eurek
ELECTRONIC ENGINEERING

IL TUO PARTNER TECNOLOGICO

touchMe
SIZE S / M / L

T//COTTON

Based on Linux™

TouchMe® Cotton **EK340**

USER MANUAL

TouchMe®, tutto il bello del touch, senza complicazioni.

Il nostro Sistema Qualità ha ricevuto la certificazione di conformità a norme ISO 9002 dal 1998. Da Gennaio 2010 siamo passatti alla ISO 9001:2008.

Our Quality System was granted the certification of conformity with ISO 9002 standards as September 1998. From January 2010 we have switched over to ISO 9001:2008.

assolutamente **made in italy.**

TouchMe® Cotton **EK340**

TouchMe®, tutto il bello del touch, senza complicazioni.

Sistema di sviluppo software su ambiente Linux per schede embedded Serie Cotton [EK340]

Eurek Debian Embedded Linux

La presente documentazione fa riferimento all'immagine di lavoro VirtualBox fornito assieme alla scheda Serie Cotton [EK340].

- Nella cartella ***/home/eurek/Progetti/iMX28/documenti*** sono presenti i file di documentazione, eventualmente i file *Gerber gli Schematici e distinte componenti.* È presente inoltre anche lo schematico dell'adattatore JTAG/Seriale di debug EK206-0.

- Nella cartella ***/home/eurek/Progetti/iMX28/bootrom*** e ***/home/eurek/Progetti/iMX28/barebox*** sono presenti i file dei sorgenti e relativi binari dei bootlets, del bootloader Barebox e del kernel Linux. Nella cartella **/home/eurek/Progetti/iMX28/*scripts*** sono presenti vari scripts tra cui il file *firmware-update.sh* da utilizzare per l'aggiornamento via **USB Pen Disk** nel caso la si voglia implementare.

- Nella cartella ***/home/eurek/Progetti/iMX28/bootrom/bootdescriptor*** sono presenti il tool per la programmazione del bootloader via USB, i file di bootlets, creati con il sistema di build.

- Nella cartella ***/home/eurek/Progetti/iMX28/bootrom/pendisk-EK340*** sono presenti i file da copiare su chiavetta USB per effettuare un aggiornamento dalle versioni precedenti del firmware (bootlets e bootloader Barebox, kernel Linux e distribuzione RootFileSystem *Debian Wheezy Edelin*)[1]

1 Ancora da testare, per ora in versione ALPHA semifunzionante

Embedded Linux Serie Cotton [EK340]
Board Support Package

Il presente documento è un breve sunto del contenuto del DVD omonimo e contiene alcune descrizioni dei tools più comuni per essere immediatamente operativi con la scheda Serie Cotton [EK340].

Caratteristiche tecniche della board Serie Cotton [EK340]

Processor	: 0
Model Name	: ARM926EJ-S rev 5 (v5l)
Features	: swp half thumb fastmult edsp java
CPU implementer	: 0x41
CPU architecture	: 5TEJ
CPU variant	: 0x0
CPU part	: 0x926
CPU revision	: 5

Hardware : Freescale MXS (Device Tree)

Clocks

mpll:	480 MHz
arm:	454 MHz
ioclk0:	480 MHz
emiclk:	205 MHz
hclk:	151 MHz
xclk:	24 MHz
ssp0:	96 MHz
ssp1:	96 MHz

Memory

RAM	: 128MB DDR2
FLASH	: eMMC 2Gbyte (mmc driven)

Firmware

Bootloader	: barebox 2013.03.0 (custom drivers)
Linux Kernel	: 3.12.1 (vanilla con patches per device tree)
OperatingSystems	: Debian Wheezy 7 ***Edelin*** (armel based/custom)

Consumi Serie Cotton [EK340]

Condizioni di misura:

- Scheda EK340-0 con modifiche
- Alimentazione con alimentatore da banco
- Multimetro ***HTItalia DMM8501*** come misuratore di corrente collegato all'uscita dell'alimentatore (**scala 400mA**)
- sensore di test one-wire collegato alla Serie Cotton [EK340]
- scheda EK206 JTAG Debug collegato alla Serie Cotton [EK340]
- cavo Ethernet collegato con processo di trasferimento file tramite comando *scp*
- processo di compressione in ram con utilizzo CPU 90%
- processo che calcola md5sum su flash eMMC (*in lettura*)

Tensione di alimentazione	**Corrente misurata**	**Potenza calcolata**
11.0V	160-175mA	1.93W
18.12V	112-124mA	2.24W

Alle precedenti condizioni si é aggiunto:

- Collegamento SDcard con processo di calcolo md5sum sul device (in lettura)
- Collegamento USB pendisk tramite cavetto OTG con processo di calcolo md5sum sul device (in lettura)

Tensione di alimentazione	**Corrente misurata**	**Potenza calcolata**
11.14V	220-240mA	2.67W
18.04V	145-150mA	2.71W

Alle precedenti condizioni si é aggiunto:

- Rimosso SDcard e collegato scheda Wifi con processo di scansione reti wifi ogni secondo (di più per ora non si puo` fare senza il driver funzionante)

Tensione di alimentazione	**Corrente misurata**	**Potenza calcolata**
11.01V	265mA	2.92W
17.95V	166mA	3.00W

Alle precedenti condizioni si é aggiunto:

- processi di scrittura sia in eMMC che in pendisk USB

Tensione di alimentazione	**Corrente misurata**	**Potenza calcolata**
11.01V	275mA	3.03W
17.95V	172mA	3.09W

La scheda funziona perfettamente con una tensione superiore a **8.5V**.

Connettori presenti nella scheda Serie Cotton [EK340]

Sulla scheda sono presenti:

- **Connettore Ethernet 10/100 su RJ45** con trasformatori e LED integrati
- Tasto di **Factory Reset** SMD (programmabile)
- Tasto di **System Reset** SMD (programmabile)
- **Led** rosso SMD
- **microUSB** device tipo AB OTG
- **USB Host tipo A**
- **Connettore microSD**
- **Modulo WiFi WF111**
- **Connettore LCD J6**
- **Connettore Speaker A/B per casse acustiche**
- **Connettore J11 Mic-In ingresso microfonico**
- **Connettore Jack Line-In Ingresso audio analogico**
- **Connettore Jack Headphone Out Uscita cuffie**
- **Selettore livello porta seriale esterna (TTL/RS232)**
- **Morsetto Phoenix serie PTSM SMD 8 Pin (J8) (Vcc 8/32, RS232, RS485, GND)**
- **Connettore DB9-M CanBus Standard**
- **Morsetto Phoenix serie PTSM SMD 6 Pin (J9) (I2C, One-Wire, GND, INT)**

Connettore J8

1	Vin VCC +
2	Vin GND -
3	UART TX
4	UART RX
5	GND con filtro
6	RS485-B
7	RS485-A
8	TERRA

- La seriale UART é chiamata */dev/ttyAPP1* mentre quella RS485 é chiamata */dev/ttyEK2*.
  *Il nome ttyAPP sta ad indicare che é una **Application UART** tipica di questa serie di processore.*

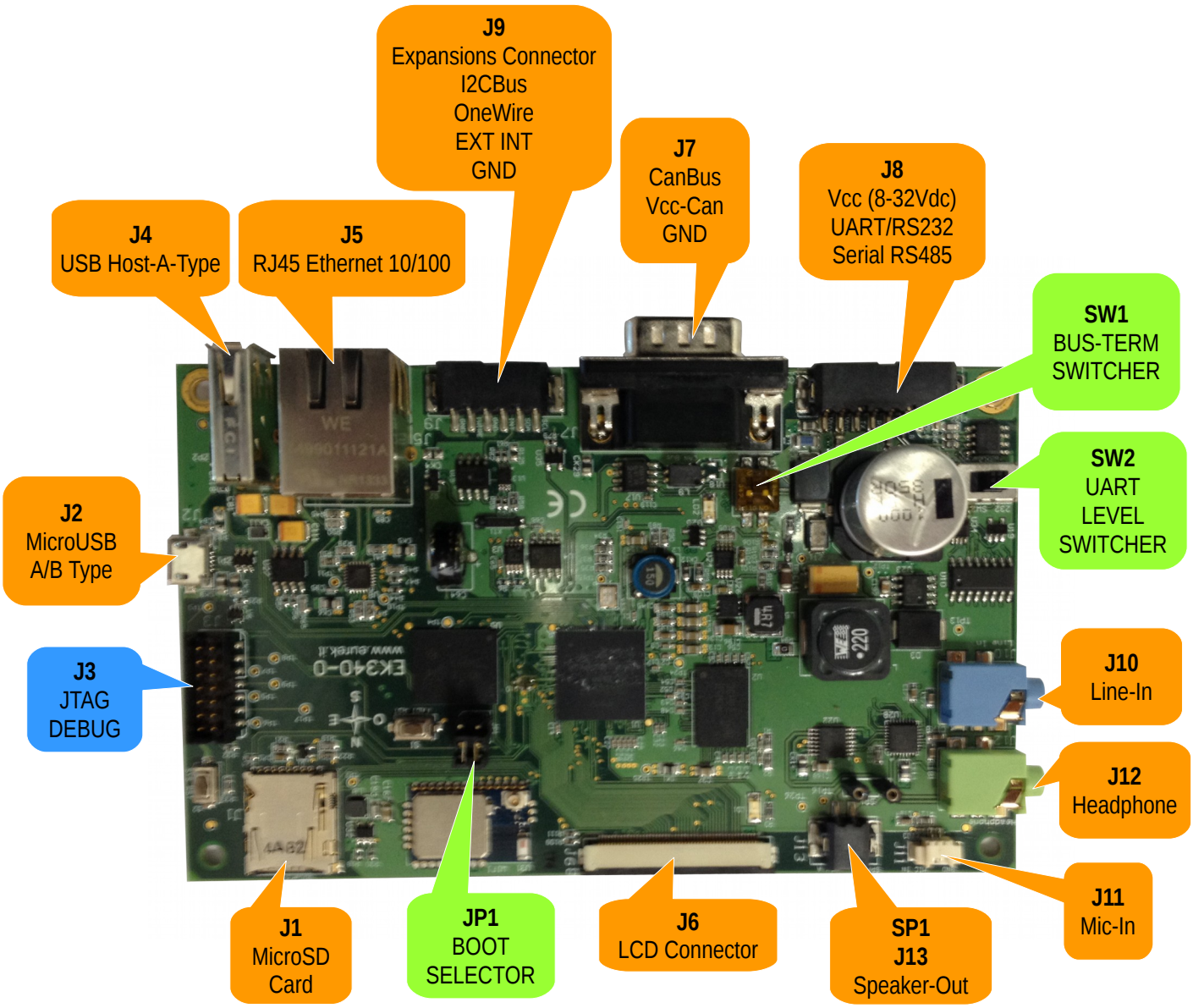
Inoltre sulla scheda è presente un doppio switch (SW1) per *abilitare/disabilitare* il terminatore di linea per la *seriale RS485* e per il bus *CAN*.

