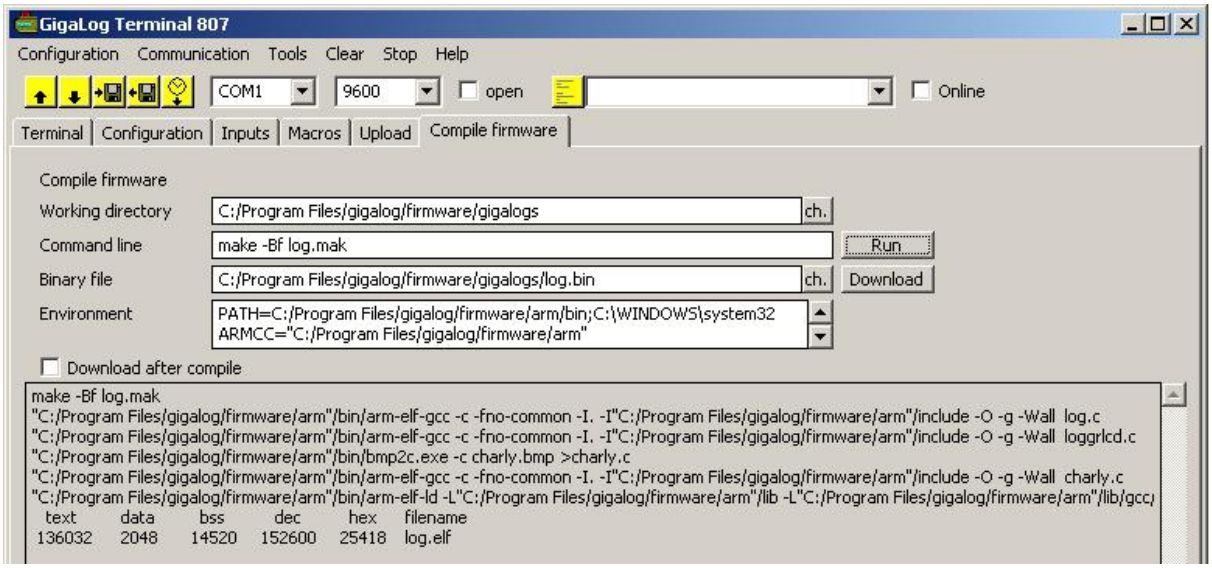
GigaLog S Firmware Programming in C

This manual shows, how to program the GigaLog S board in C.
Programming your own firmware is only necessary, when the supplied firmware can not be configured for your special application. You need experiences in C programming to change the firmware

Table of Contents

Version 1705, Mai 2017.

GigaTerm Programming Interface



	GigaLog S
Working directory	./firmware/gigalogs
Command line	make –f log.mak
Binary file	./firmware/gigalogs/log.bin
Environment	PATH=./firmware/arm/bin ARMCC="./firmware/arm"

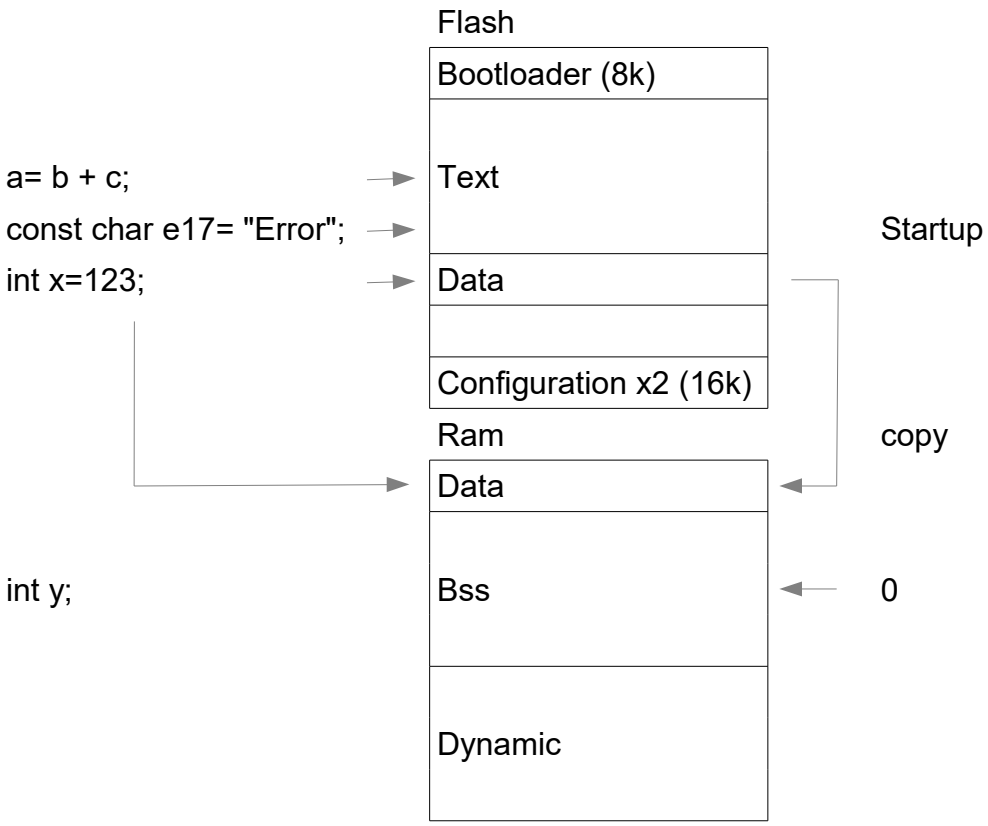
GigaTerm compiles in this tab the firmware source log.c using the log.mak file and produces log.bin.
You may then load this file into the target flash memory.

