Gigalog S Programmation du Firmware en C

Ce manuel montre la programmation du firmware de la carte GigaLog S.
Un changement du firmware de la carte GigaLog est seulement nécessaire, si la configuration du firmware ne couvre pas votre application. Ce travail demande une forte expérience en programmation en C.
Une partie de ce manuel est écrite en anglais.

Table de matières

Gigalog S Programmation du Firmware en C....1
GigaTerm Interface de programmation....2
- Fichiers dans la distribution....4
- Bootloader....5
- En cas de panne....5
- Premier contact: Recompiler firmware, chargement du firmware compilé....6
- Compilation et chargement d'un petit programme de démonstration....6
- STDIO (standard input output / entrée sortie standard)....6
- Ce qui une application doit respecter....6

Noyau multi-tâches....7
- Sémaphores....7
- Le principe d'arrière-plan....8
- ADCserver....8

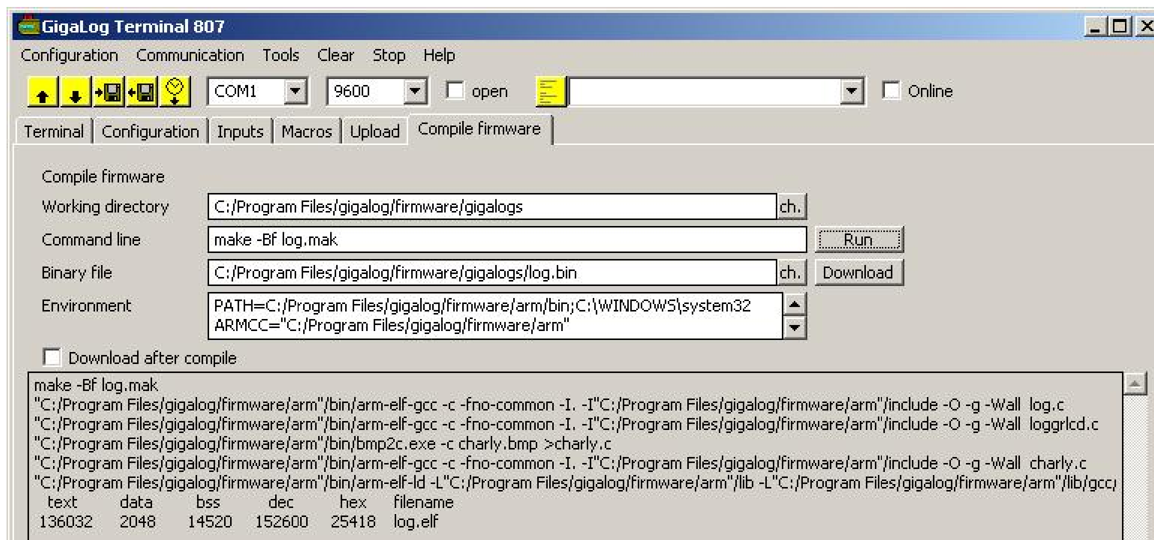
Firmware Log: Tâche personnelle....9
Fs file subsystem....11
Afficheur graphique....14
- Couleur....14
- Affichage graphique....14
- Affichage de texte....14
- Dalle tactile....14
- Polices....15
- Créer ses propres polices....15
- Affichage d'image....15
- Autres fonctions....16
- Autres fonctions dans loggrlcd.c....16
- Firmware Log: Page personnelle....16
- Lcd Boîte à outils....17

Les entrées et les sorties, là bibliothèque....19
- Types utilsés dans les programmes....19
- System Functions....19
- Precise N * 1 kHz Timer....20
- RS0 (ud), RS1(u0), RS2(u1)....20
- USB....21
- Embedded System Printf....21
- Memory functions....22
- Internal Adc....22
- Relay....22
- Lcd alpha 2x16 or 4x16....22
- Real time clock RTC....23
- Digital to analogue converter DAC....23

Controlord, 484, avenue des Guiols, 83210 La Farlède, Tél. (0033/0)4 94 48 71 74, www.controlord.fr

Version 1111, Novembre 2011.

