ISSN 2039-7941
Anno 2017_Numero 384

rapporti tecnici INGV

Implementazione del Sistema Embedded per l'Osservatorio multidisciplinare di Portopalo di Capopassero

384

Istituto Nazionale di Geofisica e Vulcanologia

Segreteria di Redazione
Francesca Di Stefano - Referente
Rossella Celi
Tel. +39 06 51860068
redazionecen@ingv.it

in collaborazione con:
Barbara Angioni (RM1)

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO

Rappresentante legale: Carlo DOGLIONI
Sede: Via di Vigna Murata, 605 | Roma

ISSN 2039-7941
Anno 2017_Numero 384

rapporti tecnici

INGV

IMPLEMENTAZIONE DEL SISTEMA EMBEDDED PER L'OSSERVATORIO MULTIDISCIPLINARE DI PORTOPALO DI CAPOPASSERO

Nicola Mario Marcucci[1], Umberto Apponi[2]

[1]**INGV** (Istituto Nazionale di Geofisica e Vulcanologia, Sezione Geomagnetismo, Aeronomia e Geofisica Ambientale)
[2]**SpacEarth Technology srl**

384

Come citare: Marcucci N.M., Apponi U., (2017). Implementazione del Sistema Embedded per l'Osservatorio multidisciplinare di Portopalo di Capopassero. Rapp. Tec. INGV, 384: 1-24.

Indice

Introduzione 7

1. Requisiti implementativi 7

2. Gli strumenti di sviluppo 7

3. Panoramica del flusso di progettazione da Vivado a PetaLinux 8

4. Petalinux: Configurazione 9

4.1 Creazione del Progetto e importazione dell'hardware 9

4.2 Configurazione di Linux 9

4.3 Creazione delle Apps utente 15

4.4 App: NTP 15

4.5 App: Avrdude 17

4.6 App: Configuration 17

4.7 App: Timer 18

4.8 Modulo: Timermodule 18

4.9 Petalinux: Creazione e caricamento delle immagini sulla microzed 18

Conclusioni e sviluppi futuri 19

Bibliografia 20

Sitografia 20

Introduzione

Il lavoro presentato in questo articolo descrive lo sviluppo del sistema operativo embeddedd dell'osservatorio di fondo mare di Portopalo di Capopassero e rappresenta lo step successivo rispetto a quanto introdotto in [Giovanetti et al., 2017]. Il sistema operativo embedded ospiterà il software di acquisizione, i tools di gestione e la sincronizzazione temporale. In particolare, analizzeremo le fasi di implementazione del sistema operativo embedded, realizzate attraverso l'uso di diversi prodotti della **Xilinx**[1]**.** Per lo sviluppo è stato utilizzato un host con sistema operativo Linux. Il lavoro è stato realizzato su Ubuntu 14.04 LTS (64 bit) con installata la suite Xilinx Vivado 2014.4 e il toolchain Petalinux 2014.4, descritti di seguito.

1. Requisiti implementativi

In fase di progettazione come descritta in [Giovanetti et al., 2017], si è deciso di sviluppare un sistema operativo embedded modellato sull'hardware già realizzato. Il sistema embedded dovrà ospitare il software di acquisizione che riceverà i dati dai sensori, collegati su porta seriale, li marcherà temporalmente e li invierà a terra tramite TCP/IP. Affinché la marcatura temporale dei dati sia affidabile, è necessario un sistema di sincronizzazione della data di sistema. Il software di acquisizione non gestirà i dati di tutti i sensori. Il sismometro e gli idrofoni ad alta frequenza trasmettono direttamente i dati a terra su ethernet. Il sismometro ha un proprio client Network Time Protocol (NTP) per la sincronizzazione temporale, mentre gli idrofoni possono ricevere un segnale di sincronizzazione Pulse per second (**PPS**[2]). Nella progettazione dell'hardware è stato quindi realizzato il timer PPS per la sincronizzazione temporale degli idrofoni. L'osservatorio sarà deposto sul fondo del mare e quindi non sarà fisicamente accessibile fino al momento del suo recupero. Questo rende necessaria la gestione da remoto del sistema (riavvio del software di acquisizione, verifica malfunzionamenti ecc). Si potrebbe inoltre rendere necessario dover caricare un nuovo firmware sull'Arduino, quindi si deve implementare la possibilità di poterlo trasferire. In caso di sviluppi successivi alla deposizione, si vuole poter aggiornare l'immagine del sistema embedded in maniera sicura. Va implementata quindi la possibilità di caricamento da remoto del firmware.

2. Gli strumenti di sviluppo

Prima di procedere nella descrizione del flusso di progetto è necessario descrivere i tool che sono stati impiegati:

- **Vivado:** ambiente di progettazione, prodotto dalla Xilinx**,** ideato per affrontare i problemi che caratterizzano la realizzazione sugli **FPGA**[3] più evoluti di System-on-Chip (SoC) sempre più grandi e complessi. Esso permette sia l'utilizzo di moduli hardware preesistenti, sia lo sviluppo di moduli custom in linguaggio VHDL o Verilog. Il SoC così progettato viene sintetizzato e caricato sull'FPGA [Un nuovo ambiente di Progettazione, 2016].
- **SDK (**Xilinx Software Development Kit): ambiente IDE, incluso nella suite Vivado, per la creazione di piattaforme software e applicazioni mirate per i processori Xilinx embedded. SDK funziona con progetti hardware creati con Vivado.
- **PetaLinux** toolchain: strumento di sviluppo Xilinx che contiene tutto il necessario per costruire, sviluppare, testare e implementare sistemi Linux embedded.
  PetaLinux consiste di tre elementi chiave: i binari delle immagini avviabili pre-configurate, un sistema linux completamente personalizzabile per il dispositivo Xilinx, e PetaLinux SDK che include strumenti e utilità per automatizzare le operazioni complesse attraverso la configurazione, la costruzione, e la distribuzione [What-is-PetaLinux, 2016].

[1]La **Xilinx** è un'azienda produttrice di dispositivi logici programmabili (PLD) e in particolare di dispositivi FPGA
[2]Un **PPS** è un segnale elettrico che indica molto precisamente l'inizio di un secondo.
[3]**FPGA** è l'acronimo di Field Programmable Gate Array. È un circuito integrato le cui funzionalità sono riconfigurabili via software.

