**Tổng Công ty Công nghiệp công nghệ cao Viettel**

**Porting SDK 2020.1 cho RU 4G 2T2R**

|  |  |
| --- | --- |
| **Họ và tên:** | Ngô Minh Khánh |
| **Đơn vị:** | Phòng Phần mềm hệ thống – Trung tâm VTBR |

**HÀ NỘI – 2023**

Mục lục

[Mục lục 1](#_Toc130809200)

[1. Giới thiệu chung 1](#_Toc130809201)

[2. Bootup RRU 4G 2T2R. 1](#_Toc130809202)

[2.1. Bringup mạch qua JTAG. 1](#_Toc130809203)

[2.2. Quá trình boot trên Zynq. 3](#_Toc130809204)

[2.3. Setup môi trường build offline. 4](#_Toc130809205)

[2.4. Cài đặt sử dụng external source (u-boot, linux-kernel). 5](#_Toc130809206)

[2.4.1. U-boot. 5](#_Toc130809207)

[2.4.2. Linux-kernel. 6](#_Toc130809208)

[2.5. Thêm Meta Layer. 7](#_Toc130809209)

[2.6. Chỉnh sửa U-boot defconfig target, config header. 8](#_Toc130809210)

[2.7. Cấu hình device-tree. 10](#_Toc130809211)

[2.8. Chỉnh sửa Flash Memory Layout. 11](#_Toc130809212)

[2.9. Cấu hình RootFS. 12](#_Toc130809213)

[2.10. Boot mạch. 12](#_Toc130809214)

[2.11. Các vấn đề chưa giải quyết. 13](#_Toc130809215)

[2.11.1. Không hiện thị serial trên kernel 13](#_Toc130809216)

[3. Tìm hiểu về hệ thống Clock. 13](#_Toc130809217)

[3.1. Hệ thống clock trên mạch RRU 4G 2T2R. 13](#_Toc130809218)

[3.2. Cấu tạo tạo AD9548. 15](#_Toc130809219)

# Giới thiệu chung

*<Tại sao/lý do cần porting SDK mới cho hệ thống 4G>*

Kế hoạch thực hiện:

|  |  |  |
| --- | --- | --- |
| **Main Task** | **Sub Task** | **Nội dung cần chuẩn bị** |
| Bootup RRU-4G-PVT2, sử dụng source base trên RRU-5G | Setup môi trường build | Setup môi trường build offline, sstate-cache tương ứng cho Zynq-7000. Thử nghiệm build BSP ZC702 với petalinux verify môi trường đã setup thành công |
| Build và nạp được lên RRU-4G | Setup petalinux build local với source code từ RRU-5G: adi-5.4.0-v3 (linux) & v2020.1 (u-boot) |
| Mạch RRU-4G bootup thành công với source base tương tự RRU-5G | Nạp bản build với XSA từ FPGA vào mạch RRU-4G, chỉnh sửa driver, uboot, fix các lỗi gặp phải trong quá trình bootup |
|
| Chỉnh sửa, thêm profile clock  156.25 cho ECPRI | Tìm hiểu topo clock đang hoạt động trên RRU-4G | Tìm hiểu topo clock hiện tại, nắm rõ driver clock, phương pháp cấu hình, chỉnh sửa profile. |
| Thêm profile clock 156.25 vào  port tương ứng cho port ETH ECPRI | Phối hợp với HW đẩy bộ profile thêm clock 156.25 vào port ETH cho giao diện ECPRI. Verify thông qua đo kiểm hoặc trạng thái đèn của cổng NIC |
| Verify topo clock mới | Sau khi porting core oran ethernet vào RRU, kiểm tra trạng thái block-lock để đảm bảo clock đã ready. Trong trường hợp lỗi, tính đến giải pháp bootup clock trước khi bootup FPGA |
| Porting Oran ethernet, các tiến trình đồng bộ ptp4l, vlan setup | Porting core Oran ethernet, chuẩn bị các driver liên quan như dma… Chuẩn bị app liên quan đến | Tham khảo cách porting từ linux-kernel trên các phiên bản RRU-5G hiện tại. Chuẩn bị các app liên quan đến tiến trình đồng bộ. Thiết lập quy trình init hợp lý. Quy trình thành công khi đọc được trạng thái, offset đồng bộ từ ptp4l, từ phía DU có thể ssh trực tiếp đến RRU thông qua giao diện ECPRI |
| Porting driver điều khiển RFIC, JESD | Tìm hiểu driver RFIC trên RRU-4G | Đảm bảo cấu hình được các thông số cơ bản trên RRU như gain, tần, chạy được ETM từ DU qua RRU, decode được và có thể chạy được dịch vụ, phục vụ quá trình phát triển software layer |
| Tìm hiểu driver JESD trên RRU-4G | Đảm bảo JESD hoạt động đúng như mong muốn |

# Bootup RRU 4G 2T2R.

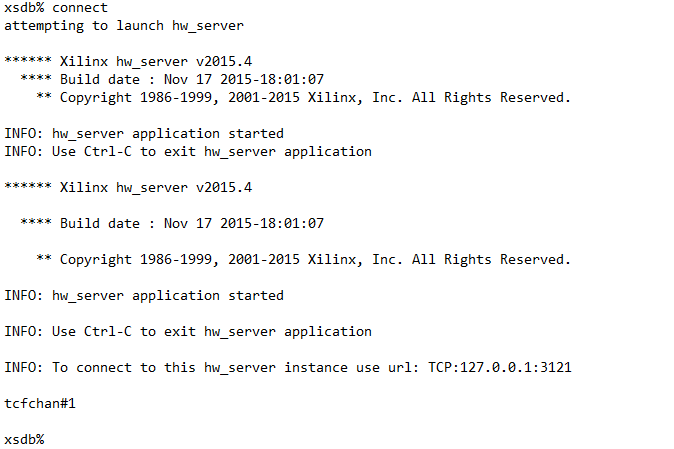
## Bringup mạch qua JTAG.

Trước khi bring-up qua JTAG cần chuẩn bị các file cần thiết sau:

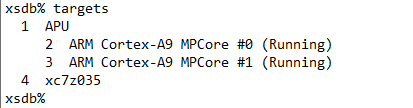
* u-boot.elf
* ps7\_init.tcl
* ps7\_init.c
* ps7\_init.h
* ps7\_init\_gpl.c
* ps7\_init\_gpl.h

Thực hiện các lệnh sau để nạp trên Xilinx SDK 2015.4 (VD: các file nạp đặt ở đường dẫn D:/bringup-rru-2t2r-4g):

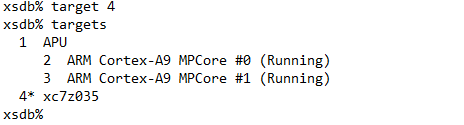
1. cd D:/bringup-rru-2t2r-4g
2. connect



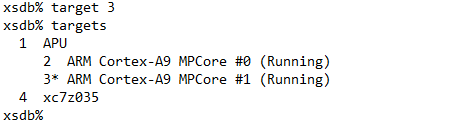
1. targets



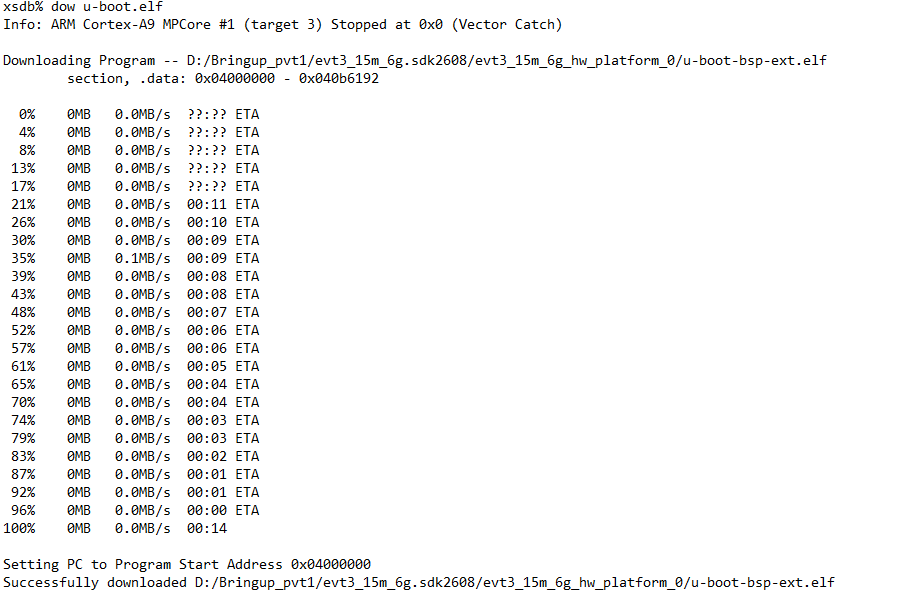
1. target 4



1. rst -srst
2. target 3



1. rst -processor
2. source ps7\_init.tcl
3. ps7\_init
4. ps7\_post\_config
5. dow u-boot.elf

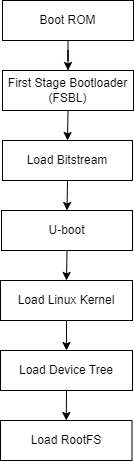


1. con

Sau khi nạp file u-boot.elf thành công, thực hiện nạp lại các file cần thiết ở mục [2.10 Boot mạch](#_Boot_mạch.).

