DE0 Nano SoC – Compilare U-Boot, Baremetal e Kernel

Data prima stesura	**31/08/2016**
Data revisione	**29/09/2016**
Revisione	**01**
Autore	**Calzoni Pietro**
Riferimenti per Download	**www.lugman.org o su richiesta**

Obiettivi del documento:
1. Riassumere quanto descritto su https://rocketboards.org in [11], [12] e [13]
2. Creare un software baremetal con DS5 Community Edition e riuscire a caricarlo
3. Compilare e sostituire un nuovo kernel, comprensivo di Device Tree

Software Utilizzato:
- ALTERA Quartus Prime (versione 16.0)
- ARM DS5 Community Edition

Scheda Target: DE0 Non SoC

INDICE

ACRONIMI, DEFINIZIONI E ABBREVIAZIONI

Sistema HOST	è il sistema sul quale sono installati i tool di sviluppo come i cross-compilatori: di fatto Linux Mint KDE 17.2
Sistema Target	sistema di destinazione, tipicamente Linux su ARM o l'FPGA
SoC	System On a Chip: sono chip che realizzano un sistema completo comprensivo di un certo numero di periferiche già implementate sul DIE.
HPS	*Hard Processor System*: in Cyclone V di fatto è l'ARM
AMBA	*ARM Advanced Microcontroller Bus Architectur*: attualmente alla versione 4, è un bus standard opensource di ARM utilizzato principalemnte su SoC.
AXI	*Advanced eXtensible Interface*: parte di AMBA, permette a sistemi multi-core di condividere memoria oltre ad altre tecnologie ARM e di effettuare transazioni ad alta velocità..
NIC-301	ARM CoreLink Network Interconnect: Sistema di interconnessione tra HPS e le periferiche, basato su bus AMBA, AXI, AHB e APB bus.
IP	Intellectual Property: componenti che è possibile installare, non necessariamente a pagamento, per estendere particolari funzionalità (es: sono IP le MegaFunction ALTERA o il soft core NIOS II)
EDS	SoCEmbedded Design Suite: di fatto DS-5 Altera Community Edition
ABI	*Application Binary Interface*: è l'interfaccia che garantisce che un programma sia compilato seguendo le regole del sistema che poi lo eseguirà; per esempio indica come viene chiamata una funzione, quanti parametri ha, come vengono allineati i dati, ecc.
EABI	*Embedded ABI*: tipicamente usate per ARM o per processori embedded.
Baremetal	programma più o meno complesso che prevede la configurazione del microprocessore ed esegue delle funzioni indipendentemente dal sistema operativo, che speso non è implementato se non in modo minimale.

1. DOWNLOAD SD CARD

Dal link [11] è possibile ottenere la SD-CARD per DE0.
In particolare alla pagina seguente:
https://rocketboards.org/foswiki/view/Documentation/SoCSWWorkshopSeriesSDCardImages
In particolare l'immagine è attualmente per Quartus II 15.1.

Quartus 15.1.2 SD Card Images

Development Kit	SD Card Image
Atlas and DE0 Nano	sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image.gz
Altera Cyclone V SoC Dev Kit	sd_card.ALTERA_CV_SOC.2016-04-26---23-27-00.image.gz

Per scriverla su chiave USB o su SD card occorre decomprimere il file e scrivere l'immagine con *dd* come è tipico in Linux.
È possibile però esplorare la SD senza doverla scrivere su chiavetta. Queste operazioni saranno comode quando si avranno altre immagini contenenti magari solo dati da esaminare:

- Decomprimere il file con gunzip *sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image.gz*: attenzione perché il file .gz verrà cancellato e rimarrà soltanto il file .image
- *fdisk –l sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image*: si notano le 3 partizioni di linux e di boot più una sconoscuta dove è installato il preloader.

```
Disco sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image: 536 MB, 536870912 byte
255 testine, 63 settori/tracce, 65 cilindri, totale 1048576 settori
Unità = settori di 1 * 512 = 512 byte
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificativo disco: 0xcb462893

                                    Dispositivo Boot      Start         End      Blocks   Id  System
sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image1            4096      528383      262144    b  W95 FAT32
sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image2          528384     1048575      260096   83  Linux
sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image3            2048        4095        1024   a2  Sconosciuto

Le voci nella tabella delle partizioni non sono nello stesso ordine del disco
```

- Essendo i settori da 512 byte, per conoscere l'offset della partizione (in termini di dimensioni) basta calcolare il punto di *Start* x 512. Esempio: .image2 comincerà all'offset 528384 x 512 = 270532608 (byte).
- Per montare la partizione su un dispositivo di loopback si può usare mount, oppure losetup. Si consiglia mount per semplificare l'unmount una volta terminato l'utilizzo. Il comando è il seguente:
  *mount -o loop,offset=$((Start * 512)) <nome_file> <punto_di_montaggio>*
  in questo caso occorre montare la partizione FAT dove ci sono tutti i sorgenti e binari dei workshop, quindi:
  *sudo mount -o loop,offset=$((4096*512)) sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image sd_card*
  ipotizzando *sd_card* come directory

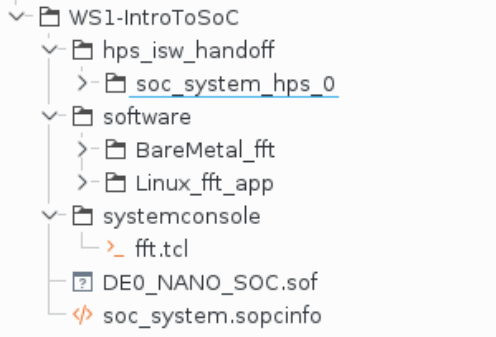
Maggiori informazioni sul montaggio delle partizioni da file immagine: http://madduck.net/blog/2006.10.20:loop-mounting-partitions-from-a-disk-image/.
L'SD Card scaricata prevede due partizioni da circa 512Mb ciascuna (una FAT e una EXT2) più una partizione piccola marcata come sconosciuta (0xA2) dove verrà installato il preloader. Questa SD può essere usata per fare i workshop, ma in questo caso si è proceduro utilizzando una SD formattata con l'immagine fornita da Terasic che quindi ha già un sistema Linux funzionante (kernel 3.13).

