Proyecto 1: Proceso de arranque de un sistema embebido BSP y entorno de desarrollo

Nombre: Ing. Luis Carlos Alvarez Mata | Carne: 200902144
Nombre: Ing. Verny Moreles Soto | Carne: 201116212

Durante el proyecto, aprendio cuál es el proceso de arranque común de un dispositivo emebido y el caso específico de Raspberry Pi 4 (RPI4), además, se utiliza las herramientas de compilación cruzada Linaro y utilizando una distribución del Proyecto Yocto.

1. Respuesta de preguntas

1.1. ¿Qué es un device tree, por qué se introdujo y por qué es tan importante?

Un "device tree" es una estructura de datos que describe los componentes de hardware de una computadora en particular para que el kernel del sistema operativo puede utilizar y los pueda administrar. Alguna información que se puede encontrar en el devicetree es la cantidad de cores que tiene el sistema, cuanta memoria, cual es la dirección donde está mapeada la memoria, cantidad de buses y que dispositivos hay conectados a los buses. El OS necesita esta información para saber cuales drivers necesita cargar. Es de suma importancia para algunas arquitecturas el devicetree ya que el kernel lo necesita para saber que hardware se está utilizando [1].

1.2. ¿Qué es un device tree overlay, qué se puede controlar con ella?

Es el formato que manejan los desarrolladores para incorporar los ficheros y ser incluidos en el kernel. Cuando el compilador parsea el primer fichero irá haciéndolo de manera ordenada, uno por uno. A Esto se le llama overlay, el cual habilita ingresar librerías, y ficheros de manera dinámica agregando nodos y realizando cambios al árbol existente. En este proyecto el agregar nuevas recetas de la manera en que lo hicimos con los meta layers es un ejemplo de este tipo de topología y manera de agregar datos a una imagen ya existente. Por ejemplo, la raspberry pi permite crear layers alternas con componentes que no siempre se incorporan utilizando lo que se llama como un devicetree parcial, y posteriormente se crea el devicetree master, el cual está compuesto por todas las ramas agregadas por medio de los overlays [2].

1.3. ¿Cuáles son los processor fuses y las bootstrap resistors y qué exactamente del proceso de arranque se puede controlar con ellos?

Los fuses son básicamente interconexiones que existen entre el core y todo el ensamblaje principal (cache, uncore, restos de cores, SA, entre otros). Los fuses se programan una única vez en la etapa de desarrollo, esto permite al desarrollador bloquear o activar ciertas regiones del procesador lo que permite segmentar los procesadores y dividirlos en función a sus características disponibles. Esto permite colocar diferentes precios al mismo procesador, ya que algunos van a tener mayores capacidades y atributos que otros dependiendo que fuses se encuentren habilitados [3].

Un circuito de bootstrap es aquel en el que parte de la salida de una etapa de amplificador se aplica a la entrada, para alterar la impedancia de entrada del amplificador. Cuando se aplica deliberadamente, la intención suele ser aumentar en lugar de disminuir la impedancia. En el dominio de los circuitos MOSFET, "bootstrapping"se usa comúnmente para significar levantar el punto de operación de un transistor por encima del riel de la fuente de alimentación para habilitar alguna configuración. A nivel de un procesador un bootstrap resistor se utiliza para

habilitar configuraciones de inicio o caracteristicas las cuales estan ligadas al hardware y son controladas pr medio de una impedancia especifica. [4, 5, 6].

Un ejemplo de esta implementacion se muestra en el circuito de selección para la configuración de un Half-Bridge, donde un bootstrap resistors se utilizan para la configuración de arranque en configuraciones para suministrar la polarización al high-side FET. Figura **??** muestra la ruta de carga de un circuito de arranque en una configuración simplificada. Cuando el low-side FET está encendido (el high-side FET está apagado), el pin HS y el switch nodo se tiran a tierra; el suministro de polarización VDD, a través del bypass capacitor, carga el capacitor de arranque a través del diodo de arranque y la resistencia [7].

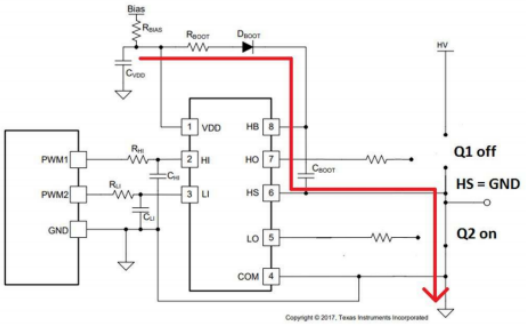

Figura 1: Bootstrap de carga. [7]

Cuando se apaga el low-side FET y se enciende el high-side, el pin HS del controlador de puerta y el switch nodo se llevan al bus de alto voltaje HV; el bootstrap capacitor descarga parte del voltaje almacenado (acumulado durante la secuencia de carga) al high-side FET a través de los pines HO y HS del controlador de puerta como se muestra en la Figura 15 [7].