The compiler used is the GNU compiler chain, from http://gcc.gnu.org/

a)
install the complete package in a directory without spaces " " in the path.

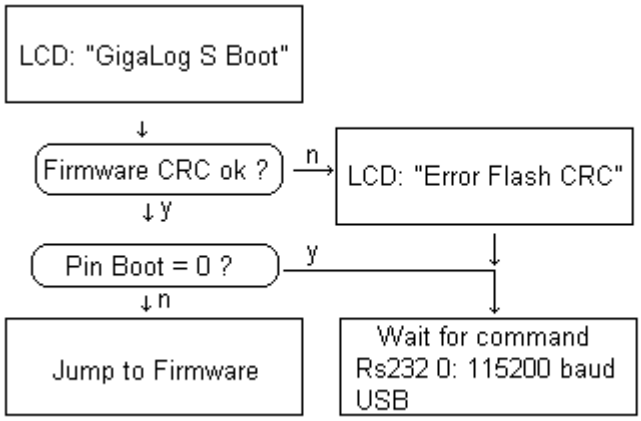
Or
b)
If you insist on having spaces in the path:
You must install Cygwin on your computer before compiling: See www.cygwin.org.
During compilation, an error message may arise about incompatible cygwin libraries on your computer.
Just read the error message and proceed as proposed.

The command line "make -Bf" always rebuilds all from the source files.
If you change this to "make -f", only needed steps will be performed. This is faster.


Flash
Bootloader (8k)
a= b + c;
Text
const char e17= "Error";
Startup
int x=123;
Data
Configuration x2 (16k)
Ram
copy
Data
int y;
Bss
0
Dynamic

Boot loader

A boot loader is installed in the first 8 k of the Flash program memory.
This program is loaded into the memory at the factory. It will not be replaced.
After Reset
The boot program verifies the CRC of the application program (data logger)
When the CRC is correct, the boot loader starts the application program.
If the CRC is not correct, or
If the boot signal on the board beside the battery is held to GND,
The boot program waits for a command on STDIO.
The command "z" clears the configuration in the Flash memory.
The command "go" starts the application.
You can now download a program into the flash memory by GigaTerm.
Each application shall always recognize the command "dl" to re-enter the boot loader to load another application.



First Help

When your application does no longer respond, or your configuration does no longer allow you to work.
Place a wire from GND to the boot pad on the board beside the battery.
Reset the board.
The board writes "Download S7" on the port RS0.
When you are on USB, enter "dl".
Enter "z" to clear the configuration.
Download a new application.
Enter "go" to start the application.

During this operation: Remove modems, Gps, or any devices, that can send messages to the board !

Files in the distribution

Directory	File	
./firmware/gigalogs	hello.c	Demo: program
	hello.mak	Demo: makefile
	lcd.c	Demo graphic: program
	lcd.mak	Demo graphic: makefile
	lcdToolkit.h	Graphic lcd toolkit header file
	lcdToolkit.c	Graphic lcd toolkit source
	gigalogSlog.h	Log: definitions header file
	log.c	Log: main program
	log.mak	Log: makefile
	loggrlcd.c	Log: graphic
	charly.bmp	Log: image
	machina.bmp	Log: image
	logPerso.c	Log: personal task
	loggrlcdPerso.c	Log: personal graphic page
./firmware/arm/include	gigalogs.h	Gigalog S: board and library header file
./firmware/arm/lib	gigalogs.ld	Gigalog S: load script for link
	gigalogs256.ld	Gigalog S: load script at91sam7x256
	gigalogs512.ld	Gigalog S: load script at91sam7x512
	gigalogscrt.o	Gigalog S: startup file
	libgigalogs.a	Gigalog S: library

Note: Gigalog S uses the at91sam7x512 microcontroller
Earlier boards used the at91sam7x256. Replace the gigalogs.ld by the gigalogs256.ld either in the library, or in the makefile for these boards.

Debug support in the log firmware

Command	
xxa	Adc info
xxbl	Disk block cache dump
xxcpu	Cpu info
xxdk	Disk info
xxm <a> <lng>	Memory dump
xxmb <a>=<b>	Write single byte to memory
xxmw <a>=<w>	Write single word to memory
xxmf	Memory free list
xxp	Parallel ports
xxreset	Reset
xxsim <n>	Set a0 to simulation mode <n>mV
xxt	Task info
xxz	Configuration memory table

First steps: Recompile firmware, load the compiled firmware

GigaTerm tab: Compile Firmware.
Working directory: Click on Ch, open ./firmware/gigalogs/log.mak
Click on Execute: Compile the program.
If there are any errors, see chapter above Programming Interface about cygwin programs.
The compilation shall terminate without any errors and show the following lines:

text	data	bss	dec	hex	filename
136000	2000	14000	152000	25000	log.elf

Binary file: Click on Ch, open ./firmware/gigalogs/log.bin
Click on Download.
Le program will be loaded into the Flash memory of the GigaLog S board within a few seconds.
When the download is finished, the program starts automatically.
Verify, that the program runs as expected.