3. Panoramica del flusso di progettazione da Vivado a PetaLinux

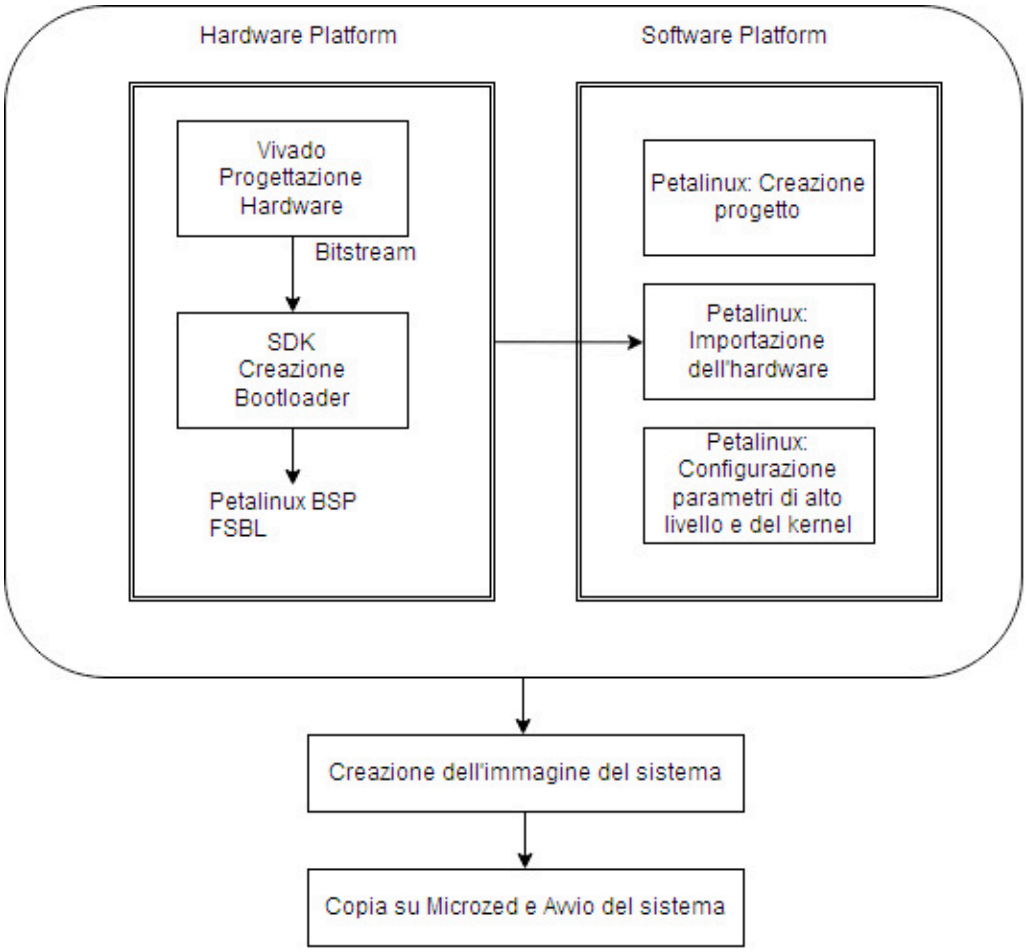

Figura 1. Fasi di progettazione e implementazione del progetto.

Dal diagramma in Figura 1 si evince che il flusso di lavoro inizia con la creazione della piattaforma hardware all'interno di un progetto dell'ambiente di sviluppo Vivado.

Una volta progettato il sistema da implementare nella logica programmabile, si genera un bitstream completo con il quale configurare l'FPGA.

Il design cosi creato viene esportato in SDK, attraverso il quale viene creato il Petalinux Board Support Package (BSP) ed il First Stage BootLoader (FSBL). Il BSP è il codice di supporto per una specifica configurazione dell'hardware di una data scheda, compatibile con il sistema operativo utilizzato [Board_Support_Package, 2016]. L'FSBL è responsabile del caricamento del bitstream sull'FPGA e della configurazione dell'architettura del sistema di elaborazione Zynq al momento dell'avvio [First Stage Bootloader, 2016].

Per la creazione della piattaforma software da far girare sulla configurazione hardware realizzata con Vivado, si usa il toolchain Petalinux. Attraverso i comandi di Petalinux si realizza un sistema Linux personalizzato, adattando i file di configurazione alle specifiche della configurazione hardware. Per i moduli hardware forniti dalla Xilinx i driver sono inclusi. Vengono generate un'immagine che contiene il sistema Linux ed una che ne permette l'avvio (bootloader). Entrambe vengono caricate su una sd card o una memoria flash interna, per essere avviate dalla Microzed.

Nel prossimo paragrafo sarà approfondita la procedura per lo sviluppo della piattaforma software, descrivendo più in dettaglio il funzionamento del toolchain Petalinux.

4. Petalinux: Configurazione

La parte del flusso di progetto relativa alla creazione della piattaforma software è incentrata essenzialmente sull'uso del toolchain Petalinux che, come esposto in precedenza è uno strumento di sviluppo Xilinx che contiene tutto il necessario per configurare e cross-compilare sistemi Linux embedded [PetaLinux 2014.4 Image with Custom Application, 2015].

Con Petalinux gli sviluppatori possono personalizzare il bootloader, il kernel di Linux, e le applicazioni Linux. Possono aggiungere nuovi kernel, driver di dispositivo, applicazioni, librerie.

Gli elementi fondamentali della piattaforma software sono il kernel linux, il filesystem rootfs, il bootloader u-boot, ed il device tree.

Device tree: È una struttura dati che descrive l'hardware. La struttura dei dati è un albero di nodi. I nodi contengono proprietà e nodi figlio, mentre le proprietà sono coppie nome-valore [Device Tree, 2016]. È utilizzata dal kernel per avere le informazioni sull'hardware effettivamente presente (ad esempio informazioni come la frequenza del clock, la tipologia ed il numero di periferiche presenti ecc).
U-Boot: È un Universal bootloader opensource utilizzato di frequente nella comunità Linux [U-boot, 2016].
Il bootloader è il programma che in fase di avvio del computer carica in memoria il kernel del sistema operativo, permettendone l'esecuzione e di conseguenza l'avvio del sistema [Boot loader, 2016].
rootfs: È un'immagine del filesystem che viene caricata in memoria RAM.
Kernel: Il kernel è il "cuore" di un sistema operativo. Fornisce tutte le funzioni essenziali per il sistema, in particolare la gestione della memoria primaria, delle risorse hardware del sistema e delle periferiche, assegnandole di volta in volta ai processi in esecuzione [Linux_(kernel), 2016]. Nel nostro caso utilizziamo un kernel **linux-xlnx-3.17**

4.1 Creazione del Progetto e importazione dell'hardware

Dopo aver installato e configurato il toolchain Petalinux, è necessario disporre del bitstream generato da Vivado, oltre al BSP e FSBL generati da SDK.

Si inizia con la creazione di un progetto all'interno dell'ambiente di Petalinux usando il comando:

```
$ petalinux-create --type project --template zynq --name <nomeprogetto>
```

Una directory con lo stesso nome del progetto contenente i file di configurazione di base verrà creata. Questa directory è la root del progetto.

A questo punto è necessario importare le specifiche hardware all'interno del progetto. Per fare questo avremo bisogno dell'FSBL e del BSP, entrambi generati da SDK. Dopo esserci assicurati di essere all'interno della root del progetto, lanciamo il comando per importare l'hardware:

```
$ petalinux-config –get-hw-description=<SDK_Export_directory>/<hw_platform_name>
```

Così facendo i file di configurazione saranno adattati alle specifiche dell'hardware.

