PetaLinux RootFS 與網路流程（含 GPIO 與 mDNS 支援）

# 1. 前置作業

在開始編譯 PetaLinux 前，需要建立專案、匯入硬體設計 (XSA)、設定 Device Tree，並在 petalinux-config 中完成基本設定。

## 建立專案

petalinux-create project --name <project-name> --template zynq # 或 zynqmp  
cd <project-name>

## 匯入硬體設計 (XSA)

petalinux-config --get-hw-description <path-to-xsa>

請記得petalinux-config若沒有特別設定如device-tree等，就不會在petalinux-build中編譯進去，下圖為官方(UG1144 2025.1)的流程圖一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

## 設定 Device Tree

petalinux-config -c device-tree

## 專案層 petalinux-config (重要)

進入 petalinux-config 主選單，必須完成以下設定：

petalinux-config

1. Image Packaging Configuration → Root filesystem type → 務必選擇 ext4  
 - 目的：確保根檔系統能正確掛載 SD 卡分割區。  
 - 否則開機可能出現找不到 rootfs 的錯誤。

2. Image Packaging Configuration → 取消勾選 Copy final images to tftpboot  
 - 一般 SD 卡開機不需要自動拷貝到 tftpboot，避免誤拷貝或權限問題。

3. Yocto Settings → 設定本機 sstate 與下載目錄（離線/加速建置）  
 - Primary sstate feed location → 指向本機解壓的 sstate-cache 路徑。  
 - Download mirror → 指向本機 yocto downloads 目錄。  
 - ⚠ 記得先到 AMD/Xilinx 官網下載對應版本 (2025.1) 的 sstate-cache，否則建置時 bitbake 會嘗試從網路下載，導致延遲或錯誤。

1. 4. 進入 **Yocto Settings → User Layers → Add**，加入下面兩個路徑

${PETALINUX}/components/yocto/layers/meta-openembedded/meta-oe

${PETALINUX}/components/yocto/layers/meta-openembedded/meta-python

${PETALINUX}/components/yocto/layers/meta-openembedded/meta-networking

# 2. RootFS 與套件安裝

額外需要的套件可透過 user-rootfsconfig 指定，或在 petalinux-config -c rootfs GUI 介面中勾選。

## user-rootfsconfig 範例

echo "CONFIG\_python3-pip" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_python3-flask" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_python-websockets" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_python-websocket-client" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_python-websocket-client" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_glibc-utils" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_emacs" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_gpio-demo" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_libgpiod-tools" >> project-spec/meta-user/conf/user-rootfsconfig  
echo "CONFIG\_peekpoke" >> project-spec/meta-user/conf/user-rootfsconfig  
這是在rootfs沒有的，可以自行加入

## petalinux-config -c rootfs 必須選的套件

1. **ldd**
   * 提供 ldd等工具，專門檢查 ELF linked library。
   * Filesystem packages-> misc->glibc
2. **binutils**
   * 提供 nm, objdump, strings 等，方便看 .so 的符號。
   * Filesystem packages-> devel->binutils
3. **gdb或gdbserver`/l**
   * 若需要進一步在板上 debug。
4. **strace**
   * 用來追蹤 openat, mmap, ioctl 等系統呼叫。
   * Filesystem packages-> console->utils->strace
5. **file**
   * 確認 ELF 類型，例如 file ./libawg\_core\_mmap.so。
   * Filesystem packages-> console->utils->file
6. **coreutils / procps**
   * 基本系統診斷工具，像 ps, top, which。
   * Filesystem packages-> misc->coreutils
7. net-tools
   * 提供 ifconfig, route 等傳統網路工具
   * Filesystem packages-> misc->net-tools
8. iproute2
   * 提供 ip 指令ip addr, ip link 等
   * Filesystem packages-> base->iproute2
9. avahi-daemon
   * mDNS 支援
   * Filesystem packages-> network->avahi
10. python3-core
    * Python 執行環境
    * Filesystem packages-> misc->python3
11. Openssh
    * SSH server/client，用於遠端登入
    * Filesystem packages-> console -> network -> openssh
12. git
    * git 環境
    * Filesystem -> console -> utils -> git
13. Ssh-server-openssh
    * openssh-server
    * Image features-> ssh-server-openssh
14. gdb
    * GNU偵錯器
    * Filesystem -> misc -> gdb