Compile and load a small demonstration program

The program ./firmware/gigalogs/hello.c is a small demonstration program, which respects all rules and toggles the LED
Working directory: Click on Ch, open ./firmware/gigalogs/hello.mak
Click on Execute: Compile the program.
Binary file: Click on Ch, open ./firmware/gigalogs/hello.bin
Click on Download.
The program accepts one single command from the serial port, or the USB:
"dl" to load another program.

STDIO: standard input / output

The application program and the boot loader work with a standard input output, called STDIO.
The STDIO can be one of

- RS0 serial port Rs232 on IDC RS0 (0)
- RS1 serial port Rs232 on IDC RS1 (1)
- RS2 serial port Rs485 (2)
- USB (3)
- Graphic LCD terminal emulation (4)

After Reset STDIO is on RS0.
When the application or the boot loader receives a character from one of these ports, it switches STDIO, and will from now on send messages to this port.
When the boot loader calls the application, or the application calls the boot loader, it passes the current STDIO as parameter.
The function Printf() writes to standard output.

What an application program has to respect

The application program must work on STDIO to receive commands.
It must recognize the command "dl" download, and jump to the boot loader to load another application.
The program shall create at least one task and start the multitasking kernel.
If not, the application cannot use the file sub system and other modules.
Each task must regularly allow other tasks to run.
This can be done by calling muxSwitch(), or at the end of time slot, if the task is preemptive.

The demonstration program ./firmware/gigalogs/hello.c is an example for a program, which respects these rules.

Mutex Multitasking kernel

The multitasking kernel allows running several tasks apparently simultaneously. The program main() may create one or several tasks, and then call muxStart to start the Mutex kernel. MuxStart will not return. Tasks may create other tasks. A task may exit.
The function muxCreateTask() creates a new task.

muxTcb *muxCreateTask(char *name, void (*fct)(int p), int p, int stacksize, int timeslice);
Name: Name for muxDump.
Fct: Name of the function, the task shall execute.
P: Parameter to be passed at the function.
Stacksize: Size of the memory reserved for the stack of the task. 2000 is a good starting value.
Timeslice: Time in milliseconds for each time slice of this task.

If this time is not zero, the task is preemptive. The kernel may at any time stop the task and run another task.
If this time is zero, the task is non preemptive. The task must call muxSwitch to allow other tasks to run.
Comment: Make this time non-zero, but call muxSwitch in each principal loop when waiting.

Some modules like the file subsystem will not work when the Mutex kernel is not running.

Interrupt functions use another stack.
An interrupt function must not call directly or indirectly muxSwitch.

muxTcb *muxCreateTask(char *name, void (*fct)(int p), int p, int stacksize, int timeslice)
Create a task

void muxStart()
Main program: Start Mutex kernel

void muxSwitch()
Dispatch current task, run another task.

void muxExit()
End of the task.

void muxDump()
Display all running tasks on STDIO. The command "xxt" of log.c calls this function.
For each task: Address of its task control block, part of the stack not used, which allows to redefine its stack size, time the task was running (very approximate).

Semaphores

En.wikipedia.org:

> A semaphore is a protected variable (an entity storing a value) or abstract data type (an entity grouping several variables that may or may not be numerical) which constitutes the classic method for restricting access to shared resources, such as shared memory, in a multiprogramming environment

volatile muxTcb *s;
Declaration of a semaphore.

Int Semaphore(volatile muxTcb **s)
Returns true, if the program could reserve the semaphore.
Returns false, if the semaphore was occupied.

void Semawait(volatile muxTcb **s)
Wait by calling muxSwitch() until Samaphore() returns true.

#define SemaFree(x) *x=0
Free a semaphore.

Background processing

Several works are placed into the background, either into a server task of the multitasking kernel, or into an interrupt function.
The main tasks do not have to handle these works any longer, and do not have to wait for input output operations to complete.

The access to the memory card by functions like fsWrite works on a memory cache that keeps several blocks of the disk in RAM. A server task transfers data from the cache to the memory card, and vice versa.

The driver of the character LCD is based on a buffer lcdbuf[32]
The application program only writes into this buffer.
The driver transfers the buffer regularly to the LCD, using an interrupt function.

The internal analogue to digital converter, the voltage of the power supply.
Based on an interrupt routine, which is called each millisecond.
The interrupt routine reads out all 8 analogue inputs.
The server reads out all ADC each millisecond and puts them into a sum.
After 10 milliseconds, the server calculates the average from the sum.
The function iadc just reads out this sum and gets the last average value over 10 ms.

ADCserver

The ADC on the GigaLog S board, ADS1258 scans automatically all 16 inputs all the time.
An interrupt signals that a result of a conversion is available.
The Log program includes a server, which keeps the data in memory.
The server calculates a sum for data storage on the memory card.
It also keeps a second sum to calculate an average of over some milliseconds.
For each input an average sum over some milliseconds is always available in memory.
The function Manalogin(int ch) just reads out this value.

Log Firmware personal task

The GigaLog firmware includes an additional task to integrate additional functions into the firmware.
The personal task has a 20 bytes long configuration area in Ram.
This area will be transferred automatically to Flash memory after any changes.
The file logPerso.c will be included in the firmware.
It has an empty personal task (from #if 1 to #endif).
It has a personal task as example (from #if 0 to #endif). This task toggles a relay rl0 according to the input a0 and two parameters stored in the configuration.

