file system drivers.

The following functions have to be provided by a low level mass storage driver so that it can be used with the BLACKSheep:

bit (*Hard_read_open)(unsigned long pa_nSector); //Hard_read_open(sector)

unsigned char (*Hard_read_byte)(void); //Hard_read_byte()

void (*Hard_read_close)(void); //Hard_read_close()

bit (*Hard_write_open)(unsigned long pa_nSector); //Hard_write_open(sector);

void (*Hard_write_byte)(unsigned char pa_cByte); //Hard_write_byte(byte)

void (*Hard_write_close)(void); //Hard_write_close()

The following example shows the registration of the SD-card driver:

MSD_RegisterDevice ("sd",

nCardSize,

&cDeviceNr,

cSDSlot,
SD_ReadOpen,

SD_ReadByte,

SD_ReadClose,

SD_WriteOpen,

SD_WriteByte,

SD_WriteClose);

The I/O functions are stored in a function table in an array identified by the shortcut of the device. In this case “sd”.

3.4 Storage Devices

3.4.1 SD Card Driver

The SD-card driver uses the SPI low level driver (SPI_Config.c) to communicate with the SD-card. Because the SD-card needs a continuous CS signal asserted during a read or write operation the driver must ask the SPI driver, for an exclusive access to the SPI calling SPI_GetExlAccess(). Once the SD-card driver obtained the exclusive control over the SPI interface all other SPI operations are locked until the SD-card driver returns the control to the SPI driver calling SPI_FreeExkAccess().

Files associated: SD_Card.c, SD_Card.h

In SD_Card.h there are to important parameters that has to be adapted on the hardware:

SD_MAX_NR_OF_CARD_SLOTS: describes how many SPI CS signals should be reserved for communication with an SD-Card. SD_FIRST_POSSIBLE_SLOT and SD_LAST_POSSIBLE_SLOT describes the first and the last SPI CS signal witch is used for SD-Card communication. They must be contiguous. You have to tell the SPI driver setup function how many SD-slots you want to use, so that the SPI driver creates the appropriate number of slave buffers.

For the upper interface to the mass storage device driver take a look to section 3.3.3.

3.4.2 CF Card Driver

The CF-card driver is currently in development.

3.5 VT100 Emulator

The VT100 emulator provides a set of basic functions to communicate with standard vt100 compatible terminal software. The BLACKSheep software itself uses only a few functions to communicate with the terminal. These functions are located in TER_emulation.c and its procedures communicate with the communication line over the COMM_Interface.c. The COMM_Write() function calls the appropriate lower level communication line functions depending on witch communication hardware is selected via defines. Currently only UART and SPI are supported.

3.6 Hardware Drivers

3.6.1 UART

Files associated: UART_Config.c, UART_Config.h

The UART is interrupt handled. The UART RX, TX and error interrupts are mapped to the event vector 10. Only one Interrupt Service Routine (ISR) handles all possible UART interrupts. The ISR is defined as reentrant so it can be interrupted by another interrupts source with higher priority. The driver creates one 256 byte buffer for receiving data and another buffer with the same size for transmitting characters. The ISR first checks witch source causes the interrupt. If it is a receive interrupt checks for a buffer overflow and than puts the character in the receive buffer if no callback function was registered. Otherwise it calls the callback function. If an error occurs an error handling is forced calling EH_SetLastError() and EH_ForceErrorHandling().

UART_Putchar() is used to fill the buffer. If the UART is currently not transmitting data, signalled by t_disabled, the UART_Putchar() fills directly the UART_THR (UART transmit holding register).

UART_Getchar () reads a character from the receive buffer.

3.6.2 SPI

Files associated: SPI_Config.c, SPI_Config.h

The SPI Interrupt Service Routine (ISR) is similar to the UART. The main difference is that the SPI uses 16 bit buffers instead of 8 bit, because the SPI interface can handle either 8 or 16 bit transfers. Another difference are the transmit buffers. Acting as master, the SPI can handle up to 7 slaves. Calling the SPI_Setup () function you can say how many slaves are connected to the SPI. For each slave the SPI driver creates an appropriate transmit buffer. You have to tell the SPI_WriteData () function in witch transmit buffer you want to read. In the current version of the SPI driver, if you say to the driver for example to open two slave channels, the channel 0 and 1 corresponding to the CS signal 1 and 2 are initialized. For tree slaves CS1,2

and 3 and so one. Take care of this if you design special hardware that uses the SPI interface in conjunction with the BLACKSheep SPI driver.