## Quá trình boot trên Zynq.

Quá trình boot của Zynq tương tự với các dòng chip dựa trên kiến trúc ARM khác, tuy nhiên do Zynq SoC bao gồm hai thành phần chính là processing system (PS) và programmable logic (PL) – FPGA, do đó cần có quá trình “Load Bitstream” để cấu hình cho thành phần FPGA. Quá trình boot của Zynq được thể hiện ở hình bên dưới



Hình 2.1. Boot Sequence của Zynq.

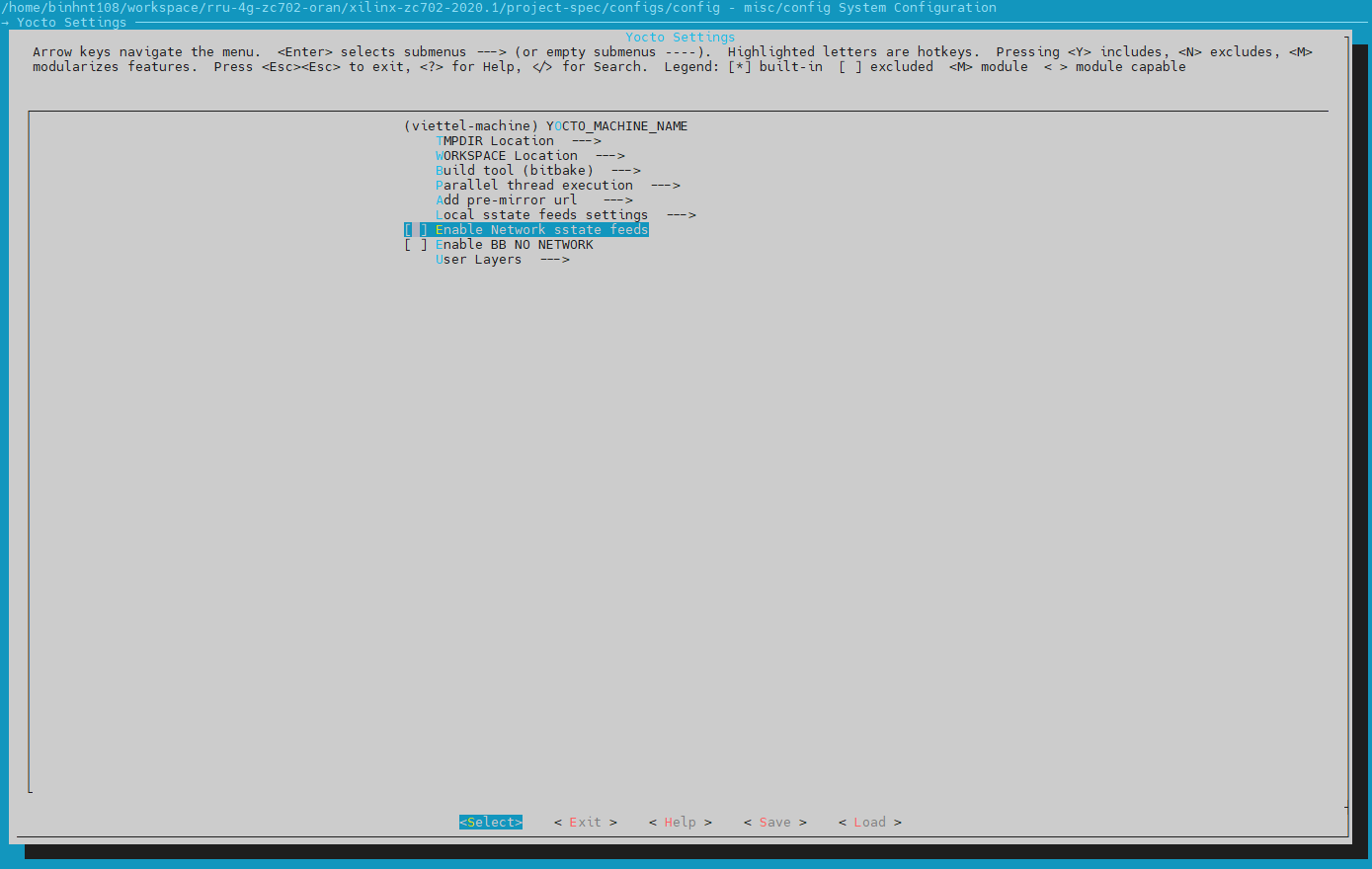
## Setup môi trường build offline.

*Thiếu bước lựa chọn option trên petalinux để fetch sstate-cache offline thay vì online fetch*

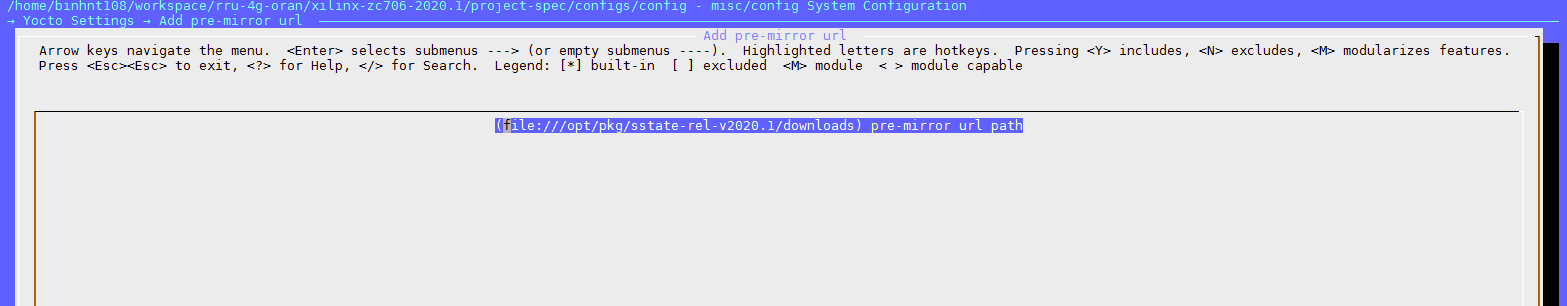
Do build server của Viettel không được kết nối Internet do đó khoog thể tải các package cần thiết cho quá trình build. Petalinux cung cấp kho lưu trữ Mirrors chứa các gói mã nguồn, các tệp cần thiết để xây dựng Linux và Sstate Feeds chứa các pre-built binary files, người dùng có thể tải các tính năng trên về local và cấu hình để sử dụng chúng ([htp://petalinux.xilinx.com/sswreleases/rel-v2020/aarch64/sstate-cache](file:///D:\khanhnm\RRU_4G_ORAN\Bao_cao\RRU_4G_ORAN.docx))

Để cài đặt môi trường build, thực hiện các bước sau sau:

Lựa chọn option để petalinux fetch sstate-cache offline: petalinux-config -> Yocto-setting -> Bỏ chọn “**Enable Network sstate feeds**”

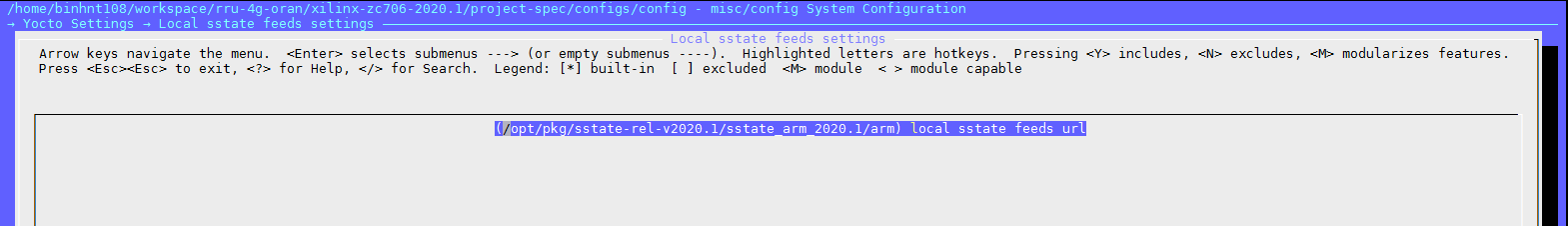


**Mirrors**: petalinux-config -> Yocto-setting -> Add pre-mirror URL -> Điền đường dẫn tới thư mục mirror



Hình 2.2. Cài đăt đường dẫn đến Mirrors

**Sstate Feeds**: petalinux-config -> Yocto-setting ->  Local sstate feeds setngs -> điền đường dẫn đến thư mục sstate



Hình 2.3. Cài đặt đường dẫn đến Sstate.