2. PRELOADER E BAREMETAL – WORKSHOP 1

Nel primo WR viene descritto come creare il preloader da inserire copiare nella partizioni nascosta (marcata 0xA2) e compilare UBOOT. Per farlo occorre aver grato una configurazione del SoC come descritto in [21] oppure utilizzare quanto descritto nel WR1.

Per semplicità seguirò quella versione, quindi:

- *sudo mount -o loop,offset=$((4096*512)),uid=<utente> sd_card.DE0_NANO_SOC.2016-04-26---23-27-00.image sd_card* dove uid è l'utente a cui si vuole dare I permessi di scrittura dull'immagine, altrimenti potrebbe scrivere solo root.
- Copiare i file dalla directory DE0_NANO_SOC alla directory del Workshop, ottenendo questa struttura:



Se il progetto fosse differente, occorrerebbe puntare alla directory del progetto contenente il file *.sopcinfo*.

ATTENZIONE: la directory WS1-IntroToSoC deve essere su un file system Linux perché al momento della compilazione il sistema dovrà eseguire funzioni di tipo UNIX come la generazione di link simbolici.

2.1. PRELOADER

È tendenzialmente l'equivalente del FSBL (First Stage Bootloader) della XILINX, ovvero la prima fase di configurazione della CPU (RAM, Flash, clock, ecc).
Queste informazioni sono generate automaticamente dal progetto sopcinfo in funzione di come è stato configurata la CPU.
Occorre quindi configurare il sistema, in particolare in termini di variabili di ambiente. Ipotizzando che la direcotory di installazione dei tool altera, in particolare del DS5, sia /opt/altera, configurare il sistema con:

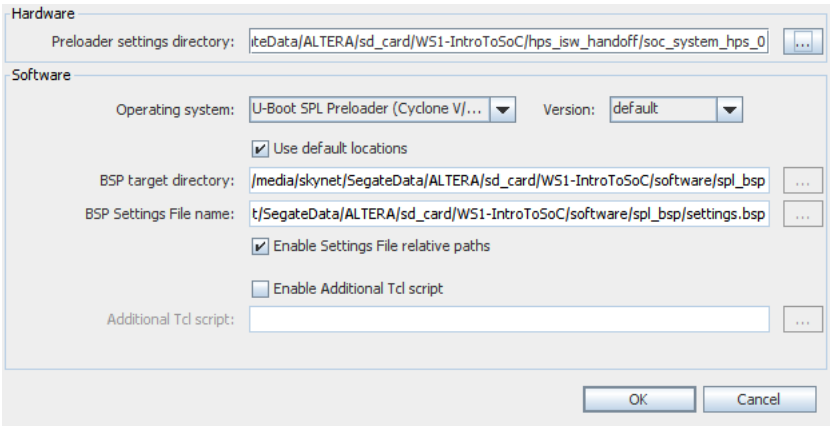
```
$ /opt/altera/16.0/embedded/embedded_command_shell.sh
bash: /bin/env.sh: File o directory non esistente
-----------------------------------------------
Altera Embedded Command Shell

Version 16.0 [Build 211]
-----------------------------------------------
```

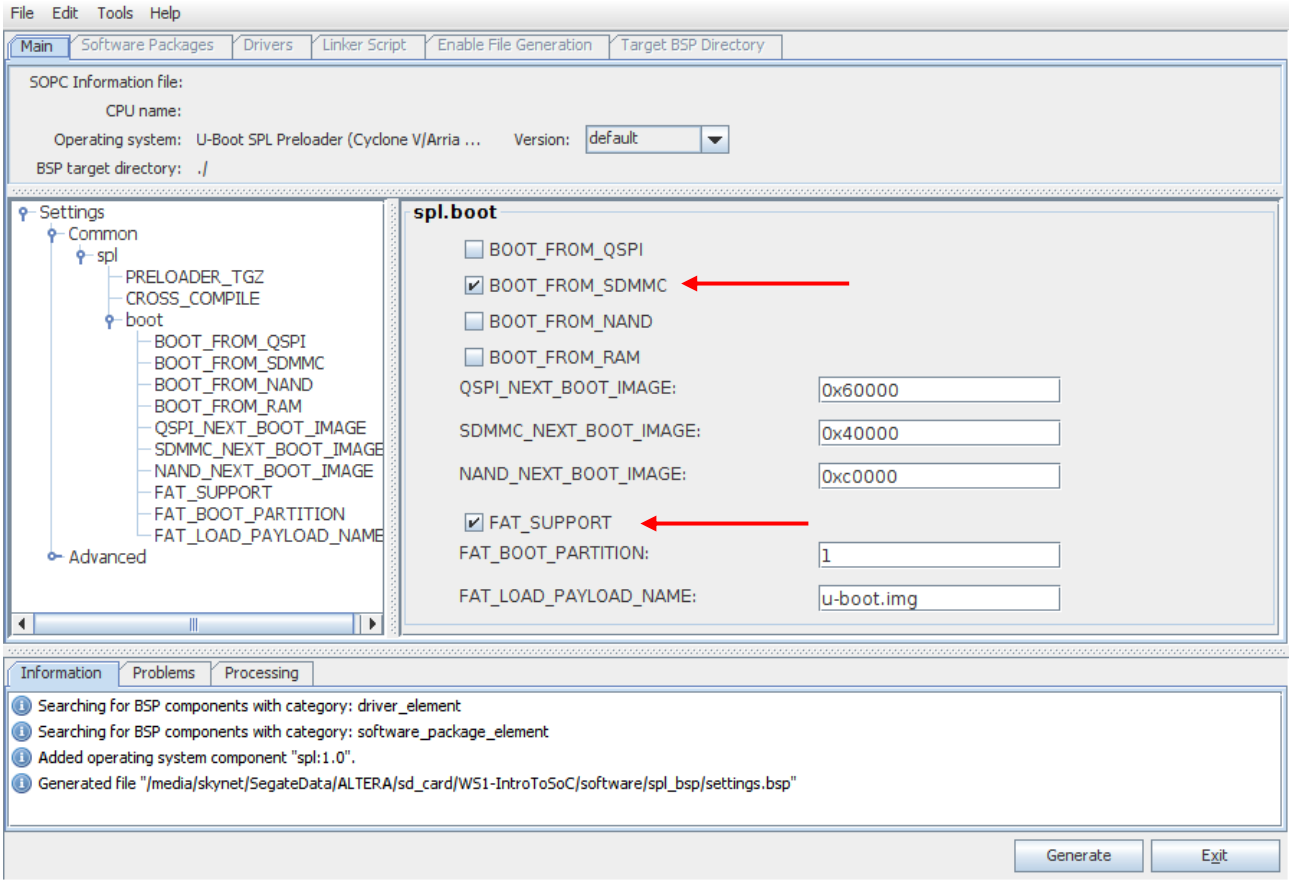
Occorre poi sapere se la RAM supporta l'ECC. Per la DE0_NANO_SOC la ram supportata è normale e quindi la macro RW_MGR_MEM_DATA_WIDTH sarà pari a 32. Cercare questa macro con *grep* in *sequencer_defines.h*.