Connettore J9

1	I2C SCL (5V)
2	I2C SDA (5V)
3	EXTERNAL INTERRUPT
4	GND
5	One Wire
6	GND

Nella pagina seguente è visibile nel dettaglio la disposizione dei connettori a bordo scheda.

LAYOUT CONNETTORI



Storage Memory

La memoria di storage **eMMC Flash** (2Gbytes) è partizionata nel seguente modo (verificabile tramite il comando *fdisk -l /dev/mmcblk0):*

```
Disk /dev/mmcblk0: 1920 MB, 1920991232 bytes
4 heads, 16 sectors/track, 58624 cylinders, total 3751936 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xdf444a80

        Device Boot      Start         End      Blocks   Id  System
/dev/mmcblk0p1            8192       24575        8192   53  OnTrack DM6 Aux3
/dev/mmcblk0p2           24576       90111       32768   83  Linux
/dev/mmcblk0p3           90112     3751935     1830912   83  Linux
```

- Partizione per bootlets e Barebox (**/dev/mmcblk0p1**) 4MB
- Partizione per parametri di configurazione Barebox, Linux Kernel e DeviceTree (**/dev/mmcblk0p2** → ***/boot***) 16MB
- Partizione per Root FileSystem e applicativo formattata NILFS2 (**/dev/mmcblk0p3**) circa 1.8Gb

L'aggiornamento del sistema può avvenire in locale tramite chiavetta **USB** e/o in remoto tramite protocollo ***TFTP*** (*tftp*) , ***FTP*** (*ftp client*), ***HTTP*** (*wget*).

PROCEDURA DI AGGIORNAMENTO FIRMWARE

Preparare la PenDisk USB con i seguenti file:

- **firmware-update.sh**
- **firmware-update-md5sum.log**
  Script di update presenti nella cartella *BSP/pendisk*

- **kernel-md5sum.log**
- **Kernel_version**
- **linux-kernel-EK340.bin**
- **dtb-EK340.bin**
  Vengono generati ad un rebuild del kernel dai sorgenti presenti nella cartella *BSP/bootrom/download/linux-kenel-3.12.1.tar.bz2 con applicata la patch apposita per il supporto alla scheda Serie Cotton [EK340]* (seguire la procedura nella sezione omonima di questo documento). I binari sono comunque presenti nella cartella di build *pendisk*.

- **modules-md5sum.log**
- **Module_version**
- **modules-EK340.tar.gz**
  Vengono generati ad un rebuild del kernel dai sorgenti presenti nella cartella *BSP/bootrom/download/linux-kenel-3.12.1.tar.bz2 con applicata la patch apposita per il supporto alla scheda Serie Cotton [EK340]* (seguire la procedura nella sezione omonima di questo documento). I binari sono comunque presenti nella cartella di build *pendisk*.

- **boot-barebox-EK340.bin**
- **boot-barebox-md5sum.log**
- **BootBarebox_version**
  Vengono generati ad un rebuild del bootloader e dei bootlets dai sorgenti presenti nella cartella di build *iMX28/barebox* e *iMX28/bootrom* (seguire la procedura omonima di questo documento). Sono comunque presenti come binari nella cartella di build *pendisk.*

N.B.: il RootFileSystem viene considerato una procedura per aggiornamento speciale e necessita al contrario dell'aggiornamento tradizionale un update via scheda ***microSD*** per cui nella procedura sopra riportata non rientra l'aggiornamento del Sistema Operativo.

La procedura di aggiornamento da PenDisk USB è la seguente:

1) Inserzione della pen disk USB contenente gli script di aggiornamento tramite adattatore **microUSB-Host A** (*non fornito, ma reperibile come cavetto in molte catene di materiale informatico*)
2) Accensione o reboot della scheda.
3) Al boot del sistema operativo la pen disk USB viene montata in automatico, e viene lanciato lo script di aggiornamento.
4) Lo script controlla la firma e calcola gli MD5sum dei file da aggiornare, e se tutti corretti provvede all'accensione del LED ed all'aggiornamento dei file necessari: Bootloader Bootlets e Barebox, Kernel Linux ed i suoi drivers (moduli), RootFS ed eventuale applicativo. Da notare che in caso di corruzione anche solo di uno dei dati presenti nella pen disk USB, il procedimento viene interrotto e segnalato tramite accensioni e spegnimenti del LED veloci (500 msec / 250 msec).
5) Al termine dell'aggiornamento il LED segnala la corretta installazione spegnendosi, altrimenti in caso di errore il LED lampeggia lentamente (1 sec / 1 sec).
6) Se tale aggiornamento prevede il reboot della scheda essa si riavviera' in maniera automatica al termine dell'aggiornamento.
7) Successivamente solo se il LED si spegne e **rimane spento** la pen disk USB si può rimuovere.

NOTA: se al terzo riavvio il LED continuasse a lampeggiare segnalando un errore, formattare la pen disk USB e ricopiarci nuovamente i file per l'aggiornamento, poiché si sospetta una memoria flash con partizione non corretta e/o rovinata. Se il problema dovesse persistere provare a sostituire la pen disk USB.

ATTENZIONE! Non rimuovere la pen-disk USB mentre il led è ancora acceso.

In remoto l'aggiornamento della scheda sarà implementato analogamente e limitato dallo spazio disponibile in memoria eMMC.

1) lo script di aggiornamento si collegherà al server per scaricare i file di aggiornamento richiesti e li copierà nella partizione di appoggio dopo averne verificato firma e MD5sum.
2) l'installazione avviene la copia ed il lanco dello script apposito per l'aggiornamento (Booloader/Barebox, Linux Kernel ed i suoi drivers [*moduli*], RootFS e/o Applicativo)
3) al termine il sistema si riavvia automaticamente

NOTA: la procedura di aggiornamento in remoto é lasciata all'applicativo utente

I2C bus
A bordo della scheda sono presenti 2 bus distinti i2c. Al primo bus sono collegati la FRAM 8K (compatibile **at24c64** indirizzo Linux **0x50**) e il RealTimeClock **RTC PCF8563** (indirizzo Linux **0x51**). Nel secondo bus i2c disponibile vi é un traslatore di livello per i 5Vdc ed un bridge per bus *1-wire* chiamato **ds2482**. Il bus OneWire e` disponibile tramite ***OWFS***. Il bus esterno **i2c** e` disponibile negli indirizzi lasciati liberi dai dispositivi interni.

L'accesso alla memoria *FRAM* é regolato dalla scrittura/lettura dello *pseudo-file* in *sysfs:*

/sys/bus/i2c/devices/0-0050/eeprom

Il *Real-Time-Clock* di sistema é automaticamente installato ed al boot ne preleva l'orologio per impostare correttamente l'orario di sistema:

```
[    1.537798] rtc-pcf8563 0-0051: chip found, driver version 0.4.3
[    1.549117] rtc-pcf8563 0-0051: rtc core: registered rtc-pcf8563 as rtc0
[    1.556107] i2c /dev entries driver
[    1.726434] rtc-pcf8563 0-0051: setting system clock to 2013-11-14 08:38:16 UTC (1384418296)
```

È possibile anche nella scheda Serie Cotton [EK340], come nei più comuni sistemi Linux se connessi in rete, la procedura di sincronizzazione tramite gli script di boot con acquisizione dell'orario da un server esterno (tipicamente *Network Time Protocol Servers*). La procedura è definita in questo modo:

- al boot si sincronizza la data del sistema operativo utilizzando come sorgente l'*hardware clock*, sempre che la batteria **SuperCAP** sia carica. Altrimenti la data di default è 01-01-1970.
- si sincronizza la data di sistema al server *NTP* tramite i servizi standard di Debian Wheezy 7
- allo spegnimento il default imposta l'hardware clock con la data del sistema operativo ottenuta fino a quel momento .

Un server NTP valido per l'Italia (per esempio) è: ntp2.ien.it.

One-Wire (1-wire (w1))
Il bus 1-wire viene implementato tramite l'utilizzo del bridge *I2CBus/1-wire* ds2482. L'accesso al 1-wire nel kernel avviene esponendo allo *userspace* l'accesso al device i2c-1 (secondo bus controller i2c integrato nel processore i.MX28). Tale operazione si è resa necessaria per l'implementazione di ***OWFS***, in quanto l'accesso al bus viene regolato da tale libreria.

USB Host
Il device USB (core ***ChipIdea - USB1***) connesso al sistema e` configurato come Host Device ed e` possibile collegare molti dispositivi riconosciuti dal sistema Linux (USB Pendisk, Tastiere, Mouse, ecc.,...). La massima corrente erogabile si attesta sui 500mA.

USB Device
Il device USB (core ***ChipIdea - USB0***) connesso al sistema é completamente configurabile e puo` assumere di essere Host (con opportuno adattatore) oppure Device a seconda del cavetto collegato e del pin USB-ID presente all'interno del cavo. E` possibile quindi collegare **USB pen disk, USB Keyboard,** oppure semplicemente collegarlo ad un PC ed impostare il funzionamento voluto tramite l'inserimento/disinserimento dei moduli/drivers opportuni. Per esempio collegando un'adattatore microUSB (A/B) con Host (A) Receptacle, esso diviene automaticamente ***Host***

Controller e quindi accettare dispositivi quali tastiere, mouse, chiavette pen disk. Viceversa, se si collega un semplice cavo microUSB (A/B) con Host (A) Connector, entra nella modalita` ***Device Controller*** e puo` simulare diversi dispositivi. Per esempio ecco come configurare verso un PC (per semplicita' con Sistema Operativo Linux) una interfaccia seriale USB.