## Cài đặt sử dụng external source (u-boot, linux-kernel).

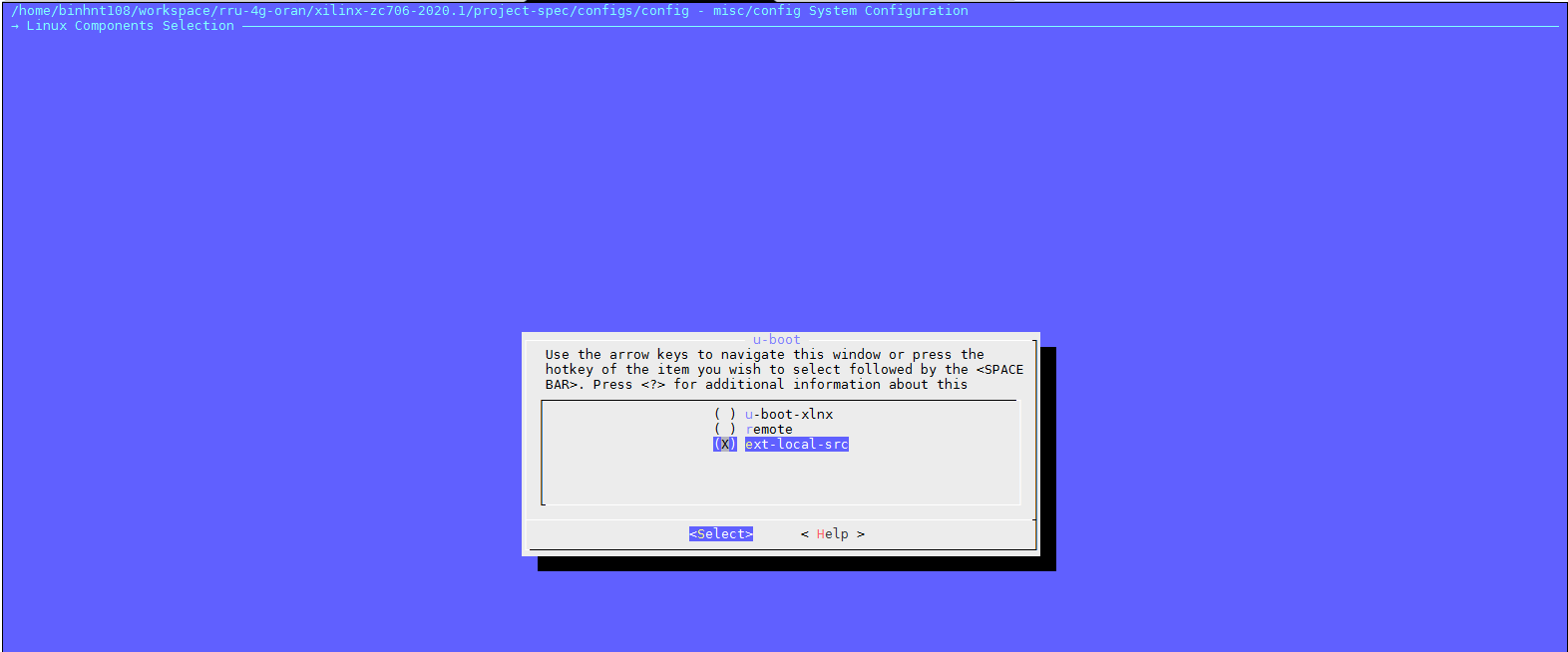
*Thiếu quy trình build lại khi chỉnh sửa source code U-boot, linux. Ví dụ: Khi chỉnh sửa source U-boot/Linux kernel thì có cần proper hết để build lại không, nếu đổi branch thì có cần cập nhật gì trên meta không?*

Để dễ dàng có thể tuỳ chỉnh, phát triển và kiểm soát các phiên bản cho U-boot hay Kernel, ta sử dụng tính năng external source để thiết lập đường dẫn tới các nguồn bên ngoài.

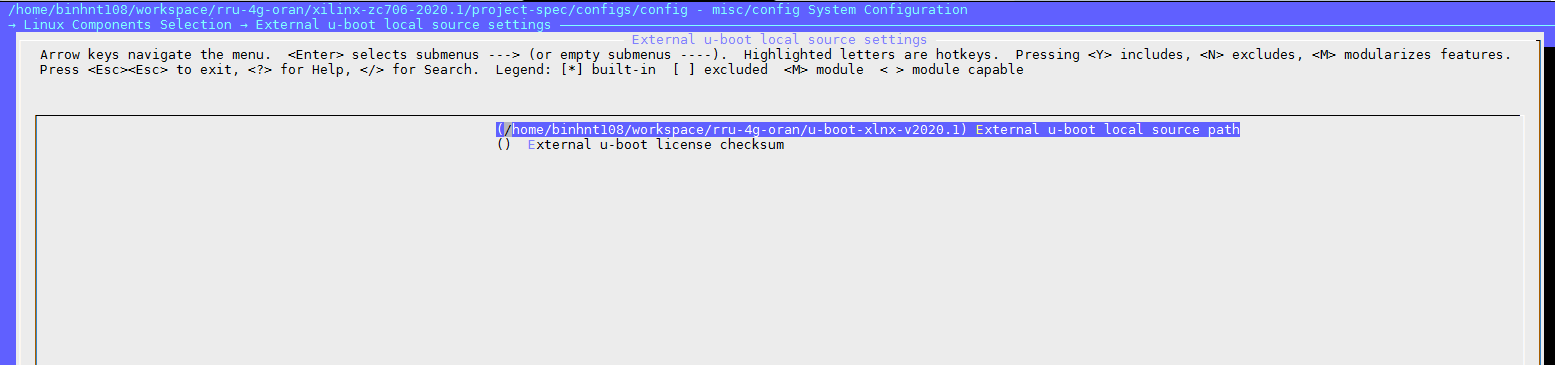
### U-boot.

Để thực hiện cấu hình ext-src cho U-boot cần thực hiện 2 bước sau:

* **Enable ext-src:** petalinux-config -> Linux Components Selection -> u-boot -> ext-local-src
* **Thiết lập đường dẫn:** petalinux-config -> Linux Components Selection -> External u-boot local source settings



Hình 2.4. Enable ext-src U-boot.