在Image Features中Init-manager修改為systemd（因為用systemd才會有machine-id）

# 3. 自訂功能擴充 (.bbappend)

在 meta-user 中新增 .bbappend 檔案，用於修正系統啟動行為。

1. 清空 /etc/machine-id → 首次開機時由 systemd 產生唯一 machine-id

2. systemd .link 設定 → 強制所有乙太網卡使用 MACAddressPolicy=persistent

3. 覆蓋 avahi-daemon.conf → 限定 IPv4、允許發布 hostname，方便 mDNS 使用

## 3.1 建立資料夾結構

cd <project>/project-spec/meta-user  
# base-files（machine-id 與 nsswitch.conf 覆寫）  
mkdir -p recipes-core/base-files

mkdir -p recipes-core/base-files/files  
# systemd-conf（persistent MAC 的 .link 檔）  
mkdir -p recipes-core/systemd/systemd-conf/files  
# avahi（mDNS 設定覆寫）  
mkdir -p recipes-daemons/avahi/avahi/files  
# （可選）image（若用 bbappend 方式安裝套件）  
mkdir -p recipes-core/images

## 3.2 base-files.bbappend（清空 machine-id 並加入 mdns 解析）

檔案：project-spec/meta-user/recipes-core/base-files/base-files\_%.bbappend

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"  
SRC\_URI += file://99-gpio.rules  
# 用 extrausers 管理帳號/群組，避免直接改 /etc/group  
inherit extrausers

do\_install:append() {  
 # 1) 首開由 systemd 重新生成 machine-id（build 階段清空）  
 if [ -f ${D}${sysconfdir}/machine-id ]; then  
 : > ${D}${sysconfdir}/machine-id  
 fi  
 rm -f ${D}/var/lib/dbus/machine-id || true

# 2) 讓 .local 解析（已有則不動）  
 if [ -f ${D}${sysconfdir}/nsswitch.conf ]; then  
 sed -i 's/^\(hosts:.\*\) dns\(.\*\)$/\1 mdns4\_minimal [NOTFOUND=return] dns\2/' \  
 ${D}${sysconfdir}/nsswitch.conf || true  
 fi

# 3) 安裝 udev 規則（讓 /dev/gpio\* 屬於 gpio 群組）  
 install -d ${D}${sysconfdir}/udev/rules.d  
 install -m 0644 ${WORKDIR}/99-gpio.rules ${D}${sysconfdir}/udev/rules.d/99-gpio.rules

}

# 目標機開機後再保險：若因任何因素少了群組/成員，補上並 reload udev

pkg\_postinst:${PN}() {  
 if [ -z "$D" ]; then  
 (getent group gpio >/dev/null 2>&1) || \  
 (addgroup gpio 2>/dev/null || groupadd gpio 2>/dev/null || true)  
 (getent passwd daemon >/dev/null 2>&1) && \  
 usermod -a -G gpio daemon 2>/dev/null || true  
 command -v udevadm >/dev/null 2>&1 && {  
 udevadm control --reload 2>/dev/null || true  
 udevadm trigger --subsystem-match=gpio 2>/dev/null || true  
 }  
 fi  
}

增加60-gpio.rules內容

project-spec/meta-user/recipes-core/base-files/files/99-gpio.rules

ACTION=="add|change", SUBSYSTEM=="gpio", KERNEL=="gpiochip\*", GROUP="gpio", MODE="0660"

## 3.3 systemd-conf.bbappend + .link（所有乙太網卡啟用 persistent MAC）

請開機後修改檔案/etc/systemd/network/10-end0.link

[Match]