Functions in logPerso.c:

void pInit(char *pconf)
This function will be called from the main program.
The parameter is the address of the configuration area.
This function may create a task

void pConfZero(char *pconf, int restore)
This function will be called when the configuration is set to zero.
It will be called with restore=0 before, and with restore=1 after clearing the configuration.
It can initialise the personal configuration

void pConfDump()
This function shall dump the configuration parameters using Printf.

int pConfSet(unsigned int ch, char *s)
This function will be called, when the user types a command pr[xxx][<ch>]=<string>
S points directly after the pr.
It may use this to execute commands, or to change the configuration.

void p1second()
This function will be called from a task each second.

void p1msecond()
Attention: Interrupt level
This function will be called from each microsecond.

float pChannelin(unsigned int ch)
Return raw value for personal channel. See configuration a<n>=vp

void pA2dkStart(void *fp, cl_time *ytime)
Called before writing a line to the analogue data file, second changed.
Fp file descriptor for eprintf()

void pA2dkStartM(void *fp, cl_time *ytime)
Like above, same second.

void pA2dkEnd(void *fp, cl_time *ytime)
Called after writing a line to the analogue data file.
Fp file descriptor for eprintf()

void pA2dkValue(void *fp, int ch, aval v)
Called before writing a value to the analogue data file.
Fp file descriptor for eprintf()
Ch channel; V raw value, use raw2real() to convert to real value.

void pLogSummary()
Called after writing frame start to data log file, before writing data.

float pLog(unsigned int ch, int amm)
Calculate value for a personal channel a_{ch}=vp. amm=0: average, -1: min, +1: max

void pLogRestart()

Called after writing all data to data log file, before writing end of frame.

int pRsGotChar(int ch, char x)
Attention: Interrupt level
Called when receiving a character on a serial line.
Ch: 0=Rs0, 1=Rs1, 2=Rs2, 3= USB
X: character.
Returns: !=0: ignore character x

void pRsGotFrame(int ch, int fp)
Called after having received a frame on a serial line in data mode
Ch: 0=Rs0, 1=Rs1, 2=Rs2, 3= USB
Fp file descriptor data file for fsPrintf()

int pRsGotCmd(int ch, char *s)
Called after having received a line on a serial line not in data mode
Ch: 0=Rs0, 1=Rs1, 2=Rs2, 3= USB, 4= graphic LCD terminal
S: command line
return !=0 ignore cmd-line

void pTouchscreen(int x, int y)
Called when somebody touched the screen

int pRqProtocol(int mode, int ch, int rqRs, int ach0, int achs, char *rbuf, volatile int *rbufcnt, volatile int *tic, int tout)
allows to add a personal protocol to talk to a remote sensor or remote board over Rs232

Variables from log.c

typedef unsigned long Ttic;

cl_time ctime
Current date and time

volatile Ttic ctimeTtic
universal 1 sec tic since 1.1.2000

volatile Ttic ctimeMtic
1 millisec tic since midnight

extern volatile Ttic64 ctimeMMtic
iuniversal 1 ms tic since 1.1.2000

Functions from log.c

int STDOUT0(int stream, char c)
int STDOUT1(int stream, char c)
int STDOUT2(int stream, char c)
int STDOUTUSB(int stream, char c)
int STDOUTLCD(int stream, char c)
File pointer for eprintf() to Rs0, Rs1, Rs2, USB, graphic LCD terminal.

int cmdExec(char *cmd, int silent)
Execute a command line.

float Manalogin(unsigned int ch)
Return current value of analogue input.
See ADCserver. See configuration, ax= parameter <m samples>

aconf *ch2aconf(unsigned int ch)
Calculates configuration entry for a channel

float raw2real(aconf *ac, float v)

Convert a raw value into a real value.
Ac: configuration entry
Ch: Analogue input channel.

int stopGo(int why)

why:	STOPGOGETGO	return state.
	STOPGOSETGO	set state to Go, return state.
	STOPGOSETSTOP	set state to Stop, return state.

Returns state: 0= Stop, 1= Go.

Fs file subsystem

The file subsystem allows access to files on the memory card.
The file subsystem is based on a buffer cache.
Mutex must be running, since it uses a server task to copy data from the buffer cache to the memory card, and vice versa.

It works on FAT16 and FAT32 file systems.
The fsFormat function creates a FAT32 file system
It handles long filenames, and subdirectories.
The current working directory is stored in the task control block of each task.
Many function use as first parameter fd.
Fd is the file descriptor from fsOpen()
Most functions return a negative value in case of an error
fsError() translates this value into a short ASCII message

A block in the buffer cache is said to be dirty, if it was changed, and is not yet written back to the disk.
The cache is called SYNC, when there is no dirty block.
Blocks in the cache can be locked. A locked block can not be replaced.
One block will be locked in the cache for the working directory of each task, if it is not the root.
One block will be locked for each open file.
Another block will be locked during a read or write operation.
You must reserve enough blocks in the cache.

The file subsystem is save against multiple accesses by several tasks.
It uses internal semaphores.