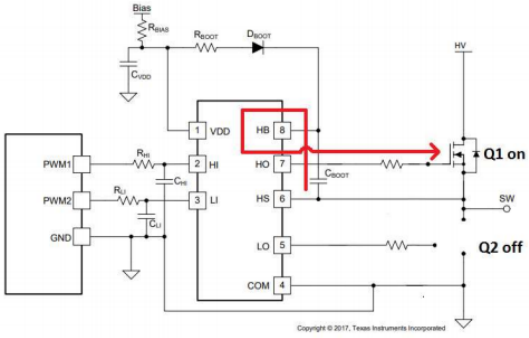

Figura 2: Bootstrap de descarga. [7]

Los procesadores ARM Cortex-M4 utilizan straps para cambian la funcionalidad de ciertos pines donde por ejemplo por medio del pin "GPIO104.es utilizado para para determinar el nivel de voltaje para la interfaz de Flash compartido. Tambien se utilizan straps para la configuración de las señales JTAG utilizadas para la depuración [8].

1.4. Investigar la ubicación predeterminada en la RAM del árbol de dispositivos en el momento del arranque para el RPI4 y busque información sobre las direcciones RAM reservadas que U-Boot usa para operar.

La direccion del device tree usado en nuestra configuracion va desde 0x3af499b0 hasta 0x3af56dc1, siendo esta la ubicación por default

La mayoria de espacios de memoria reservados se encuentra en el archivo config.txt donde se pueden encontrar los parámetros de configuración para video y el Kernel de Linux (direcciones de carga, device tree, UART / velocidades

de transmisión de la consola, etc.).

El kernel es cargado en la dirección 0x8000 y los sistemas basados en linux tienen un tamaño alrededor de 64MB, por lo cual este fragmento de la memoria será reselvado para el kernel.

Sobre NFS

¿Se carga todo el sistema de archivos en la RAM cuando monta un NFS? Si no, ¿cuál es el criterio seguido por el Kernel de Linux para cargar archivos del sistema de archivos en la RAM?

No. El sistema de archivos es almacenado en la maquina virtual, en el folder llamando rootfs el cual fue configurado previamente en los pasosl YOCTO. El kernel utiliza el folder rootfsçomo si fuera un disco de almacenamiento, por lo tanto el criterio para la carga de archivos no tiene ninguna variación con respecto a otro kernel de linux.

¿Qué pasa si usamos un Initrd? ¿Se copiaría completamente en la RAM en el momento del arranque?

Si, y para lograr esto utiliza los sigientes pasos:

1. El boot loader carga el kernel en la RAM
2. El kernel convierte initrd en un disco RAM "normalz libera la memoria utilizada por initrd
3. El dispositivo raíz está montado. si es /dev/ram0, la imagen initrd se monta como root
4. /sbin/init se ejecuta.
5. init monta el sistema de archivos raíz real"
6. init coloca el sistema de archivos raíz en el directorio raíz utilizando la llamada al sistema raíz$_p$*iroinitejecutael*/*sbin*/*initenel*
7. se elimina el sistema de archivos initrd

¿Cuáles son los beneficios y los casos de uso de ambos?

NFS nos da la ventaja que se pueden cargar sistemas de archivas de un tamaño mayor a la RAM, ademas nos da ta ventaja de poder tener el sistema de archivos de forma remota. Pero tiene la desventaja que la transferencia de datos esta sujeta al ancho de banda de la red.

Initrd nos da la ventaja de que la transferencia de datos es mas rápida, pero se ve limitado por el tamaño del la RAM

2. Resultados

Generación de la receta de Yocto

1. Creación de la carpeta meta-tec.

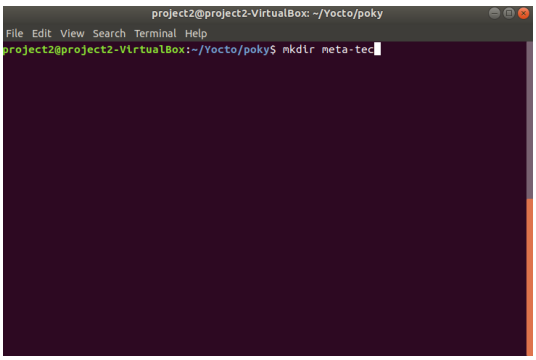

Figura 3: mkdir meta-tec.

2. Posterirormente se copia la carpeta files la cual fue dada por los profesores para utilizar el autotools. En la siguiente imagen se observa una imagen de la carpeta original, sin embargo, esta se va a modificar para ser utilizada por el código en C del trapezoide.

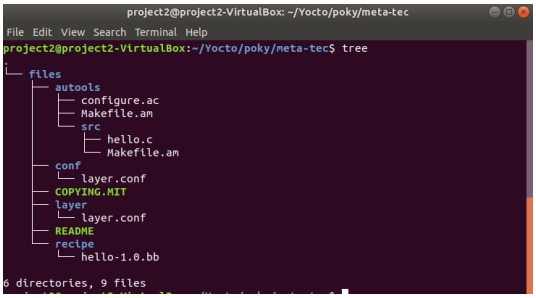

Figura 4: Carpeta File para utilizar el autotools.

