Escalabilidad de Multiplataforma sobre OpenMPI

Rafael Francisco Adorna Moreno

13 de septiembre de 2016

Agradecimientos *Me gustaría dedicar este proyecto a todas aquellas personas que han confiado en mi. En especial, a mi madre y a mi hermana por su apoyo y ayuda, mis amigos por seguir ahí incluso cuando yo no estaba y, como no, a Javier Garrigos por su paciencia a la hora de llevarlo a cabo.*

Índice

Índice de figuras

1. Resumen

Este es el documento de memoria para el proyecto final de carrera de Ingeniería Técnica Superior de Telecomunicaciones.
En este proyecto vamos a tratar de resaltar, mezclar y hacer uso de diferentes tecnologías para conseguir agrupar diferentes dispositivos bajo un mismo clúster[1] (Una agrupación de ordenadores que funcionan como un único sistema). Dichas tecnologías usadas comprenderán un rango bastante extenso y heterogéneo tales como virtualización de sistemas operativos o microprocesadores reconfigurables.
Básicamente, trataremos de configurar una FPGA[3] marca Xilinx y modelo Virtex-5 ML507[2] para que soporte un microprocesador LEON 3[4]. Luego sobre dicho sistema, instalaremos el kernel de Linux[5] para, a continuación, lograr que siempre arranque sobre una distribución de dicho kernel (Buildroot[6] para ser más concretos). A partir de ahí, instalaríamos una variante de OpenMPI[7] sobre dicho sistema operativo.

2. Tecnologías

2.1. Máquinas Virtuales

Antes de nada, deberíamos de definir qué es una máquina virtual. Una máquina virtual (VM en adelante) es una forma de poder ejecutar aplicaciones de otro sistemas operativos que, de otro modo, sería muy complicado utilizar. No es un emulador que intente aprovechar todas las características de la máquina física donde estamos trabajando, y permita ejecutar aplicaciones de otro sistema como un proceso más.
Al contrario,es un entorno virtual que simula la ejecución de una máquina genérica, configurable hasta cierto punto, dentro de nuestro sistema operativo. Dicha VM es independiente de la máquina física.

Su definición es la de "Software que simula una computadora y su sistema operativo y que puede ejecutar programas como si fuera real"[9]. Es un programa informático que crea un entorno virtual entre el sistema operativo y el hardware para que el usuario final pueda ejecutar aplicaciones en una máquina abstracta.
En muchas ocasiones surge la necesidad de probar un programa o realizar pruebas en otro sistema operativo distinto al instalado. Para ello usaremos este tipo de herramientas, configurándolo de manera que emule el sistema operativo que se quiere probar. El sistema operativo emulado debería ser totalmente independiente del sistema operativo real, conviviendo ambos en total armonía y pudiendo pasar de uno a otro con facilidad. Se pueden encontrar varios tipos de máquinas virtuales:

- **Máquinas virtuales de sistema**: Las máquinas virtuales de sistema, también llamadas máquinas virtuales de hardware, permiten a la máquina física subyacente multiplicarse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo. A la capa de software que permite la virtualización se la llama monitor de máquina virtual o hypervisor. Un monitor de máquina virtual puede ejecutarse o bien directamente sobre el hardware o bien sobre un sistema operativo ("host operating system").

- **Máquinas virtuales de proceso**: Una máquina virtual de proceso, a veces llamada "máquina virtual de aplicación", se ejecuta como un proceso normal dentro de un sistema operativo y soporta un solo proceso. La máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se detiene para cuando éste finaliza. Su objetivo es el de proporcionar un entorno de ejecución independiente de la plataforma de hardware y del sistema operativo, que oculte los detalles de la plataforma subyacente y permita que un programa se ejecute siempre de la misma forma sobre cualquier plataforma. El ejemplo más conocido actualmente de este tipo de máquina virtual es la máquina virtual

de Java.

El uso de software de virtualización permite ejecutar varias máquinas virtuales con distintos sistemas operativos sobre el mismo hardware de manera simultánea. Además estas máquinas virtuales se pueden copiar y mover a otra máquina física de manera muy sencilla, lo que proporciona una manera rápida y cómoda de hacer backups o de reutilizar máquinas existentes. Las máquinas virtuales tienen discos duros virtuales que para la máquina anfitriona (es decir, para la máquina real o "host") son simplemente ficheros de datos que se pueden copiar y manejar. La propia máquina virtual no es mas que un fichero de configuración.
La ventaja de este tipo de software es, obviamente, la flexibilidad que aportan. Por otro lado, su mayor desventaja suele ser que introducen una gran carga de trabajo en el ordenador anfitrión, pudiendo ralentizarlo hasta un punto crítico.
De entre las múltiples tecnologías disponibles, nos vamos a decantar por el uso de la tecnología VMWare[8] para gestionar nuestra máquina virtual y, más concretamente, de la aplicación VMWare Workstation Player.
Esta decisión se basa, entre otros factores, en que es un producto gratuito sin soporte que permite la emulación en plataformas x86, esto permite que cualquier usuario con una computadora de escritorio o portátil pueda emular tantas máquinas virtuales como los recursos de hardware lo permitan. Esta versión es una aplicación que se instala dentro de un sistema operativo (host) como un programa estándar, de tal forma que las máquinas virtuales corren dentro de esta aplicación, existiendo un aprovechamiento restringido de recursos.

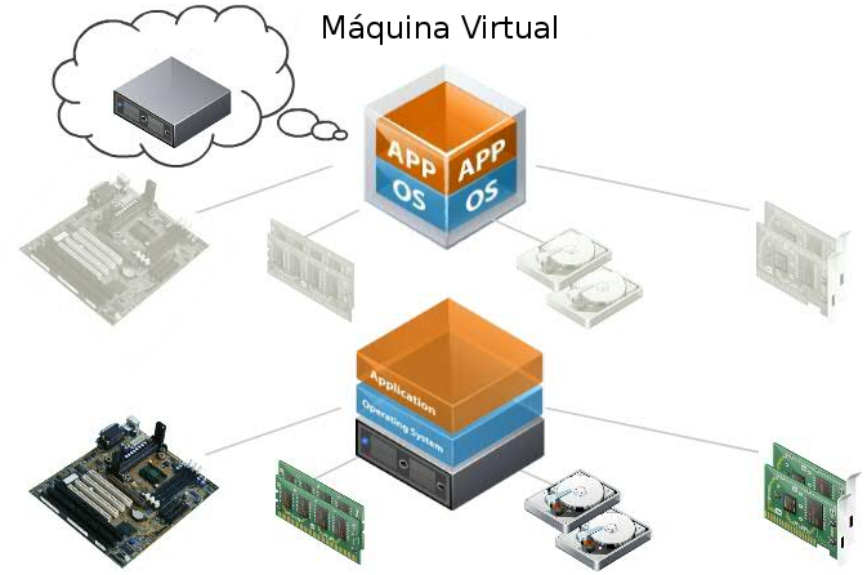

Figura 1: Máquinas Virtuales: Definición.

2.2. Leon3

El LEON3 es un modelo VHDL de un procesador de 32 bits sintetizable que es compatible con la arquitectura SPARC V8. Dicho modelo es áltamente configurable y está pensado sobre todo para los diseños SoC (System-on-a-Chip o sistema en un chip [35]). Todo el código fuente está disponible bajo la licencia GNU GPL, permitiendo un uso gratuito y libre para la educación y la investigación. Cuenta, entre otras, con las siguientes características[36]

- Arquitectura SPARC V8.
- Arquitectura Hardvard con un caché para instrucciones y datos separado.
- Áltamente configurable.
- Posibilidad de usar una versión para condiciones espaciales.
- Capacidad multiprocesador.
- Gran cantidad de herramientas disponibles para su configuración y depuración.

El procesador LEON3 es completamente parametrizable a través del uso de VHDLs genéricos. Por lo tanto, es posible instanciar varios núcleos en el mismo diseño pero con distintas configuraciones. Los modelos de diseño incluidos pueden ser configurados usando la línea de comandos o una interfaz gráfica construída a dicho fin. Esto permite que nuevos usuarios puedan configurar rápidamente sus procesadores así como otros periféricos auxiliares tales como controladores de memoria o interfaces de red.

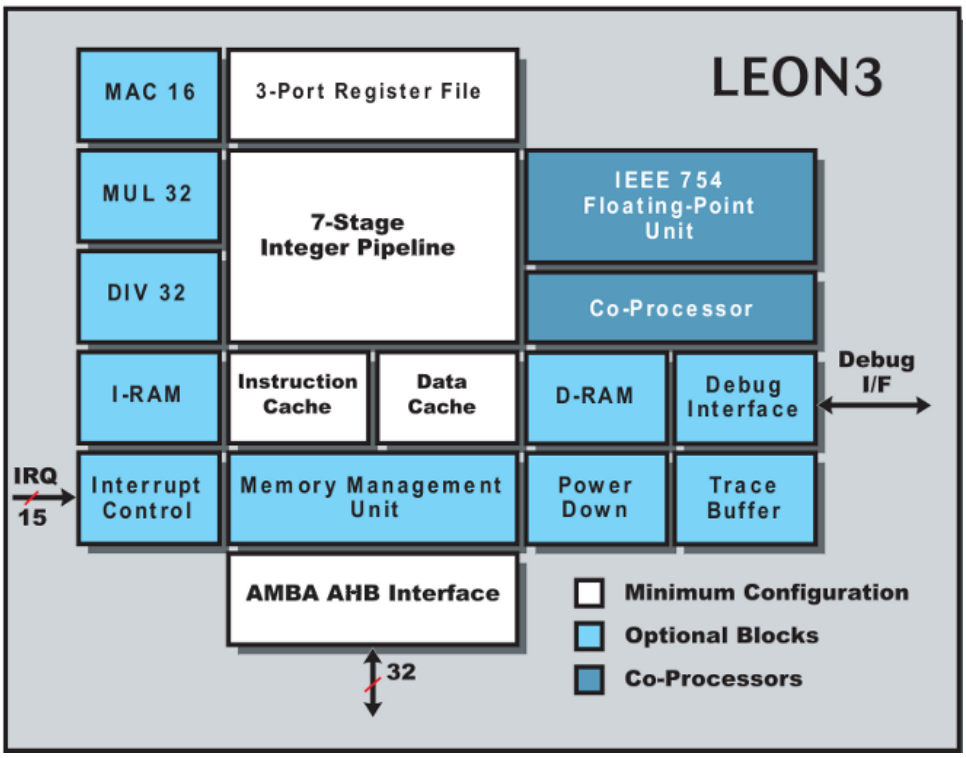

Figura 2: LEON3: Arquitectura.

Dicho modelo de procesador también puede ser sintetizado usando diferentes aplicaciones, en especial la herramienta Xilinx ISE[37]. En nuestro caso, el uso de esta tecnología vino como una imposición de nuestro proyecto: Queríamos explorar las capacidades que dicha tecnología nos ofrecía para la creación de diferentes sistemas embebidos, concretamente aquellos que pudieran ser conectados por una red Ethernet y soportaran una instanciación de este tipo de microprocesador.
Como dato extra, dejaremos constar de que hay una gran comunidad de desarrolladores alrededor de este proyecto, involucrados en dar soporte y movimiento al mismo[39]. No es de extrañar que el propio Gaisler responda a preguntas de los usuarios y el tiempo de respuesta se puede medir en horas, en vez de en días como en otros proyectos.

2.3. GNU/Linux

Antes de comenzar, intentaremos establecer una diferencia, entre los términos Linux y GNU/Linux, a fin de saber de qué estamos hablando con exactitud, en cada caso.

Linux es un kernel, es decir, el núcleo de un sistema operativo, mientras que GNU/Linux, el sistema operativo que utiliza el kernel Linux como núcleo, fue creado, y es difundido y promovido a través del Proyecto GNU, por la Free Software Foundation, organización sin fines de lucro, fundada por Richard Stallman.

A la versión de GNU ampliamente utilizada hoy en día se la llama a menudo «Linux», y muchos de quienes la usan no se dan cuenta de que básicamente se trata del sistema GNU.

Efectivamente existe un Linux, pero constituye solo una parte del sistema que utilizan. Linux es el núcleo: el programa del sistema que se encarga de asignar los recursos de la máquina a los demás programas que el usuario ejecuta. El núcleo es una parte esencial de un sistema operativo, pero inútil por sí mismo, sólo puede funcionar en el marco de un sistema operativo completo. Linux se utiliza normalmente en combinación con el sistema operativo GNU: el sistema completo es básicamente GNU al que se le ha añadido Linux, es decir, GNU/Linux. Todas las distribuciones denominadas «Linux» son en realidad distribuciones GNU/Linux.

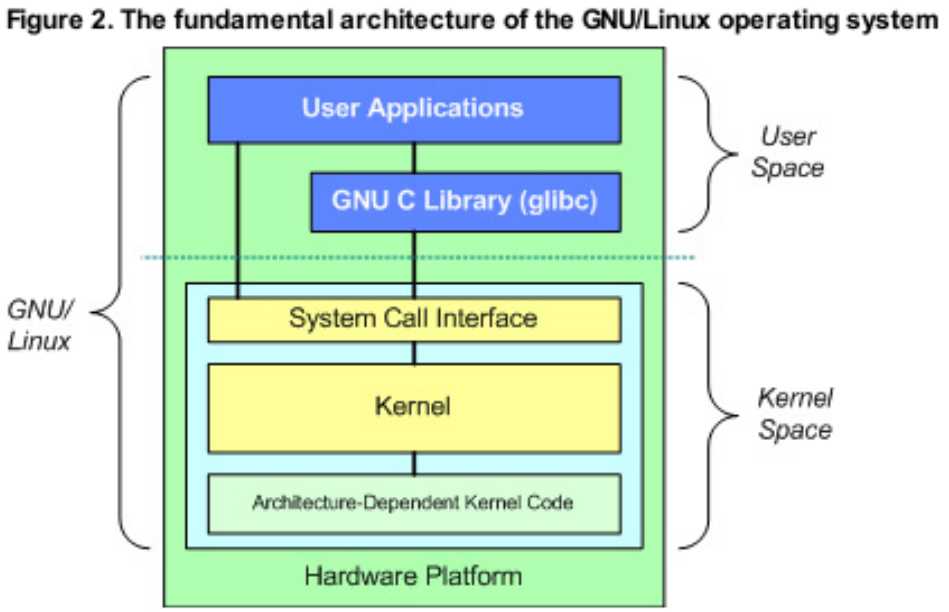

Figura 3: GNU/Linux: Arquitectura.

El núcleo Linux, parte fundamental del sistema operativo, fue desarrollado por Linus Torvals, utilizando como modelo a UNIX[42]. Una de las diferencias fundamentales entre los núcleos Linux y UNIX, es que el primero, es Software Libre, mientras que el segundo no lo es.

El proyecto GNU apoya tanto a los sistemas GNU/Linux como al sistema GNU. La Free Software Foundation[38] financió la reescritura de las extensiones de la biblioteca C de GNU relacionadas con Linux, por lo que ahora están bien integradas y los sistemas GNU/Linux más recientes utilizan la versión actual de la biblioteca sin modificaciones. Esta misma fundación también financió las etapas iniciales del desarrollo de Debian GNU/Linux.

Hoy en día existen muchas variantes diferentes del sistema GNU/Linux, comúnmente llamadas «distribuciones». La mayoría de ellas incluyen software que no es libre, ya que sus desarrolladores siguen la filosofía de Linux en lugar de la de GNU. Pero también existen distribuciones GNU/Linux completamente libres.

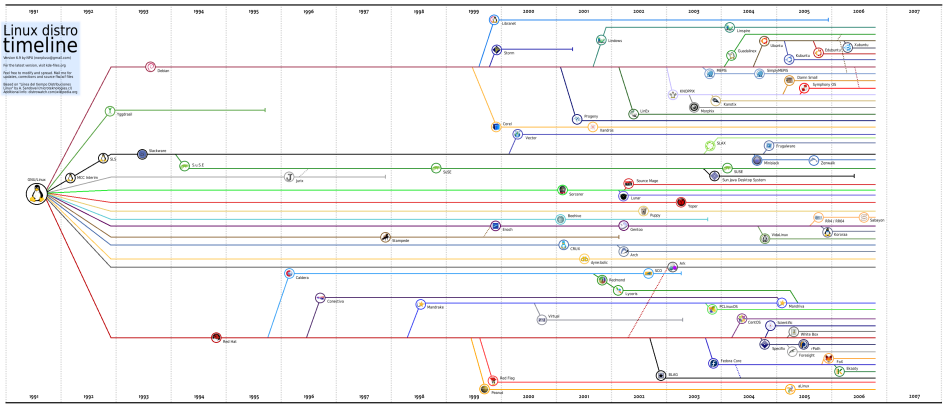

Figura 4: GNU/Linux: Distribuciones.

Cabe destacar que el sitema operativo GNU / Linux es idóneo para utilizarse en sistemas embebidos como el de nuestro proyecto: Aunque la mayoría de estos sistemas funcionan en ordenadores personales, este sistema operativo puede adelgazarse de opciones que no vamos a necesitar (Los drivers de fibra óptica, por ejemplo) para hacerlo más liviano y que cargue más rápido o se ajuste a las limitaciones de espacio de nuestro sistema.
Otra de las ventajas de usar éste sistema operativo es su soporte natural de protocolos de red: Desde el própio núcleo, dichos protocolos vienen implementados y optimizados, lo que nos da una versatilidad inigualable en el campo de los sistemas embebidos.
Además, existe una gran comunidad que trabaja para dar soporte a dispositivos de lo más dispares dentro del sistema GNU/Linux, lo cual nos da una compatibilidad bastante deseable a la hora de aventurarnos en un proyecto como éste.

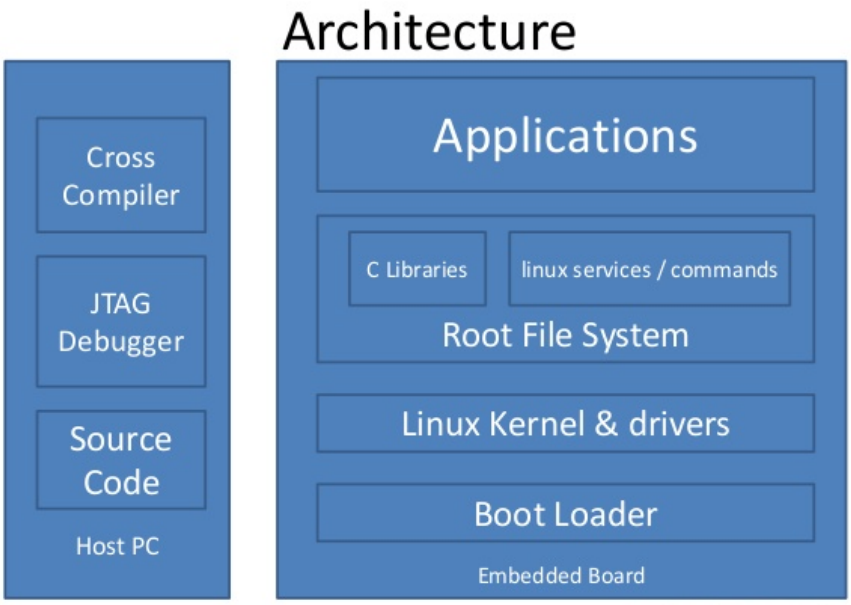

Figura 5: GNU/Linux: Sistemas embebidos.

El núcleo del sistema operativo GNU/Linux tiene una arquitectura bastante simple. Los sistemas de redes y ficheros están desplegados sobre el micro "kernel" de una manera modular. Los controladores y otras características pueden ser tanto compilados cuando creamos nuestro núcleo como añadidos después mientras éste está en funcionamiento mediante módulos. Esto nos provee de una herramienta altamente modular a la hora de construir nuestro sistema operativo para nuestro dispositivo embebido, que usa típicamente una combinación de controladores personalizados y programas para darle las funcionalidades deseadas.

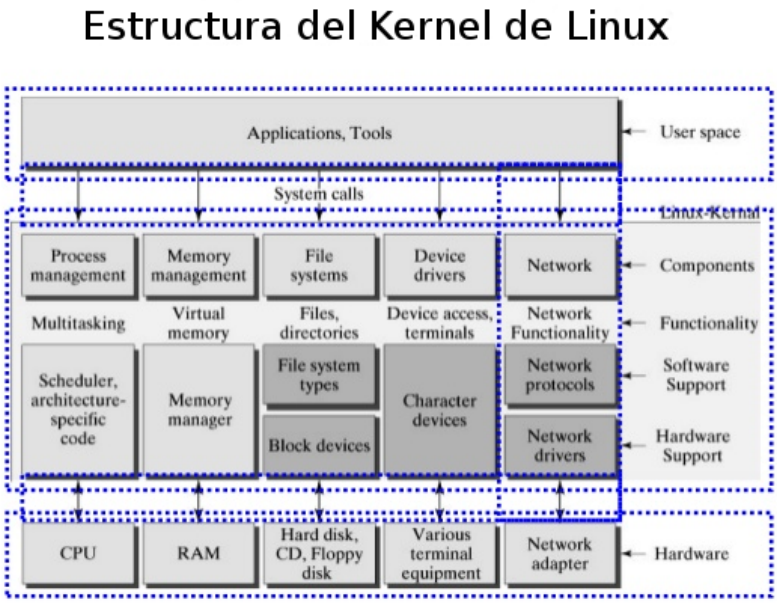

Figura 6: GNU/Linux: Arquitectura del Kernel.