GigaTerm Interface de programmation



	GigaLog S
Répertoire de travail	./firmware/gigalogs
Ligne de commande	make –f log.mak
Fichier binaire	./firmware/gigalogs/log.bin
Environment	PATH=./firmware/arm/bin ARMCC="./firmware/arm"

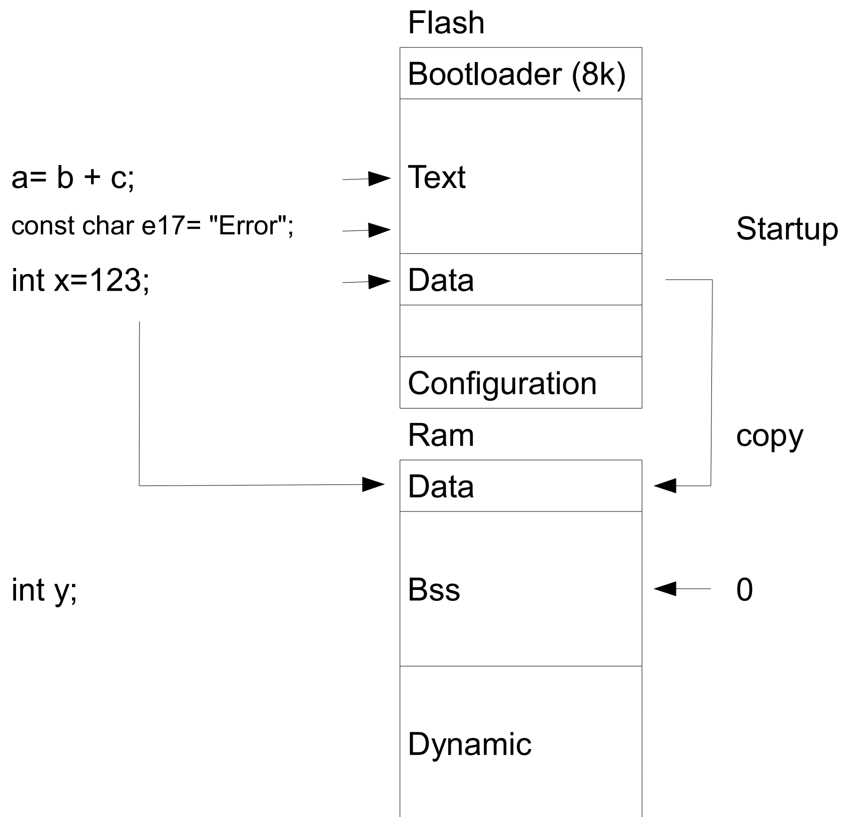
GigaTerm compile le programme source log.c avec l'aide du fichier log.mak et crée le fichier log.bin.
On peut charger ce fichier dans la mémoire Flash de la carte.

Le programme utilise la chaîne de compilation de GNU http://gcc.gnu.org/

Pour compiler il faut préalablement installer Cygwin sur l'ordinateur: Voir www.cygwin.org.
Pendant la compilation un message d'erreur peut apparaître à cause d'une incompatibilité des bibliothèques Cygwin sur votre ordinateur. Lisez le message attentivement et procédez comme proposé.

Si vous ne voulez pas installer Cygwin, ou vous avez toujours un message d'erreur comme
    "'C:/Program' n'est pas reconnu comme commande
installez le logiciel dans un répertoire sans espace " " dans le nom et dans le chemin.
Il faut aussi remplacer dans l'environnement
    ARMCC="./firmware/arm"
par   ARMCC=./firmware/arm

La ligne de commande "make -Bf" construit l'objet à partir des sources.
En changeant la ligne à "make -f" le programme ne compile que les sources nécessaires.

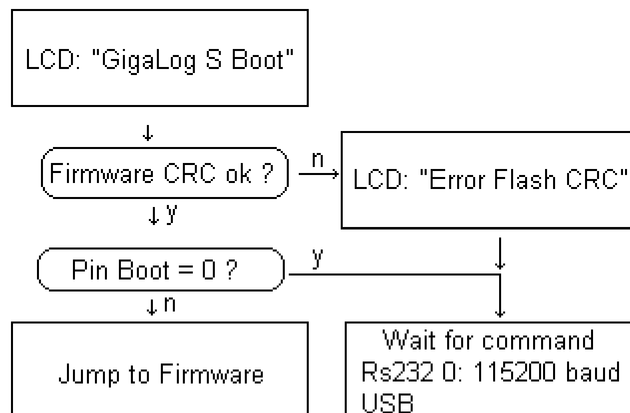

Flash
Bootloader (8k)
a= b + c;
Text
const char e17= "Error";
Startup
int x=123;
Data
Configuration
Ram
copy
Data
int y;
Bss
0
Dynamic