OriginalName=end0

[Link]

MACAddress=02:ab:cd:12:34:56

## 3.4 avahi.bbappend + avahi-daemon.conf（mDNS 設定）

檔案：project-spec/meta-user/recipes-daemons/avahi/avahi\_%.bbappend

# project-spec/meta-user/recipes-daemons/avahi/avahi\_%.bbappend

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

# 只帶 ssh.service 這個檔案

SRC\_URI += "file://ssh.service"

do\_install:append() {

install -d ${D}${sysconfdir}/avahi/services

install -m 0644 ${WORKDIR}/ssh.service \

${D}${sysconfdir}/avahi/services/ssh.service

}

# 關鍵：append 到現有 avahi-daemon 的 FILES，而不是覆蓋

FILES:avahi-daemon:append = " ${sysconfdir}/avahi/services/ssh.service"

檔案: project-spec/meta-user/recipes-daemons/avahi/files/ssh.service

<?xml version="1.0" standalone='no'?>  
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">  
<service-group>  
  <name replace-wildcards="yes">%h SSH</name>  
  <service>  
    <type>\_ssh.\_tcp</type>  
    <port>22</port>  
  </service>  
</service-group>

## 3.5 將帳號建立放在image

project-spec/meta-user/recipes-core/images/petalinux-image-minimal.bbappend

# 1) 確保基礎帳號/群組與工具都在  
IMAGE\_INSTALL:append = " base-passwd shadow "

# 2) 用 extrausers 在 rootfs 階段直接寫入 /etc/group、/etc/passwd  
inherit extrausers  
EXTRA\_USERS\_PARAMS += "\  
 groupadd -g 1001 gpio; \  
 usermod -a -G gpio daemon; \  
usermod -a -G gpio petalinux; \  
"

**3.6 （可選）image bbappend：把套件安裝進映像檔案：project-spec/meta-user/recipes-core/images/petalinux-user-image.bbappend（或你的實際 image 名稱）**

IMAGE\_INSTALL:append = " avahi-daemon python3-pip python3-flask libgpiod-tools"

# 4. 編譯與封裝

4.1 petalinux-build

petalinux-build -c base-files

petalinux-build

4.2 產出 boot 及 rootfs：  
 petalinux-package --boot --fsbl images/linux/zynq\_fsbl.elf --u-boot --fpga images/linux/system.bit --force

4.3 產出SD image

petalinux-package wic

產出的image在 image/linux下

4.4如果建立後，需要清除可依下列程序

# 針對你有改過的 base-files recipe，強制清理

petalinux-build -c base-files -x cleansstate

# 若要保險，把 rootfs 與 image 的 sstate 也清掉

petalinux-build -c rootfs -x cleansstate

petalinux-build -c petalinux-user-image -x cleansstate

# （可選）遇到怪異殘留才用最重的清理

# petalinux-build -x mrproper

4.5檢查 rootfs 是否有規則檔與群組（測試用）：

mkdir -p mnt

sudo mount -o loop images/linux/rootfs.ext4 mnt

# 檢查 udev 規則

ls -l mnt/etc/udev/rules.d/60-gpio.rules

# 檢查 group（若使用了 pkg\_postinst 方案）

grep '^gpio:' mnt/etc/group || echo "（group 將於首次開機由 postinst 建立）"

**4.7在 PetaLinux 中正確使用 .bbappend 的做法與注意事項**

**.bbappend**

**檔名必須正確對應**

* .bbappend 的檔名必須 **完全對應到原始 recipe 名稱**。

例如：

* + 修改 **minimal** → petalinux-image-minimal.bbappend
  + 修改 **full** → petalinux-image-full.bbappend
* 錯誤案例（無效）：

petalinux-user-image.bbappend

因為 Yocto 尋找不到對應的 base recipe。

正確位置應該在：

project-spec/meta-user/recipes-core/images/

檔案範例：

project-spec/meta-user/recipes-core/images/petalinux-image-minimal.bbappend

### bbappend

### 範例內容

用來移除不需要的字型、避免 update\_font\_cache 出錯：

# 移除字型相關套件

IMAGE\_INSTALL:remove = "fontconfig-utils fontconfig liberation-fonts cantarell-fonts"