Un sistema embebido también necesita realizar ciertas tareas generales, las cuales, para evitar reinventar la rueda, el "kernel" incorpora con un elenco de controladores y programas genéricos que permiten su adecuado funcionamiento.
Otro punto a favor de este sistema operativo es su soporte de sistemas multiprocesadores, lo cual lo hace idóneo desde el punto de vista de escalabilidad (Ya que no deberemos de restringirnos a modelos con un solo procesador con lo cual el rango de dispositivos a usar aumenta) y de potencia (Ya que nuestras aplicaciones van a poder funcionar en sistemas con mas de un núcleo de procesado).

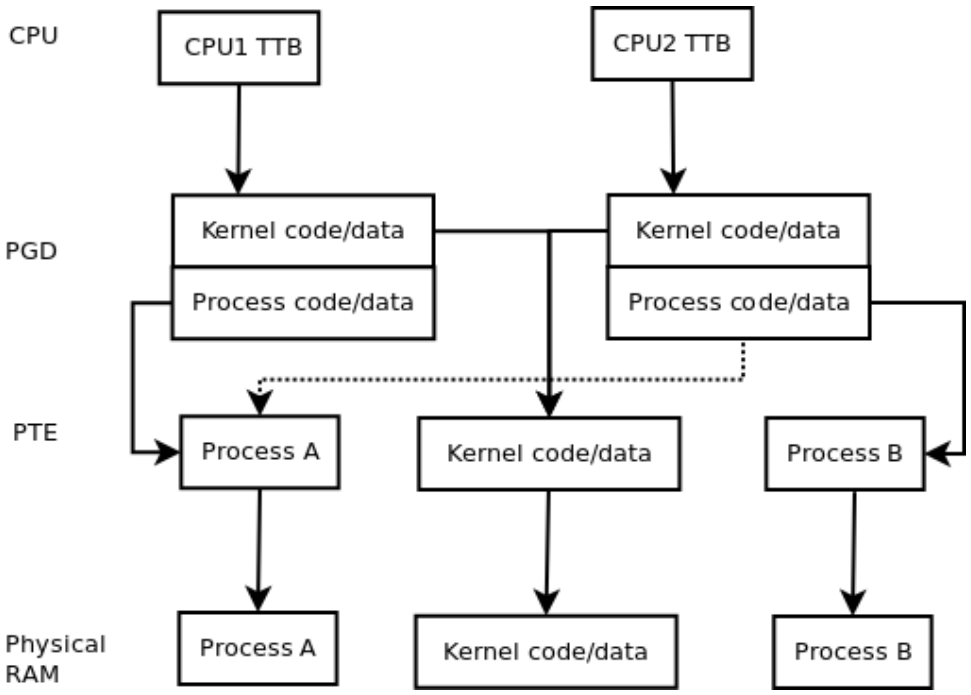

Figura 7: GNU/Linux: Multiprocesado.

Si a todos los puntos expuestos anteriormente le sumamos que es un sistema operativo gratuito, puede comprobarse por qué nos decantamos por éste software en vez de otros disponibles en el mercado.
Además, como nota a pie de página, GNU/Linux puede incorporar módulos en su núcleo para funcionar como un sistema operativo en tiempo real, lo cual aumenta las posibilidades de usar nuestro diseño en aplicaciones industriales o espaciales, por poner algunos ejemplos.

2.4. OpenMPI

La computación en paralelo es ahora parte de nuestras vidas al igual que los ordenadores, los teléfonos móviles o las antenas. Podemos encontrar ejemplos de esta tecnología en numerosos programas que usamos habitualmente como hojas de cálculo en la nube o los teclados predictores en nuestro teléfonos. Debido a su uso cada vez más común, su programación ha sido mejorada debido, en gran parte, al interés generado en ella hasta los estándares en los que se encuentra actualmente.
Pero no siempre ha sido así ya que, antes de 1990, los ingenieros no disponían de las mismas herramientas con las que contamos nosotros actualmente.
Programar aplicaciones en paralelo para las diferentes arquitecturas de computación era una tarea dificil y tediosa. Si bien es cierto que había numerosas librerías que podían facilitar dicho trabajo, no había ningún estándar aceptado y extendido.
Además, su uso no estaba tan difundido: Durante este periodo la mayoría de las aplicaciones se desarrollaban en los laboratorios de investigación y desarrollo, con lo cual, sólo una parte del ámbito académico se dedicaban al uso de esta tecnología.

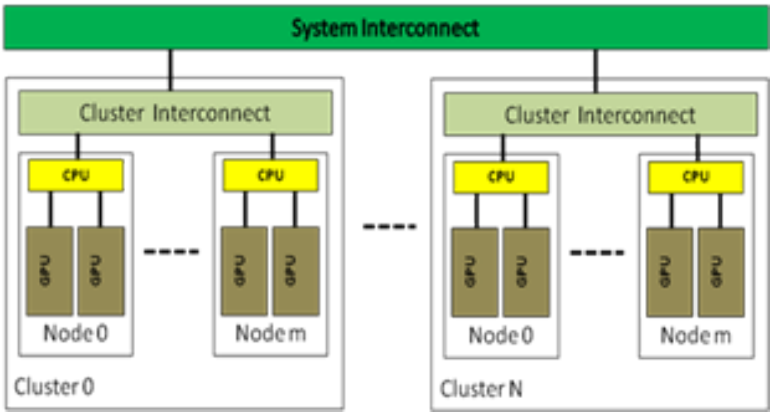

Figura 8: OpenMPI: Interoperabilidad de diferentes arquitecturas.

El modelo mas comúnmente adoptado durante esta época era el modelo de envío de mensajes: Una aplicación envía mensajes a diferentes procesos según se requiera para llevar a cabo una tarea. Es un modelo de demostrado funcionamiento para aplicaciones en paralelo.
Por ejemplo, un proceso maestro puede asignar una carga de trabajo a un proceso esclavo a través de un mensaje que describe el trabajo a realizar. Otro ejemplo es una aplicación de clasificación en paralelo que ordena los datos en procesos locales y luego pasa su respuesta a procesos vecinos para

que los ordenen a su vez con sus datos locales. Casi todas las aplicaciones en paralelo pueden ser descritas y resueltas a través del modelo de envío de mensajes.

Dado que la mayoría de librerías de esta época usaban en mismo modelo de envío de mensajes con pequeñas diferencias entre ellas, los autores de dichas librerías se reunieron en la conferencia de supercomputación de 1992[43] para definir un estándar para el envío de mensajes. Había nacido MPI, el interfaz de envío de mensajes (Message Parsing Interface por sus siglas en inglés). Este estándar de interfaces permitiría a los ingenieros escribir aplicaciones en paralelo que fueran portables entre la mayoría de arquitecturas y les permitiría aunar esfuerzos y propagar características y mejoras de las diferentes librerías.

El estándar MPI ha sido redefinido varias veces:

- MPI-1.0 (Publicado en 1994).
- MPI-2.0 (Publicado en 1996). MPI-2.0 es, en su mayor parte, Una extensión de la especificación original 1.0. MPI-2.1 y MPI-2.2 fueron publicados mas tarde, y se consistían en pequeños arreglos, cambios y adiciones a la especificación 2.0.
- MPI-3.0 (Publicado en 2012). Después vino MPI-3.1 que incorporaba algunos arreglos a la especificación 3.0.

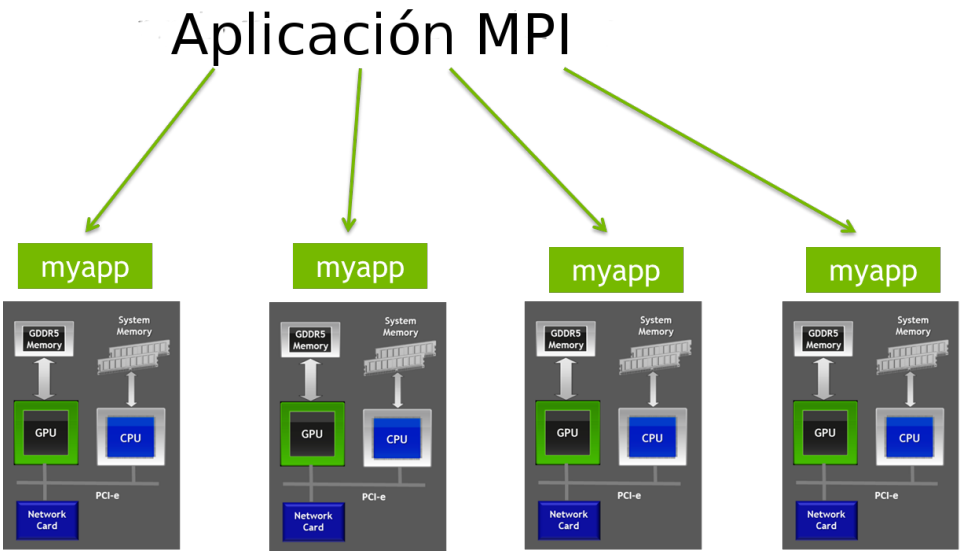

Figura 9: OpenMPI: Diagrama.

Algunos conceptos clave detrás del diseño MPI son los siguientes:

- Definición de comunicador: Un comunicador es un grupo de procesos que tienen la habilidad de comunicarse entre ellos. En este grupo de procesos, a cada uno se le asigna un único identificador y ellos se envían mensajes entre ellos usando dicho identificador.
- La comunicación está basada en operaciones de envío y recepción entre los procesos. Un proceso puede enviar un mensaje a otro proceso a traves del identificador del destinatario y una etiqueta única para identificar el mensaje. El receptor puede entonces informar de que ha recibido el mensaje y manejar los datos del mensaje de manera apropiada. Este tipo de comunicación es de tipo punto a punto.
- Habrá algunos casos donde los procesos necesitarán comunicarse con todos los demás nodos. Por ejemplo, cuando un proceso maestro necesita hacer llegar información a todos los procesos esclavos. MPI puede usar las librerías Ethernet estándar para llevar a cabo dicha comunicación.

Todas las especificaciones MPI pueden descargarse del Foro Oficial de MPI[41].

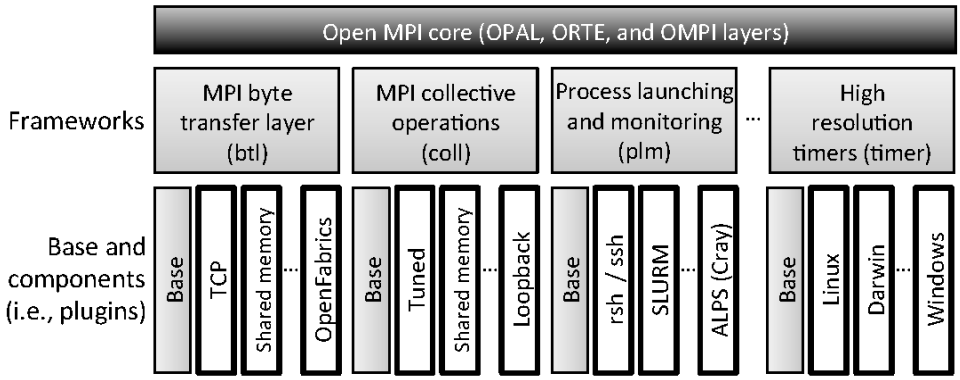

Figura 10: OpenMPI: Arquitectura.

En nuestro proyecto vamos a usar MPICH el cual es un derivado de Open-MPI, que a su vez es una implementación del estándar MPI que, actualmente, está supervisado por el Foro MPI (Un gran comité que comprende numerosos actores tanto académicos como de la industria). MPI es una API[40] (Aplication Programming Interface o Interfaz de programación de aplicaciones por sus siglas en inglés) y, como ya hemos comentado anteriormente, está estandarizada para el uso de computación en paralelo y/o distribuído.

OpenMPI es capaz de alcanzar un alto rendimiento y cuenta con una comunidad de usuarios bastante consolidada la cual ofrece un excelente soporte, documentación y versión de actualizaciones.

Open MPI tiene varios objetivos:

- Crear una completa implementación de MPI que sea libre, gratis, revisada por pares y tenga una calidad de alto nivel.
- Que dicha implementación tenga un rendimiento extremadamente alto y competitivo (Latencia, ancho de banda, ...)
- Involucrar a una comunidad de desarrolladores Open Source con la comunidad de Computación en paralelo.
- Crear una plataforma estable para el desarrollo comercial y las investigaciones académicas.
- Prevenir el problema de escisión común en otros proyectos MPI.
- Dar soporte a la mayor cantidad de plataformas.

En definitiva, trabajar con la comunidad de computación en paralelo para crear una implementación de alto estándar de MPI que pueda ser usado por una gran variedad de sistemas.

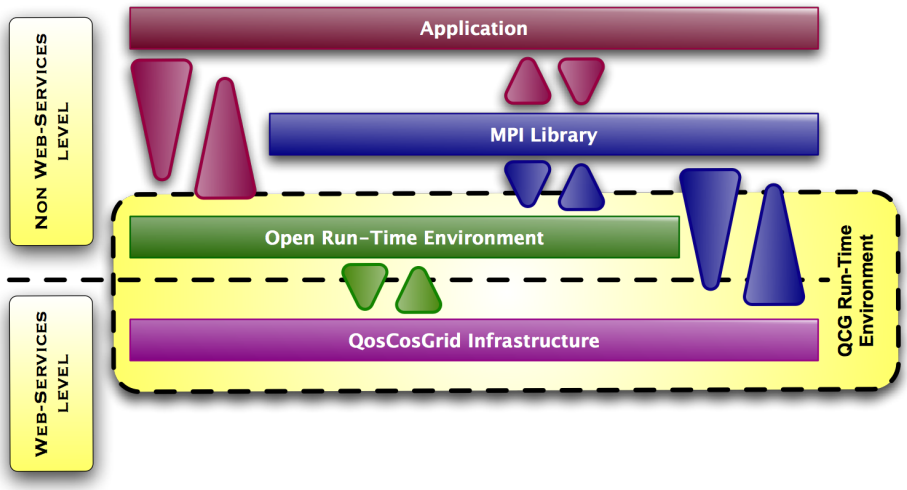

Figura 11: OpenMPI: Funcionamiento en el sistema operativo.

3. Instalación del entorno virtual.

3.1. Creación de la Máquina Virtual.

Aunque nosotros damos la máquina virtual totalmente funcional, a continuación vamos a detallar los pasos necesarios para la correcta instalación y configuración de la máquina virtual con la que vamos a interactuar con nuestra FPGA.
Nuestro primer paso seria instalar el gestor de máquinas virtuales. Para este proyecto usaremos la tecnología de virtualización VMWare.
Si necesita información acerca de qué es una máquina virtual, diríjase al capítulo Máquinas Virtuales.

Procederemos ahora a descargarnos el gestor de máquinas virtuales gratuito de VMWare: VMWare Workstation Player. En el momento de redactar éste proyecto, la versión era la 12.0 y está incluido su instalador en la carpeta "Compilables y ejecutables". Su instalación es bastante intuitiva: Basta con aceptar los valores por defecto y pulsar "siguiente" en el menú de instalación.
Una vez instalado, habrá que crear una VM con la que conectarse a la FPGA. Para ello nosotros hemos seleccionado el sistema operativo Fedora 24 de 64 bits para escritorio XFCE (La imagen usada para dicha instalación también está disponible en el directorio de ejecutables y archivos de este proyecto).
El propio gestor de VMs nos provee de un instalador sencillo para dicho sistema operativo. Simplemente pulsamos sobre "Archivo" y luego le damos a "Crear una nueva máquina virtual". Nos deberia aparecer un menú donde nos pide elegir entre una instalación típica y otra "custom" o a medida. Seleccionamos la típica tal y como aparece en la siguiente figura 12:

Figura 12: Instalación VMWare: Menú inicio.

Ahora el menú progresará para pedirnos la imagen de nuestro sistema operativo. Lo seleccionamos mediante el navegador de archivos y se nos quedará de la siguiente manera:

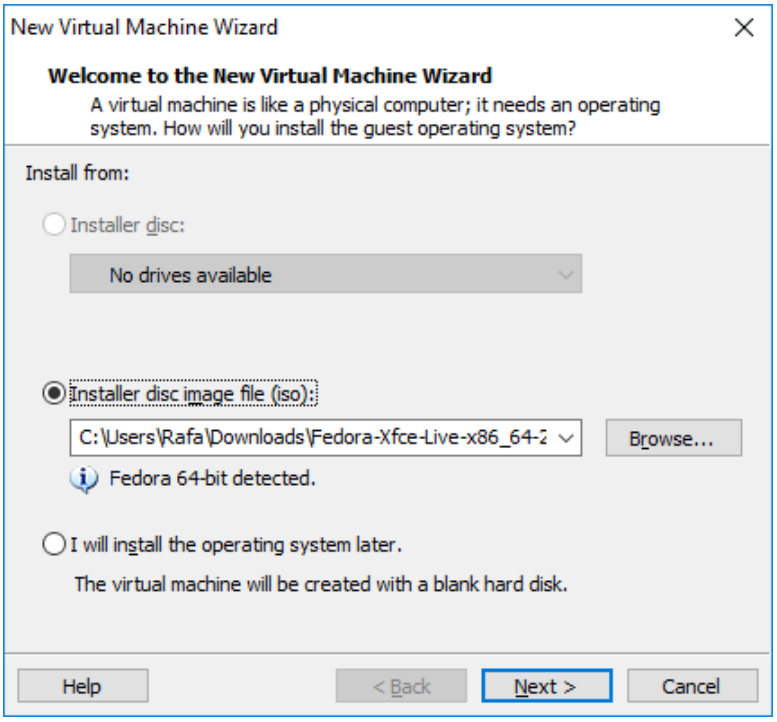

Figura 13: Instalación VMWare: Selección de archivo de imagen.

En el siguiente paso, el menú nos pedirá algunos datos referentes a nuestro usuario, tal y como podemos apreciar en la siguiente captura de pantalla 14:

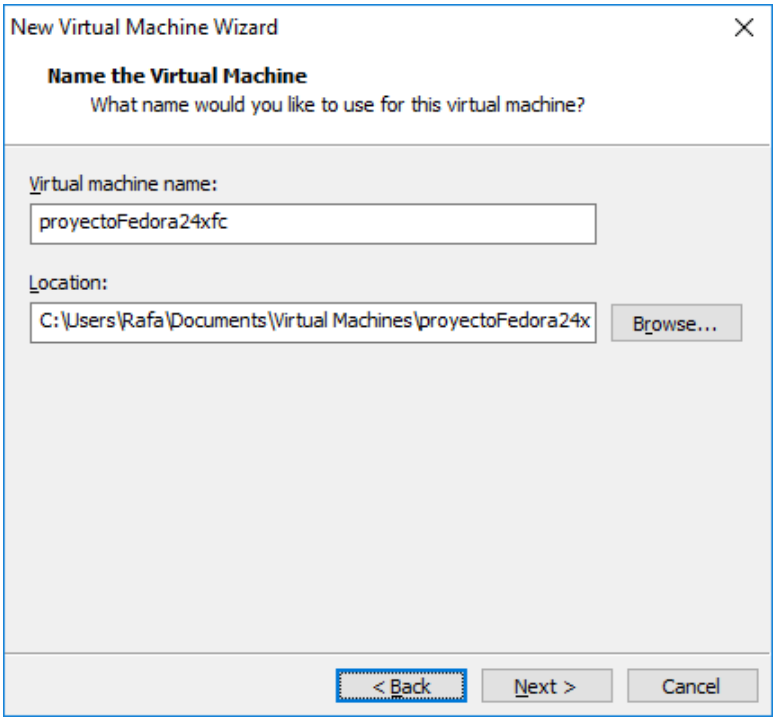

Figura 14: Instalación VMWare: Datos de usuario.