Fichiers dans la distribution

Répertoire	Fichier	
./firmware/gigalogs	hello.c	Démo: programme
	hello.mak	Démo: makefile
	lcd.c	Démo graphique: programme
	lcd.mak	Démo graphique: makefile
	lcdToolkit.h	Lcd graphique boîte à outils définitions
	lcdToolkit.c	Lcd graphique boîte à outils source
	gigalogSlog.h	Log: définitions, fichier entête
	log.c	Log: programme principale
	log.mak	Log: makefile
	loggrlcd.c	Log: graphique
	logstubgrlcd.c	Log: substitution pour graphique
	charly.bmp	Log: image
	machina.bmp	Log: image
	logPerso.c	Log: tâche personnelle
	loggrlcdPerso.c	Log: page graphique personnelle
./firmware/arm/include	gigalogs.h	Gigalog S: fichier entête pour la carte et la bibliothèque
./firmware/arm/lib	gigalogs.ld	Gigalog S: load script pour link
	gigalogscrt.o	Gigalog S: fichier de départ
	libgigalogs.a	Gigalog S: bibliothèque

Bootloader

Dans les premiers 8 kilo de mémoire Flash se trouve un programme boot.
Ce programme est chargé par nos soins. On ne touche jamais à ce programme.
Apres un reset:
Le boot verifie le CRC de la mémoire Flash du programme d'application.
Si le CRC est correct, il lance l'application.
Si le CRC n'est pas correct, ou
Si l'entrée BOOT (signal sur la carte à coté de la pile) est mise à zéro,
Boot ne lance pas l'application, mais attend une commande par le STDIO.
La commande "z" efface la mémoire prévue pour la configuration.
La commande "go" lancer l'application.
Sinon on peut charger une application par GigaTerm.
L'application doit toujours reconnaître le commande "dl" download et sauter
dans le boot pour charger une autre application.



En cas de panne.

Si votre application ne répond plus, si votre configuration ne permet plus de travailler.
Placer un fils entre GND et le point BOOT à coté de la pile sur la carte.
Taper sur Reset.
La carte répond "Download S7" sur le port RS0.
Sur USB, taper "dl"
Entrer "z" pour effacer la configuration.
Charger une autre application.
Taper "go" pour lancer l'application.

Premier contact: Recompiler firmware, chargement du firmware compilé

GigaTerm onglet: Compiler Firmware.
Répertoire de travail: Cliquer sur Ch, chercher ./firmware/gigalogs/log.mak
Cliquer sur Executer: Compilation du programme.
S'il y a des erreurs, voir plus haut pour les programmes Cygwin et la bibliothèque Cygwin.
Il faut, que la compilation se passe sans erreur et finisse avec des lignes comme

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

Fichier binaire: Cliquer sur Ch, ./firmware/gigalogs/log.bin
Cliquer sur Charger.
Le programme se charge dans la carte GigaLog S dans quelques secondes
Apres le chargement le programme se met automatiquement en route.
Verifier que le firmware marche bien.

Compilation et chargement d'un petit programme de démonstration

Le programme ./firmware/gigalogs/hello.c est un petit programme de base, qui respecte toutes les règles des chapitres suivantes, et fait clignoter la LED.
Répertoire de travail: Cliquer sur Ch, ./firmware/gigalogs/hello.mak
Cliquer sur Executer: Compilation du programme.
Fichier binaire: Cliquer sur Ch, ./firmware/gigalogs/hello.bin
Cliquer sur Charger.
Le programme hello accepte une seule commande par la liaison série: dl pour charger un autre programme.

STDIO (standard input output / entrée sortie standard)

Le programme d'application comme le programme BOOT travail avec une entrée sortie standard, le STDIO.
Le STDIO est

- soit le port série 0 (HE10 RS0)
- soit le port série 1 (HE10 RS1)
- soit le port série 2 (Rs485)
- soit le USB.