DSP Editor

- Eseguire il bspeditor lanciando il comando *bsp-editor &* possibilmente nella directory di lavoro *WS1_IntroToSOC*
- Creare un nuovo progetto: *File → New HPS BSP*
- Caricare la directory *./WS1-IntroToSoC/hps_isw_handoff/soc_system_hps_0*:



- Dopo OK, compariranno tutte le informazioni. LA ram non è ECC e quindi non occorre modificare nulla relativamente alla RAM. Il boot invece può cambiare in funzione di come si prevede di eseguire il boot appunto. Di default è settato **BOOT_FROM_SDMMC**, ma si potrebbero settare anche _FROM_QSPI/_NAND/_RAM. Per ora lascio invariato. Il Preloader poi caricherà U-Boot dalla partizione FAT e quindi occorre flaggare anche **FAT_SUPPORT**.



- GENERATE: cliccando su *Generate* verranno generati tutti i file necessari al Preloader. Si noti che questo sistema configura anche il compilatore (inteso come cross compilatore) *arm-altera-eabi-* per la compilazione. Stando alla documentazione, i file generati sono nella sottodirectory ***software/spl_bsp/***:

The following items are generated in the "<path-to-lab-work>/WS1-IntroToSoC/software/spl_bsp/" folder

File	Description
generated	Folder containing source code that was generated based on the information from the handoff folder
settings.bsp	Preloader settings file, that contains the settings from the Preloader Generator
Makefile	Makefile used to build the Preloader
preloader.ds	ARM DS-5 AE that can be used to load the Preloader

Il BSP Editor può essere chiuso.

Compilare il Preloader

Nella directory ***software/spl_bsp/*** si trovano i file generati.

- Sopstarsi nella directory ***software/spl_bsp/***
- Lanciare *make*: il sistema procede alla decompressione di alcuni sorgenti e alla compilazione degli stessi, tra cui u-boot.

ATTENZIONE: la directory WS1-IntroToSoC deve essere su un file system Linux perché al momento della compilazione il sistema dovrà eseguire funzioni di tipo UNIX come la generazione di link simbolici.

A questo punto i file generati sono (da documentazione):

The following files are built in the "<path-to-lab-work>/WS1-IntroToSoC/software/spl_bsp/" folder:

File	Description
uboot-socfpga/spl/u-boot-spl	Preloader ELF file
uboot-socfpga/spl/u-boot-spl.bin	Preloader binary file
preloader-mkpimage.bin	Preloader image with the boot ROM required header

Verifica dei file:

```
$ file uboot-socfpga/spl/{u-boot-spl,u-boot-spl.bin} preloader-mkpimage.bin
uboot-socfpga/spl/u-boot-spl:       ELF 32-bit LSB   executable, ARM, EABI5 version 1 (SYSV),
statically linked, not stripped
uboot-socfpga/spl/u-boot-spl.bin: data
preloader-mkpimage.bin:           data
```

Install Preloader

L'installazione del preloader e successivamente di u-boot verrà fatta sulla SD card di terasic per vedere se cambia qualche cosa, e dovrebbe cambiare lo splash screen di u-boot principalmente.
Dalla SD di Terasic si collegandosi in seriale si ha:

```
U-Boot SPL 2013.01.01 (Dec 29 2014 - 15:29:15)
BOARD : Terasic DE0_Nano_SoC Version-A  Board
################################################################################
##################################=                                           =#
########=-=#########################=                                      --    =#
######-   =#########################=                           ####-    =######   =#
######-   ################- -=##=##=                           ####-   ########   =#
####=       =##-     -###       ##=   -#######    ######   ####-  #####-  -   =#
####        ##   ###   ##      -##=   -=---=###=  ####==#   ####-  ####        =#
######-   ###-  ##-  -###    =#####=   ========##=  ####-     ####   ####        =#
######-   ###      =#####    ######=  ####-=####=    -=###-  ####-  #####       =#
######-    -#=     -  =##    ######=  #### -###=  #=-=####  ####-   ########-  ##
#######    =##=        ##    ######=  -#########=  #######   ####-    =######   =#
##########=######=====#####==########=     -         ---      --         ----    =#
##################################=                                           =#
################################################################################
BOARD : Terasic  DE0_Nano_SoC Version-A Board
CLOCK: EOSC1 clock 25000 KHz
```

```
CLOCK: EOSC2 clock 25000 KHz
CLOCK: F2S_SDR_REF clock 0 KHz
CLOCK: F2S_PER_REF clock 0 KHz
CLOCK: MPU clock 925 MHz
CLOCK: DDR clock 400 MHz
CLOCK: UART clock 100000 KHz
CLOCK: MMC clock 50000 KHz
CLOCK: QSPI clock 3613 KHz
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
SDRAM: 1024 MiB
ALTERA DWMMC: 0


U-Boot 2013.01.01 (Dec 30 2014 - 12:07:34)

CPU   : Altera SOCFPGA Platform
BOARD : Terasic DE0_Nano_SoC Version-A  Board
################################################################################
####################################=                                         =#
########=-=#########################=                                    --   =#
######-   =#########################=                        ####-    =######   =#
######-   ###############- -=##=##=                          ####-   ########   =#
####=       =##-     -###       ##=   -#######    ######   ####-  #####-  -   =#
####        ##   ###   ##       -##=   -=---=###=  ######=#   ####-  ####        =#
######-   ###-  ##-  -###    =#####=   =====####=  ####-     ####   ####        =#
######-   ###      =#####    #####=  ####-=###=    -=###-  ####-  #####       =#
######-    -#=     -  =##    #####=  #### -###=  #=-=####  ####-   ########-  ##
#######    =##=        ##    #####=  -#########=  #######   ####-    =######   =#
#########==######===####==#######=     -         ---      --          ----   =#
####################################=                                        =#
################################################################################
BOARD : Terasic  DE0_Nano_SoC Version-A Board
I2C:   ready
DRAM:  1 GiB
MMC:   ALTERA DWMMC: 0
In:    serial
Out:   serial
Err:   serial
Skipped ethaddr assignment due to invalid EMAC address in EEPROM
Net:   mii0
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
SOCFPGA_CYCLONE5 #
```