Una vez creado el usuario y la contraseña (En nuestra VM el usuario es "proyecto" y la contraseña es "proyecto."), tendremos que seleccionar un nombre para dicha máquina virtual y la ubicación dentro de nuestro ordenador.
Tras ello, tendremos que seleccionar las características de hardware que va a emular nuestro virtualizador. Este proceso se realiza en dos pasos dentro del menú. El primero es seleccionar el tamaño en disco. Por defecto vienen 20 GBs de espacio. Además, nos pregunta si debemos dividir el espacio en varios archivos o en uno solo. Seleccionamos varios, como podemos ver a continuación 15, con un tamaño de 80 GBs, y proseguimos:

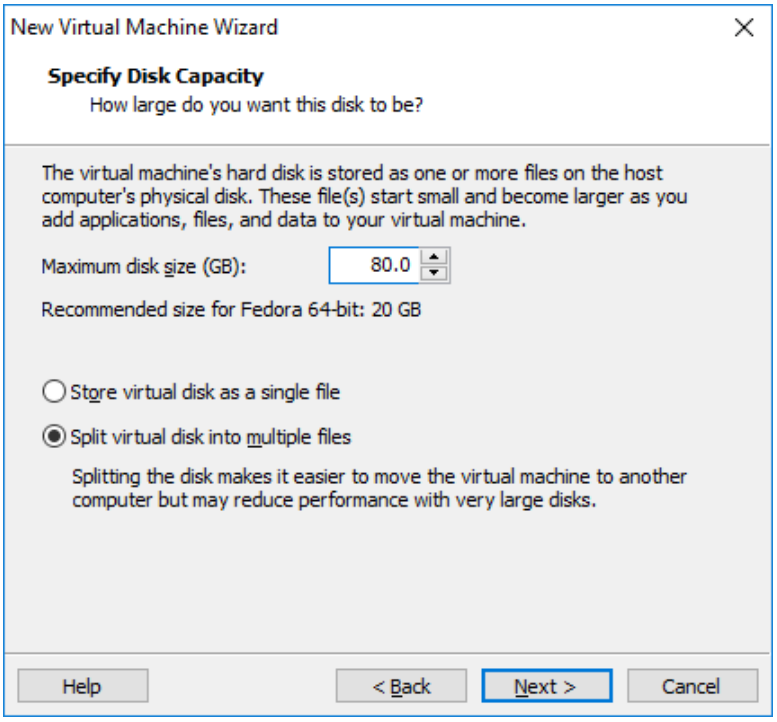

Figura 15: Instalación VMWare: Espacio en disco.

Ahora nos propondrá unas características por defecto referentes al hardware, tal y como podemos ver en la figura 16. Dependiendo de lo que queramos hacer con nuestra VM elegiremos unas u otras. Para cambiar dichas características elegiremos la opción de "Modificar características". En nuestro caso elegimos usar 2 procesadores y 4 GBs de RAM.

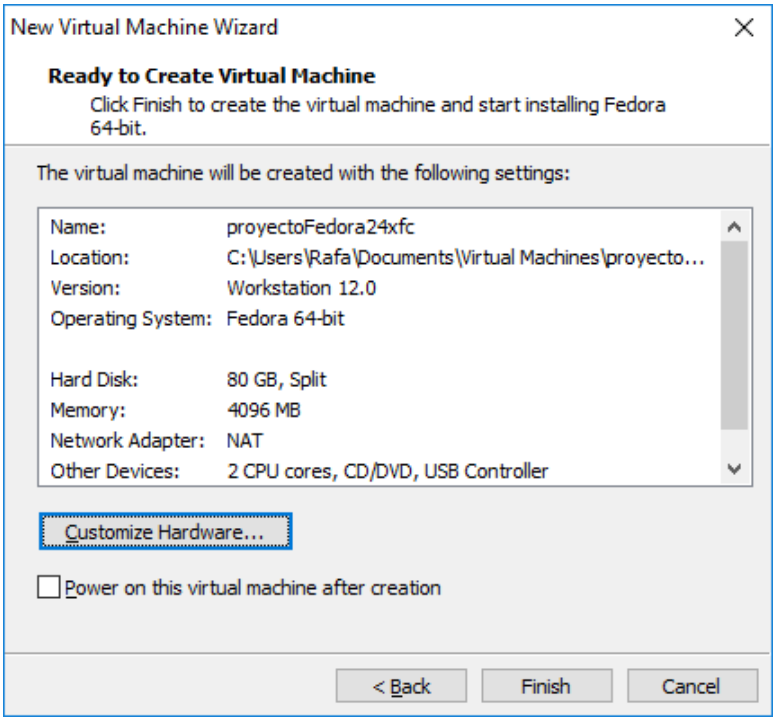

Figura 16: Instalación VMWare: Características de hardware.

Una vez seleccionadas las opciones que deseamos, pulsamos en finalizar y nos aparecerá una pantalla donde comenzará la instalación de Ubuntu. Dicha instalación se hará de manera automática, usando los datos de usuario que le hemos facilitado a VMWare en los menús anteriores. Tras finalizar dicha instalación, instalaremos las VMWare Tools, las cuales son una serie de herramientas muy útiles, sobre todo para transferir datos del Sistema Operativo host (Windows 10 en nuestro caso) al Sistema Operativo virtualizado (O guest). Podemos conseguir más información en la página principal de VMWare[12].

Tras esta última instalación, pasaremos ahora a la configuración de dicho Sistema Operativo (SO a partir de ahora) para su uso en el entorno de nuestro proyecto.[11]

3.2. Configuración de la Máquina Virtual.

En esta sección instalaremos y configuraremos el software necesario para programar e interactuar con nuestra placa. Solo resaltar el hecho de que, para ejecutar la mayoría de comandos, debemos estar logeados dentro de una consola de Linux (O Shell) como nuestro usuario por defecto. En nuestra VM funcional, el nombre del usuario sería "proyecto" y nuestra contraseña sería "proyecto.", tanto para este usuario como para root.
El primer paso sería instalar el paquete de software de Xilinx, el cual nos permitirá configurar y programar nuestra placa Virtex. En nuestro caso hemos elegido la versión ISE 14.3 por su compatibilidad, tanto con nuestro SO como con nuestro hardware Virtex. Dicho paquete puede descargarse desde la página oficial de Xilinx[13].
Lo primero que habríamos de hacer sería irnos al directorio donde tengamos el paquete de instalación de la suite ISE, descomprimirlo y ejecutar el binario de instalación gráfica. Para ello haremos lo siguiente:

```
1  # cd /home/proyecto/downloads
2  # tar -xvf Xilinx_ISE_DS_Lin_14.3_P.40xd.6.0.tar
```

Tras descomprimirlo, nos moveríamos al directorio con los ejecutables necesarios para la instalación gráfica:

```
1  # sudo dnf install ncurses-compat-lib
2  # sudo mkdir /opt
3  # chown -R proyecto: /opt
4  # cd bin/lin64/Xilinx_ISE_DS_Lin_14.3_P.40xd.6.0
5  # ./xsetup
```

Al abrir dicho script, nos aparecerá un instalador gráfico, el cual puede observarse en la figura 17:

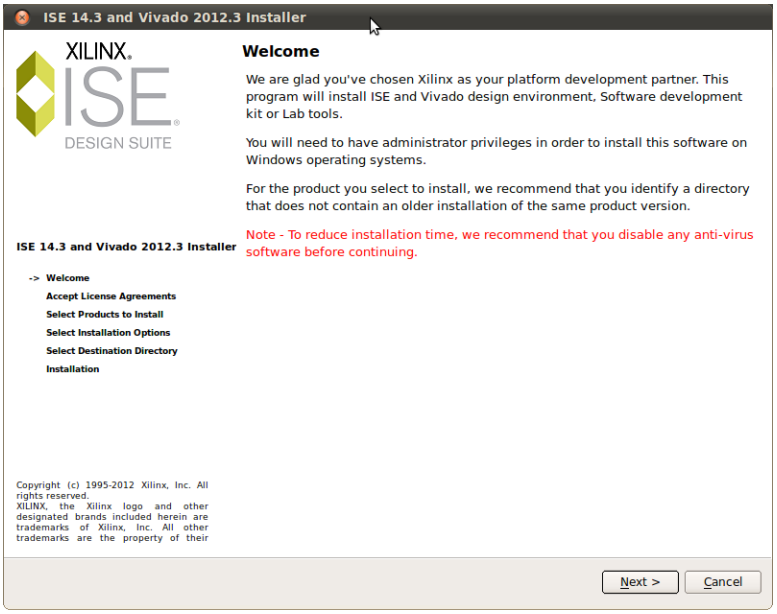

Figura 17: Instalación ISE Xilinx: Menú de inicio.

La instalación en sí es bastante directa: Simplemente basta con aceptar los acuerdos de licencia de Xilinx, seleccionar el paquete de software a instalar, seleccionar los componentes, elegir el directorio de destino y empezaría con la instalación. Acerca de estos pasos, aunque aceptando las opciones por defecto es suficiente, nos gustaría recalcar ciertos aspectos de la misma:

- Hemos instalado el paquete **ISE Design Suite System Edition + Vivado System Edition**.
- **No hemos instalado los controladores de cable**.
- El directorio destino usado en esta memoria será /**opt**/**Xilinx**.
- El tamaño del paquete de software es bastante grande. Es necesario tener alrededor de 16 GB libres.

Una vez finalizada la instalación, tendremos dicho paquete instalado en el sistema, aunque deberemos seguir instalando algunos paquetes específicos del SO para el correcto funcionamiento, tales como :

```
1  # sudo dnf install libusb-devel libftdi-devel
       fxload
2  # sudo dnf install gitk git-gui glibc-devel
```

Y cargar las variables de entorno en la sesión de consola:

```
# export PATH=\$PATH:/opt/Xilinx/14.3/ISE_DS/ISE/
    bin/lin
# cd /opt/Xilinx/14.3/ISE_DS
# chmod a+x settings64.sh
```

Cada vez que queramos abrir la suite de software Xilinx, deberemos de ejecutar los comandos que aparecen a continuación, que cargan las variables necesarias en el sistema para la correcta ejecución de la misma. Es importante destacar que dichos comandos vuelven inservible el terminal desde el que se ejecutan, por lo que deberemos abrir otra terminal si quisiéramos ejecutar cualquier otra serie de instrucciones.

```
# source /opt/Xilinx/14.7/ISE_DS/settings64.sh
# ise
```

A este efecto, hemos de reseñar que hemos creado un ejecutable que ejecuta estos comandos en el escritorio de la VM. Solo hemos de hacer doble click y se nos abrirá la suite de sofware. El contenido de este ejecutable es el mismo que los comandos ilustrados anteriormente y su nombre es "start_ise.bash". Si todo ha funcionado correctamente, debería aparecernos el ISE de Xilinx una vez ejecutados dichos comandos, como se puede apreciar en la siguiente figura 18

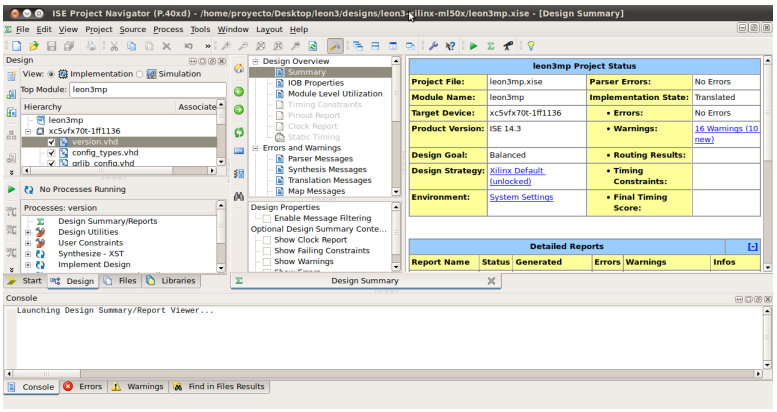

Figura 18: ISE Xilinx: Menú de inicio.

Es necesario una licencia para poder ejecutar el software a plena capacidad, con lo que se recomienda obtener una de la página de licencias de Xilinx[19], aunque estos pasos se detallarán mas adelante en la sección Licenciamiento Xilinx: Página principal.

Ahora, instalaremos el programa “gtkterm”, el cual es una sencilla interfaz gráfica que nos permitirá comunicarnos con un dispositivo a través del puerto serie. Su instalación es bastante sencilla. Basta con introducir el siguiente comando:

```
1   # sudo dnf install gtkterm
```

Con dicho programa, tendremos una utilidad funcional para controlar el puerto serie de nuestra VM.
A continuación, interactuaremos con nuestra placa para comprobar el correcto funcionamiento de todo lo instalado hasta ahora. Para ello conectaremos la placa a la corriente mediante la fuente de alimentación, colocaremos en su zócalo la memoria SDRAM y encenderemos la placa mediante el interruptor “SW1”.
Como a partir de ahora usaremos la nomenclatura oficial de Xilinx referente a sus productos, se puede consultar el manual de dicha placa en nuestro directorio de ejecutables pero también en la siguiente referencia bibliográfica [14].
Para el correcto funcionamiento de la comprobación, es necesario que los **pines de “SW3” estén con la configuración “00010101” y los de “SW6” estén con la configuración “11001010”**.
Ahora abriríamos el “Serial Port Terminal” y, con la configuración por defecto, pulsaríamos el boton "SYSACE RST". Tras efectuar dicha acción, deberíamos ser capaces de entrar en un menu donde podemos hacer chequeos de memoria, jugar al “Simon dice...” o comprobar la integridad de la SDRAM, tal y como puede observarse a continuación en la figura 19:

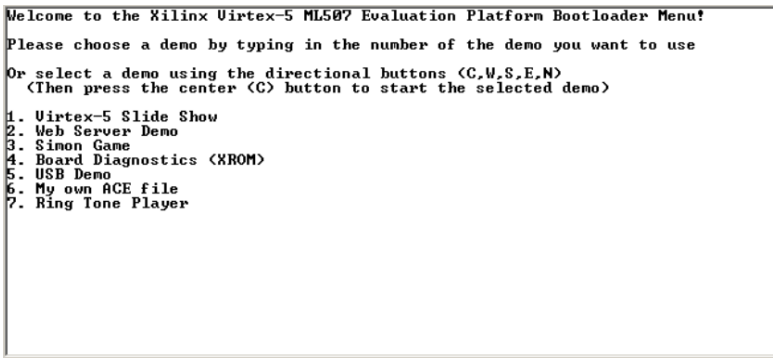

Figura 19: Virtex 5: Menú de Bootload.

Pasaremos ahora a instalar los drivers necesarios para el cable "JTAG", el cual nos permite reprogramar la placa según un diseño que hayamos previamente creado en el ISE de Xilinx. Para ello, abriremos una consola y introduciremos los siguientes comandos:

```
1  # mkdir -p /etc/hotplug/usb/
2  # cd /opt/Xilinx/14.3/ISE_DS/common/bin/lin/
       digilent/
3  # ./install_digilent.sh /opt/Xilinx/14.3/ISE_DS/ISE
```

Aceptaremos todos los valores por defecto que se nos presentan y los drivers para el cable "JTAG" estarían instalados.
Para comprobar que todo funciona correctamente, deberíamos comprobar que nuestra VM detecta el componente USB del cable "JTAG", estando éste conectado, mediante el siguiente comando:

```
1  # lsusb | grep Xilinx
```

Nos debería de dar un resultado similar al de la siguiente figura 19:

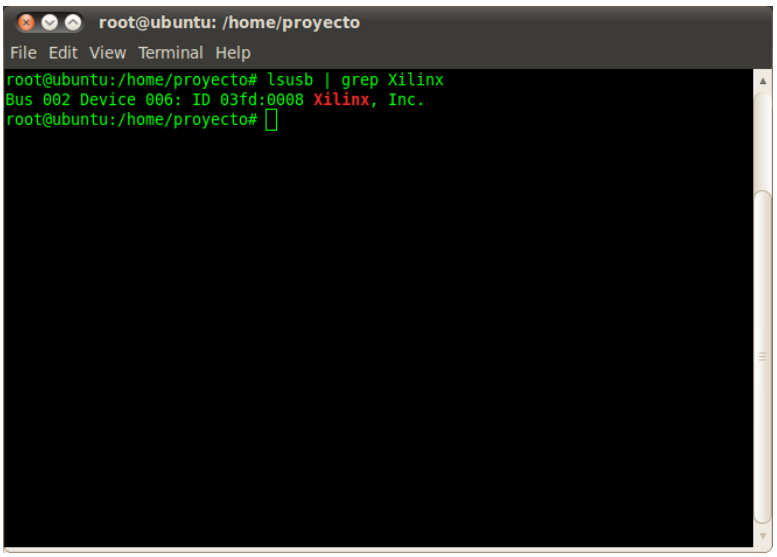

Figura 20: Virtex 5: Comprobación drivers cable JTAG.

Si no nos devolviera nada, es que el cable no está conectado o que no ha sido desconectado del host (Recordemos que estamos funcionando dentro de una VM).
Para asegurarnos de que sería posible interactuar con la placa (Estando ésta conectada usando el cable JTAG), debemos ejecutar los siguientes comandos desde la linea de consola, sin que nos dé error alguno:

```
# source /opt/Xilinx/14.7/ISE_DS/settings64.sh
# impact
```

Nos aparecerá la siguiente pantalla de bienvenida, referente a la configuración del proyecto del ISE, tal y como se puede apreciar en la siguiente figura 19:

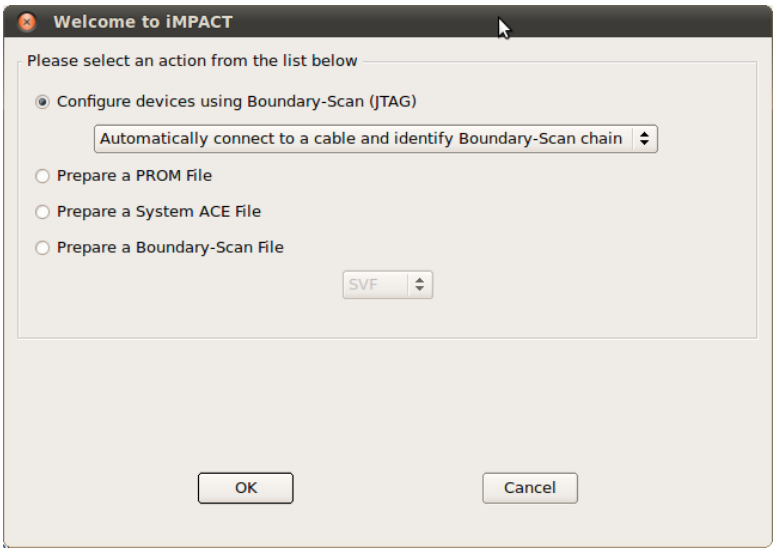

Figura 21: Virtex 5: Comprobación drivers cable JTAG Impact bienvenida.

Pulsaríamos en el botón de "Accept" y nos mostraría la siguiente figura 19, si todo funciona correctamente:

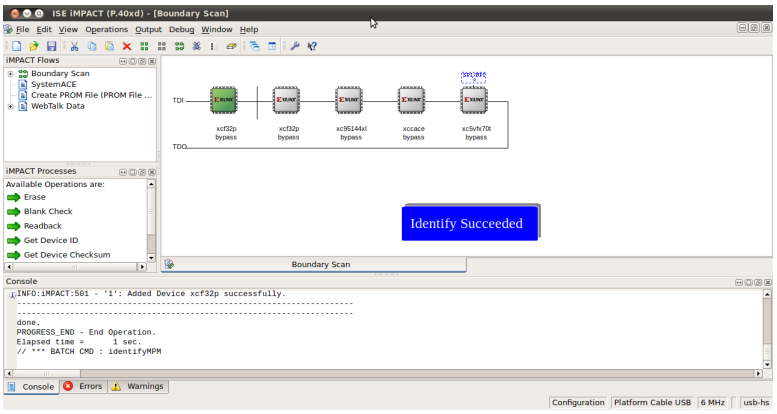

Figura 22: Virtex 5: Comprobación drivers cable JTAG Impact OK.

Si, a pesar de todo, siguiéramos teniendo problemas con el cable, tendríamos que compilar desde el repositorio "GIT" los drivers, tal y como se observa a continuación:

```
1  # sudo dnf install gitk git-gui libc6-dev
2  # cd /opt/Xilinx
3  # git clone git://git.zerfleddert.de/usb-driver
4  # cd usb-driver/
5  # make
6  # ./setup_pcusb /opt/Xilinx/14.3/ISE_DS/ISE/
7  # echo "PATH=\$PATH:/opt/Xilinx/14.3/ISE_DS/ISE/bin
      /lin" >> ~/.bashrc
8  # echo "export PATH" >> ~/.bashrc
```

Si esto no fuera suficiente, deberíamos de instalar los drivers en el "host" y recibiríamos un mensaje de error similar al siguiente:

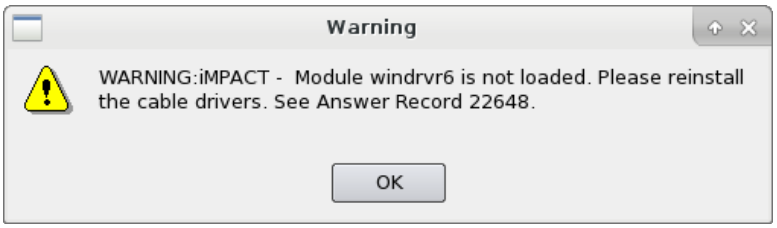

Figura 23: Virtex 5: Error de drivers en el host.