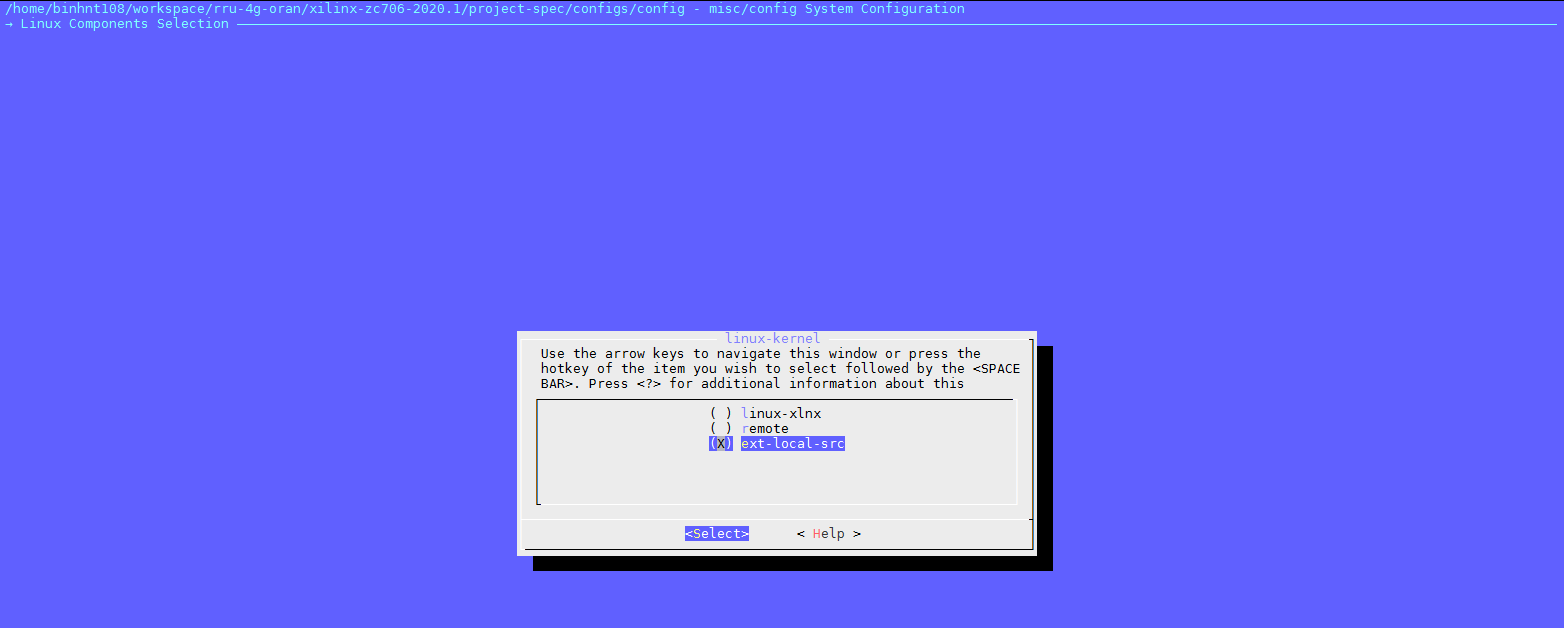


Hình 2.5. Thiết lập đường dẫn U-boot.

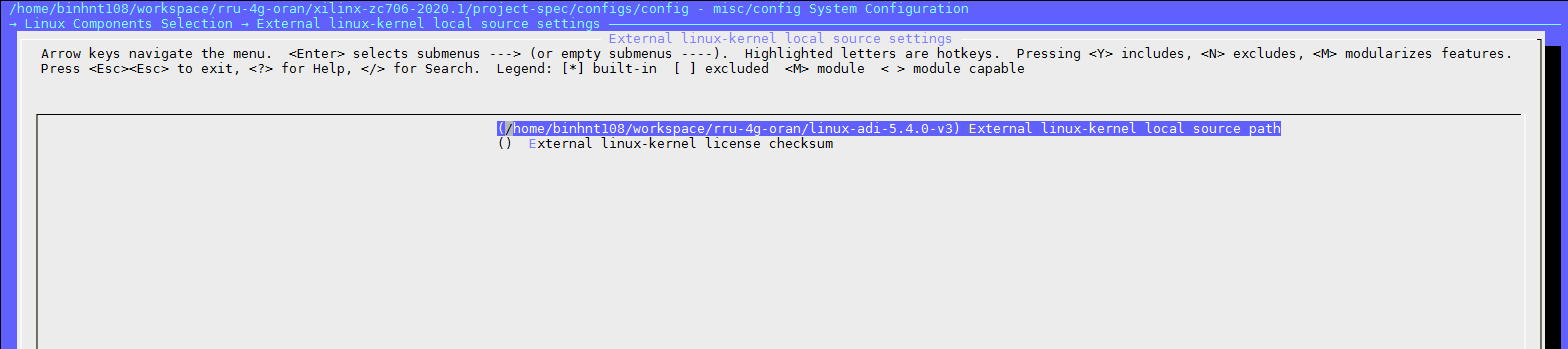
### Linux-kernel.

Tương tự như U-boot, để thực hiện cấu hình ext-src cho Linux-kernel cũng cần thực hiện 2 bước sau:

* **Enable ext-src:** petalinux-config -> Linux Components Selection -> linux-kernel -> ext-local-src
* **Thiết lập đường dẫn:** petalinux-config -> Linux Components Selection -> External linux-kernel local source settings



Hình 2.6. Enable ext-src cho linux-kernel.



Hình 2.7. Thiết lập đường dẫn cho linux-kernel.

### Cập nhật lại source.

Nếu chỉnh sửa lại source, ta có thể clean và build lại như sau

petalinux-build -c <u-boot|kernel> -x cleansstate | petalinux-build -c <u-boot|kernel>

## Thêm Meta Layer

*Cụ thể, meta-rru-bsp/meta-rru-core/meta-rru-linux chứa layer và các lớp definition cho các phần nào tương ứng? Khi nào cần sửa các file .bb của meta này?*

Trong Petalinux, các meta-layer là tập hợp các recipes và các tệp cấu hình cung cấp các gói phần mềm, thư viện và cài đặt cấu hình bổ sung để xây dựng Linux. Các meta layer được tổ chức theo cấu trúc phân cấp, trong đó mỗi lớp có thể phụ thuộc vào các lớp bên dưới nó trong cấu trúc phân cấp. Trong dự án này, project được chia thành 3 layer là meta-rru-bsp, meta-rru-core và meta-rru-linux.

Cấu hình cho các layer thực hiện như sau:

1. Tạo file ở đường dẫn *<proj\_root>/project-spec/meta-rru/meta-rru-A*
2. petalinux-cofig -> Yoctor Settings -> User Layers
3. Nhập đường dẫn tới meta layer: *${PROOT }/project-spec/ meta-rru/meta-rru-A*
4. Sửa nội dung trong file *${proot}/project-spec/meta-rru/meta-rru-A/conf/locallayer.conf*  với nội dung như sau:

#We have a conf and classes directory, add to BBPATH

BBPATH .= ":${LAYERDIR}"

#We have recipes-\* directories, add to BBFILES

BBFILES += "${LAYERDIR/recipes-\*/\*/\*.bb \

    ${LAYERDIR/recipes-\*/\*/\*.bbappend"

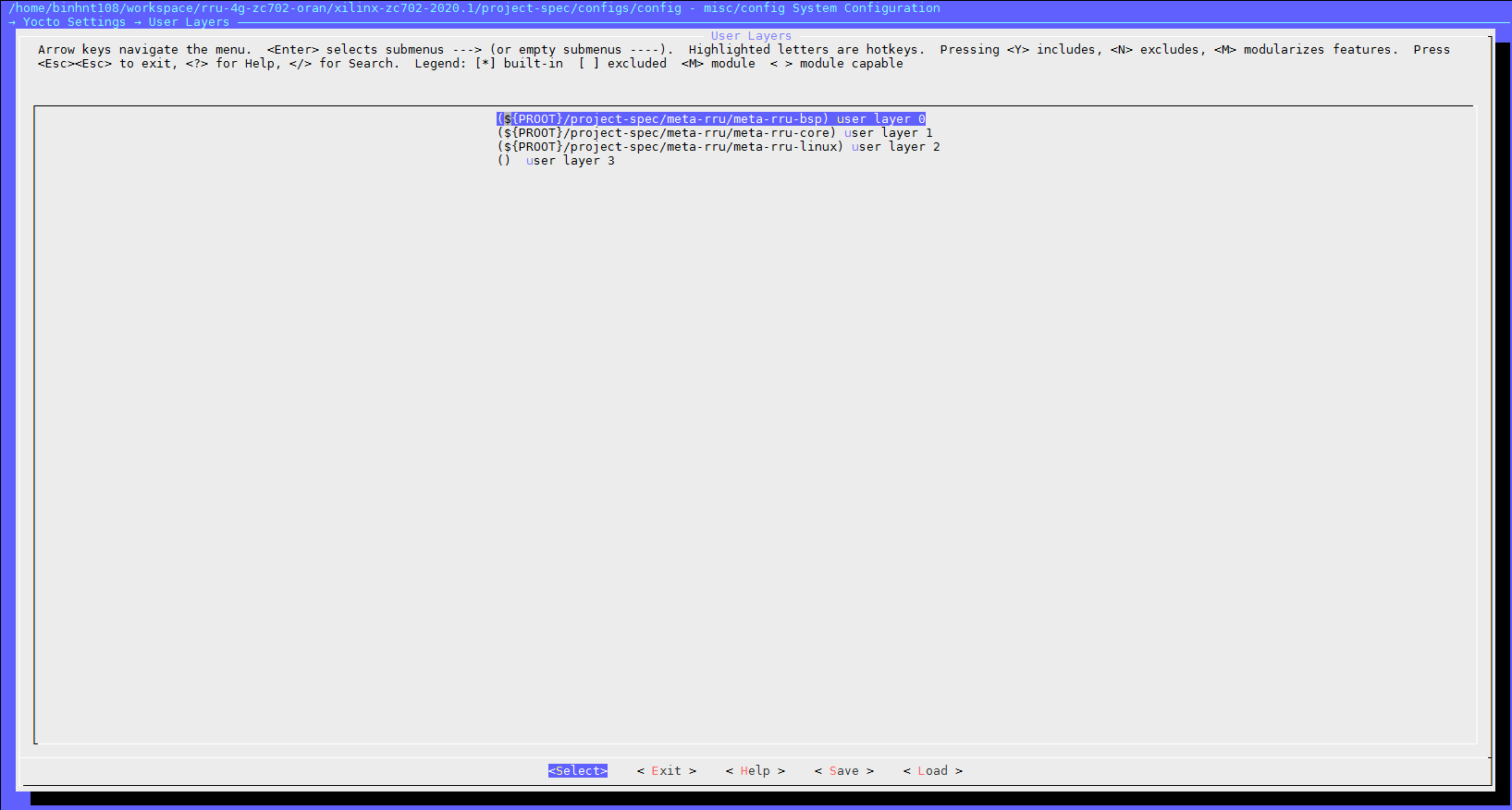
BBFILE\_COLLECTIONS += "meta-A"

