libios.so Reference

before launching an application using the library, make sure that you have loaded the necessary modules to the kernel

You can also use the command

```
$ sudo ./iosmodload.sh
```

Toggle once to reboot.

If you want to use the part of the chip 328 must send the firmware and run the 'ard_init () at the beginning of each program and ard_close () at the end.

The only difference that occurs between an i2c device and one of the 328 is that while in the first you will have a synchronous call in the second no!

For example, when you set a value to a pin board 328 ios_write function () will return immediately even if in reality it will take a few microseconds to run the operation.

This is because the internal firmios has a circular buffer of commands and executes them one at a time, this has allowed us to create an output pwm to any of the pin and the ability to execute code interpretato.L'uso of a buffer increases then the potential, even if in some circostane complicates the result.

was, however, implemented an "echo" command, with this command becomes easy to synchronize the two boards.

By submitting an "echo" the application will waiting for the answer, as this command will be the last of the tail of course once it receives the response we are sure that all the commands will be executed.

Last premise, remember that you are working with a quad core 1.6 ghz and you can send many more commands than I can run the 328.

To minimize the effects of this problem, the firmware takes all the commands of the small buffer 328 and puts them in a circular buffer very large (~ 100 commands) and then executes the command and repeat the cycle.

Even here it may be necessary the use of echo to reduce the possibility of loss of data.Still working to automate the whole.

```
FLOAT64 ios_clock_get();
```

returns the time passed since the S.O. in seconds

```
UINT32 ios_mclock_get();
```

returns the time passed since the S.O. in millisecons

```
UINT32 ios_uclock_get();
```

returns the time passed since the S.O. in microseconds

```
UINT32 ios_nclock_get();
```

returns the time passed since the S.O. in nanoseconds

```
VOID ios_msleep(UINT32 ms);
```

sleep milliseconds

```
VOID ios_usleep(UINT32 us);
```

sleep microseconds

VOID ios_nsleep(UINT32 nano);

sleep nanoseconds

BOOL ios_exported(UINT32 pin);

checks whether it is exported to a pin, to use only the pins of the chip I2C defined in the library with IOS_P00 -> IOS_P07, IOS_P10 -> IOS_P17

returns TRUE or FALSE otherwise have been exported

INT32 ios_export(UINT32 pin, UINT32 mode);

export a pin for use only with the pins of the chip I2C defined in the library with IOS_P00 -> IOS_P07, IOS_P10 -> IOS_P17.

Mode IOS_EXPORT exports pin

Mode IOS_UNEXPORT remove the pin

returns 0 for success.

IOS_ERR_PIN if the pin is invalid

IOS_ERR_OPEN if you can not open the file.

IOS_ERR_WRITE if it can not write to the file.

INT32 ios_exportall(UINT32 mode);

export / unexport all pin

INT32 ios_dir_write(UINT32 pin, UINT32 d);

set direction for pin ,you can use IOS_DIR_OUT or IOS_DIR_IN.

Use only for low level access.

INT32 ios_dir_read(UINT32 pin);

return direction pin

VOID ard_init();

If you loaded the firmware "firmios" on the shield and then use the pin of the IC 328 as those of the integrated i2c without any difference must first run the above function.

This function initializes the communication with the firmware.

VOID ard_close();

close comunication with firmware

void ard_test();

nothing

INT32 i328_make(UINT32* offset, BYTE* bytecode, CHAR* code);

the firmios can run interpreted code, but for the moment there is a bug in the compiler then left alone for future implementation.

HIOS ios_open(CHAR* t,CHAR* m);

This feature allows you to create a new device.

The first parameter selects the type of device, the second opening mode ("r" or "w") devices based only accept or opened in read or write.

The first parameter is a bit more complex, you first need to indicate which library is part of the device and successively the name of the device you want to creare.This small complexity allows you to create libraries which add support for new devices without having to recompile the library, this will be of great help terminios the terminal version of the library.

Here are the devices to be used:

"ios/u3l" ,"w"

allows the use of blue LED

The application using the following device requires superuser permissions.

Only write.

"ios/pin" ,["r" | | "w"]

Create a device associated with a physical pin of the shield. can be run in both read and write

"ios/port" ,["r" | | "w"]

Create a device associated with a maximum of 8 pins on the physical device, not necessarily adjacent.