- collegare il **cavo USB Device** dalla scheda **Serie Cotton [EK340]** al **PC**,
- attivare il driver di emulazione seriale sul connettore USB

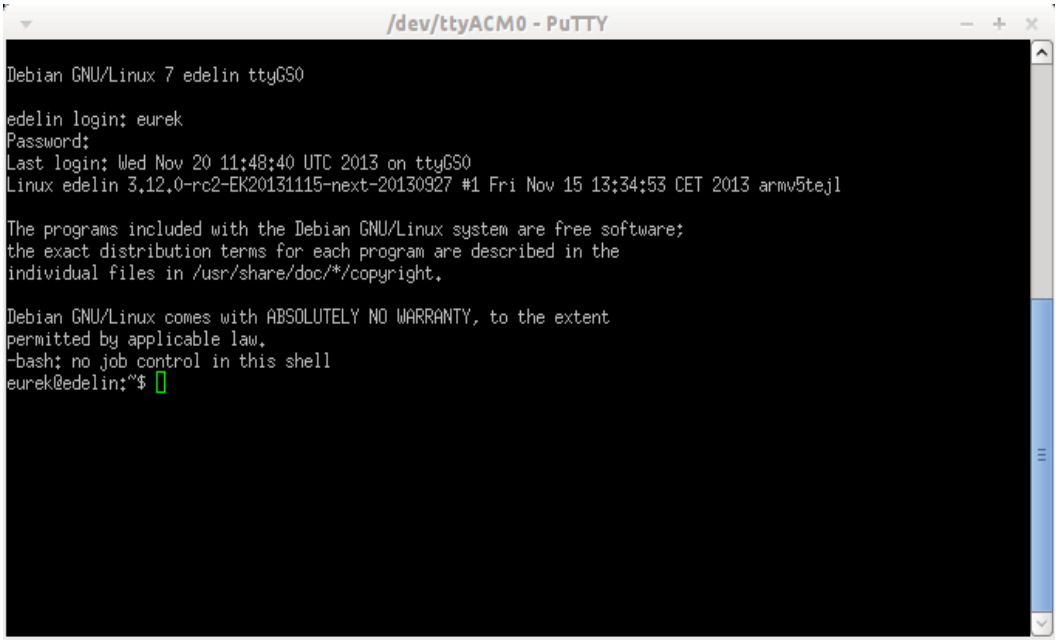
  # modprobe g_serial 2>/dev/null 1>/dev/null

- Attendere il termine dell'installazione del driver dal lato PC e verificare che il kernel sulla scheda abbia istanziato il device: **/dev/ttyGS0**
- Lanciare per esempio sulla **scheda embedded Serie Cotton [EK340]** il programma di login, per testarne il corretto funzionamento

  # /sbin/getty -L ttyGS0 115200 vt100

- Lanciare sul PC un emulatore di terminale sulla porta */dev/ttyACM0 (baudrate 115200 8N1)*

Si dovrebbe cosí verificare su quella seriale la connessione con login alla scheda embedded come nella schermata seguente:



Per ogni altro utilizzo della configurazione come Device fare riferimento ai seguenti documenti/siti presenti in Internet:

http://www.linux-usb.org/gadget/

http://elinux.org/images/8/81/Useful_USB_Gadgets_on_Linux.pdf

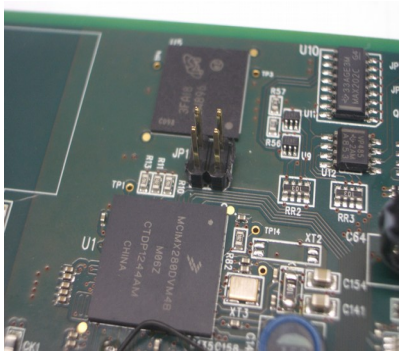
Le porte UART su Serie Cotton [EK340]
L'assegnazione delle seriali sul sistema ARM ® Embedded Serie Cotton [EK340] è la seguente (tutti i device name si trovano nel file system all'interno della cartella **/dev** creati dinamicamente da *udev*):

UART1 **ttyAPP1** → UART presente su **J8** (il cui livello e` selezionabile tramite **slider SW2**)
UART2 **ttyEK2** → RS485 presente su **J8** (il terminatore e` selezionabile tramite **SW1**)
DUART **ttyAMA0** → RS232C presente sul connettore **J3** per debug (kernel, bootloader, …)

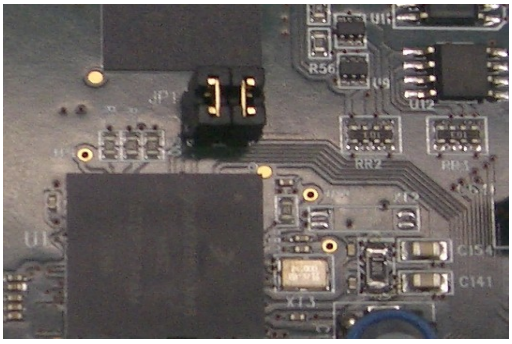
NOTA: le seriali hanno connessi solo i pin RX e TX.

Sistema di boot
La modalità di avvio della scheda (se tramite bootstrap da **microSD** oppure se da **eMMC Flash** interna) è selezionabile tramite la chiusura o apertura dei due jumpers **JP1**. Nelle foto sequenti sono rappresentate le due configurazioni ammissibili dei jumpers: in quella di sinistra (fig.1) la posizione per effettuare il *boot da eMMC Flash* (***default***) mentre in quella di destra (fig.2) la posizione per *bootstrap da microSD (RECOVERY MODE)*.

(fig.1)



(fig.2)



Watchdog
Un circuito di watchdog è un dispositivo hardware che permette di effettuare un controllo sullo stato del sistema e riavviare la macchina nel caso risulti non operativa a causa di un crash irreparabile del software. Il funzionamento solitamente è basato su un timer non pilotato dalla CPU principale che viene impostato dall'utente ad un valore stabilito e diminuisce nel tempo; quando questo valore raggiunge lo zero allora viene alzato il segnale di reset del sistema e la macchina viene riavviata.

La scheda Serie Cotton [EK340] tramite il SystemOnChip i.MX28 include tra i suoi componenti interni un circuito di watchdog programmabile che si interfaccia col sistema grazie al driver nel kernel (*/dev/watchdog*) specifico per questo scopo.

Il sistema più comodo e intuitivo è quello di utilizzare prima del lancio del software il seguente comando tramite script:

 # modprobe stmp3xxx_rtc_wdt heartbeat=[SEC]

dove [SEC] è il tempo ragionevole per il quale il software può essere impegnato ad eseguire operazioni complesse e non rinfrescare il timeout del watchdog (se viene omesso il parametro heartbeat, come default é impostato un conteggio di reset a **19 secondi**). Al primo accesso al device */dev/watchdog* viene *triggerato* il **countdown** al cui scadere il sistema provvederà ad eseguire un **watchdog reset hardware**. Se durante questo countdown il software continuasse ad accedere al

device */dev/watchdog* regolarmente, tale timeout verrebbe *ricaricato con il valore impostato all'inizio* e tutto riprenderebbe come prima. Quindi in breve se il dispositivo */dev/watchdog* non ricevesse il flag di reset prima dello scadere del tempo, oppure se il comando non venisse spedito in tempo, il sistema **si riavvierà dopo 19 secondi** o quanto indicato in **heartbeat** al caricamento del modulo. È inoltre possibile modificare il timeout in ogni momento, invocando una chiamata a *ioctl(WDIOC_SETTIMEOUT)* sul device con il parametro in secondi come spiegato nel dettagli nel seguente sito web:

http://embeddedfreak.wordpress.com/2010/08/23/howto-use-linux-watchdog/

NOTA: la rimozione del modulo dal kernel non provoca lo spegnimento del conteggio.

Strumenti di controllo in SysFS

Tramite sysfs è possibile pilotare vari pin di I/O nella maniera seguente (si utilizza il metodo standard gpio definito nei kernel Linux recenti e nello specifico con il sottosistema ***gpiolib pinctrl***:

Controllo del LED

se impostato a '0' si accende altrimenti si spegne.

```
# echo 7 > /sys/class/gpio/export
# echo out > /sys/class/gpio/gpio7/direction
# echo '0' > /sys/class/gpio/gpio7/value [ACCESO]
# echo '1' > /sys/class/gpio/gpio7/value [SPENTO]
```

Controllo della pressione del tasto FACTORY RESET

Per verificare se il tasto FACTORY RESET viene premuto, monitorare il pin apposito

```
# echo 126 > /sys/class/gpio/export
# echo in > /sys/class/gpio/gpio126/direction
```

Leggere il file */sys/class/gpio/gpio126/value* per determinare se:

- quando é 1 il tasto **NON é premuto**
- quando é 0 il tasto **é premuto**

Inoltre si puo` abilitare / disabilitare (default **DISABILITATO**) il tasto di System Reset tramite l'utility *hwreset:*

- *# hwreset 1* (abilitato)
- *# hwreset 0* (disabilitato)

Impostazione del fuso orario (timezone)

I file di configurazione interessati a questo proposito sono:
/etc/localtime che è un link simbolico al file */usr/share/zoneinfo/**Europe/Rome***
/etc/timezone che contiene solamente il tipo: *Europe/Rome*

NOTA: l'orologio hardware è impostato secondo la *timezone locale* e non UTC. Il cambio della timezone avviene come un comune sistema Debian.

Wireless Module WF111

La scheda Serie Cotton [EK340] è fornita con un modulo Wireless (**WF111**) da utilizzare con antenna esterna.

E` necessario per il corretto utilizzo che l'antenna sia inserita a dispositivo **SPENTO**.