BBFILE\_PATTERN\_meta-A = "^${LAYERDIR}/"

BBFILE\_PRIORITY\_meta-A = "6"

LAYERSERIES\_COMPAT\_meta-A = "zeus"

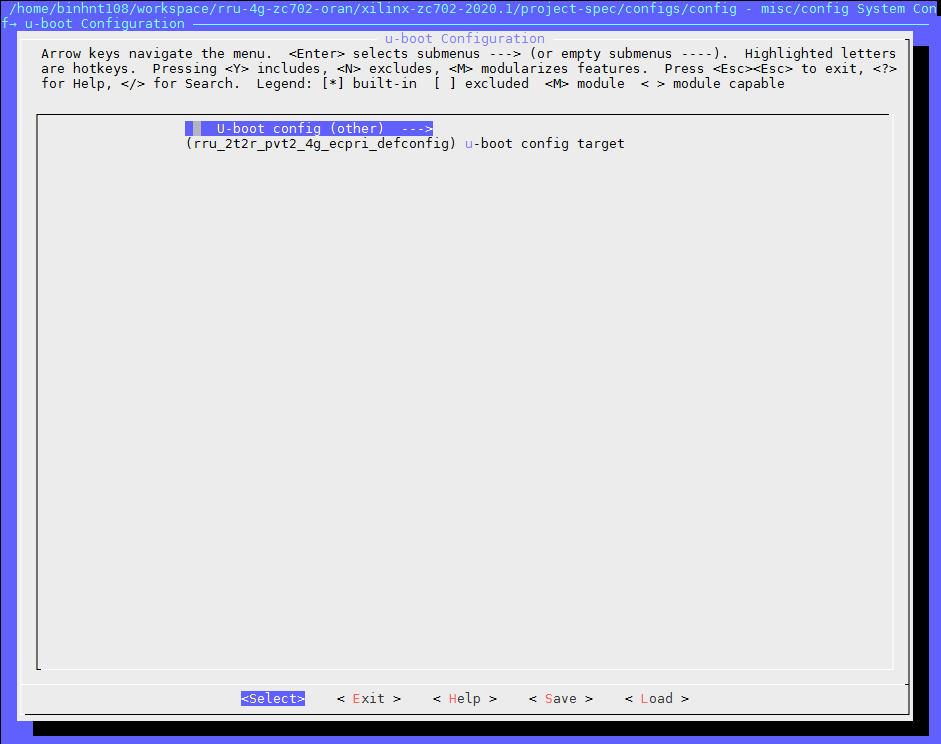
\* Trong đó, A = {bsp, core, linux}



Hình 2.8. Cấu hình layer trong Petalinux.

## Chỉnh sửa U-boot defconfig target, config header.

Chỉnh sửa U-boot defconfig thực hiện các thao tác sau: petalinux-config -> U-boot Configuration -> Điền tên file U-boot defconfig



Hình 2.9. Chỉnh sửa U-boot config target trong Petalinux

Để thiết lập đường dẫn tới uboot header file, thêm cấu hình sau vào file uboot defconfig

CONFIG\_SYS\_CONFIG\_NAME="rru\_2t2r\_pvt2\_4g\_ecpri\_config"

Sau đó, chỉnh sửa nội dung trong file rru\_2t2r\_pvt2\_4g\_ecpri\_config.h để thêm những cấu hình cần thiết cho thiết bị

#ifndef \_\_CONFIG\_2T2R\_4G\_H

#define \_\_CONFIG\_2T2R\_4G\_H

#include <configs/zynq-common.h>

#define CONFIG\_BOOTDELAY 5

#define PSSERIAL0 "psserial0=setenv stdout ttyPS0;setenv stdin ttyPS0\0"

#define SERIAL\_MULTI "serial=setenv stdout serial;setenv stdin serial\0"

#define CONSOLE\_ARG "console=console=ttyPS0,115200\0"

#define SERIAL\_MULTI "serial=setenv stdout serial;setenv stdin serial\0"

#define CONFIG\_BAUDRATE 115200

/\* Extra U-Boot Env settings \*/

#undef CONFIG\_EXTRA\_ENV\_SETTINGS

#define CONFIG\_EXTRA\_ENV\_SETTINGS \

    "fdt\_high=0x20000000\0"     \

    "initrd\_high=0x20000000\0"  \

    "scriptaddr=0x20000\0"  \

    "script\_size\_f=0x40000\0"   \

    "fdt\_addr\_r=0x1f00000\0"        \

    "pxefile\_addr\_r=0x2000000\0"    \

    "kernel\_addr\_r=0x2000000\0"     \

    "scriptaddr=0x3000000\0"        \

    "ramdisk\_addr\_r=0x3100000\0"    \

    DFU\_ALT\_INFO \

    BOOTENV \

    SERIAL\_MULTI \

    CONSOLE\_ARG \

    PSSERIAL0 \

    "set\_tftp=setenv serverip 192.167.120.238 && setenv ipaddr 192.167.120.105 && ping $serverip\0" \

    "clobstart=0x01000000\0" \

    "netstart=0x01000000\0" \

    "bootsize=0xf00000\0" \

    "bootstart=0x0\0" \

    "boot\_img=BOOT.BIN\0" \

    "load\_boot=tftpboot ${clobstart} ${boot\_img}\0" \

    "install\_boot=sf probe 0 && sf erase ${bootstart} ${bootsize} && " \

    "sf write ${clobstart} ${bootstart} ${filesize}\0" \

    "kernelsize=0xc00000\0" \

    "kernelstart=0xf40000\0" \

    "kernel\_img=image.ub\0" \

    "load\_kernel=tftpboot ${clobstart} ${kernel\_img}\0" \

    "install\_kernel=sf probe 0 && sf erase ${kernelstart} ${kernelsize} && " \

    "sf write ${clobstart} ${kernelstart} ${filesize}\0" \

    "jffs2size=0x2480000\0" \

    "jffs2start=0x1b40000\0" \

    "jffs2\_img=rootfs.jffs2\0" \

    "load\_jffs2=tftpboot ${clobstart} ${jffs2\_img}\0" \

    "install\_jffs2=sf probe 0 && sf erase ${jffs2start} ${jffs2size} && " \

    "sf write ${clobstart} ${jffs2start} ${filesize}\0" \

    "cp\_kernel2ram=sf probe 0 && sf read ${netstart} ${kernelstart} ${kernelsize}\0" \

    "default\_bootcmd=run cp\_kernel2ram && bootm ${netstart}\0"

#endif /\* \_\_CONFIG\_2T2R\_4G\_H \*/

## Cấu hình device-tree.

Trong Petalinux project, người dùng có thể cấu hình, tuỳ chỉnh device-tree trong file *<plnx-projroot>/project-spec/meta-user/recipes-bsp/devicetree/files/system-user.dtsi*. Trong quá trình build, petalinux sẽ include nội dung từ file này system-top.dts để tạo ra file device-tree cuối cùng.