Para terminar, debido a las características de la placa y del modelo VHDL que vamos a usar, deberemos obtener una licencia acorde a dicho nivel. Para ello visitaríamos la página de licencias de Xilinx [15]. Una vez introducidos nuestros datos de usuario (Sería necesario registrarse en la página), debería de aparecernos el menú de la siguiente imagen 24:

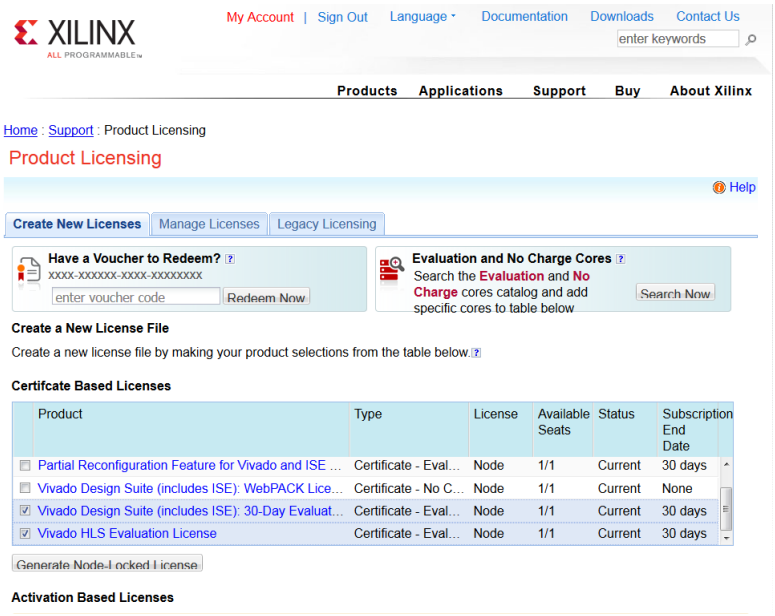

Figura 24: Licenciamiento Xilinx: Página principal.

Seleccionaríamos las mismas 2 opciones que aparencen en la anterior imagen y pulsaríamos sobre “Generate Node-Locked License”. Aparecería entonces el siguiente recuadro 25:

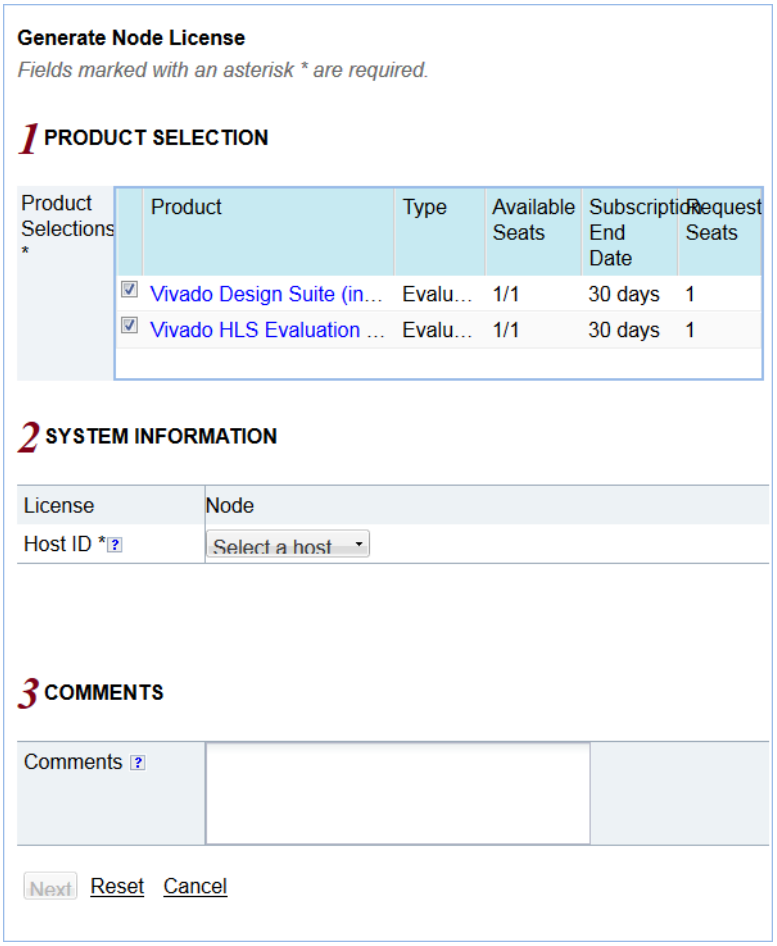

Figura 25: Licenciamiento Xilinx: Página de configuración.

Nos pediría una serie de información, como el Sistema Operativo o la MAC, para generar una licencia válida solo para un PC (O Máquina Virtual). La introducimos y nos la descargamos a nuestro entorno virtualizado.

Ahora solo tendríamos que usar dicha licencia en el IDE de Xilinx para que nos deje generar nuestro microprocesador LEON3. Para ello abrimos dicho IDE siguiendo los pasos previamente explicados. Dentro del menú "Help" elegiríamos la opción "Manage License..." y nos aparecería la siguiente ventana 26:

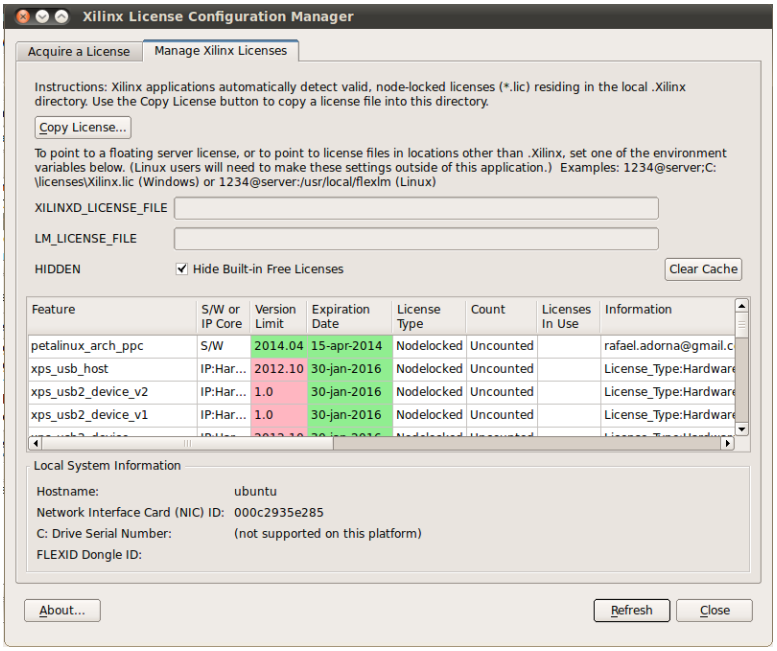

Figura 26: Licenciamiento Xilinx: Menú IDE configuración.

Pulsaríamos sobre “Copy License...” y elegiríamos el fichero de licencia. Una vez aceptado, deberiamos poder sintetizar nuestros modelos VHDL durante un mes, prorrogrables siempre que generemos de nuevo la licencia.

Si siguiéramos teniendo problemas con la licencia, deberíamos asegurarnos que la MAC de nuestra tarjeta de red coincide con el que el software de Xilinx obtiene:

```
1  # sudo dnf install redhat-lsb
2  # lmutil lmhostid
```

Nos debería de aparecer una información en este formato:

```
1  # lmutil lmhostid
2  # ...
3  # The FLEXnet host ID of this machine is ``
       NUESTRA_MAC''
```

Si por cualquier motivo nuestra MAC fuera 000000000000 en vez de la correspondiente, puede ser debido a que dicho comando solo busca la interfaz “eth0” y, si no la encuentra, asigna dicho valor a la licencia.
Para solventar este problema, deberíamos de usar los siguientes comandos:

```
1  # ip link set ens36 down
2  # ip link set ens36 name eth0
```

```
3   # ip link set eth0 up
```

Nótese que nuestra interfaz se llama “ens36”.

En nuestra VM, hemos creado un ejecutable en el escritorio para solventar éste y otros problemas relacionados con la licencia. Su nombre es “grlib_ise_setup.bash” y efectúa los comandos de arriba indicados y otros relacionados con la licencia. Se invita al lector a que revise el contenido de dicho ejecutable.

Una vez finalizada la configuración de la máquina virtual y el entorno de software necesario, pasaríamos a la configuración de la placa.

4. Instalación y Compilación de LEON3.

4.1. Compilación de LEON3.

Los ficheros necesarios para la síntesis del modelo VHDL LEON3 en la placa Xilinx ML507 los tenemos en el archivo "grlib-gpl-1.5.0-b4164.zip", incluido en nuestro directorio de archivos útiles para el proyecto (Cuya ruta de archivos, dentro de la máquina virtual es "/home/proyecto/Desktop/instaladores/").
También podemos descargarlo desde la página del proyecto Gaisler [16].
Para empezar a trabajar con él, lo que haríamos sería descomprimirlo e invocar el ejecutable que ponemos a continuación, el cual es una interfaz gráfica para configurar la placa según las características que deseemos. Para ello, haríamos lo siguiente desde el terminal:

```
1  # cd designs/leon3-xilinx-ml50x/ (Aunque nuestra
       placa sea la ml507) y ejecutamos:
2  # make distclean
3  # make xconfig
```

El uso del comando "distclean" es necesario en caso de que tengamos configuraciones previas: Limpia todas nuestras preferencias para que podamos empezar desde cero.

Debería aparecernos la siguiente interfaz gráfica:

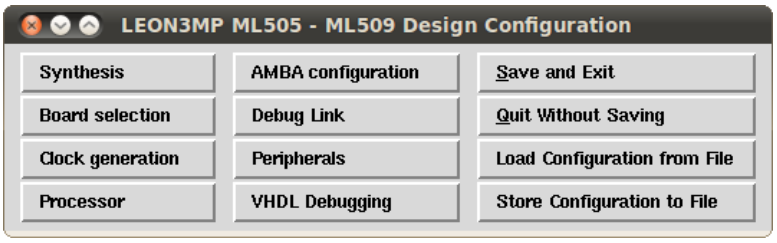

Figura 27: LEON3 configuración: Pantalla principal.

Aunque este menú es bastante complejo y puede dejarnos configurar varios "cores" dentro del microprocesador incluso, dejamos todas las configuraciones por defecto, aunque seleccionando como placa en "Synthesis" la "Xilinx-Virtex5" y en "Board Selection" la "ml507" y salimos de dicho menú pulsando sobre "Save and Exit".
Ahora, deberíamos de generar los archivos necesarios a partir del modelo VHDL con las preferencias que le hemos introducido en el paso anterior. Esto se puede hacer tanto de manera gráfica como por la línea de comandos. Aquí vamos a hacerlo mediante dicha línea de comandos (Si nos diera cualquier error, debemos de tener cargadas las variables del entorno IDE de

Xilinx, tal y como se describió anteriormente). Para ello introduciremos lo siguiente, dentro del directorio en el que nos encontrábamos antes:

```
1  # make ise
```

Este comando lanzará el proceso de compilación de nuestro archivos VHDL y puede tardar bastante, dependiendo de las características del sistema donde se ejecuten (Cerca de unos 30 minutos con 2 cores y 3 GB de RAM en nuestro caso).
Una vez sintetizados los ficheros necesarios, ahora es necesario enviarlos a la placa. Para ello, ejecutaremos lo siguiente, asegurándonos que tenemos el cable JTAG conectado y los pines con las posiciones correctas, tal y como describimos anteriormente 3.2

```
1  # make ise-prop-fpga
```

Como apunte final y, debido a las múltiples maneras que tenemos de conectarnos a la placa, hemos dejado tres compilaciones del LEON3 en la VM del proyecto, cuyas ubicaciones son las siguientes:

```
1  # /home/proyecto/Desktop/compilacion_leon_ethernet
2  # /home/proyecto/Desktop/
       compilacion_leon_no_ethernet
3  # /home/proyecto/Desktop/
       compilacion_leon_no_ethernet_fpu
```

Para conectarnos a la FPGA mediante el puerto serie o la interfaz USB, podemos usar cualquiera de ellas, pero, para conectarnos mediante un cable RJ-45 o de red, sería necesario usar los diseños de la carpeta cuyo nombre es “compilacion_leon_ethernet”.

4.2. Instalación de la interfaz GRMON.

Una vez que tenemos la placa configurada con la configuración deseada, tenemos que usar un ejecutable de Gaisler llamado "GRMON" [16] para conectarnos con la interfaz deseada (Puerto serie, Ethernet o USB). GRMON es un monitor de depuración para los procesadores de tipo LEON.
Para ejecutar dicho comando, usaremos los siguientes comandos, teniendo en cuenta que los ficheros necesarios deben estar, bien en nuestra carpeta de paquetes instaladores, bien en la página web de Gaisler [17].

```
1  # cd Desktop/instalacion/paquetes/leon3/
2  # tar -xzvf grmon-eval-2.0.54.tar.gz
3  # mv grmon-eval-2.0.54/ /opt/Gaisler/tool/
4  # echo "PATH=\$PATH:/opt/Gaisler/tool/linux/bin" >>
       ~/.bashrc
```

También deberíamos de instalar ciertos paquetes, por motivos de retro-compatibilidad, para que este comando pueda ser ejecutado:

```
1  # sudo dnf install glibc.i686 nspr-4.12.0-1.fc24.
     i686 zlib-1.2.8-10.fc24.i686 ncurses-compat-libs
     -6.0-5.20160116.fc24.i686 libusb-devel.i686
```

4.2.1. Conexión con la placa por el Puerto Serie.

Para conectarnos a la placa mediante esta conexión, una vez ejecutado los pasos anteriores, deberemos de ejecutar el comando "grmon -u" y nos debería de aparecer lo siguiente:

```
# grmon -u
GRMON2 LEON debug monitor v2.0.33 evaluation version

Copyright (C) 2012 Aeroflex Gaisler - All rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

This evaluation version will expire on 12/8/2013

using port /dev/ttyS0 @ 115200 baud
Device ID:           0x507
GRLIB build version: 4123
Detected frequency:  80 MHz

Component                              Vendor
LEON3 SPARC V8 Processor               Aeroflex Gaisler
AHB Debug UART                         Aeroflex Gaisler
JTAG Debug Link                        Aeroflex Gaisler
SVGA frame buffer                      Aeroflex Gaisler
GR Ethernet MAC                        Aeroflex Gaisler
Single-port DDR2 controller            Aeroflex Gaisler
AHB/APB Bridge                         Aeroflex Gaisler
LEON3 Debug Support Unit               Aeroflex Gaisler
LEON2 Memory Controller                European Space Agency
System ACE I/F Controller              Aeroflex Gaisler
AMBA wrapper for System Monitor        Aeroflex Gaisler
```

```
27  Generic UART                          Aeroflex
       Gaisler
28  Multi-processor Interrupt Ctrl.       Aeroflex
       Gaisler
29  Modular Timer Unit                    Aeroflex
       Gaisler
30  PS2 interface                         Aeroflex
       Gaisler
31  PS2 interface                         Aeroflex
       Gaisler
32  General Purpose I/O port              Aeroflex
       Gaisler
33  AMBA Wrapper for OC I2C-master        Aeroflex
       Gaisler
34  AMBA Wrapper for OC I2C-master        Aeroflex
       Gaisler
35  AHB Status Register                   Aeroflex
       Gaisler
36
37  Use command 'info sys' to print a detailed report
       of attached cores
38
39 grmon2>
```

Para esta conexión hemos usado el modelo generado en la carpeta "/home-/proyecto/Desktop/compilacion_leon_no_ethernet".

4.2.2. Conexión con la placa por el Puerto Ethernet.

Aún así, con esta configuración solo nos estaríamos conectando por el puerto serie a la placa. Para archivos mas grandes (Como imágenes de Linux) necesitaríamos que la placa tuviera soporte Ethernet para poder conectarlo un router y enviar la información a través de un cable de red.
Para poder conseguir esto, en el menú de configuración antes mostrado 27, deberíamos seleccionar el submenú "Debug Link" y nos aparecería la siguiente ventana 28:

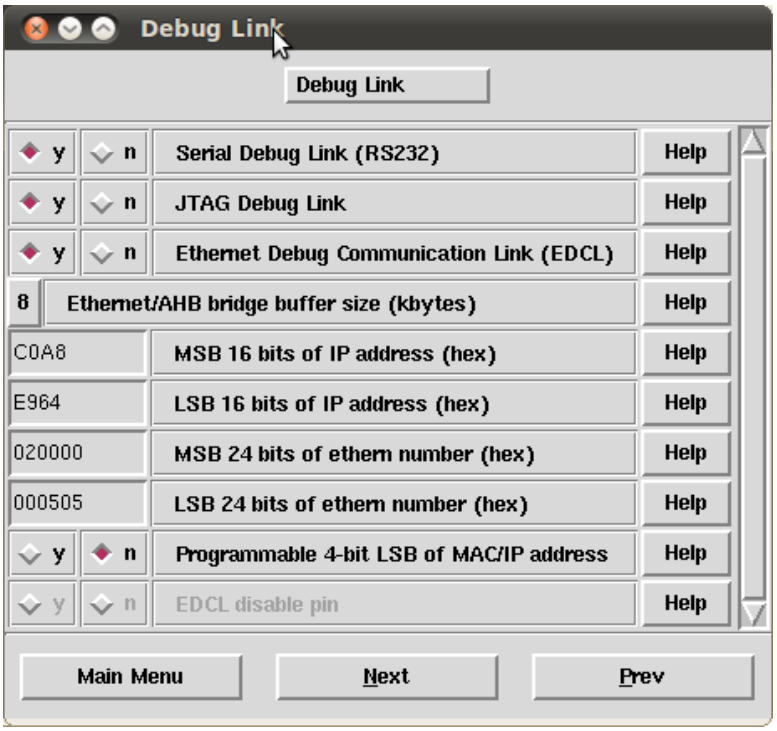

Figura 28: LEON3 configuración: Opciones Debug Link.

Lo que pretendemos en este menú es asignarle una dirección IP por defecto a la placa, para que lea los datos que nosotros enviemos a través de la conexión de red de nuestra VM.
Para ello, modificaremos los valores "MSB/LSB 16 bits of IP address (hex)" y les asignaremos una dirección que este en el rango de nuestra red Ethernet. Como ejemplo, pondré la configuración usada en mi caso.
Mi red es del tipo 192.168.88.XXX, donde el último valor puede ir desde 1 a 254. Como mi PC tiene el 78, voy a asignarle el 178, aunque nos valdría cualquier valor que no esté siendo usado por otro dispositivo en la red local. Trasladando este valor, 192.168.88.178, a Hexadecimal, vemos que nos da este valor: C0A8 58B2. Por lo tanto, tendríamos que poner dichos valores en

los campos anteriormente mencionados, quedándonos la configuración de la siguiente manera 29:

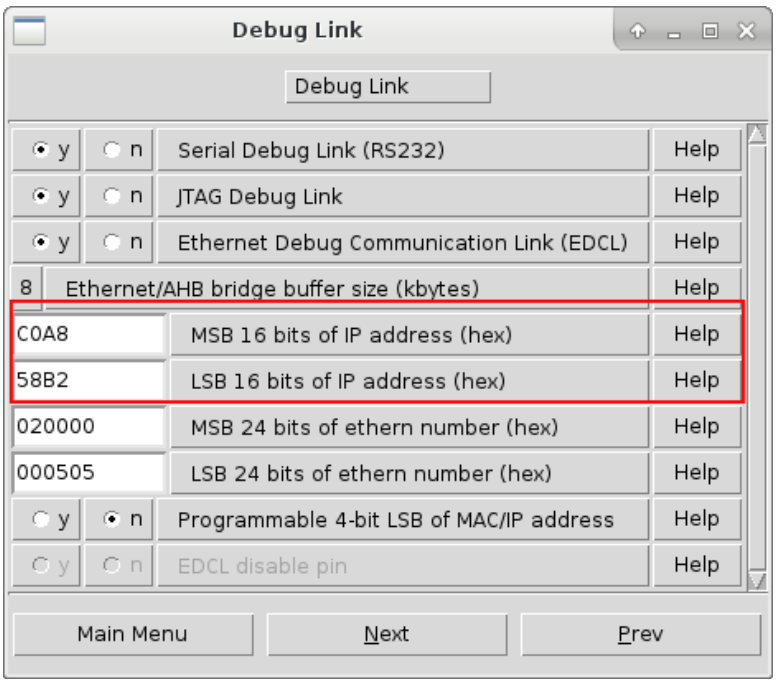

Figura 29: LEON3 configuración: Configuración Debug Link.