# 避免 Yocto 自動拉進來

BAD\_RECOMMENDATIONS += "fontconfig-utils fontconfig liberation-fonts cantarell-fonts"

如果有udev-hwdb的問題

直接把 hwdb 從 image 移除（你大多不需要它）

1. 建/改你的 image .bbappend（你用 minimal）

project-spec/meta-user/recipes-core/images/petalinux-image-minimal.bbappend

# 務必同時擋新舊名字

IMAGE\_INSTALL:remove = " systemd-hwdb udev-hwdb "

BAD\_RECOMMENDATIONS += " systemd-hwdb udev-hwdb "

# 保險：就算哪層又塞進來，也不要在 rootfs 階段跑該 hook

ROOTFS\_POSTPROCESS\_COMMAND:remove = " update\_udev\_hwdb"

ROOTFS\_POSTINSTALL\_COMMAND:remove = " update\_udev\_hwdb"

1. 乾淨重建（避免吃舊 cache

petalinux-build -c rootfs -x cleansstate

petalinux-build

4.8 如要使用 bitbake-layers指令

# # 1. 確保您在 Petalinux 專案的根目錄

# cd /home/yclin/wavegen\_z7

# 2. 取消可能干擾的動態庫路徑（UG1144 第 4 步）

unset LD\_LIBRARY\_PATH

# # 3. Source 標準 Yocto 環境

# source components/yocto/layers/poky/oe-init-build-env

# # 4. 【關鍵】手動匯出 Petalinux 的專案根目錄變數

# # 請將路徑替換成您自己的專案根目錄

# export PROOT="/home/yclin/wavegen\_z7"

# # 5. 【關鍵】將 PROOT 加入到 BitBake 的環境變數白名單

# export BB\_ENV\_PASSTHROUGH\_ADDITIONS="${BB\_ENV\_PASSTHROUGH\_ADDITIONS} PROOT"

# # 6. 現在您可以自由地執行 bitbake 指令了，不會再有 ${PROOT} 錯誤

# bitbake-layers show-appends

# 5. GPIO 權限設定（已在.bbappend中設定，這一段只是參考）

為了讓 Web (Flask/daemon) 使用者可存取 /dev/gpiochip\*，需要設定群組與 udev 規則。

# 新增群組與加入使用者  
addgroup gpio  
usermod -a -G gpio daemon  
  
# 建立 /etc/udev/rules.d/60-gpio.rules  
ACTION=="add|change", SUBSYSTEM=="gpio", KERNEL=="gpiochip\*", GROUP="gpio", MODE="0660"  
  
# 套用規則  
udevadm control --reload  
udevadm trigger --subsystem-match=gpio

# 6. 網路與服務啟動

1. 使用 mDNS (Avahi) → 系統可用 `<hostname>.local` 直接連線

2. Flask Web 服務常駐 → 可建立 systemd service 啟動 awg\_server.py

# 7. 驗證流程

## 系統啟動後驗證

# 檢查 IP 與 MAC  
ip addr  
ip link show  
  
# 確認 avahi-daemon 狀態  
systemctl status avahi-daemon  
  
# 確認 GPIO 權限  
ls -l /dev/gpiochip\*

## AWG 驗證

# 用 curl 測試  
curl -X POST http://<IP>:5000/apply -H "Content-Type: application/x-www-form-urlencoded" -d "idxA=447,448,449,450,451,452,453,454&gainA\_f=0.25,0.25,0.25,0.25,0,0,0,0"  
  
# 用前端 index.html + Flask 測試控制介面

/home/root/awg/

├── awg\_core.py # 控制硬體的邏輯

├── awg\_server.py # Flask server

├── www/ # 放靜態檔案 (CSS, JS, 圖片等)