Tuy nhiên để dễ dàng cập nhẫn và kiểm soát device-tree cùng với source linux một các dễ dàng, tạo folder rru\_2t2r\_4g và rru\_2t2r\_4g.dtsi, sau đó chỉnh sửa file device-tree.bbappend với nội dung bên dưới để copy nội dung file rru\_2t2r\_4g.dtsi từ linux source vào system-user.dtsi.

FILESEXTRAPATHS\_prepend := "${THISDIR}/files:"

SRC\_URI\_append = " \

    file://system-user.dtsi \

    "

KERNEL\_DTS\_INCLUDE\_append = " \

        ${STAGING\_KERNEL\_DIR}/arch/${ARCH}/boot/dts/rru\_2t2r\_4g \

        "

do\_configure\_append () {

    machine\_kernel\_devicetree="rru\_2t2r\_4g"

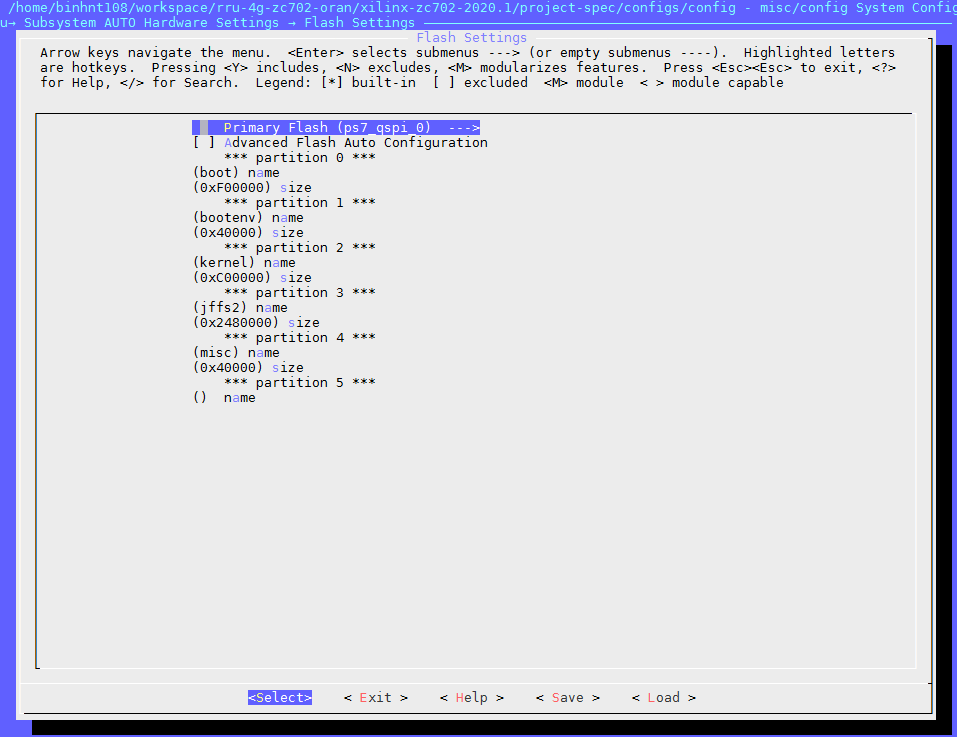
    cp ${STAGING\_KERNEL\_DIR}/arch/${ARCH}/boot/dts/rru\_2t2r\_4g/${machine\_kernel\_devicetree}.dtsi ${S}/../system-user.dtsi

}

## Chỉnh sửa Flash Memory Layout.

*Quy trình flash memory layout cần được áp dụng cả khi sử dụng các lệnh liên quan đến tftpboot, vì quy trình đó trực tiếp download và install image của U-boot, Linux kernel, rootfs,… vào các phân vùng như mình mong muốn. Nếu không chỉnh sửa trên tftpboot của U-boot => khi chạy các lệnh load và install => sai phân vùng => lỗi bootup. Ngoài ra khi lên linux kernel, việc chỉnh sửa này cũng đảm bảo thực hiện được flashcp để ghi trực tiếp image của U-boot, Linux kernel lên đúng phân vùng và đúng size (chính là tính năng updatefw của các phiên bản RRU 5G hiện tại)*

Dựa vào log quá trình boot của source 4G cũ, chỉnh sửa lại Flash Memory Layout trong Petalinux.



Hình 2.10. Chỉnh sửa memory layout trong Petalinux

Đồng thời cũng cần chỉnh sửa lại trong device-tree (<u-boot>/ arch/arm/dts/zynq-zc702.dts)

&qspi {

        #address-cells = <1>;

        #size-cells = <0>;

        flash0: flash@0 {

                compatible = "micron,n25q128";

                reg = <0x0>;

                #address-cells = <1>;

                #size-cells = <1>;

                spi-max-frequency = <50000000>;

                partition@0x00000000 {

                        label = "boot";

                        reg = <0x00000000 0x00f00000>;

                };

                partition@0x00f00000 {

                        label = "bootenv";

                        reg = <0x00f00000 0x00040000>;

                };

                partition@0x00f40000 {

                        label = "kernel";

                        reg = <0x00f40000 0x00C00000>;

                };

                partition@0x01B40000 {

                        label = "jffs2";

                        reg = <0x01B40000 0x02480000>;

                };

                partition@0x03fc0000 {

                        label = "misc";

                        reg = <0x03fc0000 0x00040000>;

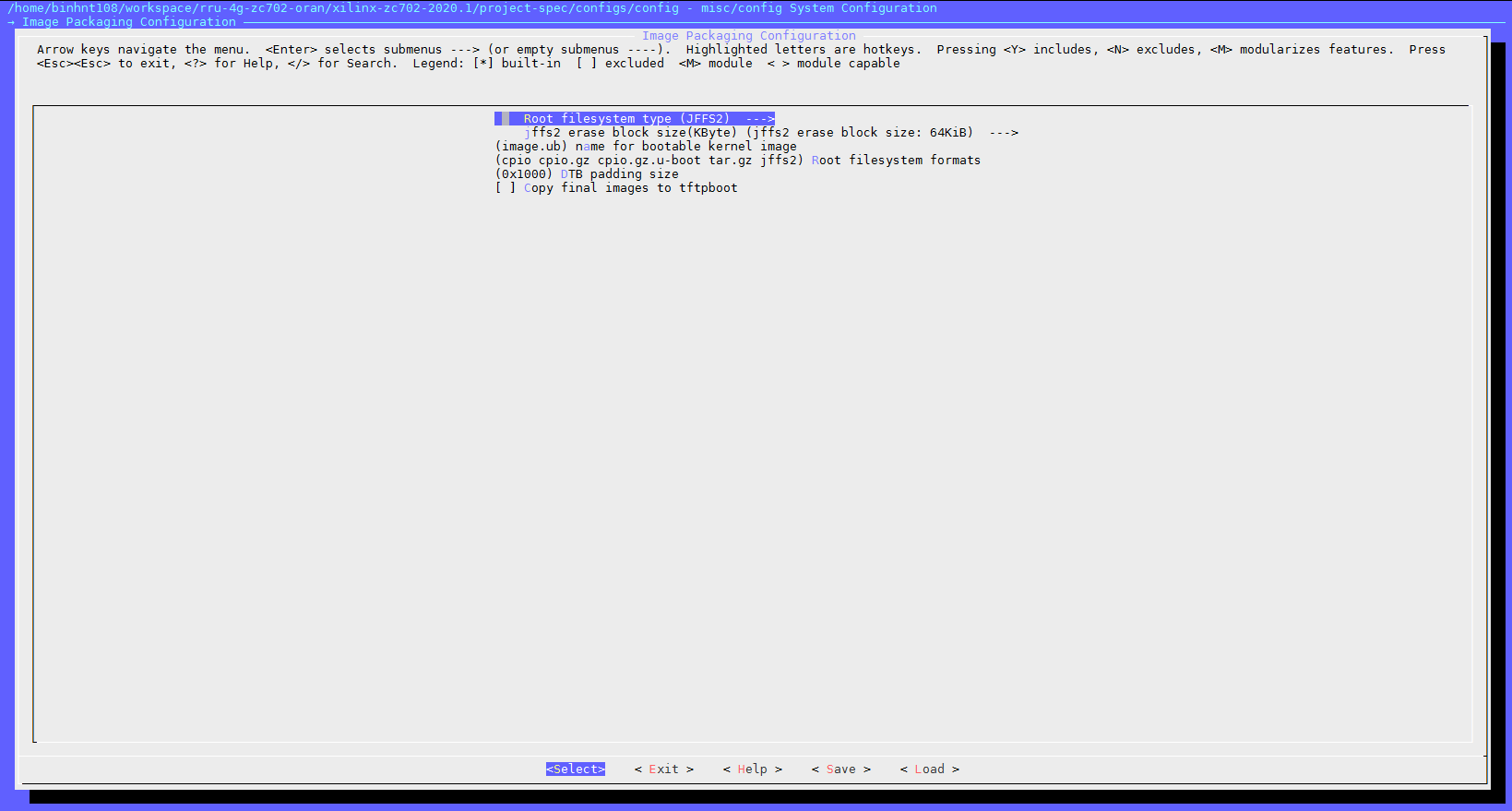
                };

        };

};

## Cấu hình RootFS.

Chỉnh sửa type cho RootFS: petalinux-config -> Image Packaging Configuration.