<path>= [<drive>][<dirname>/]*<filename>
<drive>= c: sd-card on board
<drive>= d: sd-card external

void fsInit(int files, int blocks, int param)
Initialise file subsystem. Do not call this function twice.
Max number of files. Each file costs about 32 bytes in RAM.
Number of blocks in buffer cache. Each block costs about 530 bytes of RAM.
Param: 1: no read after write, no retry

int fsOpen(char *path, int flag)
Open a file
Returns the file descriptor.
In case of an error, the return value is negative.
flag

FSO_READ	The file must exist
FSO_CREATE	If the file does not exist, create the file.
FSO_RDONLY	Only used with FSO_CREATE

int fsRead(int fd, char *buf, int lng)
Reads from the file into a buffer.
Returns the number of bytes transferred.
Returns 0 when the read arrives at the end of the file.
In case of an error, the return value is negative.

int fsWrite(int fd, char *buf, int lng)
Writes from a buffer into the file.
Returns the number of bytes transferred.
In case of an error, the return value is negative.

long fsSeek(int fd, long off, int whence)
Sets the offset in the file for subsequent read or write operations.
whence

SEEK_SET	offset is relativ to the beginning of the file
SEEK_CUR	offset is relativ to the current offset
SEEK_END	offset is relativ to the end of the file.

Returns the new absolute offset in the file.

long fsTell(int fd)
Returns the current absolute offset in the file.

int fsGets(int fd, char *buf, int lng)
Reads a line into the buffer.
A line ends with a \r, a \n, or a \r\n
These trailing end of line characters are discarded.
Returns >0 for success.
Returns 0, when the read arrives at the end of the file.

int fsClose(int fd)
Close the file
Returns 0 for success

int fsUnlink(char *path)
Remove the file from the filesystem.
Returns 0 for success

int fsRename(char *frompath, char *topath)
Rename file
Returns 0 for success

int fsDir(char *path, int (*fsDfct)(char *nm, char *dt, long l, int priv), int privat)
Pass through the current working directory and call fsDfct for each entry.

int fsPrintf(int fd, const char *fmt, ...)
Printf to file

int fsPutchar(int fd, char c)
Write a single character to the file

int fsFormat(char *n)
n= [<drive>][<volume id>]
Format the memory card.
Install a FAT32 file system on the card.

int fsMakeDir(char *path)
Create a directory

int fsChangeDir(char *path)
Change working directory of the task.

long fsInfo(int whence)
Returns information about the disk and the file subsystem
whence=

fiDKSIZE	size of the disk in k bytes
fiDKFREE	free space on the disk in k bytes
fiLASTERROR	last error code for fsError()
fiSYNC	True, when all blocks in the cache are written to disk
fiDKOK	file subsystem and the disk are ready
fiDKINFO	display disk information on standard out

char * fsError(int i)
Returns a short ASCII message for an error code.
If i is 0, takes fsInfo(fiLASTERROR)

void fsPower(int n)
Informes the file subsytem about the state of the power supply
When power is not PowerUp, the server writes all dirty blocks to the disk.
n=

- PowerUP
- PowerLOW
- PowerDOWN

void fsSignal(int n)
The main program can provide this function.
The server calls it to inform the main program about the state of the server

fsSYNC	All blocks in the cache are written to the disk
fsUNSYNC	There is at least one dirty block in the cache
fsREADOP	About to read from disk
fsWRITEOP	About to write to disk
fsOPEND	Read or write op ended.
fsIORETRY	Retry disk operation
fsIOERROR	Disk error

cl_time *fsGetTime()
The main program can provide this function.
It is needed to enter the correct time and date in the directory slot of a file.

Error numbers are negative

fEARG	Wrong argument
fEINIT	File subsystem not yet ready
fENOINIT	No previous call to fsInit()
fENODEV	No memory card
fENOFS	No File system on the memory card
fETOOMANYFILES	Too many files open
fENOENT	File not found
fEDIRFULL	Directory full
fEDISKFULL	Disk full
fEACCESS	Access denied
fEBADF	Bad file descriptor
fEFAT	Error in FAT
fEIO	Read or Write input output error
fERVERIFY	Read after Read comparison failed
fEWVERIFY	Read after Write comparison failed

void blDump()
Debug support.
Dump buffer cache on standard output.

void fsScanFat()
Debug support
Recalculate FAT free blocks count

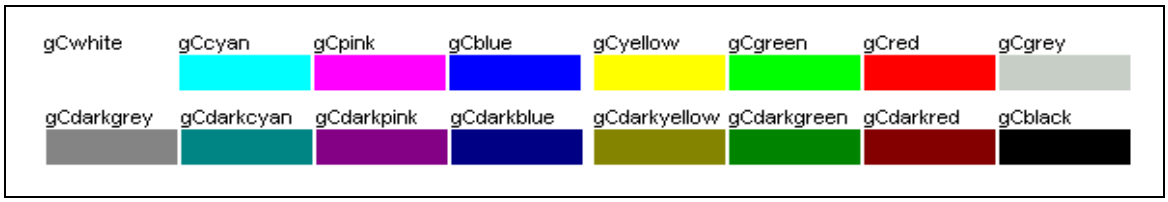
Graphic Display

Call the init function at the beginning
void glcdInit();
Only one task may write to the display at a time.
If more than one task write to the display, they must coordinate the access using a semaphore.