If you call the SPI_Setup () function with 0 slaves, the SPI is acting as slave itself. Only one transmit buffer is used in this mode and the baudrate can be set to 0 because the SPI is clocked externally by the master and so the clock signal becomes an input.

SPI_ReadData () is used two read a 16 bit value from the SPI. If you use the SPI in 8 bit mode the upper 8 bits are zero.

SPI_WriteData () is used two write a value in the appropriate transmit buffer corresponding to the slave that must receive this data.

3.6.3 Core Timer

Files associated CORET_Config.c, CORET_Config.h, Timer_Config.c, Timer_Config.h

The core timer is a hardware timer from the blackfin that runs at core clock speed. The BLACKSheep uses this timer to provide a one ms tick witch controls most of the timing critical code, like timeouts. The Sleep () function also uses this ms ticker just like the LED flashing functions. Once enabled, calling the CTGlobalTimebaseSetup () the coretimer can't be used by another entity. In CORET_Config.c there are a few functions to manipulate directly the coretimer registers. Never use these functions after the global time base is enabled. Be aware that the setup functions for the camera uses the coretimer to setup the serial control interface, so always call CTGlobalTimebaseSetup () after initializing the cameras.

3.6.4 SDRAM

Files associated: SDRAM_Config.c, SDRAM_Config.h

Call SDRAM_Setup to initialize the SD-RAM on the CM-BF5xx modules.

3.6.5 Flash

Files associated: Flash.c, Flash.h

The files contain a series of functions to manipulate the flash mounted on the CM-BF5xx. The most important are routines for erasing sectors, erasing the entire device, programming a 16 bit value into the flash and checking if the entire flash is empty. Flash.c does not support commands for reading the flash. You can do this with simple pointer operations. Be aware that the flash supports only 16 bit accesses.

To use the FLASH memory you have to enable the appropriate asynchronous memory bank. In case of the CM-BF533 this is only bank 0. In case of a CM-Bf561 these are the bank 0 and the bank 1. You can enable the banks with AMI_Setup(), a function provided by the AMI_Config.c. Be aware to enable the banks for 16 bit accesses in case of the CM-BF561.

3.6.6 PPI

Files associated: PPI_Config.c, PPI_Config.h

The functions in PPI_Config.c allow configuring the Parallel Peripheral Interfaces from the core-modules. The procedures supports the most common features of the PPI like type of interrupt (frame wise or block wise) the transferwidth (8, 16, or 32 bit), and the nr of elements divided by x and y for two dimensional DMA. The PPI_Config.c also contains the ISR for the PPI interrupt.

3.6.7 DMA

Files associated: DMA_Config.c, DMA_Config.h

The support for the DMA contains the configuration for the PPI-DMA channel and for memory DMA transfers. For memory DMA transfers you can setup a DMA-channel for a special source and destination address. You can choose the width of data transfer (8, 16 or 32 bit). To determine the end of a memory transfer there exist a function called DMA_MDMAWaitForTerminate (). Currently interrupts are not supported for memory DMA transfers.

3.6.8 Power Management

Files associated: DPM_Config.c, DPM_Config.h

Currently only changing the coreclock and systemclock frequencies is supported. For future versions also changing the core voltage will be possible.

On the dualcore variants of the Blackfin family one core must be stay in “idle” mode then the second one can change the frequency. The DPM_SetDspClocks () for the dualcore variant is reentrable. This procedure has to be called on both cores. The function communicates between the cores over the shared memory. The first core entering this function sets a flag into shared memory to communicate to the other core that he is ready to change the PLL conditions and to force the second core to go idle. If the second core enters the function he sets another flag into shared memory telling the other core that he goes idle and he is ready for changing the PLL state.