L'installazione avviene con dd scrivendo il file ***preloader-mkpimage.bin***:

- Creare una partizione con fdisk di tipo 0xA2 se non è già presente
- Individuare la partizione 0xA2 sulla SDCard
- I file che servono sono nella directory di lavoro *WS1-IntroToSoC/software/spl_bsp/*: spostarsi in questo percorso
- Individuare la partizione con *fdisk –l /dev/sdX*
- Copiare il file ***preloader-mkpimage.bin*** nella partizione individuata /dev/sdXY con dd. Il ad oggi è:

```
512+0 record dentro
512+0 record fuori
262144 byte (262 kB) copiati, 0,0884865 s, 3,0 MB/s
```

Testando or ail risultato si ha un errore perchè il sistema non riesce a trovare l'immagine di uboot:

```
U-Boot SPL 2013.01.01 (Sep 01 2016 - 13:48:13)
BOARD : Altera SOCFPGA Cyclone V Board
CLOCK: EOSC1 clock 25000 KHz
CLOCK: EOSC2 clock 25000 KHz
CLOCK: F2S_SDR_REF clock 0 KHz
CLOCK: F2S_PER_REF clock 0 KHz
CLOCK: MPU clock 925 MHz
CLOCK: DDR clock 400 MHz
CLOCK: UART clock 100000 KHz
```

```
CLOCK: MMC clock 50000 KHz
CLOCK: QSPI clock 3613 KHz
RESET: COLD
INFO : Watchdog enabled
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
SDRAM: 1024 MiB
ALTERA DWMMC: 0
reading u-boot.img
spl: error reading image u-boot.img, err - -1
### ERROR ### Please RESET the board ###
```

Il messaggio viene continuamente visualizzato in loop. Questo perché nella partizione FAT della SD non c'è u-boot.img, ma u-boot.scr fornito da Terasic. Si noti che se si rinominasse il file di terasic in u-boot.img, il sistema trova l'immagine e cerca di caricarla, ma ancora non funziona:

```
[…]
ALTERA DWMMC: 0
reading u-boot.img
reading u-boot.img
```

Anche in questo caso il messaggio è in loop.
Ciò che accade è che il Preloader è stato configurato per caricare il successivo stage di boot all'indirizzo 0x40000, ma non funziona perché in un caso non trova il programma, e nell'altro il programma non viene validato.

2.2. BAREMETAL

DS5 Community Edition non permette la generazione di un baremetal e il download tramite JTAG (USBBlaster), ma permette solo la generazione di semplici progetti baremetal che andranno caricati poi tramite il preloader.
L'obiettivo ora è caricare un programma "Hello Word" baremetal come descritto in [22], ma molto più semplice in quanto il programma FFT non interessa particolarmente.

Esattamente come U-Boot, il baremetal necessita di tutta la configurazione della CPU iniziale come indicato nelle istruzioni per la generazione del Preloader contenute nel *.sopcinfo*.
Ipotizzando di essere nella directory WS1-IntroToSoC (Linux):

- Creare la directory *qsys_headers*
- Creare tutti gli header necessari all'interfacciamento: *sopc-create-header-files ../../soc_system.sopcinfo --output-dir qsys_headers*. Questo genererà principalmente i seguenti file:

Generated File	Function
soc_system.h	The memory map as viewed by FPGA masters, in this case the mSGDMA master.
hps_0.h	The memory map of FPGA slaves as viewed by the hard ARM processors.

- Creare un progetto Baremetal. Si può fare in 2 modi:
    1. Avviare DS5 (eclipse) dopo aver configurato l'ambiente e creare un progetto baremetal. Questo è comodo perché genererà in automatico il makefile
    2. Creare manualmente il progetto andando a compilare manualmente il file .c ricordandosi di collegare l'apposito linker script e percorsi di include. I comandi si possono copiare sia da DS5 che dagli esempi FFT dei workshop Rocketboards.

  Il punto due, inizialmente dovrebbe essere più semplice, ma in futuro è bene usare il DS5 se il progetto si complica.
- Utilizzando DS5 creare un helloword e compilarlo. Nel caso in esame il programma si chiamerà testBaremetal e dalla compilazione risulterà *testBaremetal01.axf*.
- Occorre trasformarlo in binario con *arm-altera-eabi-objcopy -O binary testBaremetal01.axf testBaremetal01.bin*
- A questo punto, copiare il file .bin nella partizione FAT della scheda SD, ricollegare la scheda SD e avviare il sistema.