Hình 2.11. Cấu hình RootFS trong Petalinux

## Boot mạch.

Phương thức được sử dụng để boot mạch là tftpboot. Trong trường hợp này, server 238 (IP:192.167.120.238 hoặc 172.16.27.38) làm TFTP Server. Thực hiện các thao tác sau trong U-boot:

* run set\_tftp
* run load\_boot && run install\_boot
* run load\_kernel && run install\_kernel
* run load\_jffs2 && run install\_jffs2
* run default\_bootcmd

## Các vấn đề chưa giải quyết.

### Không hiện thị serial trên kernel

*Có thể tham khảo chỉnh sửa bbappend trên meta-rru tương ứng với phần rru-apps để ghi đè hoặc chạy process khởi tạo serial port với ttyPS0*

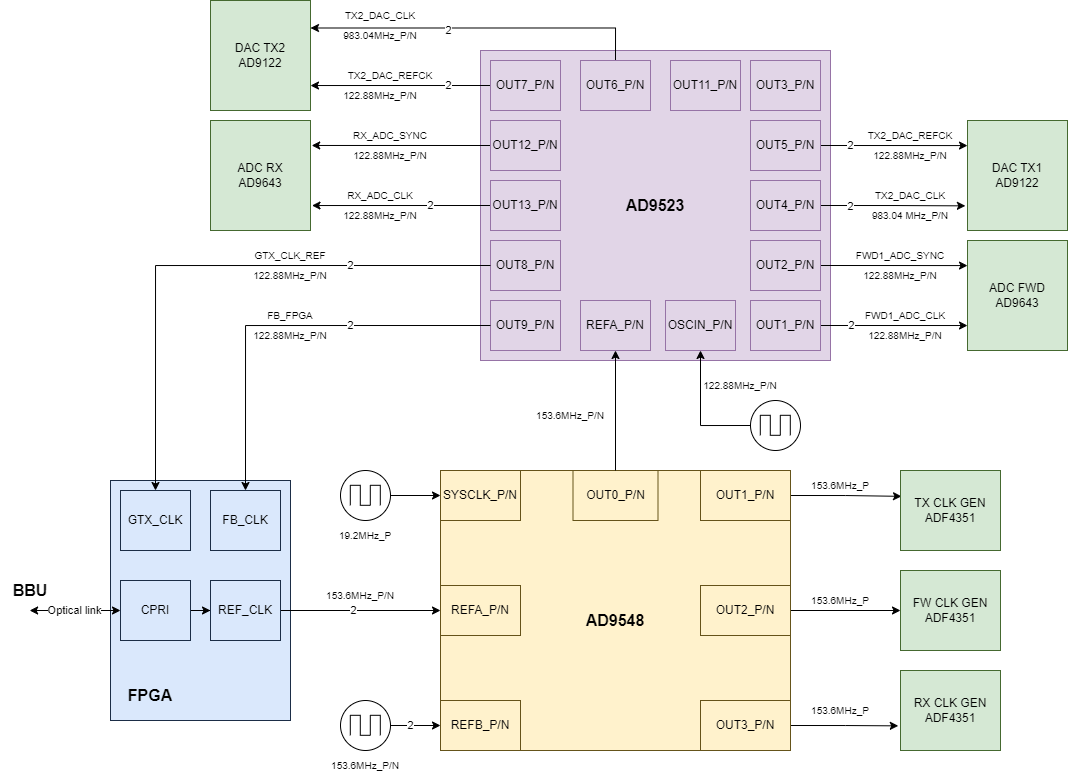
**Giải pháp (tạm thời):** Thêm đoạn sau vào file vào file /etc/inittab trên linux để cấu hình lại serial.

PS0:2345:respawn:/sbin/getty -n -L 115200 ttyPS0 vt100

# Tìm hiểu về hệ thống Clock.

## Hệ thống clock trên mạch RRU 4G 2T2R.

Hệ thống clock trên RRU 4G 2T2R được mô tả theo như hình 3.1, gồm 2 bộ tạo clock chính AD9548 và AD9523.



Hình 3.1. Sơ đồ clock trên RU 4G 2T2R

*AD9548 đóng vai trò chính là Network Synchronizer Clock, không phải vai trò chính là cleaner clock. Lý do là cung cấp thời gian chính xác, sạch để dùng cho các ứng dụng liên quan đến network (ví dụ: cung cấp tín hiệu clock cho ptp4l làm giải pháp đồng bộ giữa DU-RU, cung cấp các tín hiệu clock đầu ra có phase noise và jitter thấp, phù hợp với yêu cầu từ highspeed networking như 10G/25G cho các giải pháp CPRI/ECPRI) (Ngoài AD9548, hệ thống RRU-5G sử dụng 1 số IC clock khác như LMK05028, SI5518, đặc điểm chung đều là Network Synchronizer Clock, vai trò tương đương nhau)*

Trong đó, AD9548 đóng vai trò như là một clock cleaner để tạo ra các tín hiệu đầu ra với độ sai lệch jitter, độ nhiễu pha thấp và ổn định được tham chiếu đầu vào chính tới clock của giao tiếp CPRI tại chân REFA. AD9548 cũng có thể thực hiện chức năng switchover tới clock tham chiếu phụ (REFB) và holdover để duy trì sự ổn định của hệ thống trong trường hợp lỗi hoặc nhiễu tín hiệu tham chiếu.

AD9523 được sử dụng để làm một clock distributor để tạo ra nhiều tín hiệu clock đầu ra với các tần số cụ thể cho các phần tử khác hoạt động trên mạch.

*Các IC AD9643/AD9122/ADF4531 không phải là hệ thống clock của RU. Ví dụ AD9643 và AD9122 là các ADC/DAC để phục vụ việc chuyển đổi từ tín hiệu số sang tương tự (từ dữ liệu số sinh ra sau lớp xử lý của FPGA qua ADC chuyển thành tín hiệu tương tự, và được khuếch đại bằng PA, phát ra ngoài thông qua antenna (hay còn gọi là phía Downlink của RRU), và ngược lại, tín hiệu từ UE qua antenna đến LNA => qua DAC chuyển thành tín hiệu số, đưa vào FPGA xử lý (hay còn gọi là phía Uplink của RRU)). ?*

Bảng 3.1. Bảng các thành phần hệ thống clock.

|  |  |  |
| --- | --- | --- |
| **STT** | **IC** | **Chức năng** |
| 1 | AD9548 | Clock cleaner |
| 2 | AD9523 | Clock distributor |
| 3 | AD9643/AD9613 | ADC |
| 4 | AD9122 | DAC |
| 5 | ADF4351 | Tạo tín hiệu RF cho khối modulator |

Bảng 3.2. Các tín hiệu trong hệ thống clock.