3.6.9 Asyncronous bus interface

Files associated: AMI_Config.c, AMI_Config.h

The function AMI_Setup () allows to setup the asynchronous memory banks of the Blackfin. There is a little difference between the CM-BF533 and the CM-BF561. AMI_Config.h contains the appropriate defines for calling AMI_Setup (). You can enable either the first bank the first two banks, the first three or all banks but not crosswise, for example bank 0 and bank 2 or other combinations.

3.6.10 Camera

Files associated: CAM_Sccb.c, CAM_Sccb.h, CAM_Image.c, CAM_Image.h

CAM_Sccb.c and CAM_Sccb.h contains the control functions to communicate with the camera over the SCCB bus. This is a serial to wire interface similar to the I^2C. First you have to initialize the bus calling the CAM_Setup () function. This procedure needs the actual coreclock frequeny and the PF flags used for the SCCB (SIO_C, SIO_D) and the Power Down pin. The CAM_setup () calls the SCCB_open () function and initializes the SCCB. After this the function is trying to identify the camera connected. If a camera was found the setup function sets the appropriate camera registers for a resolution of 320*240 by 30 frames per second. After this the camera is usable and you can get a picture with GetRGBPicture () located in CAM_Image.c. The image from the camera is saved in YUV format and converted in RGB. Choose the format for the RGB with the defines in CAM_Image.h, either to view the image with the VDSP++ image viewer or in a windows bitmap compatible version. In CAM_Image.h you can find the defines on witch memory location the YUV and the RGB picture will be stored. The place where the picture is stored in memory is defined in CAM_Image.h: CAM_FRAME_BUFFER_START describes the start address where the PPI begins to store the YUV picture. CAM_RGB_BUFFER_START means the start address of the image converted to a 24 bit RGB.

3.7 Utils

3.7.1 Environment

Files associated: Enviroment.h

This files contains the most important defines. It defines for example the target platform (CM-BF533 or CM-BF561) or the core and system clock frequencies. The type of crystal, the target basis board and so on. This file should be located within the directory where the project file resides. In most driver files (located in the directory Blackfindriver) and many other files this file is included.

3.7.2 Simple Types

Files associated: simpletypes.h

This file contains a few type definitions for the most common byte oriented types.

3.7.3 Register Definition

Files associated: regDef.h

This includes the pointer register and normal register definitions for the appropriate Blackfin processor.

4 Particularities for the CM-BF561

The core of the module CM-BF561 is the Blackfin BF561. This is a dual-core version with two BF533 cores together. The BLACKSheep takes care of these particularities of the BF561. For this reason there exists a special version of the BLACKSheep for the CM-BF561. The CM-BF561 variant supports also communication between two cores via a shared memory. This is used for example to communicate between the cores during changing the clock frequencies because one changes the frequencies and the other has to stay in idle mode. The shared memory is also used to create a global 1ms tick, accessible from both cores. You can enable or disable this feature with a #define directive in dualcore.h

4.1 Boot Strategy

The CM-BF561 uses the same boot strategy like the CM-BF533. The internal blackfin bootloader loads the BLACKSheep loader into the internal instruction RAM. The BLACKSheep bootloader for the CM-BF561 then loads the executable for both cores. The BLACKSheep itself will be executed by CORE A of the CM-BF561. The code of the BLACKSHeep resides in SD-RAM beginning by 0x0. CORE B executes a tiny program that toggles the LED(s) from internal instruction memory of CORE B, using the global 1ms tick to provide the blink frequency.

4.2 Low Level Drivers

On a BF561 every interrupt from every source can be handled by either CORE A ore CORE B. To support this feature, the low level driver of the BLACKSheep differs from that of the CM-BF533. Almost every low level setup function for the appropriate hardware engine has a additional parameter for the BF561. With this parameter it is possible to select witch core has to handle the interrupt of this hardware entity.

A speciality is the DPM_SetDspClocks function in DPM_Config.c. This is a reentrable function that communicates over the shared memory. This allows changing the clock frequencies during normal program execution. The core entering first the DPM_SetDspClocks waits until the second core goes idle. Afterwards it changes the coreclock and systemclock frequencies.

5 FAQ

6 Revision History

2004 12 20	First release V1.0 of the Document
2005 05 03	Revision 1.1

A List of Figures and Tables