Se tutto va bene, con la scheda SD TERASIC il preloader configurerà la CPU e caricherà u-boot. A questo punto fermando u-boot sarà possibile caricare manualmente il programma bare metal. I comandi basi per accedere alla FAT tramite u-boot sono:

- **fatls**: permette di fare una lista dei file. La sintassi è:
  ```
  fatls <device> 0:1 <eventuale sotto-directory>
  ```
  nel nostro caso:
  ```
  fatls mmc 0:1
  ```
- **fatload**: carica un file in memoria:
  ```
  fatload <device> 0:1 <memory addr> <file>
  ```

Il problema è quindi capire qual è l'indirizzo di memoria a cui caricare il binario. Per settarlo occorre visualizzare il linker scrpt. Questo script di default è quello per CycloneV (marcato cv) nelle directory seguenti:

```
<altera_path>/embedded/host_tools/mentor/gnu/arm/baremetal/arm-altera-eabi/lib
<altera_path>/embedded/host_tools/mentor/gnu/arm/baremetal/arm-altera-eabi/lib/cortex-a9
```

I file da utilizzare sono i linker script *cycloneV-xxx.ld*. I suffissi possono essere:

- oc-ram: On Chip RAM che prevede questo indirizzamento di default:
  ```
  MEMORY
  {
    boot_rom (rx) : ORIGIN = 0xfffd0000, LENGTH = 64K
    ram (rwx) : ORIGIN = 0xffff0000, LENGTH = 64K
    ddr_sdram (rwx) : ORIGIN = 0x100000, LENGTH = 1023M
  }
  ```
- ram: Ram DDR che prevede questo indirizzamento:
  ```
  MEMORY
  {
    boot_rom (rx) : ORIGIN = 0xfffd0000, LENGTH = 64K
    oc_ram (rwx) : ORIGIN = 0xffff0000, LENGTH = 64K
    ram (rwx) : ORIGIN = 0x100000, LENGTH = 1023M
  }
  ```
- hosted: indica la presenza di un sistema operativo
- *<nulla>* indica che non c'è il sistema operativo

Le differenze tra hosted e non-hosted sono minime, ma si consiglia il non-hosted per il baremetal. È invece importante la ram perché il compilatore andrà ad indirizzare il codice in "ram", quindi il comanfo fatload dovrà utilizzare l'indirizzo definito da ram.

Quindi nel caso di On-chip Ram il comando di caricamento sarà:

```
fatload mmc 0:1 0xFFFF0000 test.bin
```

Mentre in caso di Ram sarà:

```
fatload mmc 0:1 0x100000 test.bin
```

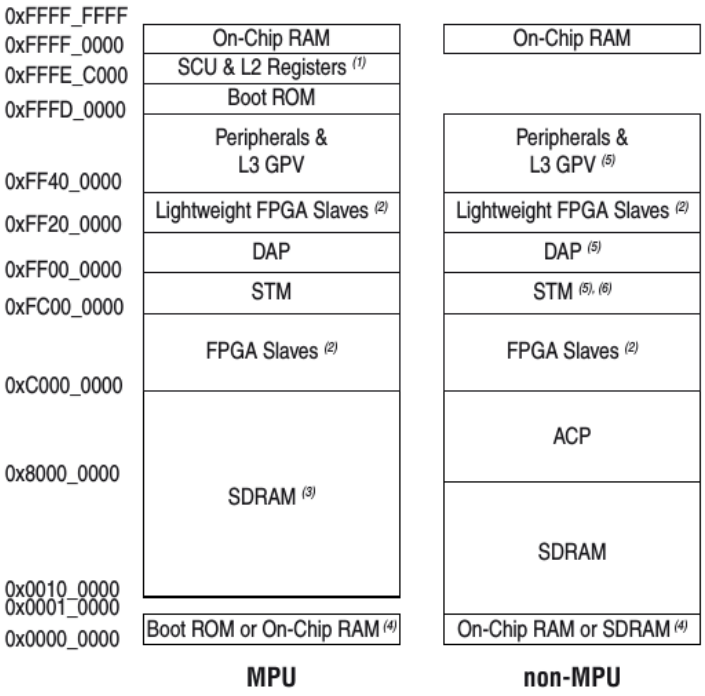
Tendenzialmente è meglio usare la ram esterna nel caso il programma sia troppo grosso.

Memory Address Space

Il linker script rappresenta lo spazio di indirizzamento del processore senza sistema operativo e quindi rappresenta la memoria fisica mappata come descritto in [5] e [6].

Nota: in Figura 1 è riportata la mappatura presa da [5]-v2012.

Si noti che la mappatura della RAM dipende da un bit di configurazione del registro `remap`, in particolare il bit `nonmpuzero`: se questo bit è 0 la mappatura della ram parte dall'indirizzo 0x0010_0000, altrimenti parte da 0x0000_0000. Analogamente anche la boot-ROM viene rimappata nel caso di modifica del bit `mpuzero`. Per maggiori informazioni [5]-7.



Notes to Figure 4–2:

(1) The SCU and L2 cache registers are located in the MPU subsystem and are not accessible from the L3 interconnect.
(2) This address range is not always accessible. For more information, refer to Table 4–3.
(3) The MPU subsystem has one master that connects to the interconnect and another master that connects directly to the SDRAM controller subsystem. The address filter registers in the MPU L2 control which MPU addresses are sent to each master. This figure assumes the filter registers contain their reset values.
(4) This address range is configurable. For more information, refer to Table 4–3.
(5) This address range is not accessible from the master peripheral interfaces. For more information, refer to "Master-to-Slave Connectivity Matrix" on page 4–6.
(6) This address range is not accessible from the DAP interface. For more information, refer to "Master-to-Slave Connectivity Matrix" on page 4–6.

Figura 1 – Memory Address Space: mappatura della memoria vista dalla CPU

2.2.1. I/O E MAKEFILE

Il preloader configura il processore e carica u-boot. Il programma che però verrà cariato da u-boot deve avere linkate tutte le funzione relative almeno allo standard output che di default è configurato sulla porta seriale a 115200kboud. Quindi ai sogenti è necessario aggiungere la parte relativa alle funzioni C che mancano. Il file ***io.c*** copiato dall'esempio FFT Baremetal di Rocketboards assolve al problema ed è il seguente:

```
#include <stdio.h>
#include <inttypes.h>

#include "socal/alt_uart.h"
#include "socal/hps.h"
#include "socal/socal.h"

#define STDOUT_FILENO 1

int _close/_fstat/isatty (int file)
{
    /* Succeeds only for STDOUT */
    return (file == STDOUT_FILENO) ? 0 : -1;
}

off_t _lseek(int file, off_t ptr, int dir)
{
    /* Succeeds only for STDOUT */
    return (file == STDOUT_FILENO) ? 0 : -1;
}
```

```
int _read(int file, void *ptr, size_t len)
{
    /* Always fails */
    return -1;
}

int _write(int file, char * ptr, unsigned len, int flag )
{
    /* Fails if not STDOUT */
    if(file != STDOUT_FILENO)
    {
        return -1;
    }

    /* Print each character to UART */
    for(int i=0; i<len; i++)
    {
        /* Wait until THR is empty*/
        while(1 != ALT_UART_LSR_THRE_GET(alt_read_word(ALT_UART0_LSR_ADDR)));
        /* Write character to THR */
        alt_write_word(ALT_UART0_RBR_THR_DLL_ADDR, ptr[i]);
    }

    /* All printed fine */
    return len;
}
```

Tutte le funzioni ritornano 0 o -1 se non viene trovato lo standard output, mentre solo la read e la write differiscono. In particolare la _read, che non serve in questo esempio, ritorna sempre -1, mentre la write accede alle risorse della USB. Senza questo file non verrà stampato nessun output.