Colours

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rouge					vert						bleu				

Each pixel on the screen is represented by a 16 bit RGB word.
The file gigalogs.h already includes some colour definitions.



Display functions

void lcdClear();
void lcdPix(uint x, uint y, int color);
void lcdRect(int x, int y, int wd, int ht, int color);
void lcdCircle(int x0, int y0, int z, int color);
void lcdHLine(int x0, int y, int xn, int color);
void lcdVLine(int x, int y0, int yn, int color);
void lcdLine(int x0, int y0, int xn, int yn, int fr, int color);

Display text

lcdSetFont() selects the font and the colour.
lcdPrintf() lcdPrintfn(), lcdPrintfc() write text to the screen. lcdPrintf() ends at the end of the screen, lcdPrintfn() ends after wd columns, when wd is <0, text is right aligned. lcdPrintfc() like lcdPrintfn(), text horizontal centre aligned.
These functions return the number of columns of the text.
lcdCwidth(), lcdCwidthS(), and lcdChight() calculate the space needed by the text on the screen.

int lcdPrintf(int x, int y, const char *fmt, ...);
int lcdPrintfn(int x, int y, int wd, const char *fmt, ...);
int lcdPrintfc(int x, int y, int wd, const char *fmt, ...);
void lcdSetFont(int font, int color);
int lcdCwidth(char c);
int lcdCwidthS(const char *s);
int lcdChight();

Touch panel

void touchScrInit(struct tscrc &t) gives as parameter a non volatile memory region to store the touch screen parameters.
int touchScr(int *y);
int Button(int x, int y, int x0, int y0, int wd, int ht);
TouchScr() returns the value of the touch screen. Returns -1, if not activ.
Button compares the values x and y with the bouton x0, y0, wd, ht.
Example: The function paintButton() places a button on the display.
The program displays two buttons "Ok" and "Cancel".
It then waits for the user to push a button.

```
void
```

```
paintButton(int x, int y, int wd, int ht, const char *txt)
{       lcdRect(x, y, wd, ht, gCgrey);
        lcdSetFont('a', gCnoir);
        lcdPrintf(x+4, y+4, txt);
}

        int x, y;
        paintButton( 50,50, 60, 30, "Ok");
        paintButton(120,50, 60, 30, "Cancel");

        for (;;) {
                muxSwitch();
                x= touchScr(&y);
                if (Button(x, y, 50,50,60,30)){
                        Ok action
                }
                if (Button(x, y,120,50,60,30)){
                        Cancel action
                }
        }
```

Fonts

The library libgigalogs.a contains two fonts

id	Font	Size
a	Arial	14
s	Arial	8

lcdSetFont() selects a font by its ID.
A font only contains standard ASCII characters, neither é nor ä.

Create your own fonts

You may write your own file of fonts that replaces the file in the library.
Use Paint to create an image, create a text zone, choose a font and paste the following text into it.

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

The text must fit to the left and to the top. Remove all empty space at the right and at the bottom.

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Create a file fonts.c

```
#include <gigalogS.h>
struct font {
        char     id;                    // identifier
        uchar ftHight;                  // hight
        unsigned int ftXsize;           // Ctable x-size
        const uchar *ftCtable;          // Ctable
        const int *ftXpos;              // var font: x pos
};
#include "fonts.txt"
```

The file .mak for two fonts font1.bmp, and font2.bmp

```
BMP2C=$(BIN)bmp2c
fonts.o: fonts.c fonts.txt
        $(CC) fonts.c
fonts.txt: font1.bmp font2.bmp
        $(BMP2C) -f a=font1 b=font2 >fonts.txt
```

Include fonts.o into your main program during the link
Fonts from libgigalogs.h are no longer available.
Identify font1 by 'a' and font2 by 'b'
In the main program choose the first font in black by

```
lcdSetFont('a', gCnoir);
```

Display an image

Three possibilities

- Create a 24 bit colour Bitmap. Compile it, using Bmp2c into a 16 bit per pixel format. Put it into the Flash memory. Use lcdImage to display it on the screen. Expensive in Flash memory. Good to display parrots and Mona Lisa.
- Create a 16 colours Bitmap. Compile it, using Bmp2c into a 4 bit per pixel format. Put it into the Flash memory. Use lcdImage to display it on the screen. Economic in Flash memory. Good enough for technical drawings.
- Create a 16 colours Bitmap. Put it on the memory card. Use lcdFimage to display it on the screen. Slow, only good for demonstration and test purposes.

Program bmp2c converts an image into a C compiler compatible format.
The make file for the image in charly.bmp:

```
BMP2C=$(BIN)bmp2c
charly.c: charly.bmp
        $(BMP2C) -c charly.bmp >charly.c
```

Compile charly.c and include it in the link in your program.
In the main program call the following function, to display the image.

```
void lcdImage(int x, int y, const u16 *info, const u16 *ip, info zoom);
```

Zoom= 2, 3, ... increases the size on the screen.
For instance:

```
extern const unsigned short charlyInfo[], charly[];
lcdImage(x, y, charlyInfo, charly, 1);
```

int LcdFimage(int x, int y, char *fname)
displays a 16 bit Bitmap file on the screen.
The function returns 0 if no error.

int
lcdF2Fimage(const u16 *inf, const u16 *ip, char *fname)
loads an image from the disk into the flash memory. Command gri <fname>.
The function returns 1 if no error.