È quindi necessario definire la configurazione del sistema prima di cross-compilare il kernel e generare l'immagine del filesystem. Petalinux mette a disposizione una serie di comandi per modificare le impostazioni del kernel e del rootfs.

4.2 Configurazione di Linux

Lanciato il comando petalinux-config si ha accesso al menù di configurazione dei parametri generali (vedi Figura 2). Qui si possono cambiare la versione del Kernel, la configurazione dell'u-boot (bootloader), le impostazioni automatiche generate nella fase di import vista in precedenza, il nome e la versione del firmware, il MAC address della scheda Ethernet e altro.

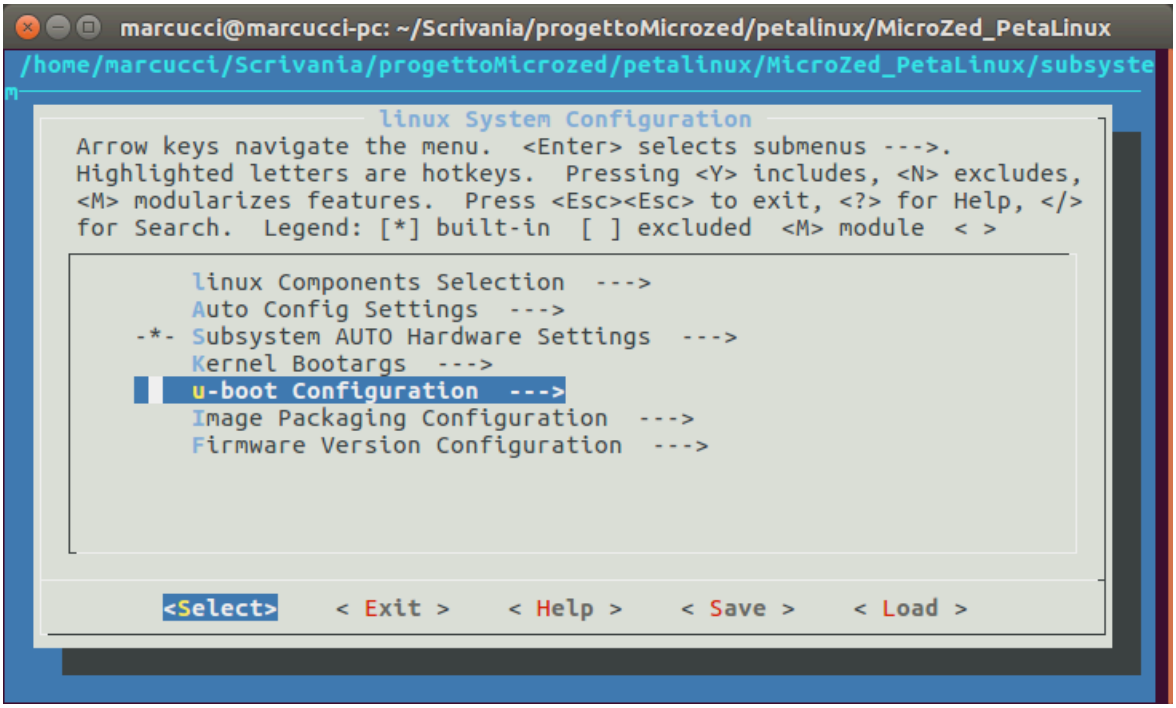

Figura 2. Menu di configurazione delle impostazioni generali (petalinux-config).

È stato necessario cambiare il tipo di storage da dove recuperare il file immagine, contenente kernel e rootfs, chiamata image.ub. Di default infatti il bootloader cerca l'immagine su una SD Card. Da menù principale → Subsystem Auto Hardware Settings → Advanced Bootable Images Storage setting → Kernel Image Settings → Image Storage Media, cambio l'impostazione da primary sd in ethernet (vedi Figura 3).

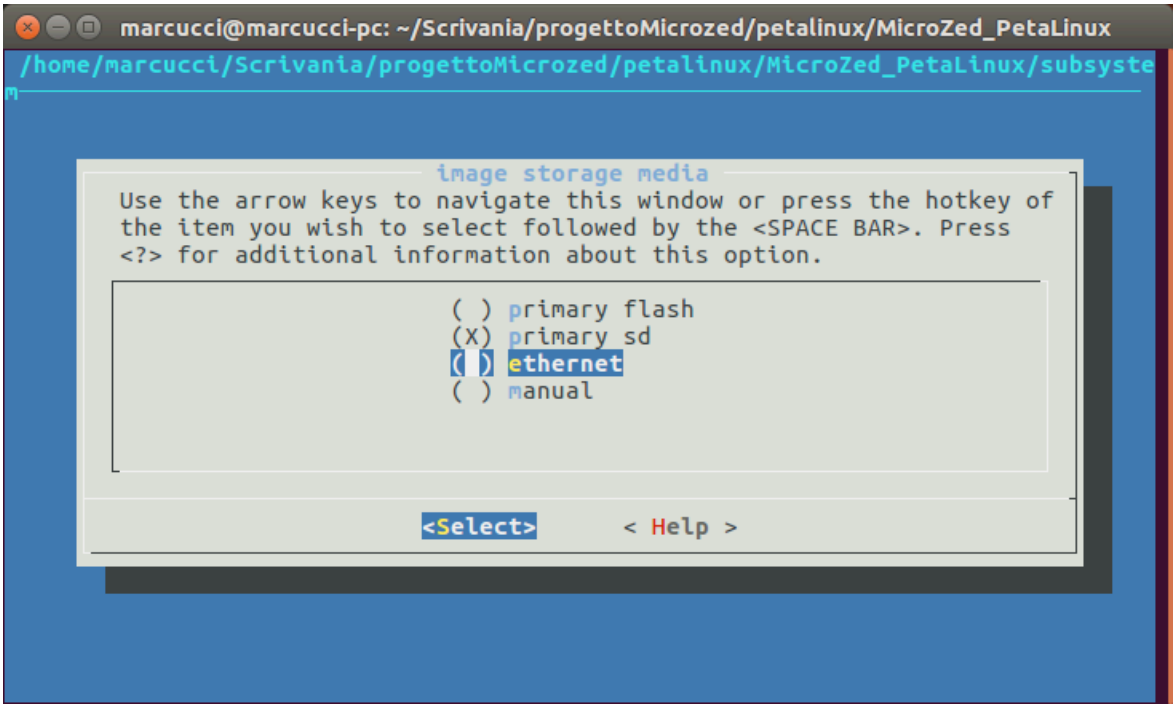

Figura 3. Cambio dell'image storage media: da primary sd a ethernet.

In tal modo il sistema farà il boot da rete recuperando l'immagine tramite il protocollo TFTP, che richiede di impostare l'indirizzo IP del server. Da U-boot configuration → TFTP Server IP imposto l'indirizzo IP del server (vedi Figura 4). Nella fase di sviluppo è stata predisposta una rete locale con un PC impostato come DHCP Server e TFTP Server e la microzed come client.