Après Reset le STDIO se trouve sur le port série 0.
Dés que l'application ou le programme BOOT reçoit un caractère par un de ces ports, le programme change la direction est répond désormais sur ce port.
Quand le Boot passe la main au programme d'application, quand l'application appelle le Boot pour charger une autre application, le STDIO passe comme paramètre d'un programme à l'autre.
La communication par le USB se passe donc exactement comme la communication par RS232.
La routine Printf() travaille sur STDIO.

Ce qui une application doit respecter.

L'application doit travailler sur les STDIO pour recevoir des commands.
L'application doit reconnaître la commande "dl" download et sauter dans le boot pour charger une autre application.
L'application doit créer des tâches et activer le noyau multi-tâches.
Sinon le programme ne peut pas utiliser le disque.
Chaque tâche du noyau doit régulièrement rendre la main aux autres tâches.
Soit par la fonction muxSwitch(), soit par le temps, si la tâche est préemtive.

Le programme ./firmware/gigalogs/hello.c est un petit programme de base, qui respecte ces règles et fait clignoter la LED.

Noyau multi-tâches

Le noyau mutex permet de créer plusieurs tâches, qui tournent en parralel.
Le programme principal main crée une ou plusieurs tâches et appelle muxStart pour lancer le noyau.
Cette routine ne revient pas.
Chaque tâche peut créer et détruire d'autres tâches plus tard.
La routine muxCreate crée une tâche.

> muxTcb *muxCreateTask(char *name, void (*fct)(int p), int p, int stacksize, int timeslice);
> name:nom d'affichage dans muxDump.
> fct: le nom de la routine à exécuter.
> p: paramètre à passé à la routine.
> stacksize: indique la taille de la pile à réserver. 2000 octets est une bonne valeur de départ.
> timeslice: le temps accordé à la tache en milliseconds.

Si ce temps est non zéro, le noyau peut à tout moment retirer le control de la tâche et appeler une autre tâche.
Si ce temps est 0, la tâche est non préemptive. Elle doit impérativement appeler muxSwitch régulièrement.
Conseil: Mettez ce paramètres non zéro, mais appeler muxSwitch dans chaque boucle principal et dans chaque attente.
Plusieurs modules comme le système de fichier ne peuvent pas travailler sans ce noyau.

Routine d'interruption.
Les interruptions utilisent leur propre pile.
Une routine d'interruption ne doit pas appeler directement ou indirectement muxSwitch().

muxTcb *muxCreateTask(char *name, void (*fct)(int p), int p, int stacksize, int timeslice)
Créer une tâche

void muxStart()
Main: Démarrer mutex.

void muxSwitch()
Changement de la tâche.

void muxExit()
Fin de la tâche.

void muxDump()
Afficher les tâches sur STDIO.
La commande "xxt" du log.c appelle cette routine.
On trouve pour chaque tâches l'adresse de son tcb, la partie de la pile non utilisé (free stack=), qui permet d'affiner la taille de la pile et le temps utilisé par la tâche (très approximatif).

Sémaphores

Fr.wikipedia.org:

> Un sémaphore est une variable protégée (ou un type de donnée abstrait) et constitue la méthode utilisée couramment pour restreindre l'accès à des ressources partagées (par exemple un espace de stockage) dans un environnement de programmation concurrente.

volatile muxTcb *s;
Déclaration d'un sémaphore.

Int Semaphore(volatile muxTcb **s)
Retourne vrai, si le programme pouvait réserver le sémaphore.
Retourne faux, si le sémaphore est occupé.

void Semawait(volatile muxTcb **s)
Attend par muxswitch() jusqu'à Sémaphore() retourne avec succès.

#define SemaFree(x) *x=0
Libérer le sémaphore.

Le principe d'arrière-plan.

Plusieurs activités du logiciel sont déplacées à l'arrière plan.
Soit en tâche du noyau multi-tâches, soit dans des routines d'interruptions.
Le programme d'application est donc libérés de ces fonctions.

L'acces au disk par les routines fsOpen etc.
Le programme d'application travail sur une mémoire cache des plusieurs secteurs du disque.
Le serveur blServer transfère les données entre le tampon et le disque.
Il assure, que le disque est rapidement à jour.

Le driver LCD est basé sur le tampon lcdbuf[32]
Le programme d'application écrit seulement dans ce tampon.
Le driver transfère les caractères à l'afficheur par une routine d'interruption.