A continuación lo único que tendremos que hacer será volver al menú principal y salvar la configuración para, a continuación, compilar una nueva imagen para la placa:

```
1  # make ise
2  # make ise-prog-fpga
```

Una vez configurada la placa con los archivos creados en el anterior proceso y conectado el cable de red entre la placa y el router, deberíamos de ser capaces de conectarnos de manera similar que con el cable serie.Para ello usaremos el siguiente comando:

```
1  # grmon -u -eth -ip 192.168.88.178 -nb
```

Obviamente tendremos que usar la IP que le hayamos asignado en los pasos anteriormente descritos.
Un detalle a tener en cuenta es que tenemos que dar de alta el servicio GRMON en nuestro firewall o no conectará a la placa, con un error de "timeout" como el que puede apreciarse en la siguiente imagen, que no nos da ningún indicio de a qué se debe:

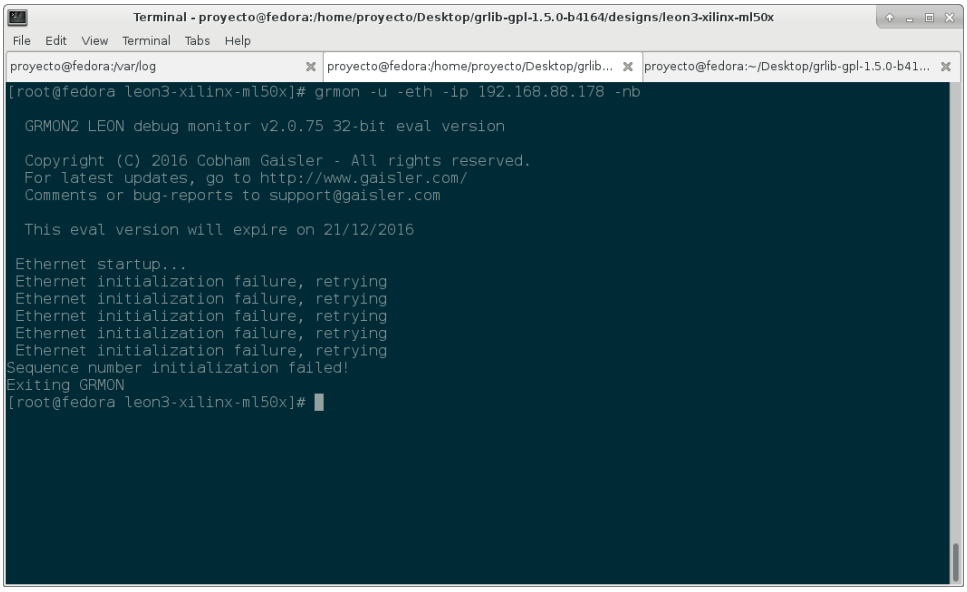

Figura 30: GRMON ethernet: Error timeout.

Para evitar esto y podernos conectar correctamente a la placa, es necesario ejecutar los siguientes comandos:

```
1  # firewall-cmd --zone=public --list-all
2  public
3  # firewall-cmd --zone=public --permanent --add-rich
     -rule='rule family="ipv4" source address
     ="192.168.88.178" accept'
4  success
```

Ahora deberíamos de ser capaces de conectarnos correctamente y, si le pedimos la lista de reglas al firewall de linux, debería de devolvernos algo parecido a ésto:

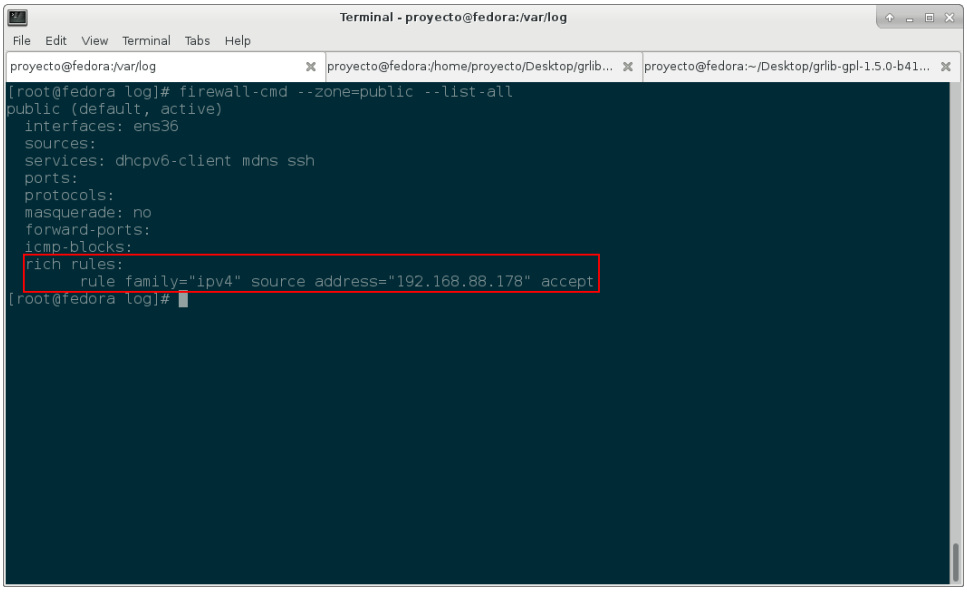

Figura 31: GRMON ethernet: Reglas del firewall.

Ésta nueva regla que añadiríamos al firewall, aceptaría todas las conexiones provenientes de la IP que hemos asignado a nuestra placa. Podríamos hacer que nuestra conexión solo usara un puerto en concreto mediante la opción "-udp" a la hora de invocar el depurador, pero hemos preferido usar una opción menos restrictiva puesto que se supone que estamos en un entorno de desarrollo y no debería preocuparnos mucho la seguridad de la red a la que estemos conectados.

4.2.3. Conexión con la placa por el Puerto USB.

Debido a que nuestro objetivo principal es usar un cluster tipo "MIPCH" usando la conexion Ethernet de nuestra imagen Linux compilada, usar el "debugger" mediante la interfaz de red supone una merma en la capacidad de dicho puerto Ethernet, lo cual no es deseable.
Por eso nosotros usaremos el tercer método aquí descrito para conectarnos y acceder al sistema LEON generado anteriormente. Basta con tener el cable USB conectado a la placa a través de nuestra interfaz XILINX USB y usar el siguiente comando:

```
1  # grmon -xilusb -u -nb
```

Solo recordar que hemos dejado una compilación con la configuración necesaria dentro de la máquina virtual en la ruta "/home/proyecto/Desktop/compilacion_leon_no_ethernet". La peculiaridad de dicha configuración reside en desactivar el puerto de red como opción de depuración, tal y como puede apreciarse en la siguiente figura 32:

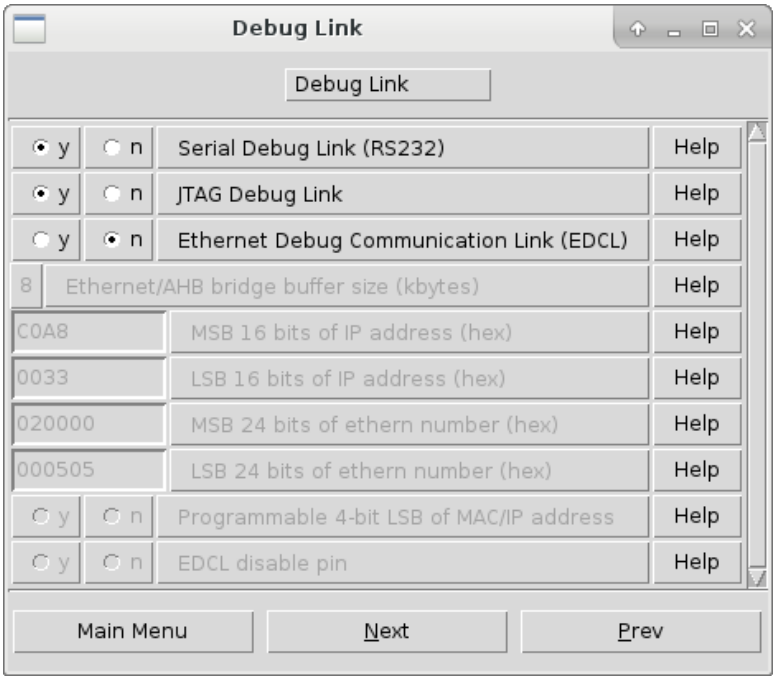

Figura 32: GRLIB compilación: RJ45 desactivado.

Un apunte que deberíamos de hacer es que el depurador GRMON solo admite el uso de modelos depuradores de XILINX DLC9G y DLC10. El resto de modelos no están soportados.
Deberíamos obtener, aunque a una velocidad menor de transferencia, una

pantalla extactamente igual a la de Conexión con la placa por el Puerto Serie.

4.2.4. Configuración de la placa con FPU.

Dado que vamos a necesitar usar comandos que van a requerir una unidad de coma flotante (O Floating Point Unit - FPU por sus siglas en inglés[45]) configuraremos por última vez nuestro modelo VHDL para que soporte una unidad FPU.
A este fin, deberemos de descargarnos las ampliaciones a los diseños que provee Gaisler en su sección de descargas bajo el nombre de "GRFPU netlists for Xilinx and Altera".
Una vez descargado dicho archivo (Que podemos encontrar en nuestra carpeta de instaladores bajo el nombre "grlib-netlists-gpl-1.5.0.tar.gz"), deberíamos de descomprimir sus contenidos en el directorio donde hayamos descomprimido el archivo "grlib-gpl-1.5.0-b4164.tar.gz ", de manera que quede del siguiente modo:

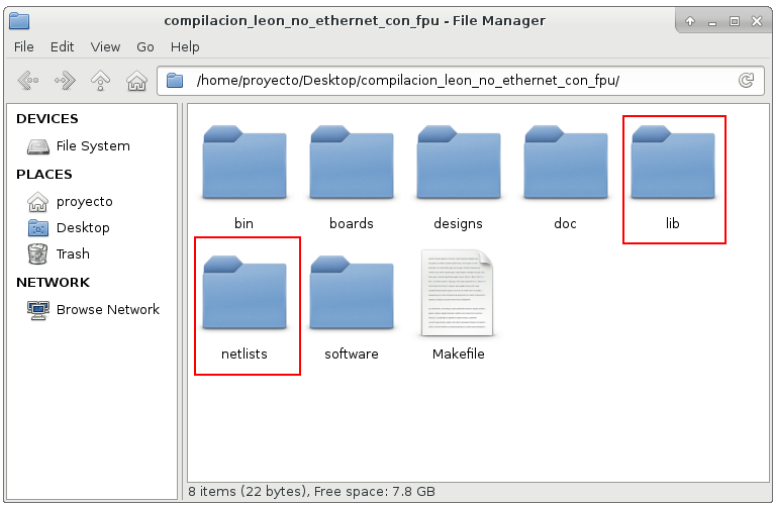

Figura 33: GRLIB compilación: Definiciones FPU en el sistema de archivos.

Ahora deberíamos de indicar al programa de compilación de Gaisler que active dicha FPU, con lo que abriríamos el menú de configuración para la síntesis VHDL y seleccionaríamos las siguientes opciones:

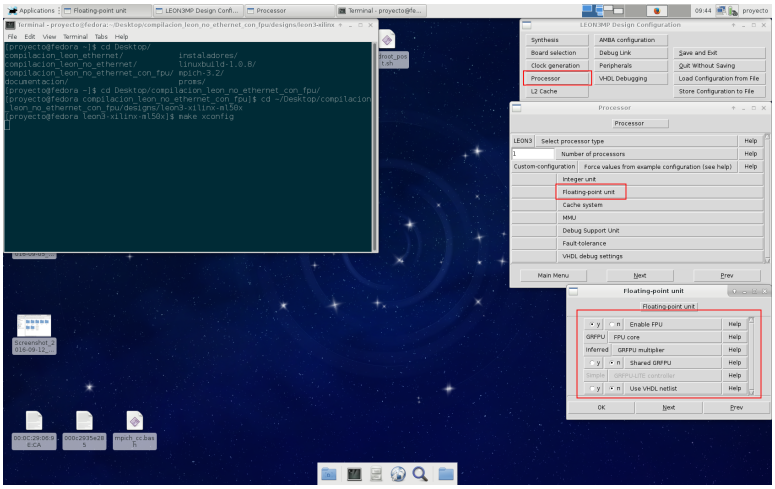

Figura 34: GRLIB compilación: Opciones en el menú de configuración.

Una vez seleccionados dichos cambios, solo nos quedaría volver a crear el archivo bitstream para, después, subirlo a la placa, tal y como explicamos en Compilación de LEON3.

5. Configuración de las imágenes de LINUX.

En las anteriores secciones hemos visto como configurar, compilar y transferir una serie de controladores para la placa que, usando una analogía con un PC de escritorio, funcionarían como el "BIOS" de la ML507.
Veremos ahora como poder configurar nuestras imágenes de Linux para poder volcarlas en la placa y poder hacer que estas funcionen bajo dicho Sistema Operativo.
Antes de eso, deberemos de configurar una serie de herramientas en nuestra Máquina Virtual para poder interaccionar correctamente con los paquetes de instalación que Gaisler nos ofrece a dicho propósito [18].
Para este proceso, hemos usado varios manuales, entre ellos el que se ofrece en la anterior página bajo el nombre de "linuxbuild-x.y.z.pdf". En nuestro caso la versión era la "1.0.8".
Como último apunte, recordaremos de nuevo que todos estos paquetes de compilación funcionan solo bajo sistemas Linux de 64 bits. Para saber si estamos usando el Sistema Operativo adecuado, ejecutamos el siguiente comando:

```
1  # getconf LONG_BIT
```

Si nos devuelve un "64", estamos en el entorno adecuado.

5.1. Instalación del compilador SPARC.

A continuación nos bajaremos el paquete necesario del repositorio de Gaisler y lo instalaremos en la ruta recomendada, además de añadirlo al "PATH" de LINUX.

```
1  # wget http://gaisler.com/anonftp/linux/linux-2.6/
       toolchains/sparc-linux-4.4.2/sparc-linux-ct-
       multilib-0.0.7.tar.bz2
2  # apt-get install gcc binutils make pkg-config git
       bison flex msgfmt gettext texinfo
3  # cp sparc-linux-ct-multilib-0.0.7.tar.bz2 /opt/
4  # cd /opt/
5  # tar -xjvf sparc-linux-ct-multilib-0.0.7.tar.bz2
6  # chmod -R 777 sparc-linux-4.4.2-toolchains/
7  # rm sparc-linux-ct-multilib-0.0.7.tar.bz2
8  # echo "PATH=\$PATH:/opt/sparc-linux-4.4.2-
       toolchains/multilib/bin" >> ~/.bashrc
```

Una vez ejecutados todos estos comandos satisfactoriamente, tendríamos que tener los ejecutables de compilación listos. Para comprobarlo hacemos lo siguiente:

```
1  # which sparc-linux-gcc
```

Y nos debería devolver lo siguiente (Teniendo en cuenta la versión que estemos usando, en mi caso la 4.4.2):

```
1  /opt/sparc-linux-4.4.2-toolchains/multilib/bin/
       sparc-linux-gcc
```

5.2. Instalación del entorno visual de configuración de imágenes de LINUX de Gaisler.

Para usar correctamente los paquetes instaladores que Gaisler provee de manera gratuita, hemos de configurar el entorno para que funcione con el "Framework" QT, el cual lo haremos desde la linea de comandos:

```
1  # sudo dnf install -y qt-devel gcc-c++ patch xterm
```

Si todo se descarga e instala correctamente, tendríamos que ser capaces de ejecutar el comando:

```
1  # make xconfig
```

En el se nos aparecerá una pantalla como la que podemos ver a continuación 35 y en la que deberemos de configurar los diferentes aspectos de la imagen de linux que vamos a hospedar en la placa.

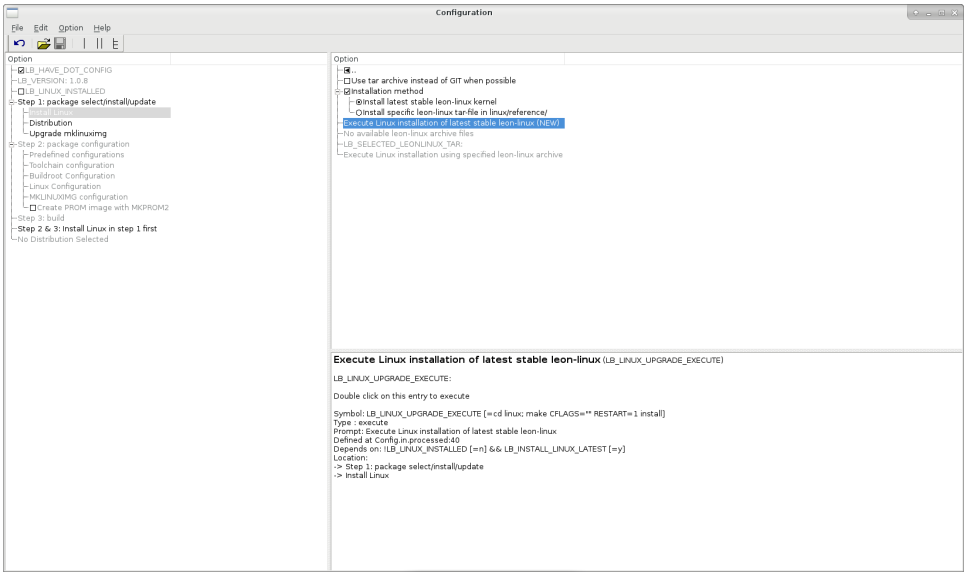

Figura 35: Linux Gaisler: Menú principal.

5.3. Compilación de Buildroot simple.

Para empezar con este proceso, deberíamos descargar la última distribución de Linux disponible para la herramienta 'linuxbuild" de Gaisler, seleccionando las siguientes opciones en el primer paso del menú:

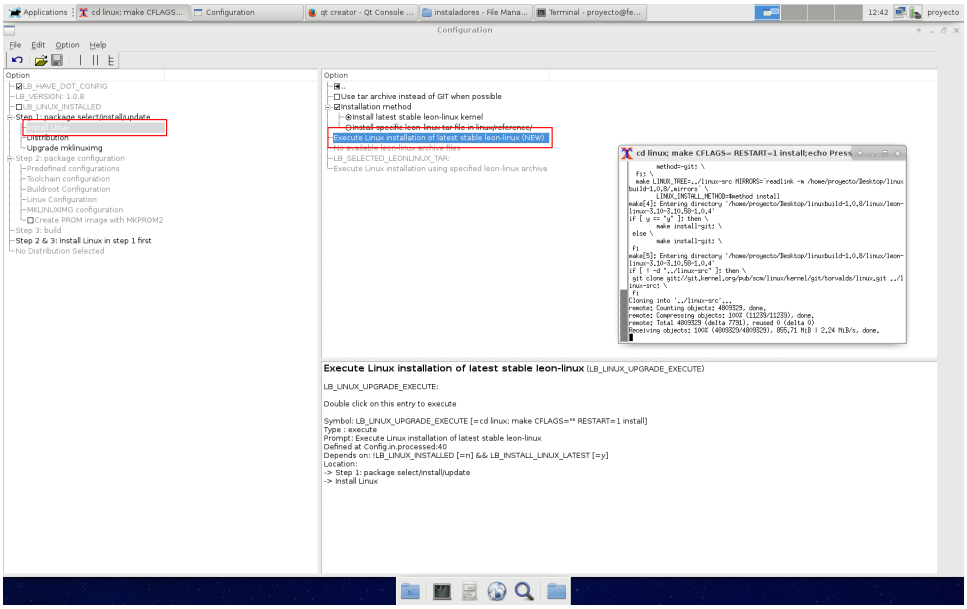

Figura 36: Linux Gaisler: Descarga de imagen Linux.

Como podemos ver en el terminal que aparece en la anterior imagen, nos estamos descargando la imagen oficial de Linux del repositorio GIT[20] de Linus Torvalds (Creador de Linux)[21].
Una vez que dicho proceso acabe, veremos que en el menú ahora nos aparecen nuevas opciones para configurar dicha imagen, como podemos ver a continuación:

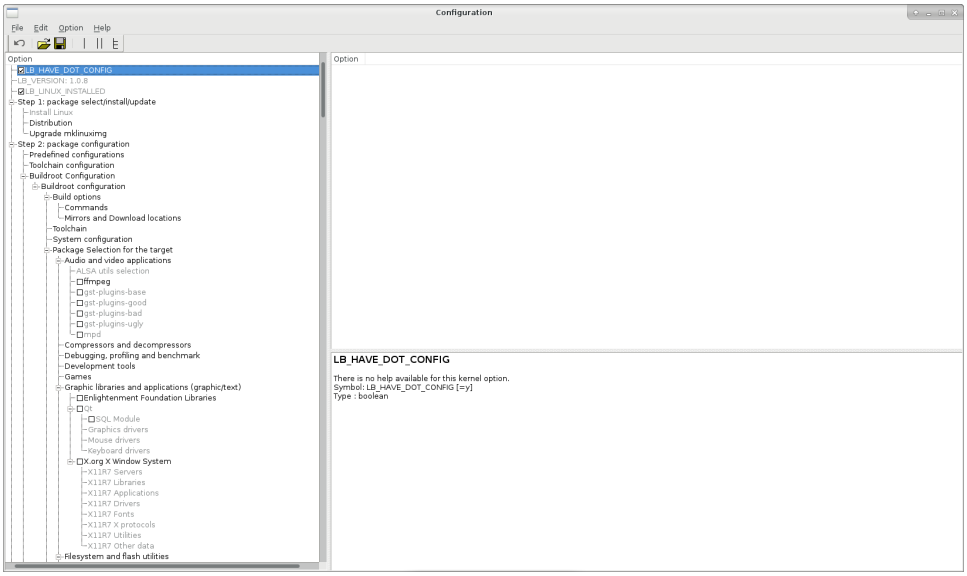

Figura 37: Linux Gaisler: Menú principal con opciones extendidas.