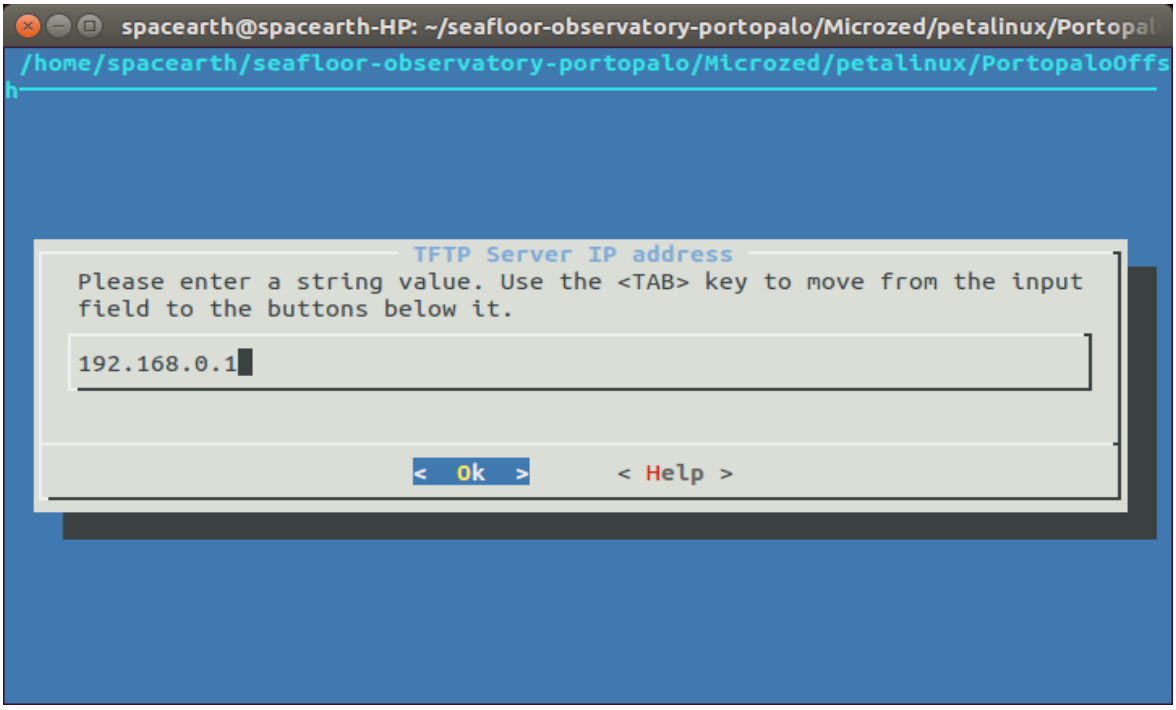

Figura 4. Impostazione dell'IP del server TFTP.

L'opzione può essere impostata modificando il file *config* in:

<root di progetto>/subsystems/linux/

```
CONFIG_SUBSYSTEM_U__BOOT_TFTPSERVER_IP="192.168.0.1"
```

Alla CPU module sono collegati diversi sensori tramite seriale RS232 e la scheda Arduino Mega del Power management Module (PMM) tramite USB [Giovanetti et al, 2017].

È stato quindi necessario abilitare alcune impostazioni nella configurazione del kernel. Per farlo, dalla root del progetto, si lancia il comando:

$ petalinux-config c kernel

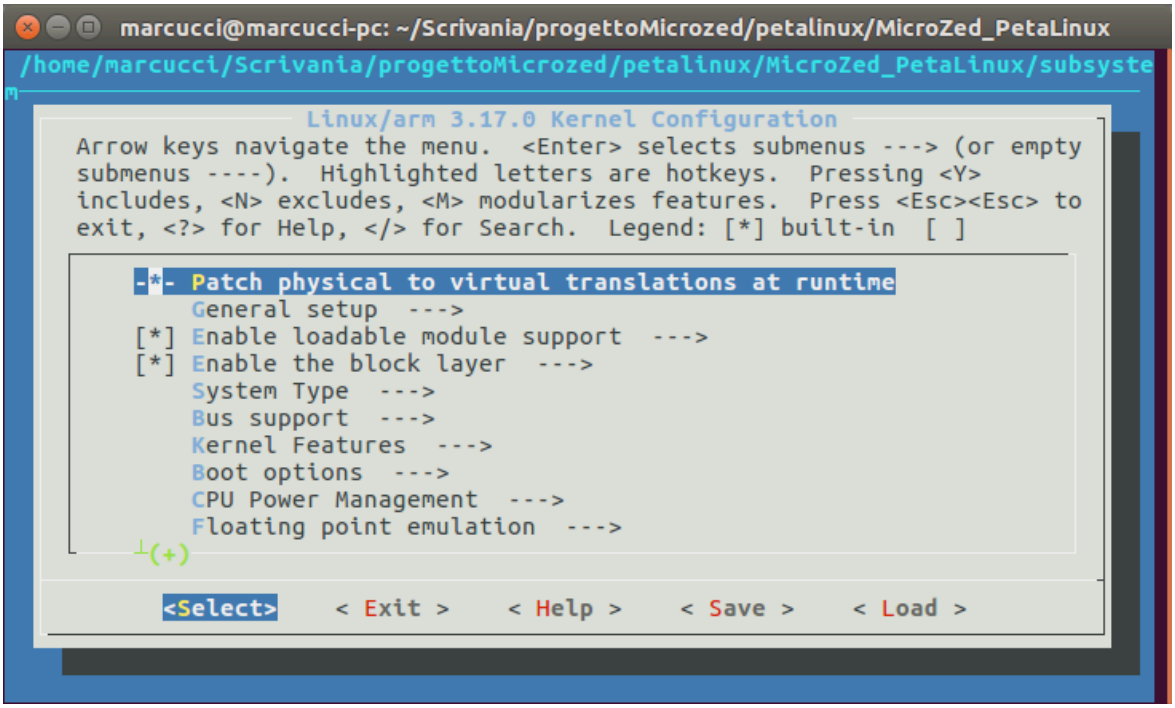

Figura 5. Menu di configurazione del kernel (petalinux-config -c kernel).