"ios/pulse" ,["r" | | "w"]

Create a device that allows the pulse on a pin, the pulse duration can be set both that direction.

Low-> Hight-> Wait-> Low

Hight-> Low-> Wait-> Hight

"ios/pwm" ,["r" | | "w"]

In write mode creates a device to generate a pwm signal from the variable duty cycle and variable frequency on any pin.

In modal reading reads the frequency and the PWM input to any pin.

In the moment the value of the duty varies from 0 to 255, but it will soon be possible to determine the minimum and maximum values to be mapped as a signal of the duty cycle

"ios/328" ,["r" | | "w"]

Create a device that accesses the extra features of the chip 328, such as "echo", the activation of the sleep / wake and many other features such as run options.

"ios/int" , ["w"]

The device interrupt to call a function or execute a command when the port pin is high or low.

To call a library containing a device must be copied to the directoty "/usr/lib" with the name of the default "libMYLIB.so" and will be referenced by:

"MYLIB/mydevice"

Given the complexity of the 'topic will be discussed in detail in a high-tutorial

INT32 ios_write(HIOS h, VOID* v, UINT32 sz, UINT32 n);

The write function is very powerful, letting you created based on the device settings are set with "ios_ioctl ()"

The first parameter of the function is the device handle returned by the "ios_open ()".

The second parameter is a pointer to the value of what we want to write to the device,

The third parameter is the size of v and the last is number of value we want to write to the device.

many aspects of writing need only two values, 0/1, to ease the transition of these values the library contains two global constants called "IOS_PIN_LOW" and "IOS_PIN_HIGH" to use them just declare them as:

```
extern const BYTE IOS_PIN_LOW ;
extern const BYTE IOS_PIN_HIGH ;
```

"ios/u3l"

u3l the device accepts only 1 byte, and the variable can take the values 0 or 1 to turn off the LED to turn on.

"ios/pin"

the device pin accepts only 1 byte, and the variable can take the values 0 for the low logic level and one for the high one.

"ios/port"

The device port only accepts one byte for writing.

each bit corresponds to the logic level of the associated pin.

The most significant bit corresponds to the pin assigned as 7 and the least significant to pin 0.

Consequently, to set to a high logic level all the pins will be enough to pass to the function one 0xFF.

"ios/pulse"

The pulse device only accepts a byte that can take only the value 1, which will perform a pulse to the pin.

"ios/pwm"

The PWM device only accepts a byte can take the following values:

0 to pause the PWM signal

1 to start the pwm signal

2 to resume after being paused

to easy use of these methods have been declared constants and can be implemented with:

```
extern const BYTE IOS_PWM_PAUSE;
extern const BYTE IOS_PWM_START;
extern const BYTE IOS_PWM_RESUME;
```

"ios/328"

The device 328 is a little more complex.

The write function can operate in three different modes depending on the options you set with the "ios_ioctl ()"

If you want to set the prescaler must pass or 2 bytes, the first will contain the pin you want to set, and the second divider prescaler (this part is to be updated)

In the mode pin direction must always move two bytes, the first containing the pin and the second containing the direction of the pin (this feature is pretty useless)

The last way is to write directly into the memory dedicated to the interpreter's .This need to send the bytecode to be run by 328.

Will write a manual that will explain more in detail the firmware in order to make best use of the device 328 that has great potential.

"ios/int"

is equal a pwm one byte for started,paused,resumed.

INT32 ios_read(HIOS h, VOID* v, UINT32 sz, UINT32 n);

read function return 1 for success or 0 otherwise

"ios/u3l"

NO read.

"ios/pin"

wen read device pin you can use unsigned char(BYTE) or short int (INT16) or int (INT32) ed value stored in v by sz, n repeat.

"ios/port"

return state for each bit corresponds to the logic level of the associated pin.

The most significant bit corresponds to the pin assigned as 7 and the least significant to pin 0.

Consequently, when read 0x77 you have al pin at High level.

"ios/pulse"

The pulse device accepts float or double where stored frequency of pulse

"ios/pwm"

The PWM device read frequncy on the pin and return median of frequency.

