



ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL
FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

“Instalación y puesta a punto de una distribución de Linux en una plataforma basada
en el procesador configurable Nios II”

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN TELEMÁTICA

INGENIERO EN ELECTRICIDAD

ESPECIALIDAD EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

Sara Rafaela Poveda Luna

Diego Gabriel Jáuregui Solórzano

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

Agradezco a Dios por tener la oportunidad de culminar una meta más como lo es mi carrera Universitaria, ya que él ha estado cuidándome y dándome fortaleza para continuar.

Gracias a la mujer que ha sido madre y padre para mi, quién me dio todo el apoyo necesario día tras día a lo largo de todos estos años y que con sus ejemplos de perseverancia, coraje me ha motivado a luchar por mis metas. Agradezco a todos que con su ayuda hayan logrado sacar adelante este proyecto. Gracias Diego Jáuregui, compañero de tesis, por todo el apoyo y gracias al Ing. Ronald Ponguillo un hombre inteligente que ha sido un gran mentor para nosotros.

Sara Rafaela Poveda Luna

Agradezco a la Pachamama,

A mis padres,

A los desarrolladores de software libre,

A Ronald, a Sara, a Diana

Al Ing. Ponguillo

Diego.

DEDICATORIA

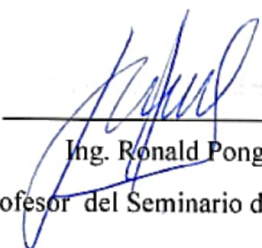
Dedico este trabajo a mi madre y hermana,
pilares fundamentales en mi vida, por ellas he
llegado a cumplir este logro.

Sara.

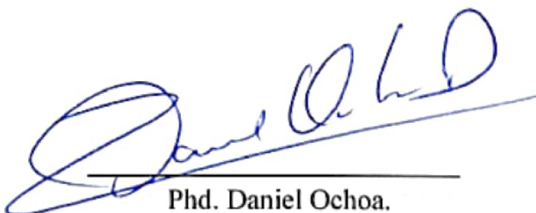
Dedicado a todos los que creen en un mundo libre
de copyright.

Diego.

TRIBUNAL DE SUSTENTACIÓN



Ing. Ronald Ponguillo.
Profesor del Seminario de Graduación



Phd. Daniel Ochoa.
Delegado del Decano

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este trabajo, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”.

(Reglamento de exámenes y títulos profesionales de la ESPOL)



Sara Rafaela Luna Poveda



Diego Gabriel Jauregui Solórzano

RESUMEN

El presente proyecto: “Instalación y puesta a punto de una distribución de Linux en una plataforma basada en el procesador configurable Nios II”, es parte del seminario de graduación “Procesadores embebidos configurables”. El proyecto fue desarrollado utilizando una tarjeta de desarrollo DE2 de Terasic Inc., por lo que nuestro proyecto estará enmarcado en las capacidades e interconexiones de sus componentes. El componente principal de la tarjeta es un FPGA de Altera de la familia Cyclone II, en el que se programa mediante las herramientas de diseño de Altera, la arquitectura de un mini-computador cuyo procesador es el Nios II y que cuenta con las interfaces suficientes para comunicarlo con los periféricos de la tarjeta.

La capacidad de los componentes del mini-computador lo hace ideal para ser usado como controlador en sistemas de propósito específico.

Como se lo hace en un PC, al mini-computador embebido en la tarjeta le instala un sistema operativo para que se encargue de administrar los recursos de su hardware. Este será una distribución de Linux que soporte la arquitectura del procesador Nios II, los dispositivos y periféricos embebidos en el FPGA y los conectados a este. Normalmente los procesadores utilizados en sistemas embebidos no cuentan con MMU, por lo que nuestra configuración de hardware tampoco la incluye. La distribución de Linux a instalar debe ser compatible con esta característica. Además de la puesta en marcha de Linux en la tarjeta DE2, desarrollamos un módulo de

hardware personalizado hecho con lenguaje HDL que lo añadiremos a la arquitectura del mini-computador y haremos uso de sus recursos desde Linux.

ÍNDICE GENERAL

| | |
|--------------------------------|------|
| AGRADECIMIENTO | I |
| DEDICATORIA | III |
| TRIBUNAL DE SUSTENTACIÓN | IV |
| DECLARACIÓN EXPRESA | V |
| RESUMEN..... | VI |
| ÍNDICE GENERAL..... | VIII |
| ÍNDICE DE FIGURAS..... | XV |
| ÍNDICE DE TABLAS | XIX |
| GLOSARIO | XX |
| INTRODUCCIÓN | XXIV |
| CAPÍTULO 1 | 1 |

| | | |
|-------|---|----|
| 1. | GENERALIDADES..... | 1 |
| 1.1 | ANTECEDENTES..... | 1 |
| 1.2 | ALCANCES Y LIMITACIONES DEL PROYECTO..... | 4 |
| 1.2.1 | ALCANCES..... | 4 |
| 1.2.2 | LIMITACIONES..... | 4 |
| 1.3 | OBJETIVOS..... | 6 |
| 1.3.1 | OBJETIVOS GENERALES..... | 6 |
| 1.3.2 | OBJETIVOS ESPECÍFICOS..... | 6 |
| 1.4 | RESULTADOS ESPERADOS..... | 8 |
| 1.5 | APLICACIONES..... | 8 |
| | CAPÍTULO 2..... | 10 |
| 2. | MARCO TEÓRICO..... | 10 |
| 2.1 | SISTEMAS EMBEBIDOS..... | 10 |
| 2.2 | PROCESADOR NIOS II..... | 13 |
| 2.3 | AVALON SWITCH FABRIC INTERCONNECT..... | 17 |
| 2.3.1 | AVALON MEMORY MAPPED..... | 18 |
| 2.3.2 | INTERFACE AVALON CONDUIT..... | 20 |
| 2.3.3 | INTERFACE AVALON TRI-STATE..... | 21 |
| 2.3.4 | INTERFACE AVALON INTERRUPT..... | 21 |
| 2.4 | DISTRIBUCIONES DE LINUX PARA NIOS II..... | 21 |