Connessione ad una rete wireless APERTA (senza password)

Prima di tutto verifcare che l'**Access-Point AP** al quale si vuole collegare sia visibile dal modulo wifi:

```
# iwlist scanning
lo                Interface doesn't support scanning.
sit0      Interface doesn't support scanning.
wlan0  Scan completed :
                  Cell 01 - Address: 00:26:BB:DE:AD:0F
                            ESSID:"open network"
                            Mode:Managed
                            Frequency:2.412 GHz (Channel 1)
                            Quality=35/40  Signal level=-75 dBm  Noise level=-100 dBm
                            Encryption key:off
                            Bit Rates:  1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                                        9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                                        48 Mb/s; 54 Mb/s
                            IE: Unknown: 000C566F6461666F6E6520383735
                            IE: Unknown: 010882848B960C121824
                            IE: Unknown: 030101
                            IE: Unknown: 32043048606C
                            IE: Unknown: 2A0100
                            IE: Unknown:
2D1A300103FF000000000000000000000010000000000000000000
                            IE: Unknown: 3D160100060000000000000000000000000000000000000000
                            IE: Unknown: 7F0100
                            IE: Unknown:
DD180050F2020101800003A4000027A4000042435E0062322F00
```

Per collegarsi alla rete aperta "**open network**" si puó procedere nel seguente modo:

```
# iwconfig wlan0 essid "open network"
# dhclient wlan0
```

Verificare l'avvenuta connessione nel sistema di network:

```
# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0  Link encap:Ethernet  HWaddr 00:07:80:01:99:63
          inet addr:192.168.43.239  Bcast:192.168.43.255  Mask:255.255.255.0
          inet6 addr: fe80::207:80ff:fe01:9963/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:133 errors:0 dropped:0 overruns:0 frame:0
          TX packets:133 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:128500 (125.4 KiB)  TX bytes:14041 (13.7 KiB)
```

A questo punto si é connessi. Per testare la bontà della connessione abbiamo ***pingato*** il server di *Google*:

```
# ping www.google.it
PING www.google.it (173.194.113.247) 56(84) bytes of data.
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp_req=1 ttl=56 time=607 ms
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp_req=2 ttl=56 time=1226 ms
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp_req=3 ttl=56 time=424 ms
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp  req=4 ttl=56 time=447 ms
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp_req=5 ttl=56 time=676 ms
64 bytes from mil01s18-in-f23.1e100.net (173.194.113.247): icmp_req=6 ttl=56 time=495 ms
^C
--- www.google.it ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5026ms
rtt min/avg/max/mdev = 424.534/646.192/1226.592/274.079 ms, pipe 2
```

Per le connessioni protette da password si puó utilizzare la procedura nel paragrafo seguente che utilizza *wpa-supplicant*.

Connessione ad una rete wireless protetta da password

Verificare che l'**Access-Point AP** sia visibile dal modulo wifi:

```
# iwlist scanning
lo                  Interface doesn't support scanning.
sit0      Interface doesn't support scanning.
wlan0  Scan completed :
          Cell 01 - Address: 00:26:BB:74:85:AF
                    ESSID:"private network"
                    Mode:Managed
                    Frequency:2.412 GHz (Channel 1)
                    Quality=24/40  Signal level=-81 dBm  Noise level=-105 dBm
                    Encryption key:on
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                              48 Mb/s; 54 Mb/s
                    IE: Unknown: 000C657572656B20756666696369
                    IE: Unknown: 010882848B960C121824
                    IE: Unknown: 030101
                    IE: Unknown: 0706495420010D1E
                    IE: Unknown: 2A0100
                    IE: Unknown: 32043048606C
                    IE: IEEE 802.11i/WPA2 Version 1
                       Group Cipher : CCMP
                       Pairwise Ciphers (1) : CCMP
                       Authentication Suites (1) : PSK
                    IE: Unknown: 2D1AAC4117FFFFFF0000000000000000000000000000000000000000
                    IE: Unknown: 3302742C
                    IE: Unknown: 3D160100400000000000000000000000000000000000000000
                    IE: Unknown: 46050200010000
                    IE: Unknown: DD180050F2020101010003A4000027A4000042435E0062322F00
                    IE: Unknown: DD0700039301720320
                    IE: Unknown: DD0E0017F207000101060026BB7485AF
                    IE: Unknown: DD0B0017F20100010100000007
```

Per collegarsi quindi all'Access-Point "**private network**" é necessario procedere modificando il file di */etc/network/interfaces* in quanto la cifratura dipende dall'*utility* ***wpa-supplicant***.

Aggiungere nel file */etc/network/interfaces* le seguenti righe:

```
iface wlan0 inet dhcp
          wpa-ssid "private network"
          wpa-psk  "***************"
```

Chiaramente al posto degli asterischi "" scrivere in chiaro la propria password.*

Successivamente per attivare l'interfaccia wireless lanciare il comando *ifup:*

```
# ifup wlan0
IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
ioctl[SIOCGIWMODE]: Input/output error
ioctl[SIOCSIWESSID]: Input/output error
unifi2: Delete key request was rejected with result -22
ioctl[SIOCSIWENCODEEXT]: Input/output error
Internet Systems Consortium DHCP Client 4.2.2
Copyright 2004-2011 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
Listening on LPF/wlan0/00:07:80:01:99:63
Sending on   LPF/wlan0/00:07:80:01:99:63
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on wlan0 to 255.255.255.255 port 67
DHCPOFFER from 192.168.143.2
DHCPACK from 192.168.143.2
unifi2: IP address assigned for wlan0
bound to 192.168.143.158 -- renewal in 6169 seconds.
```

Verificare l'avvenuta connessione nel sistema di network:

```
# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:46 errors:0 dropped:0 overruns:0 frame:0
          TX packets:46 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:16376 (15.9 KiB)  TX bytes:16376 (15.9 KiB)

wlan0  Link encap:Ethernet  HWaddr 00:07:80:01:99:63
          inet addr:192.168.143.158  Bcast:192.168.143.255  Mask:255.255.255.0
          inet6 addr: fe80::207:80ff:fe01:9963/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1107 errors:0 dropped:0 overruns:0 frame:0
          TX packets:286 errors:0 dropped:3 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:200774 (196.0 KiB)  TX bytes:72316 (70.6 KiB)
```

Come nell'esempio precedente a questo punto si é connessi. Per testare la bontà della connessione abbiamo ***pingato*** il server di *Google*:

```
# ping www.google.it
PING www.google.it (74.125.232.159) 56(84) bytes of data.
64 bytes from mil02s05-in-f31.1e100.net (74.125.232.159): icmp_req=1 ttl=53 time=306 ms
64 bytes from mil02s05-in-f31.1e100.net (74.125.232.159): icmp_req=2 ttl=53 time=229 ms
64 bytes from mil02s05-in-f31.1e100.net (74.125.232.159): icmp  req=3 ttl=53 time=458 ms
64 bytes from mil02s05-in-f31.1e100.net (74.125.232.159): icmp_req=4 ttl=53 time=373 ms
^C
--- www.google.it ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 229.601/342.184/458.873/84.535 ms
```

Per disattivare la connessione servirsi del comando *ifdown:*

```
# ifdown wlan0
Listening on LPF/wlan0/00:07:80:01:99:63
Sending on   LPF/wlan0/00:07:80:01:99:63
Sending on   Socket/fallback
DHCPRELEASE on wlan0 to 192.168.143.2 port 67
unifi2: IP address removed for wlan0
unifi2: IP address removed for wlan0
```

Implementazione di OWFS (One Wire File System)

Per attivare il filesystem ***OWFS*** é sufficente creare un mount-point per il montaggio del filesystem one-wire ed invocare l'utility owfs:

Per attivarlo basta un semplice:

```
# mkdir -p /tmp/owfs
# owfs --i2c=/dev/i2c-1:ALL -m /tmp/owfs/
```

Tramite il connettore **J 9** tutti i dispositivi *w1* (*One Wire*) connessi esternamente alla scheda Serie Cotton [EK340], verranno connessi e riconosciuti ed inseriti nella cartella predisposta proprio a questo scopo (*mountpoint*) che risiede in *ramfs (tmpfs ramdisk)* e precisamente in */tmp/owfs*.

Tutti i dispositivi riconosciuti vengono immediatamente verificati e montati entro il filesystem virtuale. Per l'utilizzo della libreria OWFS rimando alla pagina web del progetto:

http://owfs.org/

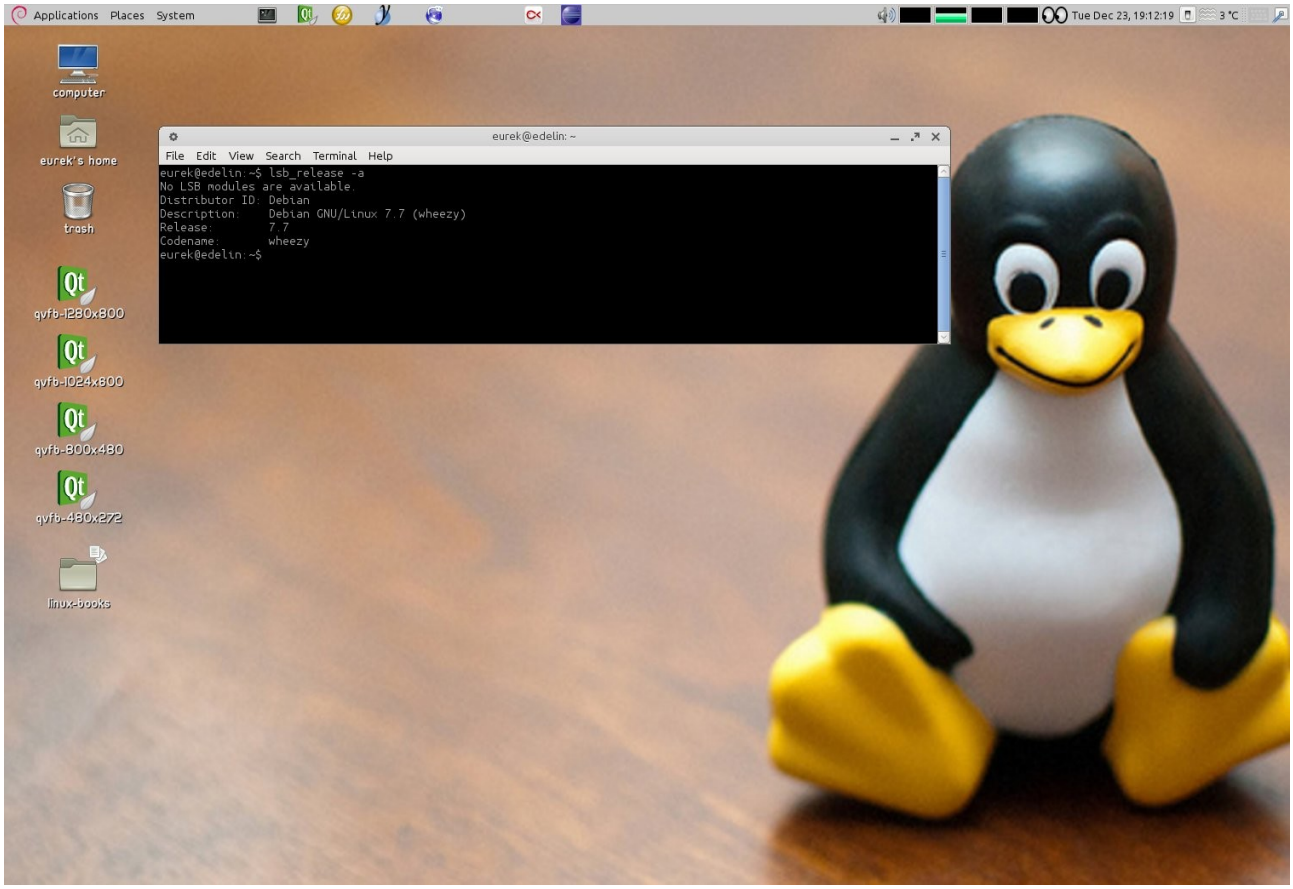
Board Support Package (BSP)