Las opciones que esta utilidad contiene son muchas y están fuera del alcance de lo que se pretende en este proyecto. Basta con saber que vamos a cargar una configuración predefinida sobre la que modificaremos algunas opciones.
Para ello, iremos a la opción "Predefined configurations" y elegiremos "lb_config_leon-linux-3.10_up_soft.tar.bz2", como puede verse a continuación:

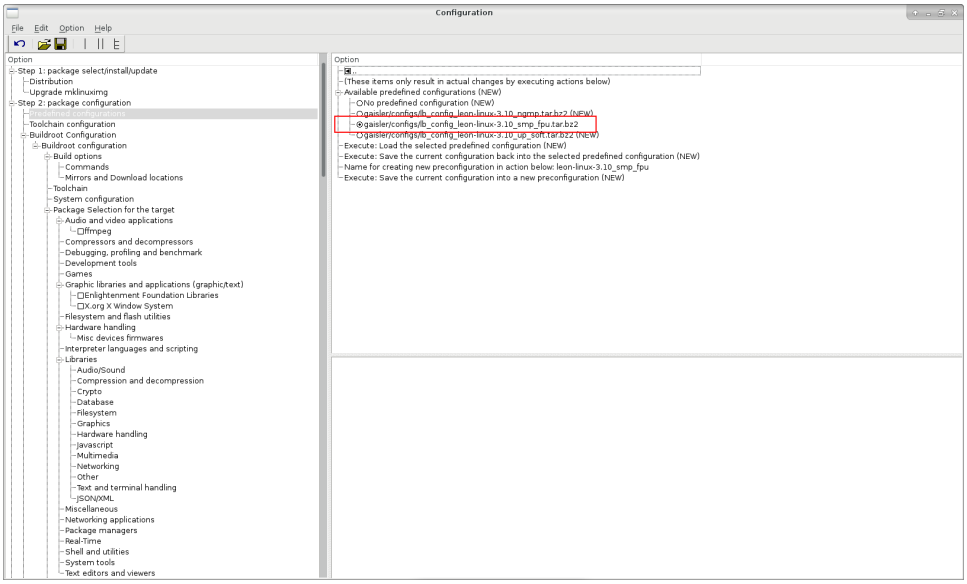

Figura 38: Linux Gaisler: Selección de configuraciones predefinidas.

Ahora solo nos quedaría lanzar la compilación para crear la imagen de carga en RAM de linux, como puede apreciarse en la figura 39 . Pero antes, deberíamos de instalar ciertos paquetes necesarios para dicha compilación:

```
1   # sudo dnf install -y bison flex texinfo
```

Con este último paso tendremos nuestro entorno configurado para usar las herramientas de Gaisler de configuración de imágenes Linux, mediante la siguiente opción:

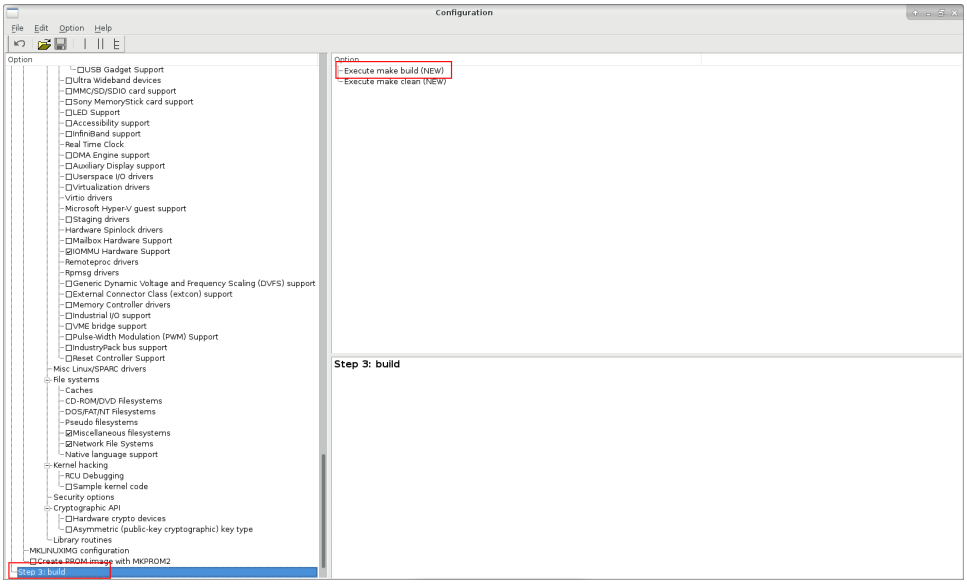

Figura 39: Linux Gaisler: Compilación de la imagen Linux.

Tras un corto periodo de tiempo, nuestra imagen debería de estar lista para ser cargada en la placa en el directorio “output” de la suite “linuxbuild”.

6. Instalación de la distribución Buildroot en la placa.

Para conseguir ejecutar la distribución Buildroot en la placa Xilinx deberíamos de iniciar el depurador "GRMON" y cargar la imagen creada en anterior apartado en la memoria de nuestra ML507 para, a continuación, ejecutarla.
Un detalle muy importante y del que no deberíamos olvidarnos es que estamos ejecutando la imagen en memoria RAM (Como puede apreciarse en la figura 40), con lo cual todo el trabajo que hagamos se perderá cuando la imagen de Linux deje de correr en la placa. En la sección Personalización de la distribución Buildroot , estudiaremos como hacer todos estos cambios persistentes o automáticos.

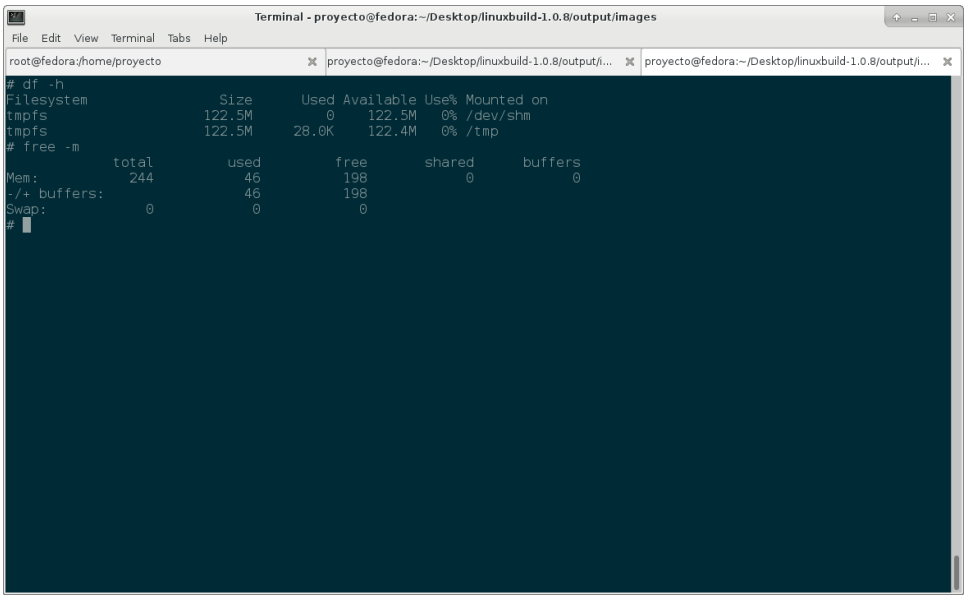

Figura 40: Buildroot: Memoria RAM y espacio en disco.

Empezaremos cambiando nuestro directorio de trabajo al del subdirectorio "output" de nuestra instalación "linuxbuild" para tener todos los archivos de nuestra anterior compilación (Las imagenes RAM de la sección Compilación de Buildroot simple) inmediatamente disponibles, sin tener que referenciarlos indirectamente, ya que esto podría ocasionarnos problemas con el depurador.
En el caso de nuestra máquina virtual, ejecutamos el siguiente comando:

```
# cd /home/proyecto/Desktop/linuxbuild-1.0.8/output
    /images
```

Una vez en dicho directorio, usando las conexiones del puerto serie o el puerto USB descritas en Instalación de la interfaz GRMON, entraríamos en el depurador "GRMON".
En nuestro caso, hemos elegido hacer dicha conexión a través del cable USB y por ello ejecutaremos el siguiente comando:

```
# grmon -xilusb -u -nb
```

Es muy importante usar la opción "-nb" ya que, sin ella, nuestra carga en la placa de la imagen RAM de Linux se interrumpirá, como puede leerse en la documentación acerca de este proceso [16].
Una vez estemos ejecutando el depurador, debemos de cargar la imagen RAM en la placa, mediante los siguientes comandos:

```
grmon2> load image.ram
```

El retorno de dicho comando debería ser el siguiente:

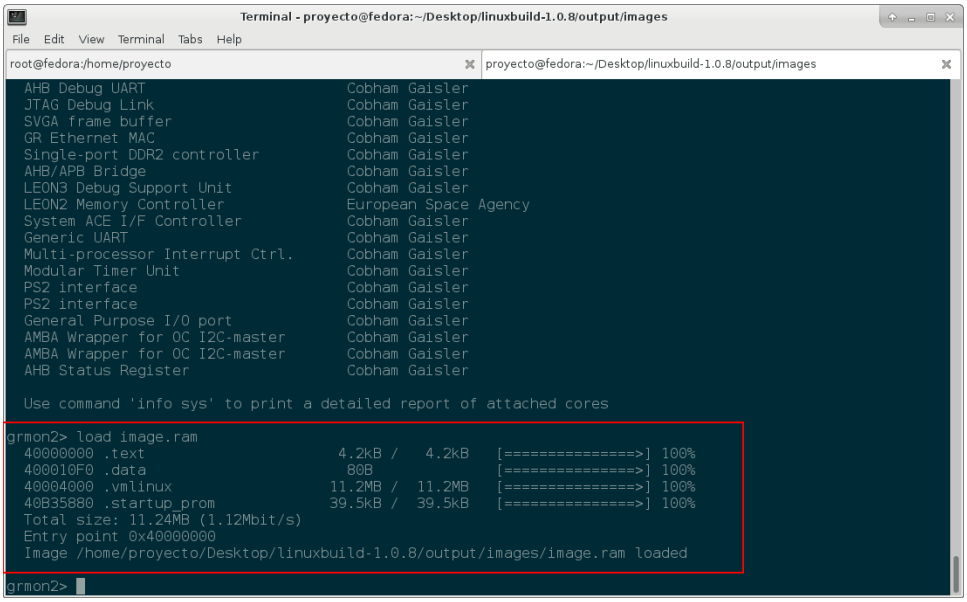

Figura 41: Grmon: Carga de imagen RAM.

Y una vez se complete, deberíamos cargar en memoria dicha imagen mediante el comando:

```
grmon2> run
```

Lo cual nos daría un texto similar al siguiente:

```
grmon2> run
PROMLIB: Sun Boot Prom Version 0 Revision 0
Linux version 3.10.58-00019-gca4e47c (proyecto@fedora
   .proyecto) (gcc version 4.4.2 (sparc-linux-ct-
   leon_multilib_basic-0.0.7) ) #2 SMP Sat Aug 13
   12:35:13 BST 2016
bootconsole [earlyprom0] enabled
ARCH: LEON
TYPE: Leon3 System-on-a-Chip
Ethernet address: 00:00:7c:cc:01:45
CACHE: 4-way associative cache, set size 4k
63MB HIGHMEM available.
OF stdout device is: /a::a
PROM: Built device tree with 19180 bytes of memory.
Booting Linux...
PERCPU: Embedded 7 pages/cpu @f108c000 s6272 r8192
   d14208 u32768
Built 1 zonelists in Zone order, mobility grouping on
   .  Total pages: 62243
Kernel command line: console=ttyS0,38400 init=/sbin/
   init
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5,
   131072 bytes)
Inode-cache hash table entries: 16384 (order: 4,
   65536 bytes)
Sorting __ex_table...
Memory: 245120k/262120k available (4260k kernel code,
    17000k reserved, 1496k data, 5696k init, 65512k
   highmem)
Hierarchical RCU implementation.
  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids
     =1.
NR_IRQS:64
Console: colour dummy device 80x25
console [ttyS0] enabled, bootconsole disabled
console [ttyS0] enabled, bootconsole disabled
Calibrating delay loop... 59.39 BogoMIPS (lpj=296960)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
Entering SMP Mode...
leon: SMP IPIs at IRQ 13
```

```
0:(1:4) cpus mpirq at 0x80000210
######## !!!! The irqmp-ctrl must have broadcast
   enabled, smp wont work !!!!! ####### nr cpus: 1
continue anyway
Brought up 1 CPUs
Total of 1 processors activated (59.39 BogoMIPS).
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
vgaarb: loaded
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
Switching to clocksource timer_cs
FS-Cache: Loaded
CacheFiles: Loaded
NET: Registered protocol family 2
TCP established hash table entries: 2048 (order: 2,
   16384 bytes)
TCP bind hash table entries: 2048 (order: 2, 16384
   bytes)
TCP: Hash tables configured (established 2048 bind
   2048)
TCP: reno registered
UDP hash table entries: 256 (order: 1, 8192 bytes)
UDP-Lite hash table entries: 256 (order: 1, 8192
   bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport
   module.
bounce pool size: 64 pages
FS-Cache: Netfs 'nfs' registered for caching
NFS: Registering the id_resolver key type
Key type id_resolver registered
Key type id_legacy registered
ROMFS MTD (C) 2007 Red Hat, Inc.
JFS: nTxBlock = 1915, nTxLock = 15320
msgmni has been set to 350
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
```

```
70 Serial: GRLIB APBUART driver
71 ffd0f7a8: ttyS0 at MMIO 0x80000100 (irq = 4) is a
      GRLIB/APBUART
72 grlib-apbuart at 0x80000100, irq 4
73 brd: module loaded
74 loop: module loaded
75 ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI)
      Driver
76 ehci-pci: EHCI PCI platform driver
77 uhci_hcd: USB Universal Host Controller Interface
      driver
78 usbcore: registered new interface driver usblp
79 usbcore: registered new interface driver usb-storage
80 usbcore: registered new interface driver usbserial
81 usbcore: registered new interface driver
      usbserial_generic
82 usbserial: USB Serial support registered for generic
83 usbcore: registered new interface driver belkin_sa
84 usbserial: USB Serial support registered for Belkin /
       Peracom / GoHubs USB Serial Adapter
85 usbcore: registered new interface driver ftdi_sio
86 usbserial: USB Serial support registered for FTDI USB
       Serial Device
87 grlib-apbps2 ffd0f548: irq = 6, base = 0xfd003400
88 grlib-apbps2 ffd0f480: irq = 7, base = 0xfd004500
89 mousedev: PS/2 mouse device common for all mice
90 usbcore: registered new interface driver usbhid
91 usbhid: USB HID core driver
92 TCP: cubic registered
93 NET: Registered protocol family 10
94 sit: IPv6 over IPv4 tunneling driver
95 NET: Registered protocol family 17
96 Key type dns_resolver registered
97 leon: power management initialized
98 /home/proyecto/Desktop/linuxbuild-1.0.8/linux/linux-
      src/drivers/rtc/hctosys.c: unable to open rtc
      device (rtc0)
99 Freeing unused kernel memory: 5696K (f05a6000 -
      f0b36000)
100 Starting logging: OK
101 atkbd serio0: keyboard reset failed on apbps2_0
102 Initializing random number generator... done.
103 Starting network...
104
```

```
105 Welcome to Buildroot
106 buildroatkbd serio1: keyboard reset failed on
       apbps2_1
107 atkbd serio0: keyboard reset failed on apbps2_0
108 atkbd serio1: keyboard reset failed on apbps2_1
109 ot login:
110 Welcome to Buildroot
111 buildroot login: root
112 #
```

Nótese que hemos tenido que presionar la tecla "Intro" cuando llegamos a la parte en la que nos muestra "Welcome to Buildroot" para que nos pregunte acerca del usuario con el que vamos a entrar en la distribución Buildroot. Dicho usuario será "root", como hemos visto en el anterior "output".
Una vez dentro de la distribución que corre en nuestra placa, se nos presentará una linea de comandos muy parecida a la que hemos estado usando en nuestra VM.

6.0.1. Configuración de la interfaz de Red en nuestra placa con Linux.

Debido a la configuración por defecto que hemos usado en los pasos anteriores, nuestra distribución viene con la interfaz de red activada pero sin configuración alguna. Para hacer que coja una IP usando el protocolo DHCP deberíamos de usar los siguientes comandos [22]:

```
1  # echo -e "auto eth0\niface eth0 inet dhcp" >> /etc
       /network/interfaces
2  # ifup eth0
```

Cabe comentar el hecho de que, para hacer más liviana la carga de la imagen Linux en RAM, hemos desactivado el protocolo ipv6 del menú "linuxbuild", como podemos observar a continuación:

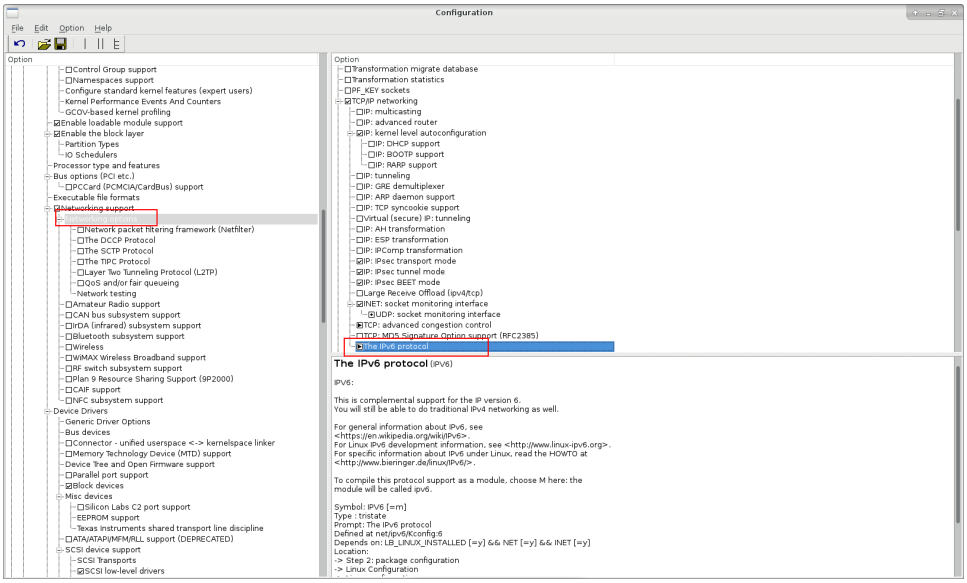

Figura 42: Buildroot: IPv6 desactivado.

Una vez ejecutados dichos comandos, deberíamos de ser capaces de hacer ping a cualquier host que esté conectado a Internet. Como ejemplo, pondremos los resultados de hemos obtenido:

```
1  Welcome to Buildroot
2  buildroot login: root
3  # echo -e "auto eth0\niface eth0 inet dhcp" >> /etc
       /network/interfaces
4  # ifup eth0
```

```
udhcpc (v1.21.0) started
Sending discover...
Sending select for 192.168.88.241...
Lease of 192.168.88.241 obtained, lease time 600
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.88.1
adding dns 194.168.4.100
adding dns 194.168.8.100
# ping -c 4 google.com
PING google.com (62.252.232.50): 56 data bytes
64 bytes from 62.252.232.50: seq=0 ttl=58 time
   =16.513 ms
64 bytes from 62.252.232.50: seq=1 ttl=58 time
   =19.249 ms
64 bytes from 62.252.232.50: seq=2 ttl=58 time
   =20.158 ms
64 bytes from 62.252.232.50: seq=3 ttl=58 time
   =21.337 ms

--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0%
   packet loss
round-trip min/avg/max = 16.513/19.314/21.337 ms
#
```

Para poder automatizar ciertos procesos, asignaremos una IP a nuestro Linux embebido en la placa ml507 usando la siguiente configuración:

```
# echo -e "auto eth0\niface eth0 inet static\niface
    eth0 inet static\naddress 192.168.88.166\
   nnetwork 192.168.88.0\nnetmask 255.255.255.0\
   nbroadcast 192.168.88.255\ngateway 192.168.88.1"
    >> /etc/network/interfaces
# ifup eth0
```

Como podemos observar, asignaremos la IP 192.168.88.166 con una máscara de red de 255.255.255.0 para que esté de acorde a nuestra red local.