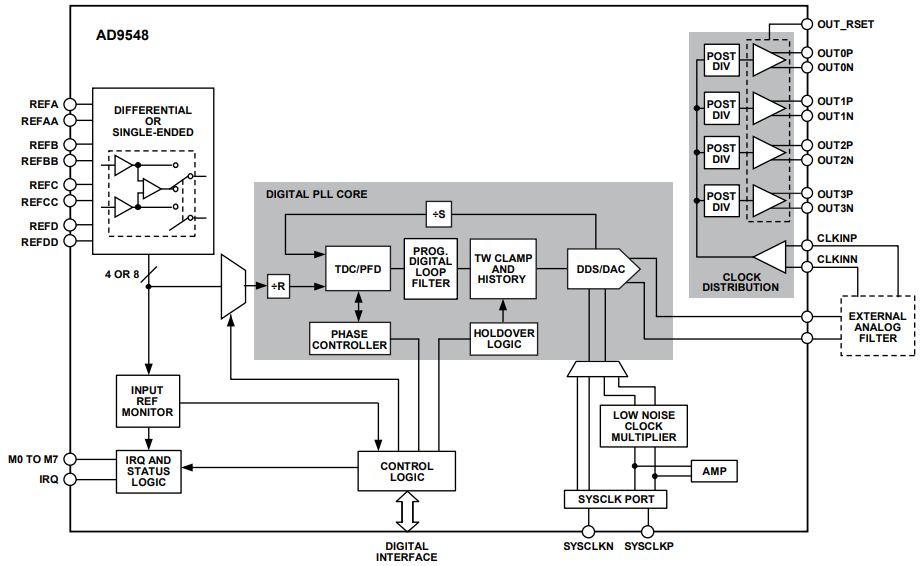
*Cần cung cấp rõ hơn giá trị của các clock trong bảng. Đối với các clock được bôi đỏ, các clock này được cung cấp từ IC nào? Clock nào cung cấp cho hệ thống CPRI của 4G chạy được? Nếu thực hiện giải pháp chuyển đổi sang E-CPRI thì cần thay đổi clock nào, giá trị bao nhiêu, thiếu clock 1PPS?*

|  |  |  |
| --- | --- | --- |
| **STT** | **Tín hiệu** | **Mô tả** |
|  | SYSCLK (AD9548) | Tạo clock hệ thống cho các phần tử bên trong AD9548 hoạt động |
|  | REFA/B (AD9548) | Clock tham chiếu |
|  | REFA (AD9523) | Clock tham chiếu |
|  | OSCIN (AD9523) | Tạo clock hệ thống cho các phần tử bên trong AD9523 hoạt động |
|  | TXx\_DAC\_CLK | Tạo clock hệ thống cho các phần tử bên trong AD9122 hoạt động |
|  | TXx\_DAC\_REFCLK | Cung cấp clock đồng bộ cho quá trình chuyển đổi DAC |
|  | RXx\_ADC\_SYNC | Clock cung cấp thông tin về điểm bắt đầu quá trình chuyển đổi ADC |
|  | RXx\_ADC\_CLK | Cung cấp clock đồng bộ cho quá trình chuyển đổi ADC |
|  | FWD1\_ADC\_SYNC | Clock cung cấp thông tin về điểm bắt đầu quá trình chuyển đổi ADC |
|  | FWD1\_ADC\_CLK | Cung cấp clock đồng bộ cho quá trình chuyển đổi ADC |
|  | GTX\_CLK | Cung cấp clock cho khối truyền thông nối tiếp GTX |
|  | FB\_FPGA |  |

## Cấu tạo tạo AD9548.

*Thiếu 1 clock rất quan trọng là 1PPS*

Để hiểu hơn về quá trình tạo và phân chia clock của một bộ clock generator, ta tìm hiểu sơ đồ khối của chip AD9548



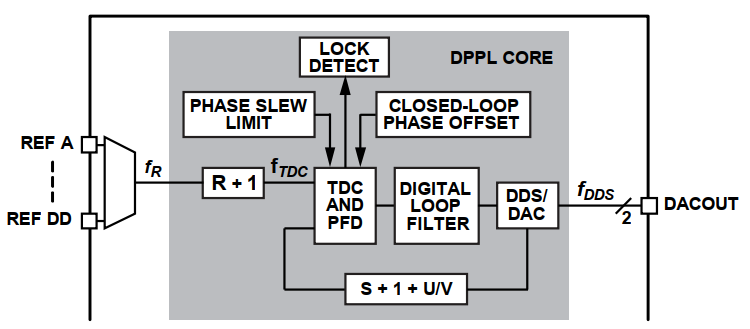
Hình 3.2. Sơ đồ khối AD9548.

Quá trình xử lý và tạo các tín hiệu clock đầu ra của AD9548 được thực hiện thông qua 3 khối chính là: Differential Input, Digital PLL và Clock Distributuion.

**Differential Input.**

Khối Differential Input chịu trách nhiệm nhận tín hiệu tham chiếu bên ngoài và chuyển đổi nó thành tín hiệu vi sai có thể được sử dụng bởi PLL. Khối này cũng bao gồm mạch để “clean” và lọc tín hiệu đến để đảm bảo nó phù hợp để sử dụng làm tín hiệu tham chiếu.

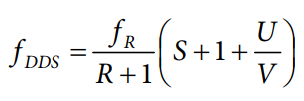
**Digital PLL.**

****

Hình 3.3. Digital PLL Core.

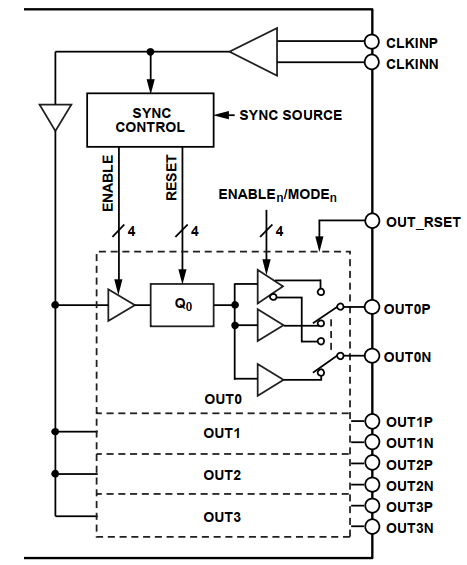
Trái tim của AD9548 là bộ PLL số. Tín hiệu tham chiếu với tần số fR được thông qua một bộ chia trước (post-divider) với tần số chia được lưu bởi 1 thanh ghi 30 bit. Sau đó tín hiệu fTDC được đưa qua TDC và PFD. TDC đo chênh lệch thời gian giữa hai tín hiệu tham chiếu và tín hiệu phản hồi bằng cách sử dụng counter đếm số chu kỳ giữa hai sườn lên hoặc sườn xuống của tín hiệu đầu vào. TDC tạo tín hiệu đầu ra số (numberic) biểu thị chênh lệch thời gian giữa hai tín hiệu. PFD so sánh pha và tần số của tín hiệu tham chiếu và tín hiệu phản hồi của DPLL và tạo ra tín hiệu lỗi được sử dụng để điều chỉnh tần số và pha của tín hiệu đầu ra. PFD hoạt động bằng cách lấy mẫu hai tín hiệu đầu vào và so sánh độ lệch pha của chúng. PFD cũng tạo ra output là một số (numberic) cho biết tín hiệu đầu ra đang nhanh hay chậm so với tín hiệu tham chiếu. Sau đó, tín hiệu sai lệch từ TDC và PFD được đưa qua bộ lọc Digital Loop Filter trước khi đưa vào làm input cho khối DAC. DAC sử dụng tín hiệu lỗi để điều chỉnh tín hiệu ra tương tự. Đầu ra tương tự này mang đặc trưng về pha và tần số tương ứng với tín hiệu clock output mong muốn.

Bộ PLL bao gồm một bộ chia phản hổi (feedback divider) với bội số là S + 1 + U/V. Trong đó, S có khoảng giá trị 30 bit và U, V có khoảng giá trị 10 bit được lưu trong thanh ghi profile. Tần số fDDS có giá trị bằng:



**Clock Distribution.**

Khối Clock Distribution tạo ra các tín hiệu đầu ra dựa trên phase và tần số của tín hiệu Sin đầu ra của PLL (hoặc từ bộ lọc tương tự ngoài) để tạo ra tín hiệu digital tại các output. Đầu ra có thể phân phối gồm 4 kênh (OUT0 – OUT3). Mỗi kênh có một bộ chia (divider) và driver chuyên biệt được mô tả như hình bên dưới



Hình 3.4. Clock Distribution

Đầu ra được được Sync Control đánh từ Q0 đến Q3 tương ứng với các kênh đầu ra là OUT0 đến OUT3. Mỗi đâu ra có bộ chia có thể lập trình được với độ lớn 30 bit. Tín hiệu đầu ra có duty cycle mặc định là 50%.

# Tham khảo

<https://docs.yoctoproject.org/2.1/ref-manual/ref-manual.html>