Assieme alla scheda viene fornita una macchina virtuale VirtualBox che permette l'utilizzo del software senza dover installare alcunché sul proprio PC. Installando semplicemente VirtualBox ed importando la macchina virtuale sul proprio sistema si è in grado in qualche minuto di essere completamente operativi.

Requisiti minimi di sistema:
CPU : x86 Pentium 4 Dual Core / Quad Core / Eight Core
RAM : 2 GB o maggiore
VIDEO : 1280x1024 24bit
INTERNET : connessione FLAT
DISCO : almeno 100 GigaBytes di spazio disponibile

La distribuzione utilizzata è una versione personalizzata di **Debian Wheezy 7** che offre alta flessibilità di configurazione e un parco software riconfigurabile in maniera semplice e modulare per essere adattato ad ogni esigenza. Nella macchina virtuale é possibile installare successivamente anche un root filesystem emulato ARM based utilizzabile tramite QEMU.

Login: utente: *eurek*
password: *eurek*



PREPARAZIONE DELLA MACCHINA VIRTUALE

Per ridurre la dimensione dell'immagine della macchina virtuale si e` optato per installare in un secondo momento il sistema Linux emulato ARM con **QEMU** ed i necessari file di build.

La procedura e` abbastanza semplice benche` necessario utilizzare un terminale per portare a termine il tutto.

WHEEZY per ARM

Il sistema consta di due file-systems distinti:

- un file system di sviluppo completo
- ed un file system ridotto per poter essere installato sulla memoria eMMC della scheda Serie Cotton [EK340]

Vengono forniti due files atti allo scopo:

- Debian-armel-wheezy-7.7-full-release-20141222-113035.tar.gz
- Debian-armel-wheezy-7.7-20141222-113035.tar.gz

Il primo e` necessario allo sviluppo della propria applicazione ed andra` installato nella cartella:

/home/eurek/debian-system/armel/wheezy

Mentre il secondo e` necessario solamente quando occorre aggiornare il sistema operativo e ricrearne uno nuovo per l'installazione via SDCard e va installato nella cartella:

/home/eurek/rootfs-edelin

Entrambi i file vanno estratti posizionandosi nelle apposite cartelle e lanciando il seguente comando:

```
$ cd ~/debian-system/armel/wheezy
$ sudo tar xvfz /path/to/file/Debian-armel-wheezy-7.7-full-release-20141222-113035.tar.gz

$ cd ~/rootfs-edelin
$ sudo tar xvfz /path/to/file/Debian-armel-wheezy-7.7-20141222-113035.tar.gz
```

da notare che /path/to/file e` il percorso dove risiedono i file!

ed attendere la fine dell'installazione...

BUILD per i.MX28

Successivamente occorre installare sul sistema di build anche il bootloader, il kernel di Linux, i drivers, il filesystem originale con la quale la scheda viene collaudata installando nella cartella

/home/eurek/Progetti/iMX28/bootrom

il pacchetto denominato:

- pre-build-images-EK340.7z

Posizionarsi nella cartella sopra indicata ed estrarlo tramite il comando **7z**:

```
eurek@edelin:~$ cd ~/Progetti/iMX28/bootrom
eurek@edelin:~/Progetti/iMX28/bootrom$ 7z x pre-build-images-EK340.7z -o./
```

Fatto cio` sara` possibile ricreare una SDCard sia accedendo direttamente alle utility di Linux, sia scrivendo un'immagine direttamente sulla SDCard (da almeno 4GB) che permettera` di ripristinare il sistema al default di fabbrica.

Inoltre sara` possibile in questo caso effetturare aggiornamenti del sistema, del kernel, ecc.,.. semplicemente lanciando una serie di script.

Nel seguente esempio installiamo sul sistema embedded un pacchetto debian non presente nel sistema operativo di default.

```
$ sudo chroot ~/debian-system/arme/wheezy
password: *****

# mount -t proc /proc proc

# apt-get install [my-debian-package]
# ...
# ...

# umount /proc
# exit
```

In questo modo avremo installato il pacchetto [my-debian-package] nel sistema **chrooted** emulato nel sistema. Il passo successivo e` quello di preparare il sistema ridotto con l'aggiunta di quel pacchetto.

```
eurek@edelin:~$ cd ~
euerk@edelin:~$ ./build_rootfs_emb.sh debian-system/armel/wheezy strip no-user
```

seguire le indicazioni ed attendere il termine della procedura. Verra` creato un file all'interno della cartella *debian-system/armel* chiamato:

- *Debian-armel-wheezy-[ANNO/MESE/GIORNO]-[HH:MM:SS].tar.gz*

Posizionarsi nella cartella rootfs-edelin, cancellarne l'intero contenuto ed installarne il nuovo:

```
eurek@edelin:~$ cd ~/rootfs-edelin
eurek@edelin:~/rootfs-edelin$ sudo rm -rf *
eurek@edelin:~/rootfs-edelin$ sudo tar xvfz ~/debian-system/armel/Debian-armel-
wheezy-7.7-YYYYMMDD-HHMMSS.tar.gz
```

ed attenderne il completamento.

A questo punto andremo a posizionarci nella cartella di build per il boot

- Progetti/iMX28/bootrom

```
eurek@edelin:~$ cd ~/Progetti/iMX28/bootrom
```

Cancelliamo il file di sistema di root (se presente):

```
eurek@edelin:~/Progetti/iMX28/bootrom$ rm rootfs-EK340.img
```

ed infine lanciare il build scegliendo il target opportuno (EK340 - scelta n.3)

```
eurek@edelin:~/Progetti/iMX28/bootrom$ ./build.sh distclean

----------------------------
---- TARGET BOARD SELECT ---
----------------------------

-(1) i.MX28 DevBoard EVK

-(2) EK330 (Riello NetMan NG)

-(3) EK340 (Generic iMX28-TFT Eurek)

-(4) EK350 (Riello/AROS)

Choice (1..3) (Press Enter when done):
```

al termine del processo (che dipendentemente dalla potenza di calcolo del proprio sistema puo` impiegare svariati minuti (se non alcune ore) ed occupare qualche Gigabyte di disco) si potra` lanciare un comando ulteriore che permette di creare un'immagine virtuale di una sdcard da poter copiare direttamente su una SDCard (da almeno 4GB) con qualsiasi sistema che permette la scrittura in tali dispositivi.

```
eurek@edelin:~/Progetti/iMX28/bootrom$ ./create_disk_image.sh
```

Copiare su una sdcard il file risultante ed inserirla sul sistema embedded in ***modalita` di recovery***, attendere la fine dell'installazione e verificare che sia stato installato il nostro pacchetto ed anche aggiornata la versione del sistema operativo in ***/etc/RootFS_version***.

COME CONNETTERSI AL SISTEMA EMBEDDED

Ci sono vari modi con i quali connettersi al sistema Linux Embedded Edelin e di seguito ne presenteremo due: via *seriale* e via *ethernet con protocollo SSH*.

Per il primo caso è sufficiente aprire una console/terminale e digitare al prompt: **screen /dev/ttyJEK0 115200** (al connettore J3 va collegato il **debug board EK206** via USB e su Linux non vanno installati i driver in quanto già presenti nel kernel in uso)

Digitare "***root***" al prompt "login as:" e come password inserire la password di default "**toor**" e premere **INVIO**.

```
Debian GNU/Linux 7 edelin ttyAMA0

edelin login: root
Password:
Linux edelin 3.12.1-EK20131115 #1 Fri Nov 15 13:34:53 CET 2013 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@edelin:~#
```

Da questo momento abbiamo accesso completo alla board embedded tramite una shell (***bash*** nello specifico) con varie utilities come *super-utente* (root).

Analogamente come per la connessione via seriale, per la connessione via SSH occorre procedere avendo conoscenza dell'**indirizzo IP** della scheda. Tipicamente potrebbe essere: **192.168.1.2** dipendentemente dal ***server DHCP*** presente sulla rete alla quale é connessa la scheda.