Other functions

int lcdPrint(char *fname);
Copies the screen into a bitmap file on the memory card.

void Rolex(int x0, int y0, int rd, int color, int hr, int min);
Displays a clock on the display.

void lcdTouchAdjust();
Adjust the touch screen parameters by the user.

Other useful functions in loggrlcd.c

void TopPaintState()	Paint top line: state
void BottomPaint(const char *txt[], int ybutton)	Paint BottomLine: 6 buttons
void BottomPaintOk()	Paint Bottom Line: "Ok" Button
int BottomSelect(int tx, int ty)	Return pressed button
void KeyboardPaint()	Paint Keyboard of command Terminal
int KeyboardKey()	Read touch screen return keyboard key
void paintButton(int x, int y, int wd, int ht, int fr, int fcolor, int bcolor, const char *txt)	Paint a button
void PalettePaint(int x0, int y0)	Paint colour palette
void KeypadPaint(int x0, int y)	Paint Keypad 0..9
int KeypadSelect(int x0, int y, int tx, int ty)	Return pressed keypad button
int tscrDebounce(int *yy)	Read touch screen, filter input

Log Firmware Personal Page

The GigaLog S graphic firmware includes a personal page to integrate easily additional functions.
The file loggrlcdPerso.c will be included in the firmware.

It has an empty personal page (from #if 1 to #endif).
It has a personal page as example (from #if 0 to #endif).
This page is a small graphic animation
It has as second page a dialogue to change parameters of the animation.

LoggrlcdPerso.c has two global entries, needed by the loggrlcd firmware.

const char *grPersoButton= "perso";
The text of the button that opens the personal page.
If the variable is zero, no personal page.

int taskgrPerso(int l);
The graphic task calls this function, when the user selects the personal page.
The function loops calling taskgrWait(), while this page is active.
It does all painting on the screen.
It has to read the touch panel, and execute the functions, the user selects.
It ends, when the user selects another page, and returns the selected page.

Lcd Toolkit

The files lcdToolkit.h and lcdToolkit.c include definitions and sources of other useful LCD functions:

void pageEnterText(const char *nm, char *txt, int lng);
Page inputline, keyboard, to enter a text line, like a name.

void pageEnterInt(const char *nm, u32 *dat);
Page inputline, numeric keypad, to enter an integer constant.

void pageEnterFlt(const char *nm, float *dat);
Page inputline, numeric keypad, to enter a float constant.

void
pageGetDate()
Enter date and time to set real time clock

void pageTerminal();
Page command terminal, to send a command as over USB or Rs232.

void fileMan();
Displays the root directory on the left side.
The menu on the bottom allows to select files and enter into subdirectories.
Selecting a file may display a small file information on the right side:
Function fileManRight() to be defined by application.
Entering into a file may cause any operation on the file:
Function fileManFile() to be defined by application.

void menuPaint(const char **menu, int lng, int xl, int xr, int y);
Paints a menu.

void bottomLine(const char **menu, int lng, int mask);
Paints a menu on the bottom line.

void barPaint(const struct barDef *bd, int erase);
Displays an input or other value as bargraph, including name, current value and x-axe.

int messageBox(int code, char *fmt, ...);
Small dialogue with answers Yes, No, Ok.

void paintCircle(int x0, int y0, int z, int color);
Paints a full circle.

void buttonPaint(int x, int y, int wd, int ht, int fr, int fcolor, int bcolor, const char *txt);
Paints a button.

int tscrDebounce(int *yy);

Debounces the touch screen and returns useful x, y values on a click.

void ButtonClear();
Clear button table.
void ButtonDefine(int x, int y, int wd, int ht);
Define a button in the button table.
int ButtonSelect(int x, int y);
Return index of a pressed button.

Example : Shut off graphic lcd, when unused since <n> seconds. Shut on by external push button

Library, Inputs, Outputs

The library libgigalogs.a contains the drivers for the inputs and outputs of the board.
The header file gigalogs.h contains their prototypes, and port definitions
Note: When using standard C library functions, the compiler may output strange messages after the link.
The standard C library libc.a is a Linux library, and may call the Linux kernel, which is not present.
Avoid these kind of functions.

Types used in the program

char	8 bits signed	*
short	16 bits	*
int	32 bits	*
long	32 bits	*
float	32 bits	*
double	64 bits	*
uchar	8 bits unsigned	
u8	8 bits unsigned	
u16	16 bits unsigned	
u32	32 bits unsigned	
uint	32 bits unsigned	
s8	8 bits signed	
s16	16 bits signed	
s32	32 bits signed	

The types marked * are the standard types of the compiler.
The others are defined in header files.

System Functions

int getSr()
Get status register of the cpu

int intEnable()
Enable interrupts.
Returns status register before operation.

int intDisable()
Disable interrupts.
Returns status register before operation.

int intSet(int)
Set status register
Parameter shall be result of former getSr(), intEnable(), or intDisable()
Returns status register

void waitUs(int n)
Wait hard n microseconds.

void wait1Us()
Wait hard 1 microsecond.

void waitMs(int n)
Wait hard n milliseconds

void waitMsMux(int n)
If mutex is running, call muxSwitch() for n milliseconds.
Else call waitMs()

void boardInit()
Init board: Set clock, enable Reset pin

void downLoad(int n, ...)
Jump to Boot downloader
n=
BOOTSERIAL
BOOTUSB
BOOTDISK

extern int versionHardware, versionBootloader
Version of the board.
Version of the bootloader.
Year*100 + month.