6.1. Configuración del protocolo SSH en nuestra placa con Linux.

Con el fin de tener el protocolo SSH activado en nuestra distribución, deberemos de activar la siguiente opción en el menú de configuración de "linuxbuild":

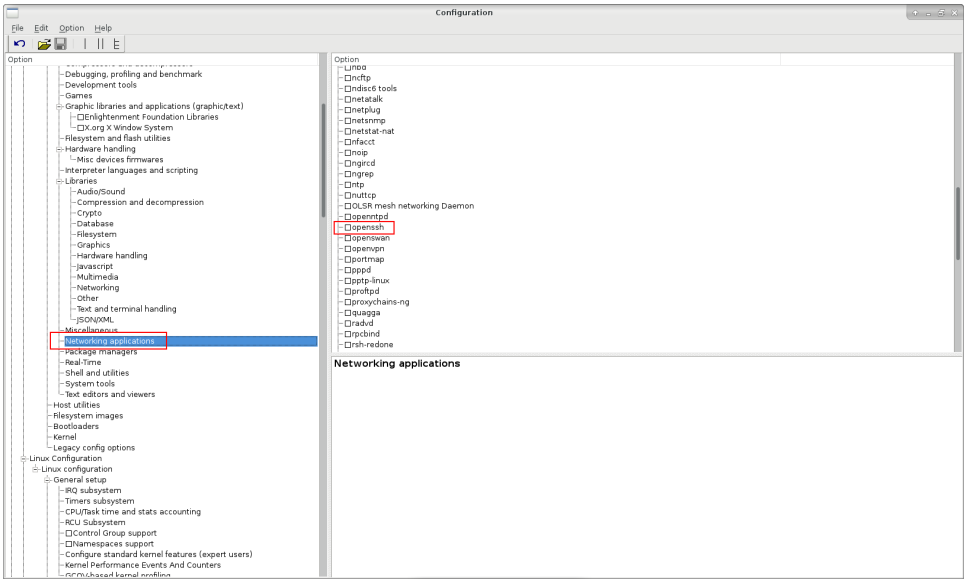

Figura 43: Buildroot: Activación del protocolo SSH.

Si todo funciona correctamente, cada vez que carguemos la imagen de la placa en la memoria RAM se generarán las claves RSA y DSA[28], lo cual es un inconveniente ya que nuestra placa tiene unos recursos más limitados que un ordenador de sobremesa normal y tarda entre 5 y 10 minutos en generarlas.
Una muestra del proceso de generación en nuestra placa es la siguiente:

```
1 Generating RSA Key...
2 Generating public/private rsa1 key pair.
3 Your identification has been saved in /etc/
    ssh_host_key.
4 Your public key has been saved in /etc/ssh_host_key.
    pub.
5 The key fingerprint is:
6 b1:42:50:e5:2e:19:c6:a9:7c:f5:a2:e4:dc:5f:94:d3
7 The key's randomart image is:
8 +--[RSA1 2048]----+
```

```
|     .....         |
|      o o          |
|       * +         |
|    . + = +    o   |
|     o * S  . + E  |
|      = = .  .  .  |
|       + .    .    |
|          . .      |
|           .       |
+-----------------+
Generating RSA Key...
Generating public/private rsa key pair.
Your identification has been saved in /etc/
   ssh_host_rsa_key.
Your public key has been saved in /etc/
   ssh_host_rsa_key.pub.
The key fingerprint is:
9b:ca:e7:15:cd:05:bf:59:a6:c1:60:e5:9e:7a:3b:de
The key's randomart image is:
+--[ RSA 2048]----+
|             +..  |
|            . *   |
|               * o|
|            o o O |
|         S . o *  |
|          o . .   |
|         o . . .  |
|      . ...   ..o |
|       oo.    .o.E|
+-----------------+
Generating DSA Key...
THIS CAN TAKE A MINUTE OR TWO DEPENDING ON YOUR
   PROCESSOR!

Generating public/private dsa key pair.
Your identification has been saved in /etc/
   ssh_host_dsa_key.
Your public key has been saved in /etc/
   ssh_host_dsa_key.pub.
The key fingerprint is:
41:83:32:4b:df:41:05:73:6d:4b:ca:a3:e4:00:61:d4
The key's randomart image is:
+--[ DSA 1024]----+
|    .+o o*oo.     |
```

```
|    .= E..+   +     |
|    . * ..o + .     |
|     . o o.+ .      |
|        +S. .       |
|         o          |
|                    |
|                    |
|
Welcome to Buildroot
buildroot login:
```

A fin de no repetir este proceso cada vez que cargamos la imagen en la placa, copiamos unas claves creadas en un proceso anterior al sistema de archivos raíz antes de que se ensamble la memoria RAM. Para más información, por favor vaya a Personalización de la distribución Buildroot.
Es interesante destacar que deberemos de iniciar el servicio SSH en nuestra máquina virtual, la cual va a funcionar de nodo maestro, para que la aplicación MPICH funcione correctamente, de la siguiente manera:

```
# systemctl start sshd.service
# systemctl enable sshd.service
```

El último comando es para que dicho servicio se inicie siempre que reiniciemos la máquina virtual.

6.2. Cambio de contraseña del usuario “root”.

Para poder establecer conexiones SSH con la configuración que hemos dejado en el punto anterior, deberíamos de establecer la contraseña de dicho usuario. Para ello ejecutaremos los siguientes comandos:

```
# passwd
Changing password for root
New password:
Bad password: too weak
Retype password:
Password for root changed by root
```

Nosotros hemos establecido la misma contraseña para nuestra imagen RAM que para nuestra máquina virtual (“proyecto.”). Una vez finalizado este paso y habiendo completado los dos anteriores, deberíamos de ser capaces de conectarnos a nuestra máquina usando el protocolo SSH:

```
ssh 192.168.88.166
root@192.168.88.166's password:
#
```

6.3. Usando BASH para computación en paralelo.

Ahora ejemplificaremos un uso de computación en paralelo ineficiente. Para ello ejecutaremos el contenido del siguiente script cuya ubicación se encuentra en "/home/proyecto/Desktop/instaladores/examples":

```
#!/bin/sh
# SCRIPT: primenumbers.sh
# USAGE : ./primenumbers.sh <Range Value>
#                       or;
#          ./ primenumbers.sh <Start Range Value> <End Range Value>
# PURPOSE: Produce prime numbers within a range lmit.


################ DEFINE FUNCTIONS HERE ##############

Usage()
{
    echo "********BAD ARGUMENTS**********"
    echo "Usage: scriptname <Range Value >"
    echo "                or                    "
    echo "Usage: scriptname <Start Range Value> <End Range Value>"
    exit 1
}

################## ARGUMENTS CHECKING ##################

[ $# -gt 2 -o $# -lt 1 ] && Usage
[ $# -eq 1 ] && Bnum=2 &&  Enum=$1
[ $# -eq 2 ] && Bnum=$1  Enum=$2 && [ $1 -gt $2 ] && Usage
[ $1 -lt 2 ] && Bnum=2

################ MAIN PROGRAM STARTS HERE ##############

calculo_simple(){
count=1

while [ $Bnum -le $Enum ]
do
  num=$Bnum
```

```
35    Prime="yes"
36    i=1
37
38    #while [ $i -lt $((num/2)) ]
39    while [ $((i*i)) -lt $((num-1)) ]
40    do
41      let i++
42      if [ $((num%i)) -eq 0 ]           # you can also
          use `expr $num % $i`
43      then
44        Prime="no"
45        break
46      fi
47    done
48
49    [ "$Prime" = "yes" ] && printf "%5d " $num && let
        count++
50
51    # count is used to print 10 values in a row
52
53    [ $count -eq 2 ] && count=1 &&  echo
54    let Bnum++
55  done
56  echo
57  }
58
59  #Dividimos el maximo
60
61  percent=9968
62  MID=$(($Enum*$percent/10000))
63
64  ssh root@node1 "sh -s $MID $Enum" < prime_single.bash
      &
65  sh -s $Bnum $MID < prime_single.bash
```

Su nombre es "prime_multi.bash" y lo que hace es usar dos llamadas en paralelo: Una a nuestra máquina virtual y la otra a nuestra placa mediante un comando SSH.
El objetivo de este script sería calcular y listar los números primos que se encuentran en un rango. Como es una actividad que tarda cierto tiempo, hemos destinado una parte de este cálculo a la placa para agilizar el procedimiento, aunque solo un porcentaje muy bajo, dada la diferencia computacional entre nuestra placa y nuestra máquina virtual.
Como podemos comprobar a continuación, los resultados son casi insatis-

factorios cuando buscamos los números primos que hay entre 2 y 100000 y damos un 0.32 % de dichos cálculo a nuestro SoC:

```
[root@fedora examples]# time ./prime_multi.bash 100000
     2     3     5     7    11    13    17    19    23
            29    31    37    41    43    47    53
        59    61    67    71
...
99661 99667 99679

real  0m33.537s
user  0m30.333s
sys 0m1.354s
[root@fedora examples]# time ./prime_single.bash 100000
     2     3     5     7    11    13    17    19    23
            29    31    37    41    43    47    53
        59    61    67    71
...
99859 99871 99877 99881 99901 99907 99923 99929 99961
    99971 99989 99991

real  0m34.415s
user  0m31.577s
sys 0m2.166s
```

Para paliar esta situación, instalaremos el entorno de computación en paralelo MPICH.

6.4. Compilando un programa para nuestro Linux en la placa ml507.

A modo didáctico, explicaremos aquí el proceso para compilar un archivo en C para general un ejecutable que funcione en nuestra placa.
Tenemos un programa que nos calcula el valor de Pi dadas "n" iteraciones. Podemos encontrar el archivo en la ruta "/home/proyecto/Desktop/instaladores/examples/calculo_pi.c" y la forma de compilarlo sería la siguiente:

```
1  # sparc-linux-gcc -msoft-float -o /home/proyecto/
     Desktop/calculopi /home/proyecto/Desktop/
     instaladores/examples/calculo_pi.c
```

Como vemos, usamos de compilador la herramienta de Gaisler "sparc-linux-gcc" para compilar dicho archivo en un ejecutable. Es importante reseñar que, como nuestro SoC Leon3 no tiene una FPU, debemos usar la opción "-msoft-float" si no queremos que la placa se quede en un estado instable y tengamos que reiniciar.
Una vez lanzado el ejecutable deberíamos de ver una salida parecida a esta:

```
1 # ./calculopi 1000
2 My value of Pi: 3.142592, math.h's value of Pi:
    3.1415926535897931
3 Absolute difference: 0.0009990007497511
```

Cuanto mayor sea el número de iteraciones que usemos en dicho programa, mayor será la precisión y, por consiguiente, tardará más.

7. MPICH - Computación en paralelo.

MPICH es una implementación de alto rendimiento y áltamente portable con licencia gratuita de MPI, el interfaz de envío de mensajes (Message Parsing Interface por sus siglas en inglés). Es un estándar en aplicaciones distribuidas en memoria para computación en paralelo.
Sus objetivos son principalmente dos:

- Proveer una implementación de MPI que soporte diferentes sistemas de computación y comunicación, incluyendo sistemas comunes (Como clústeres de ordenadores o sistemas con múltiples procesadores), redes de alta velocidad (Gigabit Ethernet, InfiniBand, ...) y sistemas de alto rendimiento propietarios (Super ordenadores como Cray o Blue Gene).
- Permitir investigaciones punteras usar MPI a través de una aplicación extensamente modular y fácilemente ampliable.

Fue desarrollada por el Laboratorio Nacional de Argonne en 1992 cuando se estaban revisando los estándares MPI. La primera versión estaba basada en la portabilidad de un protocolo anterior conocido como Chameleon (Por eso el nombre MPICH).
La nueva versión incorpora mejoras al protocolo como reducción de su huella en el Sistema Operativo y uso de sistemas Gigabit Ethernet.
Cabe destacar que en 2015 era la herramienta de computación usada en 9 de los 10 mejores supercomputadores en el mundo.[33]

7.1. Instalación de herramientas de compilación cruzada.

Debido a que nuestra instalación de Linux en la placa no cuenta con ningún compilador dentro de ella, debemos de compilar los programas que deseemos ejecutar en la placa con un compilador cargado con las opciones de la placa.
Es por esto que usaremos el paquete descrito en la sección 5.1 para crear los ejecutables que más tarde usaremos en nuestra Xilinx ml507.
A tal fin, crearemos un directorio con los paquetes de MPICH en nuestro escritorio, tras instalar los paquetes necesarios para su compilación:

```
1  # sudo dnf -y install automake libtool cvs lzma
       gperf help2man
2  # tar -xf /home/proyecto/Desktop/instaladores/mpich
       -3.2.tar.gz -C /home/proyecto/Desktop/
```

Como siempre, podemos encontrar el archivo con el código fuente de MPICH en nuestra carpeta de instaladores.
Continuaremos especificando las opciones necesarias para crear un ejecutable válido de MPICH para nuestra placa:

```
1 ./configure --host=sparc-leon-linux-gnu  CC=sparc-
     leon-linux-gnu-gcc --disable-fortran --with-atomic
     -primitives=no --disable-cxx --prefix=/home/
     proyecto/Desktop/instaladores/mpichroot/
     mpichcompiladoleon/
2 make
3 sudo make install
```

Como puede observarse en el "configure" de la figura anterior, desactivamos numerosas opciones de MPICH para ser capaz de crear un ejecutable que podamos portar a nuestra placa y especificamos que copiaremos todo el conjunto de archivos al directorio "/home/proyecto/Desktop/instaladores/mpichroot/mpichcompiladoleon/".
Dicha necesidad de limitar nuestro ejecutable viene dada por el compilador de Gaisler para Leon: Simplemente, no está preparado para ciertas operaciones que han de ser desactivadas.
Tras unos minutos, nuestros ficheros deberían de estar listos en la dirección que hemos asignado en el "configure". Ahora copiaremos dichos archivos a nuestra placa mediante el comando "scp". Como vamos a tener que acceder bastante a la placa usando el protocolo SSH, copiaremos nuestra identidad en la placa y, así, nos ahorraremos de escribir la contraseña siempre que queramos acceder a ella:

```
1 ssh-copy-id 192.168.88.166
2 /bin/ssh-copy-id: INFO: Source of key(s) to be
     installed: "/root/.ssh/id_rsa.pub"
```

```
3 The authenticity of host '192.168.88.208
    (192.168.88.166)' can't be established.
4 ECDSA key fingerprint is SHA256:
    bbh6Mx2di0uIn1cKRF3PkDaWOTWkQ7X+fw3n58X22uQ.
5 ECDSA key fingerprint is MD5:38:bb:23:0d:0c:f2
    :82:69:52:d1:b2:29:b2:9e:16:42.
6 Are you sure you want to continue connecting (yes/no)
    ? yes
7 /bin/ssh-copy-id: INFO: attempting to log in with the
     new key(s), to filter out any that are already
    installed
8 /bin/ssh-copy-id: INFO: 1 key(s) remain to be
    installed -- if you are prompted now it is to
    install the new keys
9 root@192.168.88.166's password:
10
11 Number of key(s) added: 1
12
13 Now try logging into the machine, with:   "ssh
    '192.168.88.166'"
14 and check to make sure that only the key(s) you
    wanted were added.
```

Ahora copiaríamos los archivos en la ruta que deseemos en la placa. Nosotros nos hemos decantador por usar "/opt/" pero no afecta para nada el resultado final:

```
1   # scp -r /home/proyecto/Desktop/instaladores/
      mpichroot/mpichcompiladoleon/mpich-3.2 root@192
      .168.88.166:/opt/
```

Por supuesto, tenemos que añadir los ejecutables recién copiados a la placa en su listado de ejecutables, como hemos hecho en pasos anteriores, así que usaremos los comandos:

```
1   # export PATH=/opt/mpich-3.2/bin:$PATH"
2   # export LD_LIBRARY_PATH="/opt/mpich-3.2/lib:
      $LD_LIBRARY_PATH"
```

Es muy importante que este paso sea ejecutado para el usuario correcto en la placa, en nuestro caso el usuario "root" y que sea añadido al archivo correcto. Podríamos efectuar lo siguiente para ver si está todo configurado correctamente:

```
1   # cat /etc/profile
2 # ~/.bashrc: executed by bash(1) for non-login
    interactive shells.
```

```
3 ...
4 export PATH=/opt/mpich-3.2/bin:/bin:/sbin:/usr/bin:/
     usr/sbin:/usr/bin/X11:/usr/local/bin
5 export PATH
6 LD_LIBRARY_PATH=/opt/mpich-3.2/lib:
7 ...
```

El siguiente paso sería asignar una entrada DNS a todos los componentes de nuestro clúster. Lo haremos asignando entradas estáticas en el archivo "/etc/hosts" de la siguiente manera:

```
1   # cat /etc/hosts
2 127.0.0.1    localhost
3 192.168.88.208   node0
4 192.168.88.166   node1
```

La primera entrada es la de por defecto de Linux y las otras dos son nuestras. Esta modificación debería de hacerse en la placa y en la máquina virtual a fin de que puedan encontrarse usando el mismo "hostname".
Para finalizar, deberíamos de crear un archivo parecido al anterior pero en el que especificamos cuál es el número de procesos por nodo que usaremos. El contenido sería algo así:

```
1   # cat /opt/mpich-3.2/utils/machinefile
2 node0:2
3 node1
```

Básicamente, es una lista donde nos asocia el número de procesos que se pueden ejecutar de máximo en un nodo y su nombre. Dicho nombre ha de coincidir con el que tenemos en el archivo "/etc/hosts". Podemos observar que en nuestra configuración el nodo maestro usará dos procesos de máximo por aplicación y el nodo esclavo solo uno.
Una vez finalizado todos estos pasos, ya tendríamos nuestro laboratorio preparado para ejecutar algunos programas de prueba usando MPICH.

7.2. Programa de prueba MPICH.

Hemos creado un programa de prueba basado en el clásico "Hello World" cuyo código fuente puede encontrarse en la ruta descrita a continuación [34]:

```
# cat /home/proyecto/Desktop/instaladores/examples/
   mpi_hello.c
```

El programa es bastante sencillo y su contenido es el siguiente:

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int rank;
  char hostname[256];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  gethostname(hostname,255);

  printf("Hello world!  I am process number: %d on
     host %s\n", rank, hostname);

  MPI_Finalize();

  return 0;
}
```

Esta aplicación tratará de contactar con todos los nodos de nuestro clúster y lanzar tantos procesos como sea posible para imprimir dicho text por pantalla. Podemos ver el resultado a continuación usando el ejemplo de nuestra carpeta de ejecutables:

```
# mpiexec -n 2 /opt/mpich-3.2/utils/mpi_hello_x64
Hello world!  I am process number: 0 on host fedora.
   proyecto
Hello world!  I am process number: 1 on host fedora.
   proyecto
Hello world!  I am process number: 2 on host
   buildroot
```

Algún otro ejemplo, como el cálculo del número Pi basado en un ejemplo dado por la propia organización MPICH:

```
1   # mpiexec -n 4 /opt/mpich-3.2/utils/cpi_x64
2 Process 2 on fedora.proyecto
3 Process 1 on fedora.proyecto
4 Process 3 on buildroot
5 Process 0 on fedora.proyecto
6 pi is approximately 3.1416007769231249, Error is
     0.0000064433333318
7 wall clock time = 0.000049
```

8. Personalización de la distribución Buildroot.

Para agilizar todos los cambios efectuados en los anteriores apartados, usaremos una utilidad de la herramienta "linuxbuild" en la cual podemos ejecutar un "script" que introduzca cambios en nuestro sistema de archivos antes de que se ensamble la imagen RAM.
Con este fin hemos creado el archivo "buildroot_post.sh" en nuestro escritorio. La idea es que antes de que se ensamble la imagen RAM de nuestra distribución Buildroot, hagamos los cambios que creamos necesarios en el sistema de archivos de dicha distribución embebida.[26][27]
Por supuesto, dicho archivo ha de ser ejecutable para que nuestro creador de imágenes pueda usarlo:

```
1   # sudo chmod 777 /home/proyecto/Desktop/
      buildroot_post.sh
```

El submenú donde se encuentra la opción que debemos usar para informar a la herramienta "linuxbuild" de que ha de usarlo puede verse en la siguiente figura:

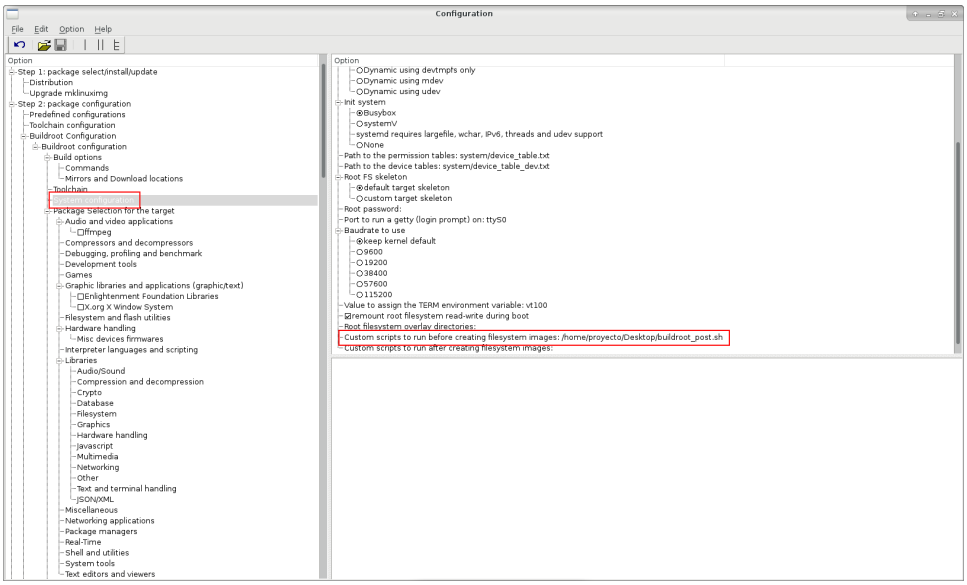

Figura 44: Buildroot: Personalización del sistema de archivos.

8.1. Interfaz de red eth0.

Con el fin de asignar una IP fija a la imagen de Linux, efectuaremos los siguientes cambios en la herramienta "linuxbuild" que nos proporciona Gaisler:

```
###############Networking##############
echo "auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.88.166
    network 192.168.88.0
    netmask 255.255.255.0
    broadcast 192.168.88.255
    gateway 192.168.88.1" > $TARGET/etc/network/
       interfaces

echo 'sleep 3;echo "nameserver 192.168.88.1">/etc/
   resolv.conf' > $TARGET/etc/init.d/S99local
chmod 755 $TARGET/etc/init.d/S99local
```

Como podemos observar, asignamos la IP "192.168.88.166". Además, creamos un ejecutable en el directorio "/etc/init.d/" para poder poner nuestra ruta por defecto como servidor DNS preferido.

8.2. Configuración del protocolo SSH.

Copiamos unas claves generadas anteriormente y las copiamos al sistema de archivos cada vez que creamos la imagen que se carga en la RAM.
Las rutas de los archivos son las siguientes:

```
/etc/ssh_host_key
/etc/ssh_host_key.pub
/etc/ssh_host_rsa_key
/etc/ssh_host_rsa_key.pub
/etc/ssh_host_dsa_key
/etc/ssh_host_dsa_key.pub
/etc/ssh_host_ecdsa_key
/etc/ssh_host_ecdsa_key.pub
```

Y el comando necesario para copiar dichos archivos desde nuestra máquina virtual sería:

```
  # scp 192.168.88.166:/etc/ssh_host_* /home/proyecto
     /Desktop/instaladores/keys_ssh/
```

Ahora bien, como el demonio SSH (En Linux este servicio es un demonio[29]) está configurado para no aceptar conexiones sin contraseña y nuestro usuario "root" funciona sin contraseña, deberíamos de crear una a tal efecto. Usaremos el comando:

```
1  # passwd
```

Tras asignarle una contraseña a dicho usuario, deberíamos de poder conectarnos a la placa usando el protocolo SSH.
En cambio, si ejecutamos un comando de conexión SSH estándar como el siguiente:

```
1  # ssh root@192.168.88.166
```

Obtendremos el siguiente mensaje de error (Tras introducir la contraseña):

```
1 root@192.168.88.166:
2 WARNING: Your password has expired.
3 Password change required but no TTY available.
```

Esto es debido a que la configuración de usuarios de nuestra distribución embebida viene con un bug.
En el archivo de configuración "/etc/shadow" podemos ver que el campo de "Días desde que la constraseña fue cambiada" está a 0. Si miramos la documentación acerca de dicho archivo, veremos que los días se referencian desde el 1 de Enero de 1970[30] y, por lo tanto, nuestro sistema siempre creerá que nuestra contraseña está caducada [31].

```
1  #cat /etc/shadow
2
3  ...
4  root:XXXXXXX:0:0:99999:7:::
5  ...
```

Este error tiene como causa que nuestra distribución se inicializa con la fecha en el 1 de Enero de 1970 si no aplicamos configuración alguna en el menú de configuración y, al cambiar la contraseña del usuario "root" (Que inicialmente no tiene), dicho campo en el archivo "/etc/shadow" cambia a 0. Para resolverlo basta con cambiar dicha configuración antes de crear la imagen RAM o con introducir un valor alto en el primer "0" en la entrada de "root" en el archivo "/etc/shadow", como podemos ver a continuación:

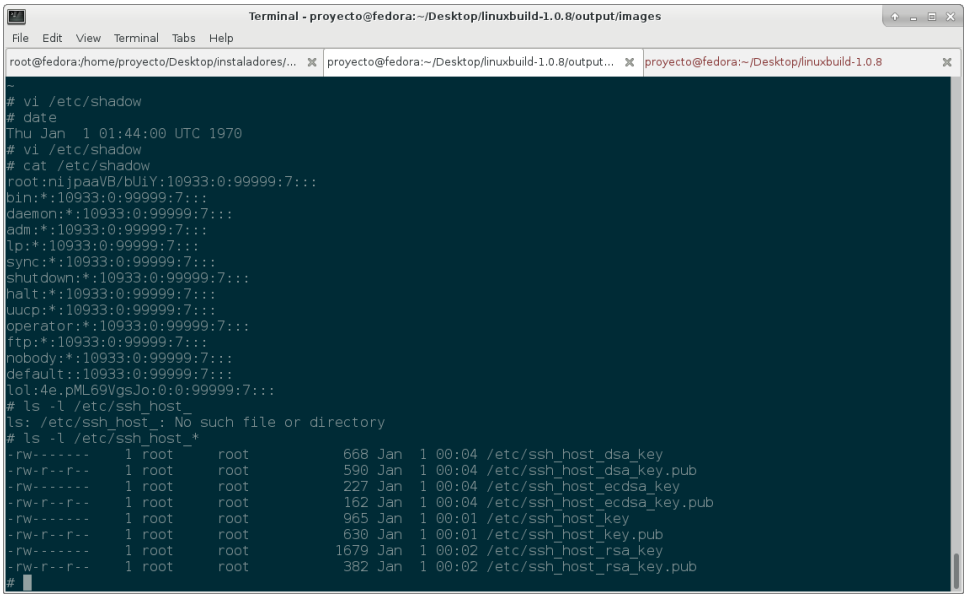

Figura 45: Buildroot: Archivos relacionados con el protocolo SSH.

Una vez copiadas las claves SSH, las pondremos en nuestra carpeta de instaladores y haremos que el script "buildroot_post.sh" las copie en nuestro distribución antes de ensamblarla en la imagen RAM, asignándoles los permisos correctos de escritura y lectura.

8.3. Cambio de contraseña del usuario root.

Para evitarnos el tedio proceso de crear una contraseña para root y cada vez que inicialicemos la RAM, hemos preferido crear dicha contraseña en el menú de configuración de "linuxbuild".
Podemos encontrar dicho campo configurable en :

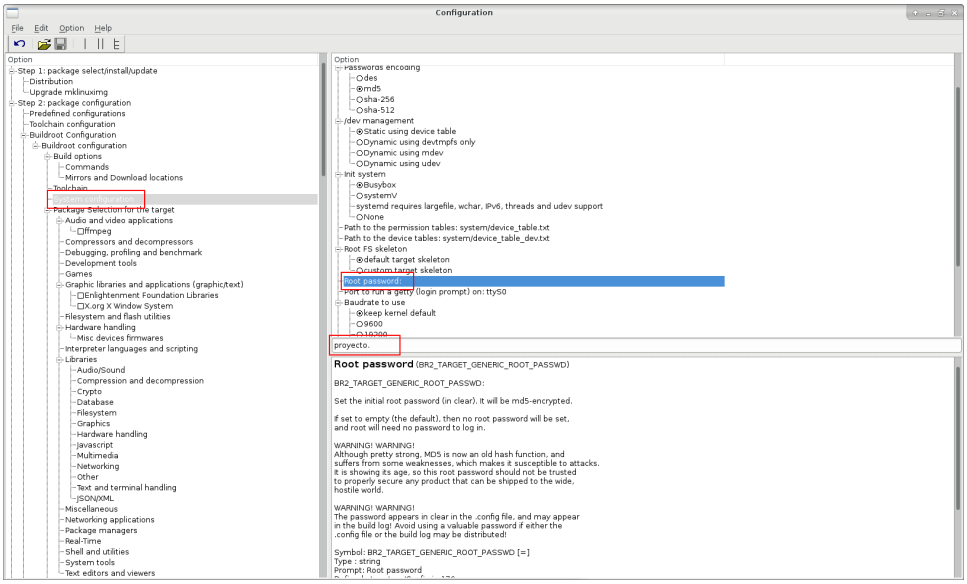

Figura 46: Buildroot: Contraseña de “root”.

Hemos decidido usar la contraseña “proyecto.” que es la misma usada en nuestra máquina virtual.
Si lanzamos la compilación de la imagen RAM veremos que llega un punto en el que fallará debido a un error en el comando “sed”. Esto es debido a un bug en nuestra herramienta “linuxbuild” y, en vez de parchear el código fuente, hemos decidido introducir el valor correcto usando el script de precompilación detallado en [**?**].
El código que hace tal fin es el siguiente[32]:

```
1  ...
2  sed -i "s/root:/root:rhIAKl.iMvnDg
       :10933:0:99999:7:::/g" $TARGET/etc/shadow
3  ...
```

8.4. Copia de archivos de MPICH.

A fin de contar con todos los archivos ya desplegados en nuestra imagen RAM, hemos creado las siguientes entradas en el archivo de personalización:

```
1 ###############MPICH###############
2 #Copiamos lo compilado para la arquitectura leon3
3 rm -rf $TARGET/opt/mpich-3.2
4 mkdir -p $TARGET/opt/mpich-3.2
```

```
5  cp -rf /home/proyecto/Desktop/instaladores/mpichroot/
       mpichcompiladoleon/* $TARGET/opt/mpich-3.2
6  #Hacemos cambios en SSH para que acepte claves RSA
7  sed -i "s/#RSAAuthentication yes/RSAAuthentication
       yes/g" $TARGET/etc/sshd_config
8  sed -i "s/#PubkeyAuthentication yes/
       PubkeyAuthentication yes/g" $TARGET/etc/
       sshd_config
9  #Copiamos las claves RSA de la VM y le damos los
       permisos correctos
10 mkdir $TARGET/root/.ssh
11 chmod 700 $TARGET/root/.ssh
12 cp /home/proyecto/Desktop/instaladores/mpichroot/
       authorized_keys $TARGET/root/.ssh
13 chmod 600 $TARGET/root/.ssh/*
14 #chown root: $TARGET/root/.ssh/*
15 #Modificamos el archivo hosts
16 echo "192.168.88.208  node0
17 192.168.88.166  node1" > $TARGET/etc/hosts
18 #Creamos el enlace simbolico para que esten en el
       path
19 echo "export PATH=/opt/mpich-3.2/bin:/bin:/sbin:/usr/
       bin:/usr/sbin:/usr/bin/X11:/usr/local/bin
20 export PATH
21 LD_LIBRARY_PATH="/opt/mpich-3.2/lib:$LD_LIBRARY_PATH"
22 export LD_LIBRARY_PATH" >> $TARGET/etc/profile
```

Como podemos ver por los comentarios del script, son los mismo pasos usados en la creación manual de los ejecutables en apartados anteriores.

9. Análisis de resultados y conclusiones.

El desarrollo de nuevos sistemas embebidos y el auge del "Internet de las cosas" (O IoT por sus siglas en inglés - Internet of Things[44]) hace posible el uso de librerías de programación paralela/distribuida bien establecidas en dichos sistemas.
Hemos presentado un caso de estudio en el que, no solo dada una arquitectura de procesado fuera de lo común sino también un entorno de arquitectura heterogéneo, se han detallado los diferentes pasos a seguir en cada una de las diferentes tecnologías para conseguir el correcto funcionamiento de dicho procesado en paralelo usando la librería MPICH.
En este caso, hemos comprobado como podemos:

- Instalar una máquina virtual acorde a nuestras necesidades.
- Configurar el sistema operativo Linux para el objetivo de nuestro proyecto.
- Usar el entorno de desarrollo IDE de Xilinx.
- Utilizar los paquetes provistos por Gaisler para crear un SoC en nuestra placa.
- Integrar los dos apartados anteriores para lograr desplegar el sistema resultante en nuestra placa.
- Crear y optimizar una imagen Linux que se adapte a las necesidades de nuestra plataforma, explorando diferentes aspectos.
- Integrar dicho sistema en nuestro entorno de desarrollo, usando diferentes programas para ver su funcionalidad y escalabilidad.
- Desarrollar un clúster MPICH y observar su funcionamiento.

El sistema resultante podría ser usado para aprovechar las diferentes FPGAs que tiene el área de Electrónica, Tecnología de Computadoras y Proyectos para sacar partido a dichos dispositivos, que de otra manera estarían apagados y sin usar en las zonas de almacenamiento, y usarlos en cálculos complejos o simulaciones que necesiten una alta carga de procesado.
Aunque la propuesta inicial de este proyecto ha sido cumplida con creces, sería interesante comprobar la compatibilidad de sistemas multiprocesador para aumentar el rendimiento de los integrantes del clúster o de otras arquitecturas más establecidas en dispositivos móviles como puede ser ARM.
Enumeramos dichas tareas futuras que podrían emprenderse, en nuestra opinión, a continuación:

- Uso de Leon4 en vez de Leon3.

- Completa automatización usando una imagen ROM.
- Utilizar la arquitectura ARM en vez de SPARC para los nodos esclavos.
- Desarrollo de programas más complejos de MPICH.
- Uso de múltiples nodos.
- Usar la distribución Buildroot más actual.
- Desplegar un SoC con multiprocesador.
- Uso de todas las funcionalidades de GRMON para depurar errores de nuestra plataforma.
- Uso de tecnología Gigabite Ethernet en vez de Ethernet.

10. Bibliografía

Referencias

[1] "Definición de clúster."
"http://wwwclusterinformatica.blogspot.co.uk/2011/05/cluster-informatica.html"
Consultado el 07/09/2016

[2] "Página principal de la placa ML507 de Xilinx."
"http://www.xilinx.com/products/boards-and-kits/HW-V5-ML507-UNI-G.htm"
Consultado el 07/09/2016

[3] "Página de la Wikipedia sobre FPGAs."
"http://es.wikipedia.org/wiki/Field_Programmable_Gate_Array"
Consultado el 07/09/2016

[4] "Página principal del LEON3."
"http://gaisler.com/index.php/products/processors/leon3"
Consultado el 07/09/2016

[5] "Página principal del proyecto linux."
"https://www.kernel.org/"
Consultado el 07/09/2016

[6] "Página principal de la distribución Buildroot."
"https://buildroot.org/"
Consultado el 07/09/2016

[7] "Página principal del proyecto OpenMPI."
"https://www.open-mpi.org/"
Consultado el 07/09/2016

[8] "Introducción a VMWare Player."
"https://cadascu.files.wordpress.com/2013/06/guia-vmware.pdf"
Consultado el 07/09/2016

[9] "Máquinas virtuales en la Wikipedia."
"http://es.wikipedia.org/wiki/M%C3%A1quina_virtual"
Consultado el 07/09/2016

[10] "Manual de VMWare Player."
"https://www.vmware.com/pdf/desktop/vmware_player60.pdf"
Consultado el 07/09/2016

[11] “Instalación de Fedora en VMWare Workstation Player.”
“https://www.youtube.com/watch?v=9gQOX2KDczI”
Consultado el 07/09/2016

[12] “Instalación de VMWare Tools en Linux.”
“https://kb.vmware.com/selfservice/microsites/search.do?language=en_US
&cmd=displayKC&externalId=1014294”
Consultado el 07/09/2016

[13] “Página oficial de descargas de la suite ISE de Xilinx.”
“http://www.xilinx.com/support/download/index.html/content/xilinx/en/
downloadNav/design-tools.html”
Consultado el 07/09/2016

[14] “Manual de la placa Xilinx ML507 Virtex.”
“http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf”
Consultado el 07/09/2016

[15] “Página principal de licenciamiento de Xilinx.”
“http://www.xilinx.com/support/licensing_solution_center.htm”
Consultado el 07/09/2016

[16] “Página principal de GRMON.”
“http://www.gaisler.com/index.php/products/debug-tools/grmon”
Consultado el 07/09/2016

[17] “Página de descarga de GRMON.”
“http://www.gaisler.com/index.php/downloads/debug-tools”
Consultado el 07/09/2016

[18] “Página de Gaisler acerca de los diferentes IDEs y SDKs de Linux.”
“http://www.gaisler.com/index.php/downloads/linux?task=view&id=160”
Consultado el 07/09/2016

[19] “Página de licencias de Xilinx.”
“https://secure.xilinx.com/webreg/register.do?group=esd_oms&tab=CreateLicense”
Consultado el 07/09/2016

[20] “Repositorio de Linux en GIT.”
“https://github.com/torvalds/linux”
Consultado el 07/09/2016

[21] “Biografía de Linus Torvalds.”
“http://www.linfo.org/linus.html”
Consultado el 07/09/2016

[22] “Configuración de la red en Buildroot.”
“http://lists.busybox.net/pipermail/buildroot/2011-April/042907.html”
Consultado el 07/09/2016

[23] “Guía de instalación de MPICH”
“http://www.mpich.org/static/downloads/3.2/mpich-3.2-installguide.pdf”
Consultado el 07/09/2016

[24] “Información acerca de la arquitectura SPARC por Gaisler.”
“http://www.gaisler.com/doc/sparcv8.pdf”
Consultado el 07/09/2016

[25] “Manual de instalación de MPICH.”
“https://www.mpich.org/static/downloads/3.2/mpich-3.2-installguide.pdf”
Consultado el 07/09/2016

[26] “Usando Buildroot en un proyecto real.”
“http://elinux.org/images/2/2a/Using-buildroot-real-project.pdf”
Consultado el 07/09/2016

[27] “Manual de Buildroot en GIT.”
“https://github.com/maximeh/buildroot/blob/master/docs/manual/customize-rootfs.txt”
Consultado el 07/09/2016

[28] “Diferencias entre claves RSA y DSA.”
“https://www.quora.com/What-is-the-difference-between-RSA-and-DSA”
Consultado el 07/09/2016

[29] “Diferencias entre procesos, servicios y demonios en Linux.”
“http://stackoverflow.com/questions/3612846/what-are-the-behavioral-differences-between-a-daemon-and-a-normal-process”
Consultado el 07/09/2016

[30] “Archivos passwd y shadow en Linux.”
“http://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html”
Consultado el 07/09/2016

[31] “Hilo en Busybox sobre problemas con SSH.”
“http://lists.busybox.net/pipermail/buildroot/2011-March/042165.html”
Consultado el 07/09/2016

[32] “Buildroot y la contraseña de administrador.”
“http://www.armadeus.org/wiki/index.php?title=
How_to_set_the_default_root_password”
Consultado el 07/09/2016

[33] “Página oficial de MPICH.”
“https://www.mpich.org”
Consultado el 07/09/2016

[34] “Ejemplos de programas MPI.”
“https://hpcc.usc.edu/support/documentation/examples-of-mpi-programs/”
Consultado el 07/09/2016

[35] “¿Qué es un SoC?”
“http://gadgets.ndtv.com/mobiles/features/tech-101-explaining-the-soc-641756”
Consultado el 07/09/2016

[36] “Características del LEON3.”
“http://www.gaisler.com/index.php/products/processors/leon3”
Consultado el 07/09/2016

[37] “Suite de diseño ISE de Xilinx.”
“https://www.xilinx.com/products/design-tools/ise-design-suite.html”
Consultado el 07/09/2016

[38] “Página principal de la Free Software Foundation.”
“https://www.fsf.org”
Consultado el 07/09/2016

[39] “Lista de correo de Gaisler en Yahoo.”
“https://beta.groups.yahoo.com/neo/groups/LEON_SPARC/info’
Consultado el 07/09/2016

[40] “Definición de API.”
“http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/”
Consultado el 07/09/2016

[41] “Foro Oficial de MPI.”
“http://www.mpi-forum.org/”
Consultado el 07/09/2016

[42] “Primer anuncio sobre Linux.”
“http://www.thelinuxdaily.com/2010/04/the-first-linux-announcement-from-linus-torvalds/”
Consultado el 07/09/2016

[43] “Conferencia de supercomputación de 1992.”
“http://dl.acm.org/citation.cfm?id=147877
&picked=prox&CFID=833944919&CFTOKEN=52178934”
Consultado el 07/09/2016

[44] “Internet of Things explicado.”
“https://hipertextual.com/2015/06/internet-of-things”
Consultado el 07/09/2016

[45] “Diferencias entre ALU y FPU.”
“http://computadoras.about.com/od/conoce-procesadores/a/Que-Son-La-Alu-Y-La-Fpu-De-Un-Procesador.htm”
Consultado el 11/09/2016