A questo punto occorre il Makefile. Anche questo è stato copiato dall'esempio FFT. Le parti salienti sono:

```
[…]
EXAMPLE_SRC := test.c io.c
C_SRC       := $(EXAMPLE_SRC)

LINKER_SCRIPT := cycloneV-dk-ram.ld

MULTILIBFLAGS := -mcpu=cortex-a9 -mfloat-abi=softfp -mfpu=neon
CFLAGS    := -g -O3 -Wall -Werror -std=c99 $(MULTILIBFLAGS) -I$(HWLIBS_ROOT)/include -I. -
Iqsys_headers -I$(SOCAL_ROOT)
LDFLAGS := -T$(LINKER_SCRIPT) $(MULTILIBFLAGS)
[…]
ELF ?= $(basename $(firstword $(C_SRC))).axf
OBJ := $(patsubst %.c,%.o,$(C_SRC))
BIN = $(basename $(firstword $(C_SRC))).bin

.PHONY: all
all: $(BIN)

.PHONY: clean
clean:
    $(RM) $(ELF) $(OBJ) $(BIN)
    $(RM) *.map
    $(RM) *.objdump

%.c: $(HWLIBS_ROOT)/src/hwmgr/%.c
    $(CP) $< $@

$(OBJ): %.o: %.c Makefile $(QSYS_HDR_DIR)/$(QSYS_HDR)
    $(CC) $(CFLAGS) -c $< -o $@

$(ELF): $(OBJ)
    $(LD) $(LDFLAGS) $(OBJ) -o $@
    $(NM) $@ > $@.map
    $(OD) -d $@ > $@.objdump

$(BIN): $(ELF)
    $(OC) -O binary $(ELF) $(BIN)
```

Si noti che *qsys_headers* viene inclusa come directory, ma di fatto non viene usata in un programma "helloword" in quanto non si accede a nessuna periferica, in particolare a nessuna periferica FPGA.

3. U-BOOT – WORKSHOP 2

Per compilare u-boot occorre avere compilato il preloader come descritto in 2.1. il metodo più veloce prevede che quando si compila il preloader, automaticamente scarica uboot aggiornato. In altre parole nella directory *spl_bsp/uboot-socfpga/* ci sono i sorgenti compilabili con un "make".
Quallo completo è il seguente:

- export CROSS_COMPILE=arm-linux-gnueabihf-
- clonare il repository di u-boot: git clone https://github.com/altera-opensource/u-boot-socfpga.git
- cd u-boot-socfpga
- git tag -l rel*
- git tag -l ACDS*
- git checkout -b ws2_v2013.01.01_16.04.01 rel_socfpga_v2013.01.01_16.04.01_pr
- git branch
- clear u-boot directory: make mrproper
- configurazione per l'FPGA CyclonV: make socfpga_cyclone5_config
- compilare: make

A questo punto dovrebbe essere comparsa l'immagine boottabile ***u-boot.img***.

SD Terasic

Se già si dispone della SE della Terasic, il file u-boot.bin appena creato va copiato nella FAT, ma rinominato come *u-boot.src*

Nuovo preloader

Se nella SD è stato messo il nuovo preloader, il file u-boot.bin va semplicemente copiato perché questo preloader era stato configurato per caricare appunto questo file.

Eseguendo il boot si vede che lo splash screen è cambiato e viene visualizzato solo una volta quanto segue:

```
U-Boot SPL 2013.01.01 (Sep 01 2016 - 13:48:13)
BOARD : Altera SOCFPGA Cyclone V Board
CLOCK: EOSC1 clock 25000 KHz
CLOCK: EOSC2 clock 25000 KHz
CLOCK: F2S_SDR_REF clock 0 KHz
CLOCK: F2S_PER_REF clock 0 KHz
CLOCK: MPU clock 925 MHz
CLOCK: DDR clock 400 MHz
CLOCK: UART clock 100000 KHz
CLOCK: MMC clock 50000 KHz
CLOCK: QSPI clock 3613 KHz
RESET: COLD
INFO : Watchdog enabled
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
SDRAM: 1024 MiB
ALTERA DWMMC: 0
reading u-boot.img
reading u-boot.img


U-Boot 2013.01.01-00134-g7dd0473 (Sep 03 2016 - 00:28:18)

CPU   : Altera SOCFPGA Platform
BOARD : Altera SOCFPGA Cyclone V Board
I2C:   ready
DRAM:  1 GiB
```

```
MMC:   ALTERA DWMMC: 0
In:    serial
Out:   serial
Err:   serial
Skipped ethaddr assignment due to invalid EMAC address in EEPROM
Net:   mii0
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
SOCFPGA_CYCLONE5 #
```

4. COMPILAZIONE DEL KERNEL

A detta di ALTERA, il kernel vanilla dovrebbe essere costantemente allineato ai prodotti Cyclon/Arria.
Seguiranno quindi i metodi adottati. Il migliore dovrebbe essere quello descritto in WS2 [12], ma ad ogni modo verranno elencati i vari metodi per come sono stati testati.