Precise N * 1 kHz Timer

The timer runs at prescale * 1000 Hz
The timer calls user defined server routines.
To avoid one long interrupt each millisecond, the timer uses a higher rate, like 4 kHz.
It calls only one function at each interrupt.
When a millisecond elapses, the timer calls all outstanding functions.

void piInit(int prescale)
Initialize timer.
Prescale can be 1, 2, 3, 4, 5, 6, 8, 10

void piOff()
Switch timer off.

void piticFct(void (*piFct)(int ms))
Ask the timer, to call piFct each millisecond.
The timer calls the function and passes as argument the number of elapsed milliseconds, usually 1.

void piticFctOff(void (*piFct)(int ms))
Stop calling piFct.

RS0 (ud), RS1(u0), RS2(u1)

void udInit(int baud)
Initialise port RS0
The port uses interrupts on incoming characters

int udPut(char c)
Output a character.
This function may hard wait for the transmitter to be ready.

void udPurge()
Hard wait until the transmitter is empty.

void udGotchar(char x)
This function must be in the main program.
The interrupt function calls it, when a character arrived.

void u0Init(int baud)
Initialise port RS1
The port uses interrupts on incoming characters

int u0Put(char c)
Output a character.
This function may hard wait for the transmitter to be ready.

void u0Gotchar(char x)
This function must be in the main program.
The interrupt function calls it, when a character arrived.

void u1Init(int baud)
Initialise port RS2
The port uses interrupts on incoming characters

int u1Put(char c)
Output a character.
This function may hard wait for the transmitter to be ready.

void u1Gotchar(char x)
This function must be in the main program.
The interrupt function calls it, when a character arrived.

USB

void usbInit(void)
Init usb driver.

void usbGotchar(char x)
This function must be in the main program.
The interrupt function calls it, when a character arrived.

int usbPutchar(char x)
Output a character.
This function may wait using muxSwitch() for the transmitter to be ready.

int usbPutFree()
Returns number of characters usbPutchar() will accept without wait.

Embedded System Printf

Do not use standard C printf(), sprintf(), etc.
Do not use these functions in interrupt functions.
The basic putChar() functions may wait, using muxSwitch()

int putChar(char c)
Main prog must provide this function for stdout Printf()
This function may use muxSwitch() to wait.

int putCharP(int stream, char c)
intern: calls putChar()

int Printf(const char *fmt, ...)
Prints on stdout, calling putChar()

int eputchar(void *fu, char c)
Outputs one character on fu;
Fu is either a function like putCharP()
Or a file descriptor, output from fsOpen(), calls fsPutchar()

int eprintf(void *fu, const char *fmt, ...)
Prints on fu, using eputchar(fu, c)

int veprintf(int (*putfct)(int stream, char), int stream, const char *fmt, va_list ap)
Stdarg version of eprintf()

int esprintf(char *buf, int maxlen, const char *fmt, ...)
Sprintf() version prints into a buffer. Maxlen size of buffer including trailing 0.

int vesprintf(char *buf, int maxlen, const char *fmt, va_list ap)
Stdarg version of esprintf()

#define STDOUT putCharP

Memory functions

void *Malloc(size_t len, char *txt)
Allocate memory.
Txt indicate a text for the “xxmf” command. Maybe 0.
Returns 0, if no more memory available.
Malloc may muxSwitch, do not use on interrupt level.
The command xxmf displays used and unused memory blocks.

void Mfree(void *p)
Free memory, that had been allocated by Malloc(), or Mrealloc()

void *Mrealloc(void *p, size_t len, char *txt)
Reallocate memory block with a new size.
P points to a memory, that had been allocated by Malloc(), or Mrealloc()

void printfreelist()
Debug support.
Print list of current free memory blocks on stdout.

int totalfreeram()
Returns the total free ram in bytes in the heap.

void flWrite(char *flash, char *ram, int lng)
Write to Flash memory.
No restriction on addresses and length.

void dumphex(char *p, u32 addr, int cnt)
Dump memory to standard output.

Internal Adc

Based on a 1ms server.
The server reads out all adc each millisecond.
It puts them into a sum.
After 10 milliseconds, the server calculates the average from the sum.

void iadcInit()
Initialise internal adc server.

int iadc(int ch)
Get current value from iadc ram.
Last average sum over the last 10 ms period.

int boardVoltage()
Returns input voltage in millivolt.

float boardTemperature()
Returns board temperature in °C

Relay

void relaySet(int ch, int value)
Set solid state relay.

Lcd alpha 2x16 or 4x16

Based on a 1ms server.
The server transfers lcdbuf[] to the LCD.
The main program only has to write into lcdbuf[]

extern char lcdbuf[65]
64 characters

void lcdInit(int on)
Init server.

void lcdContrast(int x)
Set contrast.

Real time clock RTC

Battery buffered real-time clock ds1302

void rtcRead(char *r, int cnt)
Read RTC ram to ram.

void rtcWrite(char *r, int cnt)
Write to RTC ram.

Digital to analogue converter DAC

Four channel 12 bit digital to analogue converter DAC7554.
Optional on the board

void dacSet(int ch, int v)
Set Output channel to value.