3. Se crean las carpetas donde estarán las recetas correspondientes para generar el código ejecutable en C.

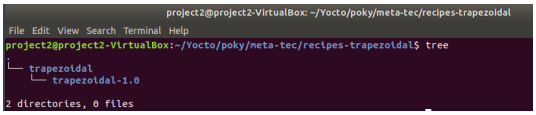

Figura 5: Jerarqía de folders para preparar la receta

4. Se procede a crear el esquema de archivos dentro de la carpeta trapezoidal-1.0 la cual se va a utilizar para compilar el coódgio C. A continuación se muestran la jerarquía de carpetas, así como los archivos Makefile.am y el configure.ac, vitales en el autotool.

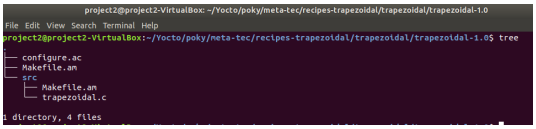

Figura 6: Jerarqía de archivos para compilar el códgio C

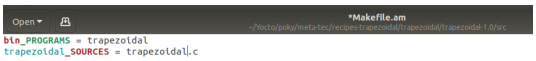

Figura 7: Archivo Makefile.am

5. Se ejecuta el autoreconf –install.

6. Se procede a correr el ./configure para crear el entorno de compilación.

Figura 8: Archivo configure.ac

Figura 9: Archivo configure.ac

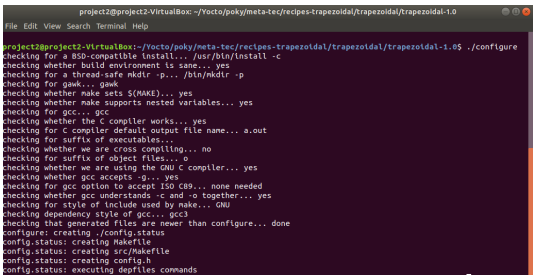

Figura 10: Entorno de compilación creado

7. Se procede a generar el binario.

Figura 11: Generación del binario

8. Generación del paquete para que Yocto entienda que esto va a ser incorporado en la imagen con el comando make distcheck.

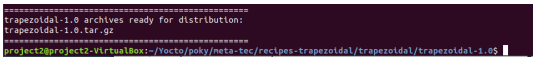

Figura 12: Compresión de la imagen

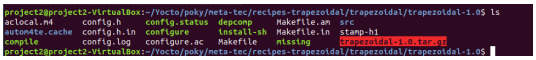

Figura 13: Imagen en el formato tar.gz.

9. Se procede a modificar el archivo layer.conf y la receta trapezoidal-1.0.bb. Posteriormente se puede observar como la jerarquía ya se encuentra como la necesitamos para generar la imagen con el autotool.

10. Se procede a correr el comando source oe-init-build-env, y posteriormente el btbake core-image-base para generar la imagen de Yocto. Posteriormente se copia en la carpeta /var/nfs/rootfs/ y está lista para ser cargada en la Raspberrypi4.

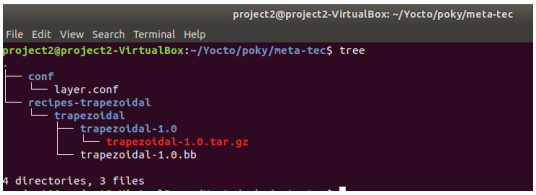

Figura 14: Esquema final para generar la imagen con autotool

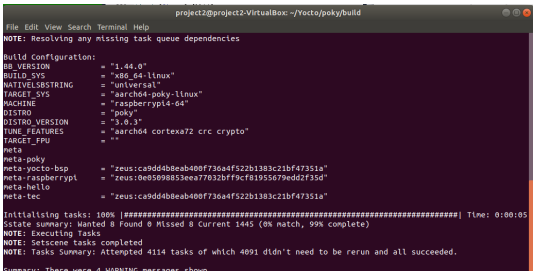

Figura 15: Imagen de Yocto creada.

Referencias

[1] Raspberry pi 4 - device tree. Library Catalog: blog.stabel.family. [Online]. Available: https://blog.stabel.family/raspberry-pi-4-device-tree/

[2] FrankBau/raspi-repo-manifest. Library Catalog: github.com. [Online]. Available: https://github.com/FrankBau/raspi-repo-manifest

[3] Intel's management engine. Library Catalog: puri.sm. [Online]. Available: https://puri.sm/learn/intel-me/

[4] "IEEE 100: the authoritative dictionary of IEEE standards terms."

[5] M. J. M. Pelgrom, *Analog-to-digital conversion*, 2nd ed. Springer, OCLC: ocn838102317.

[6] J. P. Uyemura, *CMOS logic circuit design*. Kluwer Academic Publishers.

[7] M. Diallo, "Bootstrap circuitry selection for half bridge configurations," p. 10.

[8] Microchip Technology Inc., "Cryptographic embedded controller - CEC1712." [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/CEC1712-Data-Sheet-DS00003416A.pdf