4.1. METODO DI ROBERT C NELSON

Ad ogni modo si procede come indicato in [23]:

- Download dei sorgenti del kernel:
  `git clone https://github.com/RobertCNelson/socfpga-kernel-dev`
  La versione dei sorgenti è divisa in modo simile a quelli del kernel vanilla, quindi mainline, stable e long term.
- Portarsi nella directory *socfpga-kernel-dev*
- Muoversi nel branch desiderato. Si noti che quello più aggiornato è "master", ma non sembra venire popolato.
  `git checkout origin/master -b tmp`
  altre possibilità ad oggi sono `origin/4.4.x` o `origin/4.6.x`: io scelo la `4.6x`
- Lanciare la compilazione con *./build_kernel.sh*: lo script identificherà il sistema operativo su cui si sta sviluppando (Mint in questo caso) e verificherà se mancano dei pacchetti da installare. Nel mio caso dovevo eseguire:
  ```
  sudo apt-get update
  sudo apt-get install device-tree-compiler lzma lzop
  ```

  ATTENZIONE: questo script però scarica il compilatore Linaro, mentre avendo già quello ALTERA si preferisce usare quello, quindi occorrerebbe modificare lo script. Comunque Linaro è di solito più aggiornato rispetto al compilatore ALTERA.

- Lo script sembra poi essersi fermato una volta scaricato il kernel dando un errore "fatale" sul fatto che non era in grado di determinare la mail. Ad ogni modo non dovrebbe essere un problema.
- Preconfiguro il kernel al default: `make ARCH=arm socfpga_defconfig`
  Si noti che questo comando sul kernel vanilla non funziona, ma di fatto dovrebbe semplicemente scrivere il .config con quello minimo necessario per ALTERA
- Configurando il kernel (`make ARCH=arm menuconfig`) si può vedere che la parte relativa ad ALTERA è stata configurata.
- Compilare con: `make ARCH=arm CROSS_COMPLIE=arm-linux-gnueabihf- zImage -j4`
- Compilare i moduli: `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules -j4`
- Installare i moduli in una directory temporanea [24]:
  - Creare per esempio: `mkdir ../modules`
  - Installare i moduli: `make ARCH=arm modules_install INSTALL_MODPATH=../modules`

Nonostante avessi richiesto il kernel 4.6.x il sistema ha compilato la minline 4.8.0, per altro non ancora rilasciata in *stable*.
Il primo caricamento del kernel si è piantato alla configurazione del CAN, senza dare un Kernel Panic. Un errore simile lo avevo avuto in XILINX quando la periferica veniva configurata, ma non veniva attivata la sua linea di clock e di fatto quindi non rispondeva. Non so se questo è lo stesso problema.

Il kernel non è stato (ancora) configurato; l'unica cosa che è stata tolta è il bus PCI che non è sicuramente presente. Quindi il fatto che il CAN venga caricato erroneamente potrebbe essere dovuto al device tree che è ancora quello vecchio.

> **Nota**: a questo punto di vorrebbe prelevare il .config dal kernel Terasic, ma il ripristino del vecchio kernel sembra non funzionare, quindi o ripristino la partizione (compreso il preloader) oppure si prova con il devicetree

4.1.1. RICOMPILAZIONE DEL DEVICE TREE PER DE0-NANO-SOC INTERNO AI SORGENTI

- Portarsi nella directory dei sorgenti del kernel: *socfpga-kernel-dev/KERNEL/*
- Cercare i .dts: `find -iname "*.dts" | grep -i de0`

Il risultato mostra che esiste un DTS per la DE0: *./arch/arm/boot/dts/socfpga_cyclone5_de0_sockit.dts*
La compilazione manuale del DTB è descritta bene in [25]. Analizzando il Makefile del kernel è però possibile ricorrere al seguente comando:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- socfpga_cyclone5_de0_sockit.dtb
```

In realtà già il WS2 [12] spiega che è possibile eseguire questo definendo anche quale file DTS utilizzare qualora si vogliano fare delle modifiche:

```
make ARCH=arm CONFIG_DTB_SOURCE=arch/arm/boot/dts/<your-dev-board>.dts <your-dev-board>.dtb
```

Ad ogni modo, il primo comando da come risultato:

```
DTC     arch/arm/boot/dts/socfpga_cyclone5_de0_sockit.dtb
```

Il file va copiato nella FAT con il nome *socfpga.dtb*.

Il sistema a questo punto funziona! *uname –r* restituisce: **4.8.0-rc4-00264-g0141af1**

4.2. COMPILAZIONE DEL KERNEL SECONDO WS2

Secondo WS2 [12] la procedura manuale per la compilazione del kernel è la seguente e dovrebbe essere la stessa descritta in e implementata nello script utilizzato al paragrafo 4.1:

- Creare una directory di test, esempio *./tmp* e spostarcisi dentro
- Clonare il repository `git clone https://github.com/altera-opensource/linux-socfpga.git`
  Questo repository punta effettivamente a tutte le release del kernel rilasciate da altera. L'ultima ad oggi è la 4.6 con la 4.8 in rilascio (Figura 2).
- Entrare nella directory appena creata (linux-socfpga) e:

  `git branch`
  restituirà il branch master

  `git tag -l rel*`
  mostrerà tutti i branch non master disponibili. In questo esempio si selezionerà l'ultimo stabile: *rel_socfpga-4.6_16.09.01_pr*. Vedere [25] per maggiori informazioni.

  `git checkout rel_socfpga-4.6_16.09.01_pr`
  checkout dovrebbe creare la copia locale di *rel_socfpga-4.6_16.09.01_pr*.

A questo punto non dovrebbe essere necessario applicare patch e si procede alla compilazione del kernel:

```
make ARCH=arm socfpga_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4 zImage
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4 modules
mkdir ../modules
make ARCH=arm INSTALL_MOD_PATH=../modules modules_install
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- socfpga_cyclone5_de0_sockit.dtb
```

Copiando kernel, moduli e DTB si dovrebbe essere tutto OK, infatti *uname –r* restituisce **4.6.0-00177-gf9fb6e1**.

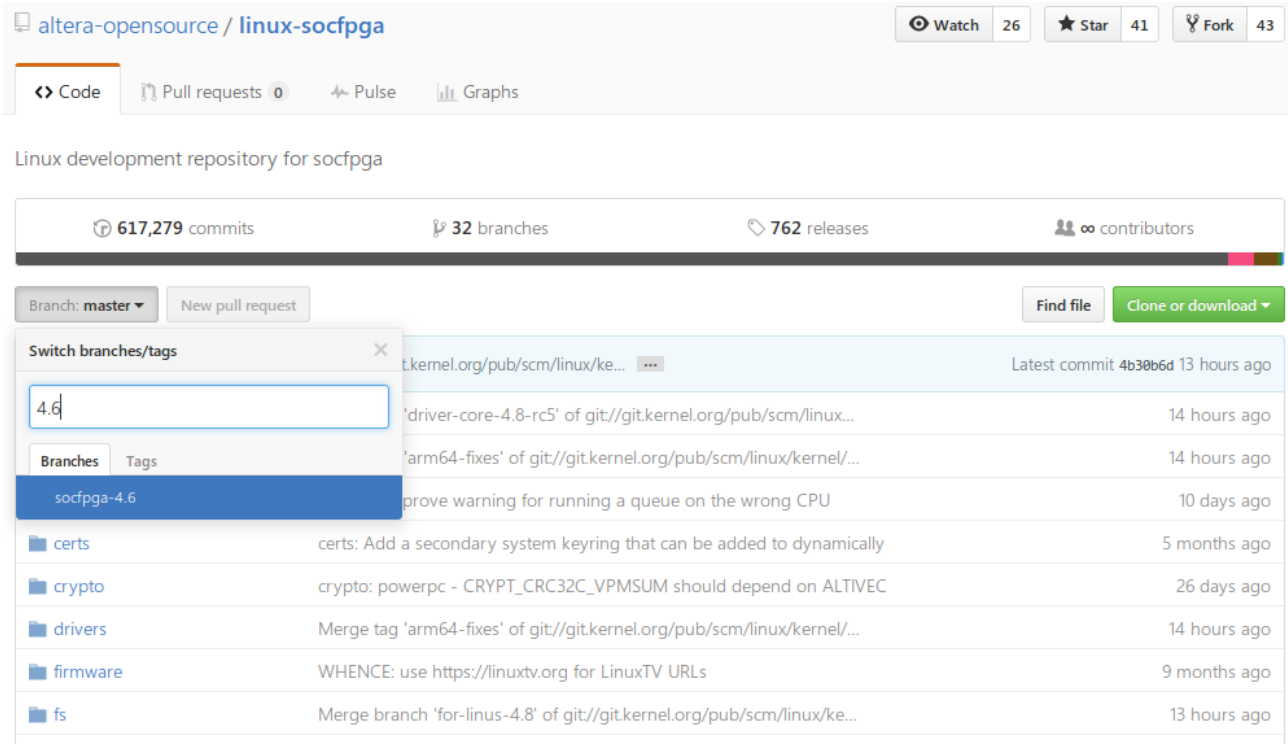


Figura 2 – Repository del Kernel per ALTERA SoC

Qualora vengano aggiunti dei moduli, la ricompilazione aggiungerà un suffisso "-dirty" al kernel in quanto si sta generando un tree diverso da quello ufficiale sul quale non è stato eseguito un commit.
Cercando nel codice, la versione del kernel si può trovare in:

```
include/config/kernel.release:          4.6.0-00177-gf9fb6e1-dirty
include/generated/utsrelease.h:         #define UTS_RELEASE "4.6.0-00177-gf9fb6e1-dirty"
```

5. RIFERIMENTI E LINK

[1] Terasic DE0 Nano SoC board
[2] Manuale DE0 Nano SoC
[3] Cyclone V Device Handbook volume 1 – Device Interface and Integration – cv_5v2.pdf
[4] Cyclone V Device Handbook volume 2 – Transceivers – cv_5v3.pdf
[5] Cyclone V Device Handbook volume 3 – Hard Processor System – Technical Reference Manual – cv_5v4.pdf (indirizzo alternativo versione 2012)
[6] Cyclone V Address Map
[7] *DE0-Nano-SoC_My_First_FPGA.pdf*: Tutorial Terasic (distribuiti con la documentazione della scheda)
[8] *DE0-Nano-SoC_My_First_HPS.pdf*: Tutorial Terasic (distribuiti con la documentazione della scheda)
[9] *DE0-Nano-SoC_My_First_HPS-Fpga.pdf*: Tutorial Terasic (distribuiti con la documentazione della scheda)
[10] Lista dei workshop di riferimento: Altera SoC Workshop Series
[11] RocketBoards Workshop 1: WS1– Intro to Altera SoCDevices for SW Developers
[12] RocketBoards Workshop 2: WS2 Linux KernelIntroduction for Altera SoCDevices
[13] RocketBoards Workshop 3: WS3 Developing Drivers for Altera SoC Linux
[14] Golden System Reference Design: AV CV GSRD 16.0 User manual – fasi per la generazione di un sistema Linux per dispositivi ALTERA.
[15] ALTERA SoC Embedded Design Suite User Guide – *ug_soc_eds.pdf* (ug-1137 del 2015.08.06) https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_soc_eds.pdf
[16] SoC HPS System Generation Using Qsys - https://www.youtube.com/watch?v=8BehnPg8IvM
[17] SoC HPS Quartus II Integration (HW/SW Hand-off) - https://www.youtube.com/watch?v=L8FMSy7Uxjc
[18] Preloader and U-boot Generation for Altera Cyclone V SoC - https://www.youtube.com/watch?v=vS7pvefsbRM
[19] Introduzione alle FPGA (*FPGA-Introduction.pdf*): http://lugman.org/index.php/Documentazione

[20] Elenco documentazione Cyclone V - http://rocketboards.org/foswiki/view/Documentation/CycloneVSoCLinks
[21] Talk ARM, FPGA & Linux - Linux Day 2015 – LUGMan
[22] BareMetal Application per DE0_NANO_SOC caricato al boot
[23] Ottenere e compilare il Kernel Linux, U-Boot, ecc per Cyclone V SoC per DE0-Nano-SoC
[24] Building External Modules: indica quali parametri e macro servono per la compilazione del kenel
[25] Embedded Linux Beginner Guide – Guida di Rocketboards abbastanza chiara e completa sulla creazione di u-Boot, Kernel e devicetreee anche in relazione all'FPGA partendo dalla configurazione dell'HPS. Comprende anche link a guide e video di approfondimento.