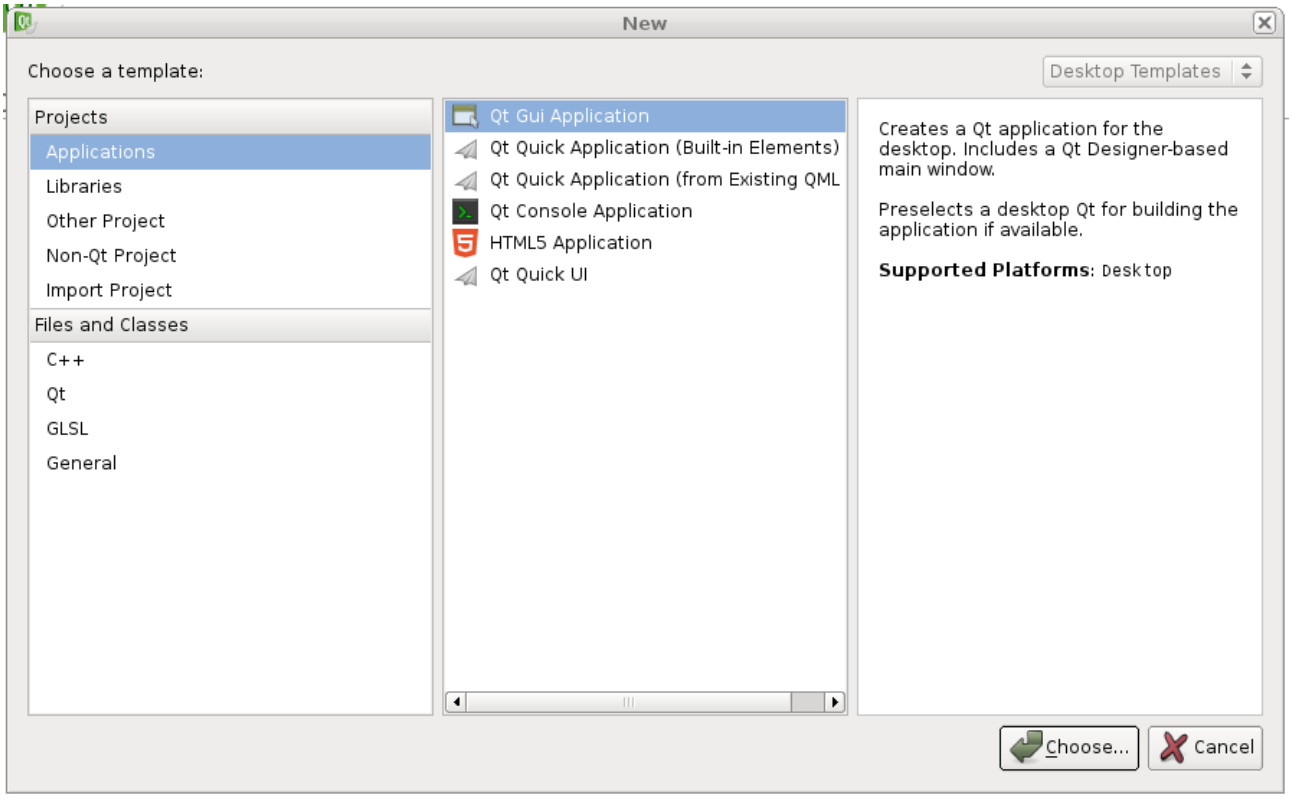
```
# ssh root@192.168.1.20
root's password:
Debian GNU/Linux 7 edelin ttyAMA0

edelin login: root
Password:
Linux edelin 3.12.1-EK20131115 #1 Fri Nov 15 13:34:53 CET 2013 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@edelin:~#
```

Utilizzo di Qt Creator

La libreria prescelta per la progettazione e debug di interfacce grafiche sulla scheda basata su **Edelin** é la famosa Libreria Qt (nello specifico la versione 4.8.4). Vi sono anche altre librerie presenti nel sistema ma per semplicità in questo documento faremo riferimento alla sola libreria Qt.

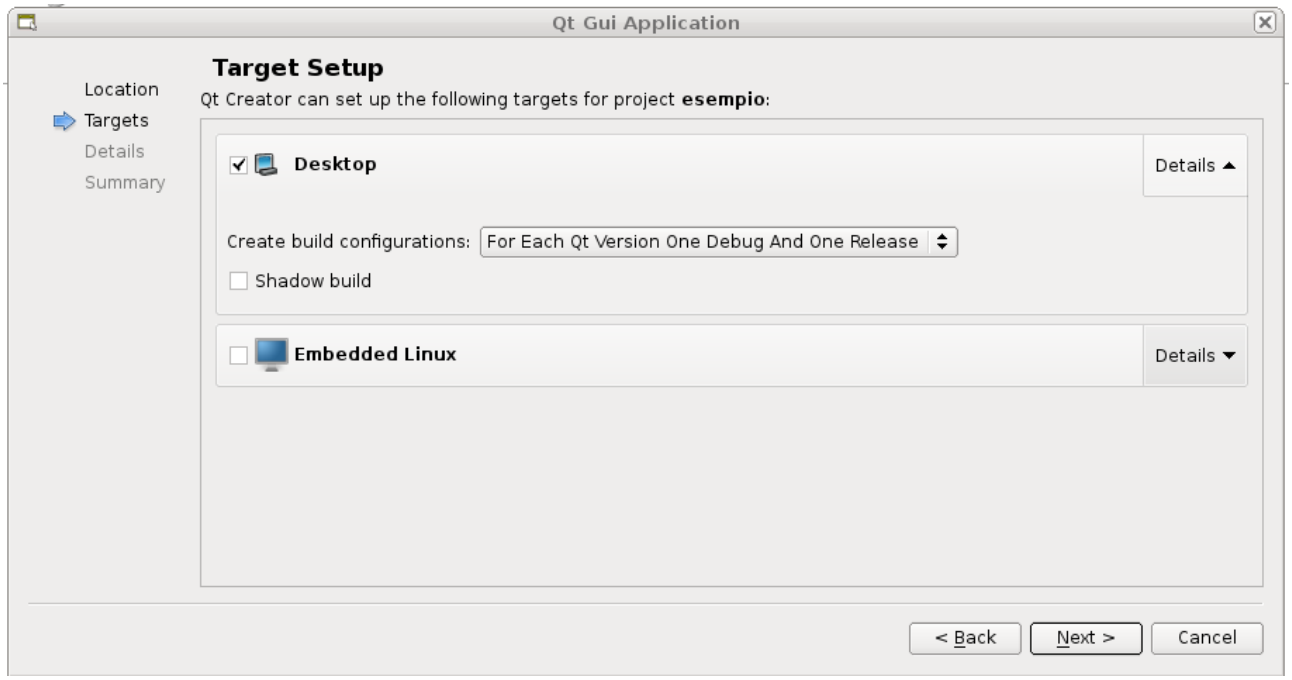
Per testare fin da subito la semplicità con la quale é progettato il sistema di sviluppo, si puó procedere aprendo il programma **Qt Creator** posto sotto la voce ***Application → Programming → Qt Creator*** .

Dal menu proncipale selezionare ***File → New File or Project***, comparirà la schermata seguente:



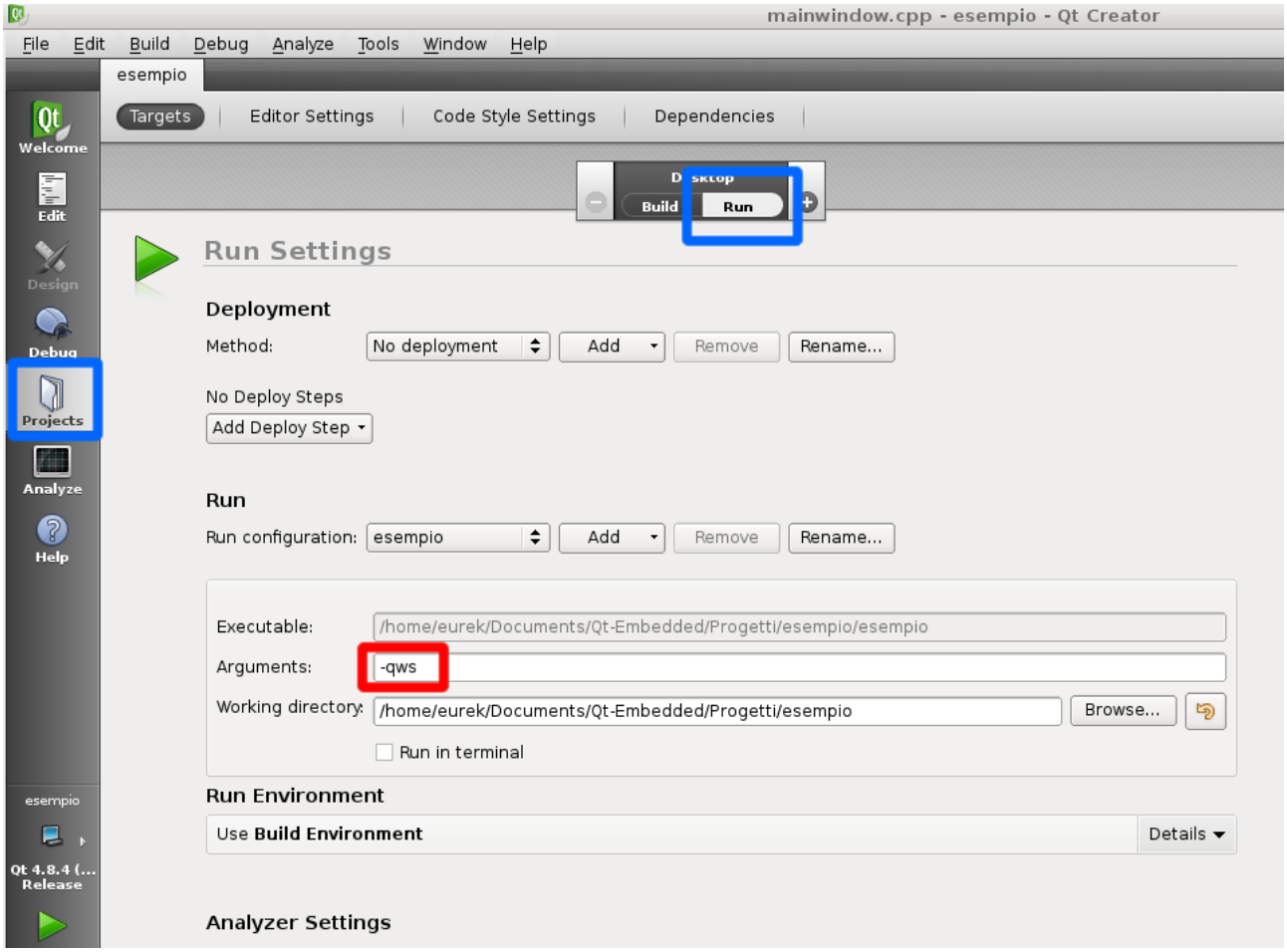
Dopo aver selezionato ***Qt Gui Application*** e premuto il tasto ***"Choose"*** verrà richiesto dove memorizzare i file di progetto ed il nome del progetto stesso. Selezionare la directory ***/home/eurek/Documents/Qt-Embedded/Progetti*** e come nome il semplice ***esempio***.