Les entrées analogiques internes du microcontrôleur, la tension d'entrée.
Une routine d'interruption lit chaque milli seconde toutes les entrées analogiques.
La routine stocke dans un tampon la moyenne sur 10 milli secondes.
La routine iadc lit une valeur du tampon et retourne donc immédiatement.
La fonction boardVoltage retourne la tension d'entée en millivolts.

ADCserver

Le convertisseur analogique numérique sur la carte GigaLog S ADS1258 scanne automatiquement les 16 entrées analogiques. Dès q'un résultat est disponible, une interruption informe le programme pour récupérer la donnée.
Le programme log.c contient un serveur, qui stocke régulièrement les données en mémoire.
Les données sont stockées pour l'enregistrement du moyen sur la carte à mémoire.
En même temps le serveur accumule une deuxième somme de données pour calculer le moyen sur quelques échantillons dans quelques millisecondes.
Pour chaque entrée sa valeur déjà filtrée est donc directement disponible à tout moment.
La routine Manalogin(int ch) retourne cette valeur.

Firmware Log: Tâche personnelle

Le firmware de GigaLog S a prévu une tâche personnelle, pour faciliter l'intégration des fonctions spécifiques pour un projet.
La tâche peu stocker jusqu'à 20 octets dans la configuration en RAM. Ces données sont automatiquement transférées an mémoire Flash.
Le firmware inclus le fichier logPerso.c.
Le fichier contient une tâche vide (entre #if 1 et #endif)
Le fichier contient également un exemple (entre #if 0 et #endif)
L'exemple contrôle le relais rl0 à partir de l'entrée a0 et deux paramètres stockés dans la 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(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 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.

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

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

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.

s32 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, s32 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.
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 fsRename(char *frompath, char *topath)
Rename file
Returns 0 for success

int fsUnlink(char *path)
Remove the file from the filesystem.
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

Afficheur graphique

On doit appeler la routine glcdInit() au début.
void glcdInit();
Une seule tâche peut écrire sur l'écran en même temps.
Si on veut travailler par plusieurs tâches, il faut une coordination par exemple par sémaphores.

Couleur

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

Chaque pixel est representé par un mot de 16 bit RGB
Le fichiers gigalogs.h contient des définitions pour quelques couleurs:

gCwhite	gCcyan	gCpink	gCblue	gCyellow	gCgreen	gCred	gCgrey
gCdarkgrey	gCdarkcyan	gCdarkpink	gCdarkblue	gCdarkyellow	gCdarkgreen	gCdarkred	gCblack

Affichage graphique

```
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);
```

Affichage de texte

La routine lcdSetFont() choisit la police et la couleur.
Les routine lcdPrintf(), lcdPrintfn(), lcdPrintfc() écrivent sur l'écran. lcdPrintf() s'arrête à la fin de l'écran, lcdPrintfn() après wd colonnes. Si wd est < 0, alignement à droite. lcdPrintfc() comme lcdPrintfn(), alignement au centre.
Ces routines retourne le numéro de colonnes utilisés.
Les routnes lcdCwidth(), lcdCwidthS() et lcdChight() calculent l'espace d'un texte sur l'écran.

```
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();
```

Dalle tactile

void touchScrInit(struct tscrc &t) passe une mémoire non volatile pour garder les valeurs de la dalle tactile.
int touchScr(int *y);
int Button(int x, int y, int x0, int y0, int wd, int ht);
TouchScr() retourne la valeur de la dalle tactile. Elle retourne -1, si la dalle n'est pas active.
Button compare des valeurs x et y avec un bouton x0, y0, wd, ht.
Exemple: La routine paintButton() ecit un bouton sur l'écran.
Le programme écrit deux boutons "Ok" et "Cancel".
Il attend que l'utilisateur presse un bouton.

```
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
        }
}
```

Polices

La bibliothèque libgigalogs.a contient deux polices.

id	Police	Taille	
a	Arial	14	
s	Arial	8	

On choisit une police par le fonction lcdSetFont() par son id.
Une police contient des caractères ASCII standard, sans é, à, ü, etc.

Créer ses propres polices

On peut créer son propre fichier de police qui remplace le fichier de la bibliothèque.
On crée avec Paint une image d'une grande largeur et d'une petite hauteur. On crée une zone de texte, choisit sa police et colle le texte suivant là-dedans.