│ └── index.html # 前端控制頁面

└── templates/ # (如果將來用 Jinja2 模板，現在可以先不用)

# /etc/systemd/system/awg.service

[Unit]

Description=AWG Flask Server

After=network.target

[Service]

ExecStart=/usr/bin/python3 /home/petalinux/awg/awg\_server.py

WorkingDirectory=/home/petalinux/awg

Restart=always

User=daemon

Group=gpio

[Install]

WantedBy=multi-user.target

systemctl daemon-reload

systemctl enable awg.service

systemctl start awg.service

# 8.在 PetaLinux 目標板上編譯 .so

# 產生共享函式庫（不需要 libgpiod）

gcc -O3 -fPIC -shared -o libawg\_core\_mmap.so awg\_core\_mmap.c

A) 臨時用環境變數（最簡單）

export LD\_LIBRARY\_PATH="$PWD:$LD\_LIBRARY\_PATH"

gcc -O3 -o test\_awg test\_awg.c -L. -lawg\_core\_mmap

./test\_awg

B) 重新連結時內嵌 rpath（推薦，之後不用設環境變數）

gcc -O3 -o test\_awg test\_awg.c -L. -lawg\_core\_mmap -Wl,-rpath,'$ORIGIN'

# 之後直接

./test\_awg

### 2. 編譯時決定是否啟用 Debug

* **啟用 Debug**（有輸出）：

**gcc -O3 -o awg\_server awg\_server\_raw\_mmap.c -L. -DDEBUG -lawg\_core\_mmap -Wl,-rpath,'$ORIGIN'**

* **關閉 Debug**（無輸出）：

**gcc -O3 -o awg\_server awg\_server\_raw\_mmap.c -L. -lawg\_core\_mmap -Wl,-rpath,'$ORIGIN'**

**可用gdm**

gcc -g -pthread -Wall -DDEBUG -o awg\_server     awg\_server\_raw\_top.c     awg\_server\_raw\_direct.c     awg\_server\_raw\_queue.c     awg\_server\_raw\_notify.c     awg\_core\_mmap.c

正式環境

**gcc -pthread -Wall -O2 -o awg\_server awg\_server\_raw\_top.c awg\_server\_raw\_direct.c awg\_server\_raw\_queue.c awg\_server\_raw\_notify.c awg\_core\_mmap.c**

**2.** 建立服務檔 /etc/systemd/system/awg\_server.service：

[Unit]

Description=AWG TCP Server

After=network.target

[Service]

ExecStart=/home/petalinux/awg\_server

WorkingDirectory=/home/petalinux

Restart=always

RestartSec=2

User=root

Group=root

[Install]

WantedBy=multi-user.target

執行 systemctl enable awg\_server 開機時啟用

或是 systemctl start awg\_server 直接執行

如要停止可以使用 systemctl stop awg\_server

50ms, Python flask + POST + python gpio

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 電路 的圖片

AI 產生的內容可能不正確。

33ms Remote, python udp client, Python udp + c mmap

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 編輯 的圖片

AI 產生的內容可能不正確。

20ms Remote, python ws client, Python web socket + python gpio

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 編輯 的圖片

AI 產生的內容可能不正確。

22ms Remote, python ws client, Python websocket + c mmap

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 軟體 的圖片

AI 產生的內容可能不正確。

6ms In local , Python udp server + c mmap + python udp client

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 編輯 的圖片

AI 產生的內容可能不正確。

1.7ms In local Pure C libgpiod

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 編輯 的圖片

AI 產生的內容可能不正確。

1.5ms In local , Python ws server + c mmap + python ws client

一張含有 螢幕擷取畫面, 多媒體軟體, 文字, 繪圖軟體 的圖片

AI 產生的內容可能不正確。

300µs Pure C mmap and /dev/mem

一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 編輯 的圖片

AI 產生的內容可能不正確。