A questo punto viene richiesto di selezionare quale tipo di Target, e per il nostro caso si puó optare per il semplice ***Desktop*** (e non il piú complesso ***Embedded Linux***) e ***deselezionare*** la voce *Shadow build*



Confermare tutte le domande di rito (se si usa un sistema di versione come *CVS*, se si vuole una *mainWindow* come oggetto principale, ecc.,...) premendo in successione il tasto ***"Next"*** fino alla fine della configurazione che viene confermata con il tasto ***"Finish"***. Al termine si aprirà Qt Creator con il nostro progetto completo di sorgenti ecc.,...

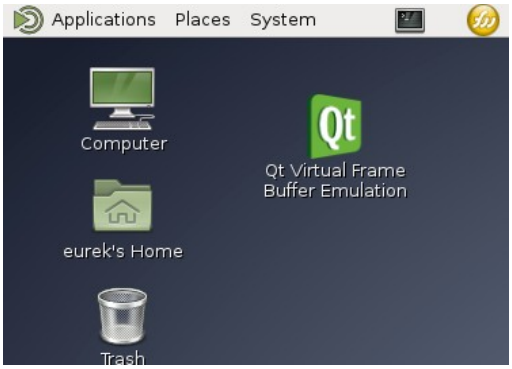
Selezionare dal menú laterale la voce ***Project*** e nel riquadro di ***Run*** → ***Arguments*** aggiungere il suffisso ***-qws*** per informare il sistema che la nostra applicazione dovrà essere eseguita in un sistema *Qt Embedded* con *QWS Server:*



Tale procedura si rende necessaria per poter **unificare** l'ambiente grafico sia sul PC che su scheda Embedded poiché ogni sistema embedded é (quasi) sempre *avaro di risorse* (come dimensione della RAM e della potenza di calcolo rispetto ad un PC tradizionale). Vi potrebbero essere degli attributi e/o caratteristiche che porterebbero a risultati differenti su *Qt per X11* rispetto a *Qt per Embedded*, e quindi per uniformare il comportamento del layout della Gui si é scelto di optare per **Qt-Embedded** (-qws) anche nel PC di sviluppo.

Lanciando ***Build*** → ***Run Qmake*** si re-impostano i file di configurazione corretti per un build su Qt Embedded.

Per verificare che tutto sia funzionante lanciare il framebuffer virtuale dal desktop di dimensione 800x480 pixels e con profondità colore di 16 bit tramite l'icona appropriata:



che rispecchia cosí le dimensioni e le caratteristiche di un comune display da 7" [2].

Infine con un run (***Build → Run***) si puó vedere la propria applicazione girare sul framebuffer virtuale.

A questo punto una volta testata e debuggata la propria interfaccia sul PC nel frame buffer virtuale é possibile preparare l'applicazione per il lancio sul processore presente sulla scheda embedded Edelin.

Procedere con i passi indicati di seguito:

- Ripulire il *tree dell'applicazione* da tutti i file generati automaticamente dal sistema di build tramite l'esecuzione del clean tramite ***Build → Clean all***
- **Chiudere** il progetto di Qt Creator
- Copiare il progetto nella cartella del **chroot** nel percorso apposito (sono necessari i permessi di superuser) utilizzando una shell:

```
$ sudo cp -R  /home/eurek/Documents/Qt-Embedded/Progetti/esempio \
/home/eurek/debian-armel/wheezy/home/eurek/Documents/Qt-Embedded/Progetti
```

- Eseguire il **chroot** sulla Debian Wheezy 7 **Edelin per ARM**:

```
$ sudo chroot /home/eurek/debian-armel/wheezy
# su - eurek
```

- Lanciare lo script per la preparazione delle variabili di ambiente:

```
$ source /home/eurek/Documents/Qt-Embedded/prepareQt.sh
```

2 : é possibile modificare le impostazioni delle dimensioni dello schermo e della profondità colore, semplicemente modificandone i rispettivi valori nelle *informazioni-proprietà* dell'icona.

- Entrare nella directory del progetto e invocare in sequenza il ***QMake*** per generare il *Makefile* e i file *.pro* adatti alla versione emulata ARM:

```
$ cd ~/Documents/Qt-Embedded/Progetti/esempio/
$ qmake  -project
$ qmake  -makefile
$ qmake
$ make  -j 2
```

```
/opt/qt/bin/uic mainwindow.ui -o ui_mainwindow.h
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB
-DQT_NETWORK_LIB -DQT_CORE_LIB -DQT_SHARED -I/opt/qt/mkspecs/qws/linux-generic-
g++ -I. -I/opt/qt/include/QtCore -I/opt/qt/include/QtNetwork -I/opt/qt/include/QtGui
-I/opt/qt/include -I. -I. -I. -o main.o main.cpp
/opt/qt/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB
-DQT_SHARED -I/opt/qt/mkspecs/qws/linux-generic-g++ -I. -I/opt/qt/include/QtCore
-I/opt/qt/include/QtNetwork -I/opt/qt/include/QtGui -I/opt/qt/include -I. -I. -I. mainwindow.h -o
moc_mainwindow.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB
-DQT_NETWORK_LIB -DQT_CORE_LIB -DQT_SHARED -I/opt/qt/mkspecs/qws/linux-generic-
g++ -I. -I/opt/qt/include/QtCore -I/opt/qt/include/QtNetwork -I/opt/qt/include/QtGui
-I/opt/qt/include -I. -I. -I. -o mainwindow.o mainwindow.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB
-DQT_NETWORK_LIB -DQT_CORE_LIB -DQT_SHARED -I/opt/qt/mkspecs/qws/linux-generic-
g++ -I. -I/opt/qt/include/QtCore -I/opt/qt/include/QtNetwork -I/opt/qt/include/QtGui
-I/opt/qt/include -I. -I. -I. -o moc_mainwindow.o moc_mainwindow.cpp
g++ -Wl,-O1 -Wl,-rpath,/opt/qt/lib -o esempio main.o mainwindow.o moc_mainwindow.o
-L/opt/qt/lib -lQtGui -L/opt/qt/lib -lQtNetwork -lQtCore -lglib-2.0 -ldbus-1 -lpthread
```

- Al termine della compilazione otterremo un file eseguibile in formato ARM:

```
$ file esempio
```

```
esempio: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared
libs), for GNU/Linux 2.6.26, BuildID[sha1]=0xcf89a9a50df17e3c0865bc29ca73b23477145bf9, not
stripped
```

UN ULTIMO CONSIGLIO...

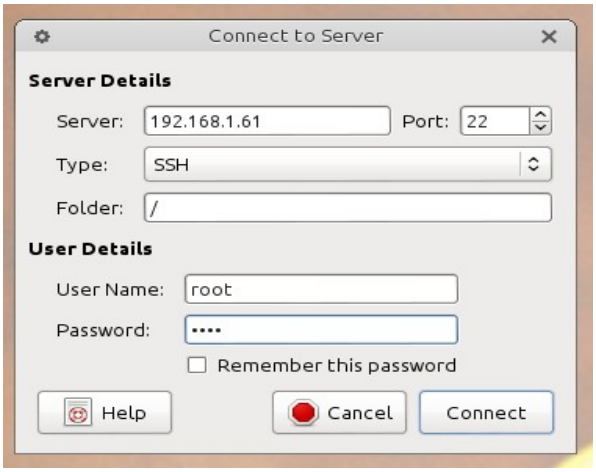
Durante lo sviluppo della propria applicazione si potrà rendere necessario che le eventuali immagini (*jpeg, gif, png, svg*) siano all'interno di una sotto-cartella del progetto con un nome tipo: *images.*
Ció é il default per tutti gli esempi di Qt ed é anche quello che Qt suggerisce, soprattutto per come il codice faccia riferimento a quella cartella con il path relativo "*./images/....*".

SISTEMA DI TRASFERIMENTO DI FILE NEL SISTEMA EMBEDDED

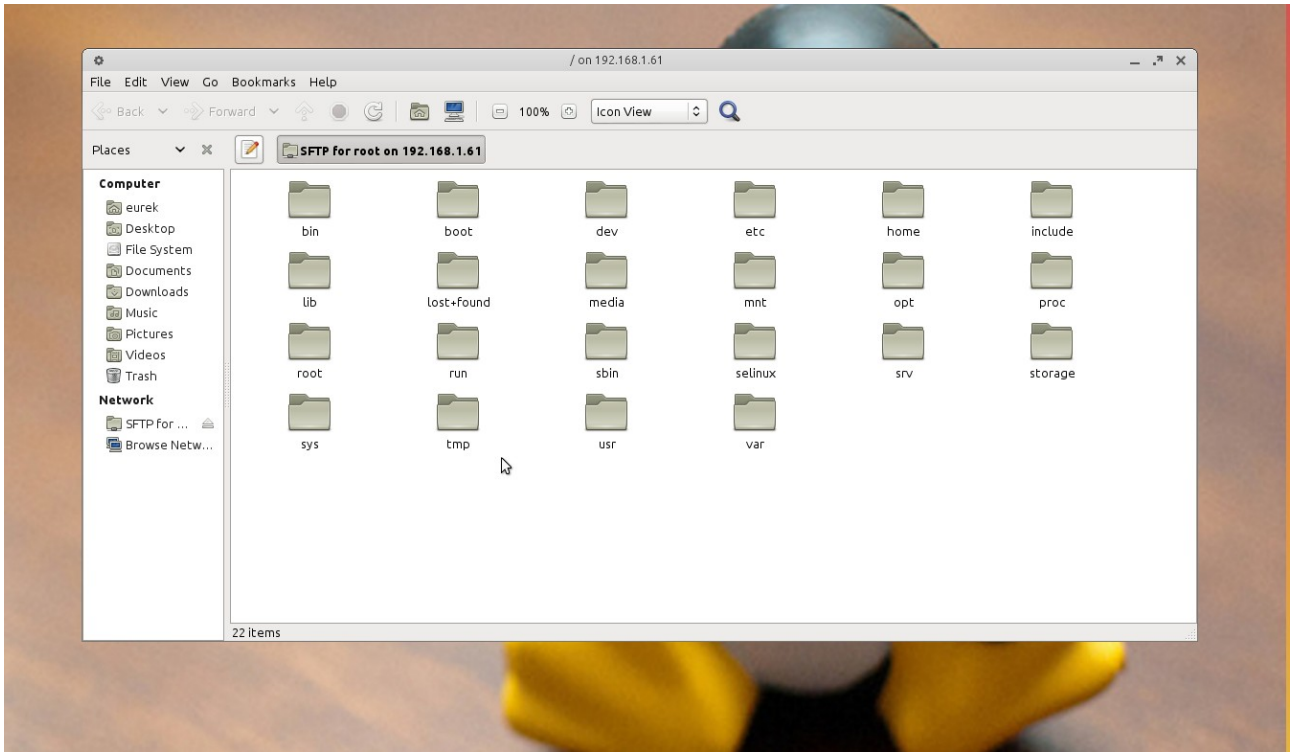
Durante lo sviluppo della propria applicazione puó essere utile trasferire di una certa quantità di informazioni (file di configurazione, immagini, eseguibili) direttamente nel sistema embedded. Ció é notevolmente semplificato dal fatto che sulla scheda embedded sono attivi per default un server **SSH** (OpenSSH) ed eventualmente un server **FTP** (vsftp, ma che di default accetta connessioni di tipo *anonymous* e che quindi andrebbe esplicitamente configurato con policy adeguate).