Le opzioni richieste possono essere abilitate tramite il menu, di cui alla Figura 5, oppure modificando direttamente il file di configurazione *config* situato in:

<root di progetto>/subsystems/linux/configs/kernel

Di seguito le opzioni attivate con la sintassi prevista all'interno del file di configurazione:

Per attivare l'usb e il relativo adattatore seriale:

```
CONFIG_USB_ANNOUNCE_NEW_DEVICES=y
CONFIG_USB_ACM=y
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_CONSOLE=y
CONFIG_USB_SERIAL_GENERIC=y
CONFIG_USB_SERIAL_CYPRESS_M8=y
CONFIG_USB_SERIAL_FTDI_SIO=y
```

Di default 4 seriali erano abilitate. Dal momento che sull'FPGA sono state sintetizzate 10 seriali UART, sono state modificate le seguenti impostazioni per consentire al sistema operativo di riconoscerle. Ne sono state impostate 12, perché il sistema permette di abilitare le seriali a multipli di 4.

```
CONFIG_SERIAL_8250_NR_UARTS=12
CONFIG_SERIAL_8250_RUNTIME_UARTS=12
```

Per accedere alla configurazione del rootfs si lancia il comando:

$ petalinux-config -c kernel (Apre l'interfaccia in Figura 6)

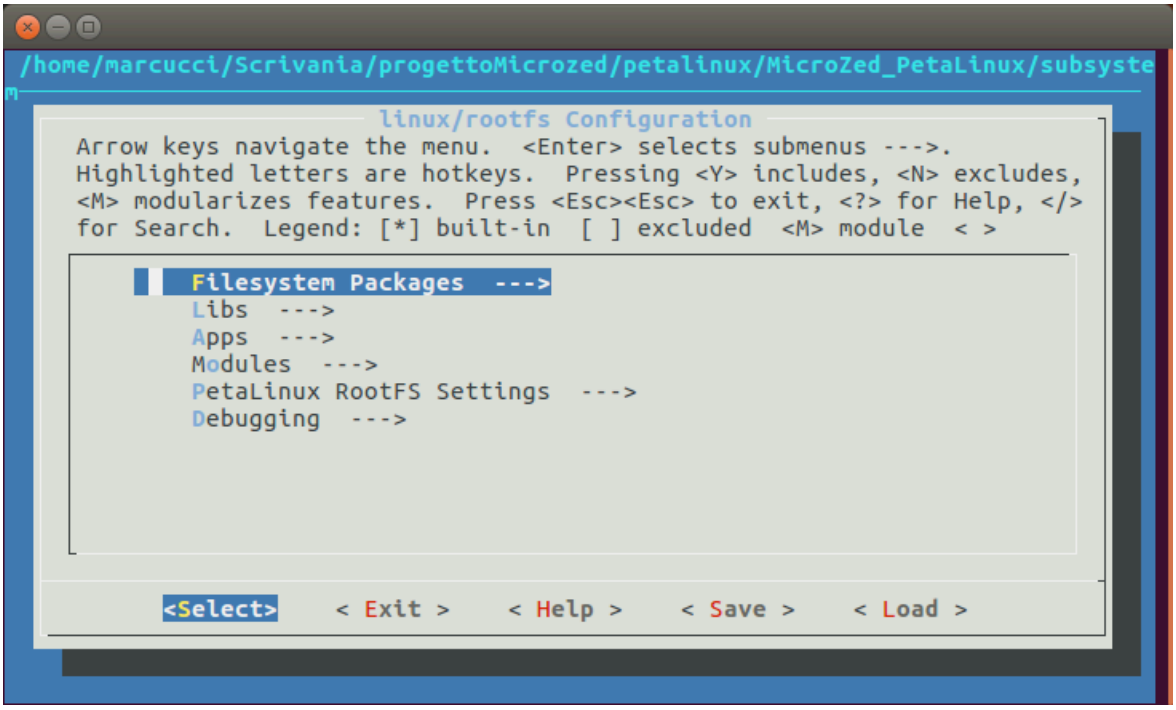

Figura 6. Menù di configurazione di rootfs (petalinux-config -c rootfs).

Per consentire l'accesso remoto alla shell Linux è stato necessario attivare l'**SSH** server. Da Filesystem Packages → console/network → dropbear sono stati attivati i pacchetti **Dropbear** e Dropbear-openssh-sftp-server (vedi Figura 7).

Questi stessi pacchetti possono essere attivati dal file di configurazione localizzato qui:

<root di progetto>/subsystems/linux/configs/rootfs

Le equivalenti opzioni nel file *config* sono:

```
CONFIG_ROOTFS_PACKAGES_DROPBEAR=y
CONFIG_ROOTFS_PACKAGES_DROPBEAR_OPENSSH_SFTP_SERVER=y
```

Dropbear è un server e client SSH abbastanza leggero, particolarmente indicato per i sistemi embedded.

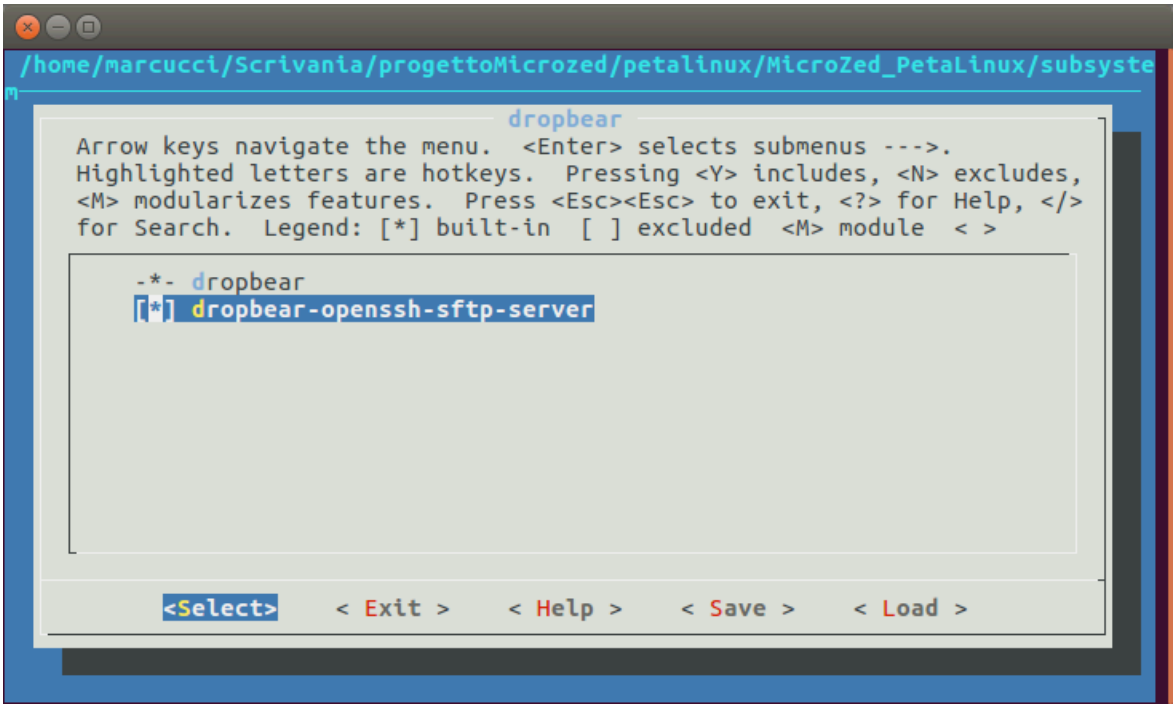

Figura 7. Rootfs: Abilitazione dei pacchetti SSH server e client.