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

Le texte doit coller à gauche et en haut. Il faut tout éliminer au bas et à droite.

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

On crée un fichier 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"
```

Le fichier .mak pour deux polices font1.bmp et 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
```

Il faut inclure fonts.o pendant le link.
Les polices de la bibliothèque libgigalogs.h ne sont plus disponibles.
La police font1 est identifiée par 'a', la deuxième par 'b'.
Dans le programme principal on choisit la première police en noir

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

Affichage d'image

Trois alternatives.

- Créer un fichier Bitmap 24 bits. Compiler le fichier par Bmp2c en 16 bits par pixel. Charger l'image dans la mémoire Flash. Afficher l'image par lcdImage. Cher en mémoire Flash. Parfait pour afficher un perroquet ou la Joconde.
- Créer un fichier Bitmap 16 couleurs. Compiler le fichier par Bmp2c en 4 bits par pixel. Charger l'image dans la mémoire Flash. Afficher l'image par lcdImage. Economique en mémoire Flash. Satisfaisant pour des croquis techniques.
- Créer un fichier Bitmap 16 couleurs. Placer le fichier sur la carte à mémoire. Afficher l'image par lcdFimage. Lent, conseillé pour une démonstration ou faire des testes.

Le programme bmp2c convertit l'image en format compatible compilateur C.
Dans le fichier.mak

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

Et il faut compiler charly.c et inclure charly.o pendant le link
Dans le programme principal, on appel la routine suivante:

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

Zoom= 2,3,… pour agrandir l'image.
Par exemple:

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

LcdFimage(int x, int y, char *fname) affiche l'image à partir d'un fichier sur la carte à mémoire.
La routine retourne 0 en cas de succès..

Autres fonctions

int lcdPrint(char *fname);
Impression de l'image sur l'écran dans un fichier sur la carte à mémoire.

void Rolex(int x0, int y0, int rd, int color, int hr, int min);
Affiche une horloge sur l'écran à la position, radius, couleur, temps.

void lcdTouchAdjust();
Declenche une operation, qui permet à l'utilisateur de régler la dalle tactile.

Autres fonctions dans 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

Firmware Log: Page personnelle

Le firmware de GigaLog S a prévu une page graphique personnelle, pour faciliter l'intégration des fonctions spécifiques pour un projet.
Le firmware inclus le fichier loggrlcdPerso.c.
Le fichier contient une page vide (entre #if 1 et #endif)
Le fichier contient également un exemple (entre #if 0 et #endif)
L'exemple est une petite animation graphique.
La page a dans une deuxième page un dialogue pour changer les paramètres de l'affichage.

LoggrlcdPerso.c doit fournier deux entrées globales utilisées par le firmware loggrlcd.

const char *grPersoButton= "perso";
Texte du bouton, qui ouvre la page personnelle.
Quand la variable est zéro, pas de page personnelle.

int taskgrPerso(int l);
La tâche qui contrôle l'afficheur appelle cette fonction, quand l'utilisateur a choisit la page.
La fonction tourne dans une boucle appelant taskgrWait() pendant que la page reste active.
Elle est responsable pour l'affichage de l'écran.
Elle doit lire la dalle tactile et exécuter les fonctions, que l'utilisateur a choisi.
Elle finit, quand l'utilisateur a choisi une autre page et retourne la page choisi.

Lcd Boîte à outils

Les fichiers lcdToolkit.h et lcdToolkit.c contient des définitions et les sources des autres fonctions pour l'afficheur graphique.

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 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.

Les entrées et les sorties, là bibliothèque

La bibliothèque Libgigalogs.a contient des fonctions pour les entrées et sorties de la carte.
Le fichier gigalogs.h contient des prototypes de ces fonctions et des définitions sur les ports de la carte.
Attention: Si on utilise une fonction standard du C, et le compilateur donne des erreurs bizarres après le Link: La bibliothèque libc.a est une bibliothèque Linux qui peut faire appel au noyau Linux, qui ici n'est pas présent. Il faut éviter cette fonction standard.

Types utilsés dans les programmes

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

Les types * sond des types du compilateur. Les autres sont dans gigalogS.h.

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)
Allocate memory.
Returns 0, if no more memory available.

void free(void *p)
Free memory, that had been allocated by malloc(), or realloc()

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

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.