Si potranno trasferire file sul sistema nei percorsi voluti, semplicemente lanciando il client dell'ambiente grafico per la connessione remota:

Selezionare successivamente il tipo di protocollo (***SSH***) ed inserire utente e password (per esempio **root** con password **toor**):



Si aprirà una finestra di connessione verso il sistema embedded come mostrato nella schermata seguente:



A questo punto utilizzando il mouse é possibile spostare/cancellare/rinominare i file sul sistema embedded in maniera molto semplice ed intuitiva.

É possibile inoltre, utilizzando il sistema di **upgrade automatico su USB** usufruire di un comodo sistema di aggiornamento via PenDisk USB che permette di aggiornare il *Bootloader*, il *Kernel Linux*, i *Device Drivers*, (ed il *Sistema Operativo Debian Wheezy Edelin in modalita` crittografata)* ed inoltre anche l'applicativo utente, il tutto sfruttando l'hook (gancio) al file */etc/rc.local* adibito proprio a questo scopo.

Raccomandazioni per il porting su embedded

Come effettuare il porting del software in ARM Linux

La maggior parte del software che gira sulla scheda embedded è scritto in C. Il C non è tipicamente un linguaggio direttamente portabile, a differenza di Java. Per scrivere codice veramente portabile in C, in genere richiede dello sforzo aggiuntivo. Le informazioni seguenti, descrivono i problemi comuni che potremmo incontrare quando desideriamo portare applicazioni in Linux ARM, specialmente da un Linux x86.

Problemi di portabilità del C

Ci sono un certo numero di aree nelle quali il comportamento delle definizioni di un programma C dipendono fortemente dall'architettura sulla quale il programma sta girando. E` un comportamento che dipende dalle peculiarità del Sistema Operativo, dal compilatore, dalle librerie ed infine dalla CPU.

Signed vs. Unsigned

Lo *standard C* dice che i tipi char possono essere signed od unsigned di default. Su Linux x86, *char* è di tipo *signed* di default. Su Linux ARM, il tipo char è *unsigned* di default.

Il paragonare un *char* ad un numero con segno negativo, restiuirà sempre 0, perchè il *char* è sempre unsigned e quindi positivo.

Allineamento di un puntatore

Su molte architetture di varie CPU, il sistema di memoria richiede che il caricamento di valori nelle variabili più grandi di un byte, deve essere correttamente allineato. Tipicamente, questo significa che le quantità a 2-byte deveno essere allineate ad un indirizzo pari, una quantità a 4-byte deve essere allineata su un indirizzo multiplo di 4 ed alcune volte addirittura di 8. Dipendentemente dalla CPU e dal sistema operativo, il caricamento ed il salvataggio di variabili in modo non-allineato può provocare alcune eccezioni, segnali che possono o meno essere manipolati dal sistema operativo stesso, o possono essere arrotondati in maniera silenziosa nell'allineamento non voluto.

Nei sistemi x86 l'allineamento non impone alcuna restrizione, così alcuni programmi scritti per x86 possono fare uso di allineamenti erronei per altre architetture. Linux ARM per default, allinea al boundary più appropriato per quel determinato tipo di variabile. Questo può essere addirittura considerata una *feature*, perché ci permette di ruotare valori, memorizzando e caricando con tipi differenti di allineamento di puntatori. (Ma c'è un'istruzione di *rotate* che potrebbe essere eseguita più velocemente per questo tipo di operazioni. ;-)

Dimensionamento delle strutture ed allineamento

Ecco un consiglio:

```
struct foo_t {   u16 x; } __attribute__ ((packed));
```

Utilizziamo sempre l'attributo *__attribute__ ((packed))* per avere le stesse dimensioni della struttura su qualsiasi architettura.

L'attributo *packed* dice al compilatore GCC di impacchettare la struttura foo_t in 2 bytes (tipo u16), invece che espanderla in 4 bytes come è lecito aspettarsi in una architettura a 32bit.

Utilizzare l'Overlay della Memoria per convertire i DataTypes

Questa operazione è veramente **NON** portabile. Il codice **DEVE** essere scritto cosicché l'allineamento, la dimensione e l'endianness siano tutti correttamente manipolabili su tutte le architetture supportate.

Problemi di Endianness

Ci sono due layout di base della memoria utilizzati dalla maggior parte dei computers, e sono definiti come *big endian* e *little endian*. Sulle macchine *big endian*, il byte più significativo di un oggetto in memoria è memorizzato nell'indirizzo meno significativo (il più vicino allo zero). In maniera opposta, nei sistemi *little endian*, il byte meno significativo è memorizzato nell'indirizzo più vicino allo zero.

Vediamo un esempio:

```
int x = 0xaabbccdd;
unsigned char b = *(unsigned char *)&x;
```

In una macchina *big endian*, b potrebbe ricevere il byte più significativo di x, 0xaa. Sulle macchine *little endian*, b potrebbe ricevere il byte meno significativo di x: 0xdd.

L'architettura x86 è *little endian*. Molti processori ARM supportano entrambi i modi, ma generalmente sono utilizzati nel modo *little endian*.

I problemi di tipo Endian possono verificarsi sotto due condizioni:

- Quando condividiamo dati binari tra macchine di endianness differenti.
- Quando effettuiamo casting di puntatori tra tipi differenti in dimensione.

Nel primo caso, i dati appaiono nella locazione corretta, ma saranno interpretati in maniera differente in macchine diverse. Se una macchina *little endian* memorizza 0xaabbccdd in una locazione, una macchina *big endian* potrebbe leggerla come 0xddccbbaa.

Nel secondo caso, una macchina *little endian* non ha nessun problema: un *char*, uno *short*, oppure un *int* memorizzato in una variabile di tipo *int* avrà lo stesso indirizzo. Su una macchina del tipo *big endian*, se volessimo essere in grado di memorizzare uno *short* e leggerlo come un *int* dovremmo incrementare il puntatore cosicché il MSB cada proprio nel posto giusto. E questo a volte può essere fonte di comportamenti errati del codice.

NOTE

NOTE

TouchMe®, tutto il bello del touch, senza complicazioni.

touchMe SIZE S / M / L SMALL T//COTTON touchMe SIZE S / M / L MEDIUM T//COTTON touchMe SIZE S / M / L LARGE

CARATTERISTICHE TECNICHE // TECHNICAL FEATURES*

Dimensions	**TouchMe Cotton** Small - Total Frame size mm 123 x 85 **TouchMe Cotton** Medium - Total Frame size mm 186 x 130 **TouchMe Cotton** Large - Total Frame size mm 251 x 170
Power supply	10-20Vdc or 21-38Vdc (you can ask for your customised version when placing the order)
Microprocessor	Freescale i.MX28-6 ARM926 @454MHz
Memory	SDRAM DDR2: 128 MiB Flash eMMC: 2 GiB FRAM: 8 KB (non-volatile, writing without waiting time and without limits in re-writing).
LCD	LCD 4.3" with resistive touch-screen - Resolution 480x272 [**TouchMe Cotton** Small] LCD 7" with resistive touch-screen - Resolution 800x480 [**TouchMe Cotton** Medium] LCD 10,1" with resistive touch-screen - Resolution 1024x600 [**TouchMe Cotton** Large]
Real Time Clock	SuperCap buffered calendar Clock. It guarantees at least one week without power supply before losing date and time
Ethernet	1 Ethernet Port 10/100 Mb Plug with activity/link warning led
Wireless	Wi-Fi b/g/n Module with connector for external antenna
USB 2.0	1 USB Host connector, type A receptacle 1 USB OTG connector, type A/B microUSB receptacle
microSD card	microSD card slot with push-push type connector and automated retention of the card
Audio	3.5mm jack stereo connector for line-in input 2-pins connector for microphone (microphone accessory with 50mm lead) 3.5mm jack stereo connector for earphones output output pins for Mono-loudspeaker on terminal or small loudspeaker mounted on the board, as an alternative
Connection	UART - serial connector with RS232 or TTL 3V or 5V levels (switch selectable) RS485 bus connector with line-end terminator resistor-pack (selectable via dip-switch) CanBus DB9 connector with line-end terminator resistor-pack (selectable via a dip-switch) I2C (Two Wire Interface) Bus 5V levels (3.3V on request) connector with an optional Interrupt digital input One-Wire connector
Jtag Port	Yes with custom connector
Software	Debian Wheezy Linux, Qt 4 Libraries and GCC Compiler

ACCESSORI // ACCESSORIES

JEK-KEY-EK206	USB JTAG Emulator with debug UART
MIC-EK340	Microphone with 50mm lead
WIFI-ANT	WiFi Antenna kit

* soggetto a modifiche senza preavviso // *subject to change without notice*

eurek ELECTRONIC ENGINEERING

Eurek s.r.l.
Via Celletta 8/b
40026 Imola (BO) - Italy
www.eurek.it

Tel: **0542 609120**
Fax: **0542 609212**
P.IVA: **00690621206**
CF: **04020030377**

Based on Linux™