L'ethernet PHY (physical layer) è costituito da un chip esterno all'FPGA. Pertanto le informazioni che lo riguardano non sono accessibili a Petalinux nel workflow finora descritto. Tali informazioni vanno aggiunte dallo sviluppatore nel file *system-top.dts*

I file dts costituiscono i sorgenti dai quali Petalinux compila il device-tree vero e proprio. Quelli che contengono le informazioni relative ai componenti sintetizzati nell'FPGA sono generati automaticamente al momento dell'importazione della piattaforma hardware. Il file system-top.dts può essere modificato dallo sviluppatore.

System-top.dts modificato:

```
&uart1{
device_type = "none";
    };

&gem0 {
        phy-handle = <&phy0>;
        phy-mode = "rgmii-id";
        ps7_ethernet_0_mdio: mdio {
                phy0: phy@0 {
                compatible = "marvell,88e1510";
                device_type = "ethernet-phy";
                reg = <0>;
                };
        };
};
```

4.3 Creazione delle Apps utente

Petalinux ci permette di creare applicazioni, definite all'interno di questo ambiente di sviluppo come l'insieme di sorgente e MakeFile per generare e trasferire nel filesystem gli eseguibili. Il comando di seguito riportato ci permette di generarne una:

$ petalinux-create --t apps --template c --name <nome app> --enable

- con l'opzione -t specifichiamo che stiamo creando applicazioni utente,
- con --template specifichiamo il linguaggio (c, c++),
- con --name definiamo il nome dell'applicazione,
- con --enable abilitiamo l'applicazione, ovvero informiamo petalinux che vogliamo includere questa applicazione nella compilazione.

Il comando crea una nuova directory <nome app> nel seguente path:

<root di progetto>/components/apps/

all'interno viene generato un sorgente c denominato<nome app>.c e il Makefile per la compilazione del medesimo. Modificando il file <nome app>.c si può scrivere la propria applicazione.
Petalinux mette a disposizione uno script *targetroot-inst.sh* per trasferire dei file nel rootfs o modificare file e directory già esistenti in fase di compilazione.

Nel nostro sistema sono state implementate le seguenti applicazioni: *ntp*, *avrdude*, *timer* e *configuration*.

Ntp e *avrdude* sono applicazioni opensource non disponibili nella distribuzione Petalinux che è stato necessario configurare e cross-compilare per il nostro sistema embedded.

Timer è un'applicazione che è stata sviluppata ad hoc per gestire il timer hardware, mentre *configuration* è un'applicazione che serve a copiare nel filesystem gli eseguibili, gli script ed i file di configurazione necessari.

4.4 App: NTP

Per implementare la sincronizzazione temporale è stato necessario installare il pacchetto NTP, che non era disponibile tra i pacchetti standard di Petalinux.

L'**NTP**, è un protocollo per sincronizzare gli orologi dei computer all'interno di una rete a commutazione di pacchetto, quindi con tempi di latenza variabili ed inaffidabili. L'NTP è un protocollo client-server appartenente al livello applicativo che utilizza la porta UDP 123 [Network Time Protocol, 2016].

La soluzione preferibile è usare il demone NTP, *ntpd*, configurando uno o più server NTP di riferimento nel suo file di configurazione /etc/ntpd.conf. In questo modo, oltre ad aggiornare l'orologio di sistema, ntpd ne stima l'errore sistematico, ed è in grado di correggerlo, evitando un andamento irregolare del tempo, e migliorando la precisione quando l'host non è connesso alla rete [Network Time Protocol, 2016].

Nel nostro caso utilizziamo come server NTP dedicato un Server Time Lantime M300 della Meinberg con ricevitore GPS incorporato. Tale server verrà installato presso i laboratori di terra di Portopalo di Capopassero.

Per sincronizzare manualmente l'orologio di sistema con quello di un server, da console, con privilegi di root, è necessario usare il comando: ntpdate<ntp server >

Come primo passo è stata creata l'app all'interno del progetto, con la procedura descritta nel paragrafo 4.3. Erano possibili due soluzioni implementative: configurare il *Makefile* per compilare i sorgenti dell'ntp all'interno della procedura di compilazione di Petalinux, oppure cross-compilare e copiare gli eseguibili nel rootfs. Si è optato per questa seconda soluzione.

Il pacchetto *ntp-4.2.8p4,* contenente il sorgente del servizio, è stato recuperato dal sito http://www.ntp.org/. Gli eseguibili sono stati generati per cross-compilazione con i seguenti comandi:

$ ./configure --prefix=< directory di destinazione> --host=arm-xilinx-linux-gnueabi CC=arm-xilinx-linux-gnueabi-gcc LDFLAGS="-W1,--gc-sections" CFLAGS="-ffunction-sections –fdata-sections -Os" --with-yielding-select=yes

$ make

$ make install
--prefix specifica la directory dove verranno inseriti i file generati dalla compilazione,

--host specifica l'architettura di destinazione (nel nostro caso arm-xilinx-linux-gnueabi).

CC definisce il compilatore. Nel nostro caso è stata usata una variante del gcc messa a disposizione da Petalinux, *arm-xilinx-linux-gnueabi-gcc*

LDFLAGS="-Wl,--gc-sections"
CFLAGS="-ffunction-sections -fdata-sections -Os"

definiscono le opzioni di compilazione e linking.

Il file di configurazione dell'NTP, *ntp.conf* , è stato creato con i seguenti parametri:

```
driftfile /media/ntp.drift
logfile /var/log/ntp.log
pidfile /var/run/ntpd.pid

statistics loop stats peerstats clockstats
filegenloopstats file loopstats type day enable
filegenpeerstats file peerstats type day enable
filegenclockstats file clockstats type day enable

server 192.168.0.5

restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery

restrict 127.0.0.1
restrict ::1

minpoll 30
maxpoll 60
```

minpoll e *maxpoll* definiscono il range espresso in secondi entro cui effettuare l'interrogazione al Server NTP per sincronizzare l'orologio.

driftfile /var/log/ntp.drift

Abilitiamo il salvataggio del drift su file. Il file drift è usato per memorizzare l'offset della frequenza tra il clock di sistema in funzione alla sua frequenza nominale e la frequenza necessaria per rimanere sincronizzato con UTC. L'uso del drift file riduce il tempo richiesto per raggiungere un tempo stabile e accurato. Il valore è calcolato e memorizzato nel drift file una volta per ora. Come vedremo, tale valore sarà utilizzato per correggere il valore della frequenza del clock.

Per consentire l'esecuzione del demone NTP all'avvio del sistema, è stato creato lo script bash*S91ntpd*. Gli eseguibili e il file di configurazione sono stati trasferiti in:

<root di progetto>/components/app/ntp

Il *MakeFile* è stato modificato come segue:

```
(...)
install:
        $(TARGETINST)ntp.conf /etc
        $(TARGETINST) ./bin/calc_tickadj /bin
        $(TARGETINST) ./bin/ntpd /bin
        $(TARGETINST) ./bin/ntpdate /bin
        $(TARGETINST) ./bin/ntpdc /bin
        $(TARGETINST) ./bin/ntp-keygen /bin
        $(TARGETINST) ./bin/ntpq /bin
        $(TARGETINST) ./bin/ntptime /bin
        $(TARGETINST) ./bin/ntptrace /bin
        $(TARGETINST) ./bin/ntp-wait /bin
        $(TARGETINST) ./bin/sntp /bin
        $(TARGETINST) ./bin/tickadj /bin
        $(TARGETINST) ./bin/update-leap /bin
```

al fine di consentire in fase di compilazione generale di trasferire i vari file su menzionati negli opportuni path del rootfs.

4.5 App: Avrdude

Come accennato nel capitolo precedente, il firmware di Arduino è scaricabile da remoto per permettere aggiornamenti. A questo scopo si è utilizzato il software avrdude, per il quale è stata generata una app.

Avrdude è un programma di utilità per scaricare / caricare / manipolare il contenuto delle ROM e le EEPROM dei microcontrollori AVR utilizzando la tecnica di in-system programming (ISP). La versione usata è la 6.2 [Avrdude, 2016].

L'eseguibile e il file di configurazione, *avrdude.conf*, sono stati generati per cross compilazione. Il *MakeFile* è stato modificato in maniera analoga a quanto fatto per l'NTP.

L'eseguibile generato presentava un malfunzionamento che portava al crash del programma in fase di trasferimento del firmware del nostro modello di Arduino. Il blocco di codice mal funzionante causava una divisione per zero, corretta tramite l'applicazione di una patch al sorgente.

Esempio di trasferimento con avrdude:

$ avrdude -v -p m2560 -c wiring -P /dev/ttyACM0 -b 115200 -D -F -U flash:w:/media/Blink.cpp.hex -C /etc/avedude.conf

4.6 App: Configuration

È un'applicazione fittizia il cui compito è solo quello di trasferire nel rootfs eseguibili, script e modificare file di configurazione già esistenti.

Tutti i trasferimenti di file, script e modifiche al filesystem rootfs vengono implementate in questa app.

Makefile per l'appConfiguration:

```
(...)

STR = "/dev/mmcblk0p1      /media        auto      defaults,sync  0  0"

(...)

install:
        $(TARGETINST) -a $(STR) /etc/fstab
        $(TARGETINST) -d ntpd /etc/init.d/
        $(TARGETINST) -d S91ntpd /etc/rc5.d/
        $(TARGETINST) -d S92loadmodule /etc/rcS.d/
```

Il *Makefile* effettua le seguenti modifiche al filesystem:

- aggiunge una riga al file /etc/fstab per montare in automatico all'avvio del sistema la SD Card;
- Copia in */etc/init.d/* lo script per gestire il servizio NTP (start, stop, restart);
- Copia in */etc/rc5.d/* lo script S91ntpd per attivare il demone NTP all'avvio del sistema;
- Copia in */etc/rcS.d*lo script S92loadmodule per caricare il modulo kernel timermodule, nonché per creare il dispositivo /dev/timer.

4.7 App: Timer

Applicazione sviluppata in questo lavoro. Conterrà l'implementazione dell'algoritmo per la sincronizzazione. Il servizio timer dovrà eseguire le operazioni necessarie a sincronizzare il Timer PPS con il tempo di sistema, che a sua volta è sincronizzato mediante protocollo NTP. Tali operazioni sono:

- leggere il delay dal timermodule;
- calcolare i fattori correttivi;
- settare opportunamente il contatore del timer.

4.8 Modulo: Timermodule

Oltre alle applicazioni viste in precedenza, Petalinux ci permette di creare moduli del kernel. Il comando è molto simile a quello usato per le app:

$ petalinux-create -type modules -name timermodule --enable

la directory *timermodule* viene creata in

<root di progetto>/components/modules/

Essa contiene il file timermodule.c , un modulo del kernel modificato per implementare il driver per il Timer PPS realizzato nell'ambito del progetto [Giovanetti et al, 2017].

Lo script *S92loadmodule* è stato implementato per:

- caricare il modulo all'avvio (modprobe timermodule&)
- creare nel filesystem il device */dev/timer* (mknod / dev/timer c 36 0 &)

Codice S92loadmodule:

```
1  #!/bin/sh
2
3  echo"Loading Timer module:"
4  modprobetimermodule&
5  echo"Creating timer device:"
6  mknod /dev/timer c 36 0 &
```

4.9 Petalinux: Creazione e caricamento delle immagini sulla microzed

Creazione immagine contenente il kernel e rootfs

A questo punto abbiamo configurato il kernel e il rootfs e possiamo procedere alla compilazione dell'immagine che la Microzed userà per avviare il sistema operativo embedded.

Petalinux ci mette a disposizione il comando petalinux-build:

petalinux-build compila kernel, applicazioni e moduli realizzati in precedenza e genera il file *image.ub*.

È anche possibile compilare soltanto una singola applicazione o modulo:

petalinux-build -c rooft/<nome app o nome modulo>

Creazione immagine di boot
petalinux-package --force --boot --fsbl "<SDK_Export_directory>/zynq_fsbl/Debug/zynq_fsbl.elf" --fpga "<SDK_Export_directory>/<hw_platform_name>/<bitstream_file>.bit" --u-boot

--**fsbl** carica il bitstream programma fpga al momento dell'avvio,
--**fpga** indica dove trovare il file bitstream,
--**u-boot** crea l'u-boot automaticamente.

Il comando genera il file BOOT.BIN che verrà caricato sulla SD Card, mentre l'image.ub verrà scaricata dall'u-boot recuperandola dal server TFTP, come si vede nella successiva Figura 8.