If set n=2 and pass vector unsigned int (UINT32) the first parameter is frequency the secon is a duty on.

"ios/328"

The read device 328 is a little more complex.

For read direction pass integer v where is setted a pin to read direction and stored always in v the direction pin.

If is set IOS_IOCTL_MEM_STA you read memory 328.

"ios/int"

Not used.

INT32 ios_ioctl(HIOS h, INT32 req, VOID* v);

The ioctl function is used to set the operating mode of the device or to perform additional functions to only read / write.

"ios/u3l"

IOS_IOCTL_U3L_MMC

Set blue led flashing when reading from emmc

IOS_IOCTL_U3L_SD

Set blue led flashing when reading from sd card.

IOS_IOCTL_U3L_TIMER

enable timer mode

IOS_IOCTL_U3L_TIMER_ON

Set time on of blue led

IOS_IOCTL_U3L_TIMER_OFF

Set time off of blue led

IOS_IOCTL_U3L_ONESHOT

Turn on and of the led

IOS_IOCTL_U3L_GPIO

I could not get it to work.

"ios/pin"

Accept only

IOS_PIN_SET

to set real pin to device.

"ios/port"

IOS_IOCTL_PORT_SET_0

IOS_IOCTL_PORT_SET_1

IOS_IOCTL_PORT_SET_2

IOS_IOCTL_PORT_SET_3

IOS_IOCTL_PORT_SET_4

IOS_IOCTL_PORT_SET_5

IOS_IOCTL_PORT_SET_6

IOS_IOCTL_PORT_SET_7

Set bit to pin

IOS_IOCTL_PORT_SET_A

need to pass int vector with all pin.

"ios/pulse"

IOS_IOCTL_PULSE_SET

Set pin to pulse

IOS_IOCTL_PULSE_MODE

If set 1 when write 1 on pulse activate LOW-HIGH-DELAY-LOW

If set 0 when write 1 on pulse activate HIGH-LOW-DELAY-HIGH

IOS_IOCTL_PULSE_US

Set DELAY of pulse.

IOS_IOCTL_PULSE_MODERET

if 0 read return frequency else duty

"ios/pwm"

IOS_IOCTL_PWM_WAIT

Wait end pwm

IOS_IOCTL_PWM_SET

Set pin for pwm

IOS_IOCTL_PWM_SET_DUTY

set duty on from 0 to 255

IOS_IOCTL_PWM_SET_FQ

set frequency for pwm.

IOS_IOCTL_PWM_SET_TOUT

Set duration of pwm

IOS_IOCTL_PWM_END_MODE

Set status of pin when finish pwm

IOS_IOCTL_PWM_READ_RESET

Restart reading value pwm

IOS_IOCTL_PWM_ELAPSE

Return the time elapse from starting pwm

"ios/328"

IOS_IOCTL_328_ECHO

Send echo to 328

IOS_IOCTL_328_WAKE

Wake 328

IOS_IOCTL_328_SLEEP

force sleep

IOS_IOCTL_328_OPZ

Set opzion

IOS_IOCTL_328_MAXTO

IOS_IOCTL_328_PRE_SET

set prescaler

IOS_IOCTL_328_MEM_STA

Start position memory to write.

IOS_IOCTL_328_NOP

328 execute NOP you con use for asyn wake.

IOS_IOCTL_328_OPPC

Start position bytecode

IOS_IOCTL_328_DIR

Set direction 328 pin

IOS_IOCTL_328_OVDT

IOS_IOCTL_328_FREERAM

"ios/int"

IOS_IOCTL_INT_SET

Set pin for interrupt

IOS_IOCTL_INT_MODE

with IOS_INT_CBK set callback function, IOS_INT_EXECL set mode execution command

IOS_IOCTL_INT_MODE_SET

Set function or command to execute

IOS_IOCTL_INT_PARAM

Param to pass at callback function

case IOS_IOCTL_INT_STAT

set when interrupt execute , 1 when pin is high 0 when pin is low

IOS_IOCTL_INT_BOUNCE

Set minimal duration on microsecond of stat for execute interrupt

IOS_IOCTL_INT_WAIT

Wait end interrupt fot now wait infinite.

VOID ios_close(HIOS h);

closes a device

FAST EXAMPLE: