0.1 Soluzione

0.1.1 Prerequisiti

Una distribuzione linux, dove sono presenti i seguenti pacchetti:

1. build-essential
2. u-boot-tools
3. gparted
4. git
5. libncurses-dev
6. libssl-dev
7. bison
8. flex
9. device-tree-compiler

Se non fossero presenti utilizzare il seguente comando.

```
sudo apt install build-essential u-boot-tools gparted git libncurses-dev libssl-dev bison
    flex device-tree-compiler
```

Clonare le seguenti repository:

1. https://github.com/Xilinx/linux-xlnx
2. https://github.com/Xilinx/u-boot-xlnx
3. https://github.com/Xilinx/device-tree-xlnx

Scaricare un filesystem, nel caso specifico è stato scelto Linaro:

```
https://releases.linaro.org/debian/images/developer-armhf/latest/
```

Prima di eseguire qualsiasi operazione, come ad esempio la cross-compilazione del kernel, è necessario configurare il terminale da cui si lavorerà. La configurazione consisten nell'esportare alcune variabili d'ambiente, lanciando lo script prepare_environment.sh riportato in seguito:

```
#!/bin/bash
VIVADO_PATH=$HOME/Vivado
VIVADO_VERS=2018.3
SDK_PATH=$HOME/SDK
SDK_VERS=2018.3
source $VIVADO_PATH/$VIVADO_VERS/settings64.sh
source $SDK_PATH/$SDK_VERS/settings64.sh
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
export PATH=$HOME/Scrivania/UART_KERNEL_MODE/linux-xlnx/tools/:$PATH
```

0.1.2 Compilazione u-boot

Posizionarsi all' interno della cartella u-boot-xlnx. Prima di procedere alla compilazione bisogna modificare parte del file zynq-common.h presente in:

```
include/configs/zynq-common.h
```

La porzione di codice da modificare è la seguente

```
"sdboot=if mmcinfo; then " \
"run uenvboot;" \
"echo Copying Linux from SD to RAM... && " \
"load mmc 0 ${kernel_load_address} ${kernel_image} && " \
"load mmc 0 ${devicetree_load_address} ${devicetree_image} && " \
"load mmc 0 ${ramdisk_load_address} ${ramdisk_image} && " \
"bootm ${kernel_load_address} ${ramdisk_load_address} ${devicetree_load_address}; "
```

e va modificata come segue

```
"sdboot=if mmcinfo; then " \
"run uenvboot;" \
"echo Copying Linux from SD to RAM... && " \
"load mmc 0 ${kernel_load_address} ${kernel_image} && " \
"load mmc 0 ${devicetree_load_address} ${devicetree_image} && " \
"bootm ${kernel_load_address} - ${devicetree_load_address}; "
```

Una volta fatto ciò è possibile procedere alla compilazione. Dal terminale in cui era stato lanciato lo script prepare_environment, eseguendo i seguenti comandi,

```
make CROSS-COMPILE=arm-linux-gnueabihf- zynq_zybo_config
make -j 2 CROSS-COMPILE=arm-linux-gnueabihf- u-boot.elf
```

verrà generato il file u-boot.elf

0.1.3 Creazione immagine di boot

La successiva operazione da effettuare la creazione del First Stage Boot Loader, responsabile del caricamento del bitstream della configurazione del Processing System (PS) della Zynq all'avvio. Avviare SDK, dal menù "File ->Launch SDK" all'interno del progetto Vivado. Una volta avviato è necessario creare un nuovo progetto da "File -> New -> Application Project". Dopo aver scelto un nome al progetto, cliccando su Next apparirà la seguente schermata

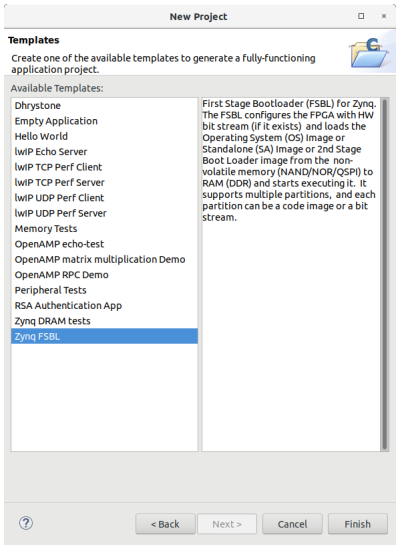

Figure 1: Creazione First Stage Boot Loader

Selezionare il template “Zynq FSBL” come in figura e cliccare su Finish.
Dal pannello “Project Explorer” selezionare il progetto appena creato e dal menù “Xilinx” selezionare “Create Boot Image”. Si aprirà la seguente finestra:

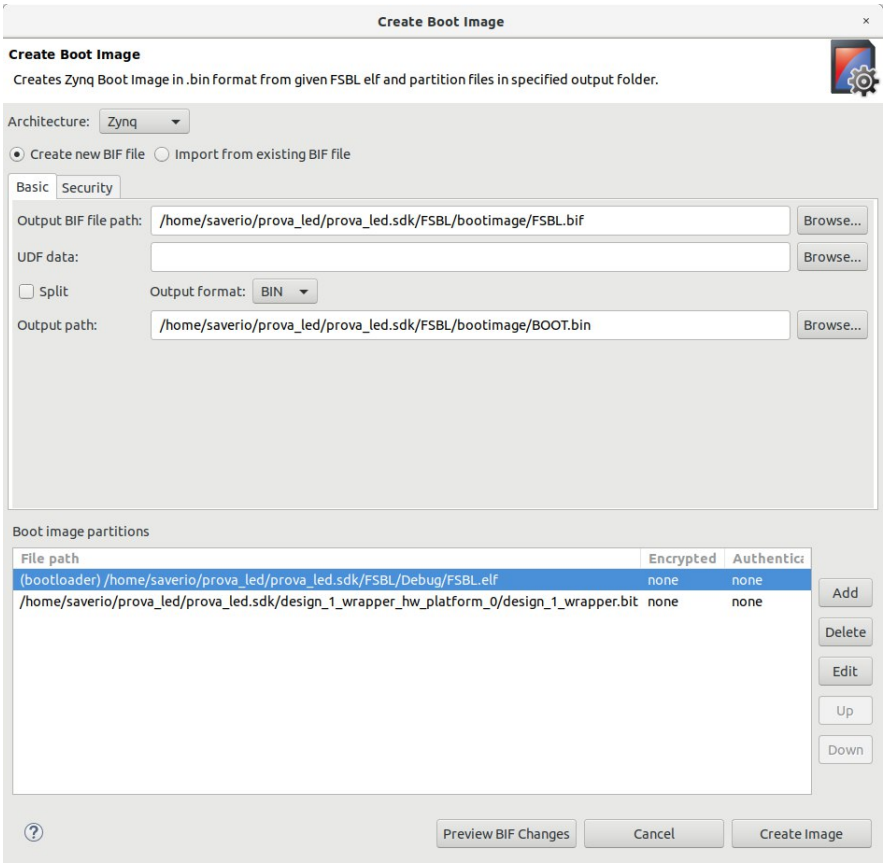

Figure 2: Creazione immagine di Boot

Cliccare “Add -> Browse...” e selezionare il file u-boot.elf generato dalla compilazione di u-boot.elf. Se l’aggiunta è andata a buon fine si potrà visualizzare il file u-boot.elf nella lista Boot image partitions.

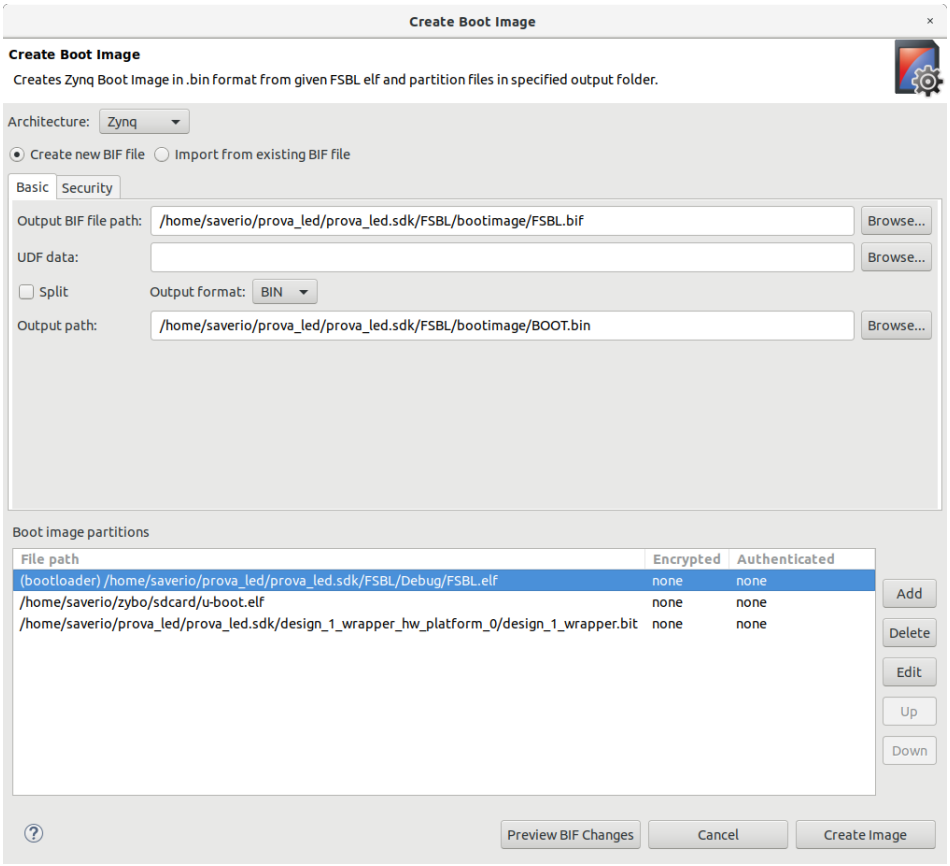

Figure 3: Creazione immagine di Boot

L'ultima operazione da effettuare è quella di cliccare su "Create Image". Se il processo è andato a buon fine verrà creato, nel path identificato dal percorso "Output path", il file BOOT.bin.

0.1.4 Compilazione del Device Tree

La successiva operazione è quella di creare un Device Tree Blob, il quale ha il compito di descrivere l'architettura hardware al sistema operativo. Prima di procedere alla creazione del dtb è necessario aggiungere le repositories DTS di Xilinx a quelle disponibili in SDK. Dal menù "Xilinx" selezionare "Repositories", cliccare su "New..." e selezionare la cartella device-tree-xlnx precedentemente clonata. Se l'aggiunta è andata a buon fine si visualizzerà la voce device-tree-xlnx tra le repositories locali.

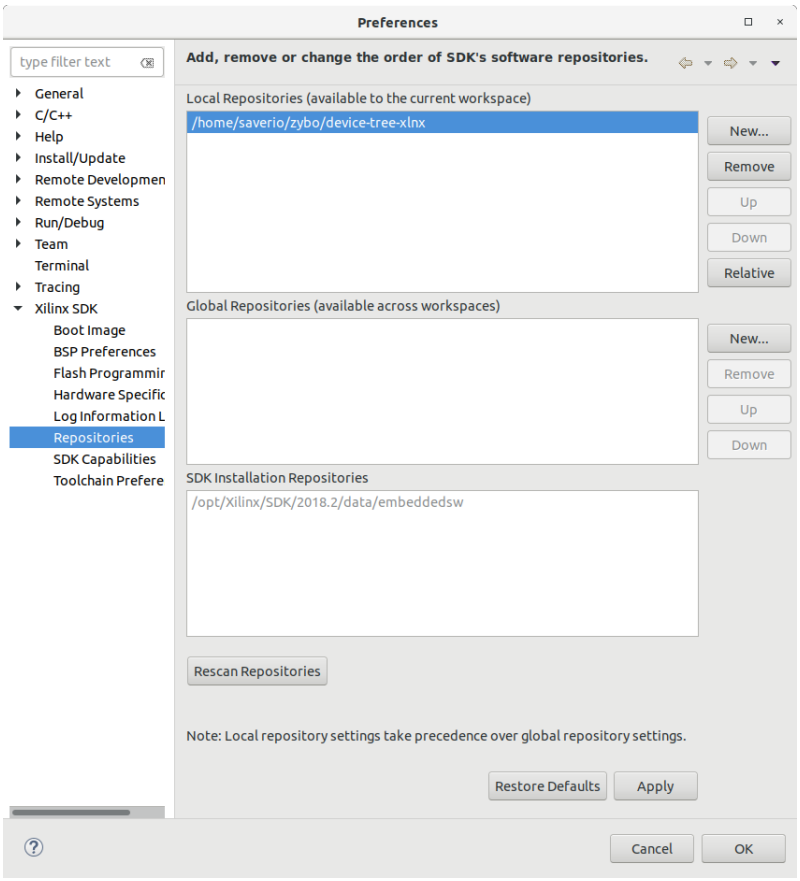

Figure 4: Aggiunta repositories device-tree-xlnx

A questo punto è possibile procedere con la creazione del device-tree. Creare un "Board Support Package" dal menù "File -> New"

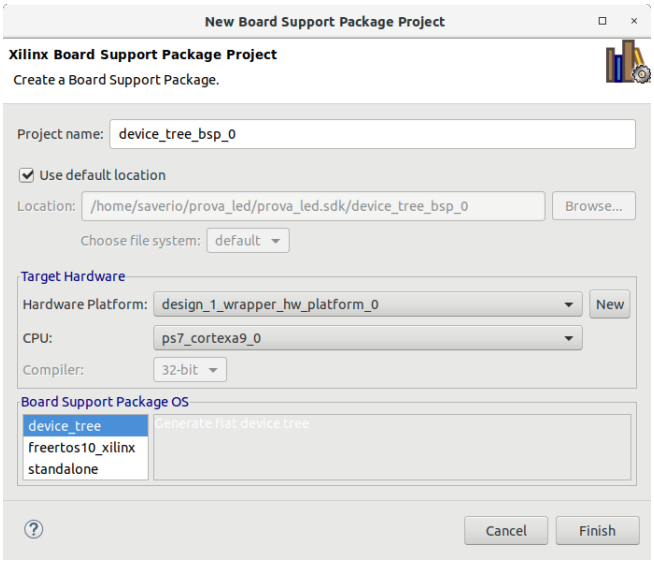

Figure 5: Creazione Board Support Package

Selezionare device_tree come mostrato in figura e cliccare su "Finish". Il progetto appena creato sarà composto dai seguenti file:

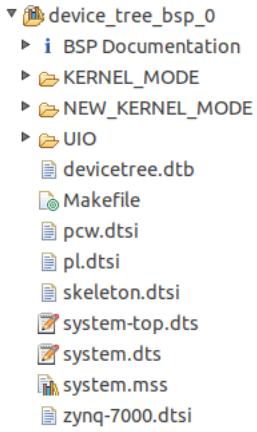

Figure 6: Progetto Board Support Package

Nel file system-top.dt è necessario modificare i parametri di boot per il kernel come segue:

```
bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4
    rootwait devtmpfs.mount=1 earlycon";
```

L'ultima operazione da effettuare è compilare il device-tree-source ottenendo il device-tree-blob. Aprire un terminale nella cartella del progetto relativo al Board Support Package e lanciare il seguente comando

```
dtc -I dts -O dtb -o devicetree.dtb system-top.dts
```

0.1.5 Creazione della uImage

A questo punto è possibile procedere con la compilazione del Kernel scaricato. Recarsi nella cartella linux-xlnx e lanciare i seguenti comandi

```
make xilinx_zynq_defconfig
make menuconfig
```

Il secondo comando aprirà il menù di configurazione seguente:

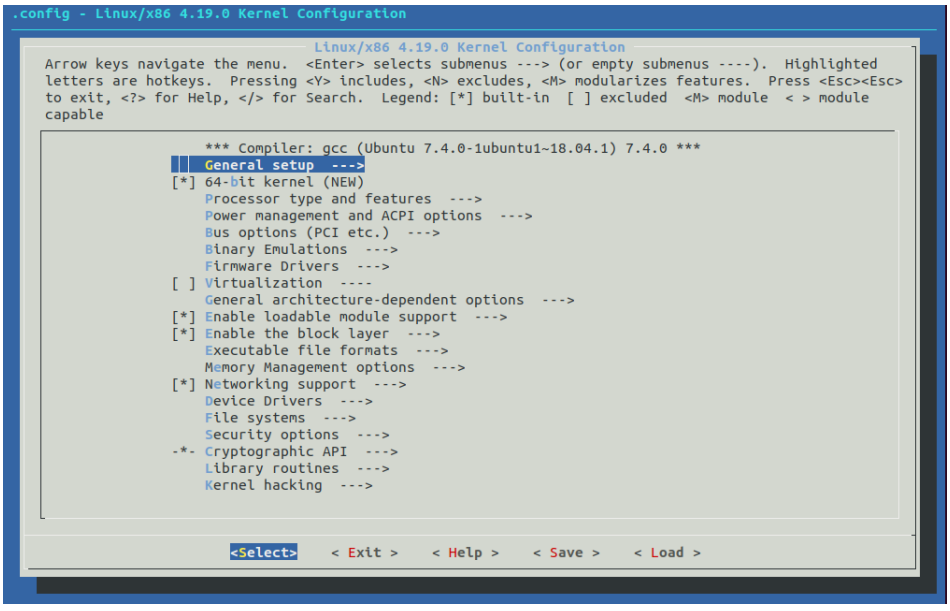

Figure 7: Menu di Configurazione del Kernel

Da questa schermata è possibile modificare la configurazione del kernel. Per abilitare il supporto ai driver Userspace I/O che verranno trattati nel seguenti capitolo bisogna recarsi in General Setup e verificare che la spunta Userspace I/O sia asserita. Per la definitiva compilazione lanciare i seguenti comandi:

```
make -j 4
make -j 4 UIMAGE_LOADADDR=0x8000 uImage
```

Verrà creato un file uImage all'interno della directory "arch/arm/boot/"

0.1.6 Preparazione SD card

Formattare una SD card con tre partizioni divise nel seguente modo:

1. 4MB spazio non allocato (root)
2. 1GB FAT32 (BOOT)
3. lo spazio rimanente EXT4

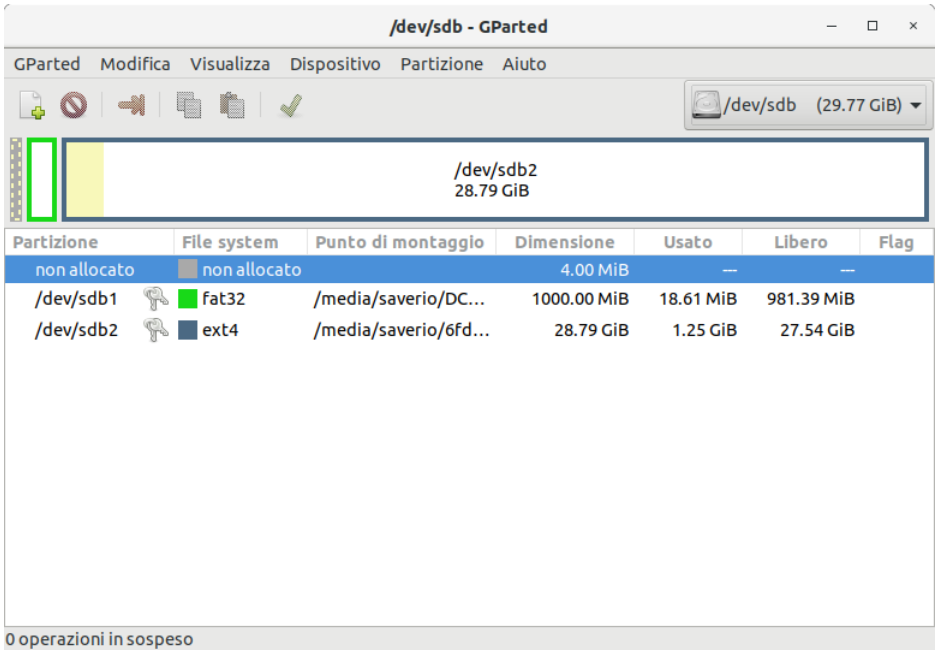

Figure 8: Parzionamento SD Card con GParted

Copiare infine sulla partizione di BOOT FAT32 i file Boot.bin, devicetree.dtb e la uImage. Nella partizione EXT4 va invece instanziato il filesystem di root. Per effettuare questa operazione è necessario scompattare l'archivio .tar, scaricato in precedenza dal sito di Linaro, ed impartire il seguente comando:

```
rsync -azv "path_cartella_scompattata_linaro"/binary/ "path mount point nella partizione
    EXT4"
```