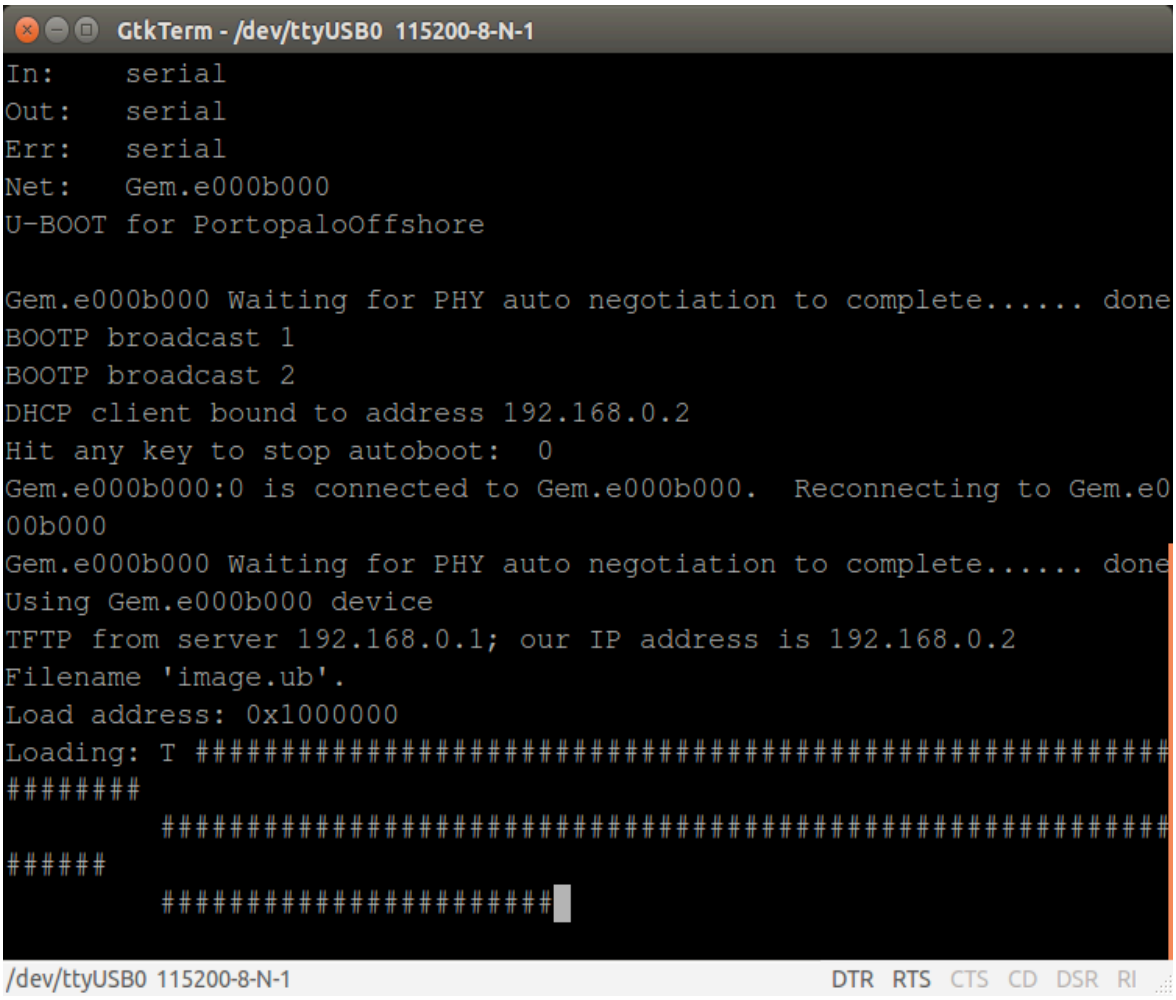

Figura 8. Microzed in avvio: Recupero immagine dalla rete.

Conclusioni e sviluppi futuri

In conclusione è stato implementato il sistema operativo Linux Embedded conformemente all'hardware realizzato in house [Giovanetti et al., 2017].

La gestione remota del sistema è stata realizzata attraverso l'installazione del pacchetto Dropbear SSH, che implementa un server SSH efficiente pensato per sistemi embedded.

La possibilità di telecaricamento di un nuovo firmware per arduino è stata realizzata tramite l'installazione del tool avrdude, il cui codice eseguibile è stato ottenuto per cross-compilazione.

Il caricamento da remoto del firmware del sistema, che contiene l'immagine del sistema operativo configurato come descritto nei paragrafi precedenti, è stato implementato attivando l'opzione per il caricamento da rete dell'immagine.

La sincronizzazione temporale del sistema operativo è stata implementata attraverso l'installazione e configurazione del servizio NTP, i cui file eseguibili sono stati ottenuti per cross-compilazione.

Un timer hardware realizzato in house [Giovanetti et al., 2017], chiamato Timer PPS, si occupa di trasmettere data e ora agli idrofoni ad alta frequenza tramite un PPS modulato. Lo scatto del secondo attivato dagli idrofoni, a seguito di un segnale PPS generato dal Timer PPS, avviene in maniera asincrona rispetto allo scatto del secondo del sistema operativo.

Nell'ambiente descritto in questo articolo si dovrà sviluppare un driver per la sincronizzazione dello scatto del secondo del sistema operativo e lo scatto del secondo del Timer PPS. In questo modo i dati degli altri sensori e degli idrofoni ad alta frequenza saranno temporalmente sincronizzati.

Lo step successivo sarà poi lo sviluppo del software di acquisizione, che girando all'interno del sistema embedded, beneficerà di una minore complessità realizzativa, poiché non dovrà ricostruire (calcolando le latenze introdotte da step intermedi di trasmissione) la data e ora corretta per marcare i singoli campioni letti.

Bibliografia

Giovanetti G., Apponi, U. & Marcucci, N.M., (2017). *Il sistema elettronico per l'alimentazione, il controllo e l'acquisizione dati del nuovo Osservatorio multidisciplinare di Portopalo di Capopassero*. Rapporti Tecnici INGV, 373.

Sitografia

[Avrdude, 2016]. http://www.nongnu.org/avrdude/
[Board_Support_Package, 2016]. https://it.wikipedia.org/wiki/Board_Support_Package
[Boot loader, 2016]. https://it.wikipedia.org/wiki/Boot_loader
[Device Tree, 2016]. https://en.wikipedia.org/wiki/Device_tree
[First Stage Bootloader, 2016].
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/sdsoc_doc/topics/introduction/concept_sdsocpl_first_state_bootloader.html
[Linux_(kernel), 2016]. https://it.wikipedia.org/wiki/Linux_(kernel)
[Network Time Protocol. 2016]. https://it.wikipedia.org/wiki/Network_Time_Protocol
[PetaLinux 2014.4 Image with Custom Application, 2015].http://www.syfer.com.au/assets/s502-00001-a.pdf
[U-boot, 2016]. http://www.wiki.xilinx.com/U-boot
[Un nuovo ambiente di Progettazione, 2016]. http://www.elettronicanews.it/un-nuovo-ambiente-di-progettazione/
[What-is-PetaLinux, 2016]. https://www.quora.com/What-is-PetaLinux

ISSN 1590-2595

Quaderni di Geofisica

http://istituto.ingv.it/l-ingv/produzione-scientifica/quaderni-di-geofisica/

I Quaderni di Geofisica coprono tutti i campi disciplinari sviluppati all'interno dell'INGV, dando particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari, che per tipologia e dettaglio necessitano di una rapida diffusione nella comunità scientifica nazionale ed internazionale. La pubblicazione on-line fornisce accesso immediato a tutti i possibili utenti. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-7941

rapporti tecnici INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/rapporti-tecnici-ingv/

I Rapporti Tecnici INGV pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico e di rilevante interesse tecnico-scientifico per gli ambiti disciplinari propri dell'INGV. La collana Rapporti Tecnici INGV pubblica esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-6651

Miscellanea INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/miscellanea-ingv/

La collana Miscellanea INGV nasce con l'intento di favorire la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV (sismologia, vulcanologia, geologia, geomagnetismo, geochimica, aeronomia e innovazione tecnologica). In particolare, la collana Miscellanea INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli ecc..

Coordinamento editoriale e impaginazione
Centro Editoriale Nazionale | INGV

Progetto grafico e redazionale
Daniela Riposati | Laboratorio Grafica e Immagini | INGV

Via di Vigna Murata, 605
00143 Roma
Tel. +39 06518601 Fax +39 065041181

http://www.ingv.it

Istituto Nazionale di Geofisica e Vulcanologia