| | | |
|-----------|---|----|
| 2.5 | μ CLINUX..... | 23 |
| 2.6 | DIFERENCIAS ENTRE LINUX Y μ CLINUX..... | 25 |
| 2.6.1 | ADMINISTRACIÓN DE MEMORIA VIRTUAL..... | 25 |
| 2.6.2 | DIFERENCIAS EN EL KERNEL..... | 26 |
| 2.6.3 | ASIGNACIÓN DE MEMORIA (KERNEL Y APLICACIONES) | 28 |
| 2.6.4 | APLICACIONES Y PROCESOS..... | 34 |
| 2.6.5 | LIBRERÍAS COMPARTIDAS..... | 37 |
| 2.6.6 | LIBRERÍA C USADA CON μ CLINUX..... | 37 |
| 2.6.7 | DISTRIBUCIÓN DE μ CLINUX..... | 38 |
| 2.7 | DISEÑO DEL SISTEMA EN CHIP PROGRAMABLE (SOPC) | 39 |
| 2.8 | FLUJOGRAMA DEL PROCESO DE DESARROLLO..... | 40 |
| 2.9 | HARDWARE Y TOOLCHAIN DE COMPILACION DE UCLINUX | 43 |
| 2.9.1 | HARDWARE..... | 43 |
| 2.9.1.1 | TARJETA TERASIC ALTERA DE2..... | 43 |
| 2.9.1.2 | HARDWARE SOPC..... | 45 |
| 2.9.1.2.1 | REQUERIMIENTO MÍNIMOS DE UCLINUX | 46 |
| 2.9.1.2.2 | .. SYSTEM_0..... | 47 |
| 2.9.1.2.3 | MODULO PWM..... | 52 |

| | |
|---|----|
| 2.9.2 ENTORNO DE DESARROLLO PARA LA COMPILACIÓN DEL KERNEL DE μ CLINUX | 55 |
| 2.9.2.1 MAQUINA HOST CON LINUX | 55 |
| 2.9.2.2 TOOLCHAIN DEL CROSS-COMPILER DE NIOS II | 56 |
| 2.9.2.3 UCLIBC | 58 |
| 2.9.2.4 GNU BINUTILS | 58 |
| 2.9.2.5 μ CLINUX-DIST -MENÚ DE CONFIGURACIÓN KCONFIG | 59 |
| 2.9.2.6 PROGRAMACIÓN DE UN MÓDULO DE KERNEL PARA EL DRIVER DEL MÓDULO PWM | 59 |
| CAPÍTULO 3 | 61 |
| 3. DISEÑO E IMPLMNTACIÓN | 61 |
| 3.1 INSTALACIÓN DE μ CLINUX EN LA TARJETA DE2..... | 61 |
| 3.1.1 CÓDIGO FUENTE DE μ CLINUX PARA ALTERA NIOS II. | 62 |
| 3.1.2 ESTRUCTURA DEL DIRECTORIO DEL TOOLCHAIN DEL COMPILADOR CRUZADO NIOS2GCC-20080203 | 63 |
| 3.1.3 ESTRUCTURA DEL DIRECTORIO DE LAS FUENTES NIOS2-LINUX-20090730..... | 66 |
| 3.1.4 SELECCIÓN DE LOS RAMALES DE DESARROLLO | 69 |
| 3.1.5 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE UCLINUX CON LOS PARÁMETROS POR DEFECTO | 72 |

| | |
|--|-----|
| 3.1.6 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE μ CLINUX CON PARÁMETROS PERSONALIZADOS | 80 |
| 3.1.7 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE μ CLINUX CON PARÁMETROS PERSONALIZADOS PARA INCLUIR HARDWARE ADICIONAL Y FUNCIONES DE DEPURACIÓN DE APLICACIÓN DE USUARIO | 101 |
| 3.1.8 AÑADIR UN MÓDULO CARGABLE PARA EL CONTROLADOR DEL PWM CORE..... | 104 |
| 3.1.8.1 DEPURACIÓN REMOTA DE APLICACIÓN DEL TARGET DESDE EL HOST..... | 108 |
| 3.1.8.2 COMPILACIÓN Y DEPURACIÓN DE LA APLICACIÓN | 109 |
| 3.1.8.3 COMPILACIÓN POR LÍNEA DE COMANDO EN CONSOLA | 109 |
| 3.1.8.4 COMPILACIÓN Y DEPURACIÓN UTILIZANDO UN ENTORNO INTEGRADO DE DESARROLLO COMO ECLIPSE IDE | 110 |

| | |
|---|-----|
| 3.1.8.4.1 CONFIGURACIÓN DE ECLIPSE CDT PARA UTILIZAR EL COMPILADOR CRUZADO DE UCLINUX | 110 |
| 3.1.8.4.2 CONFIGURACIÓN DEL DEPURADOR (DEBUGGER)..... | 112 |
| 3.1.8.5 COMPILACIÓN DE LA APLICACIÓN..... | 115 |
| CAPÍTULO 4..... | 118 |
| 4. RESULTADOS..... | 118 |
| 4.1 EJECUCIÓN DE μ CLINUX-NIOS2..... | 118 |
| 4.1.1 TERMINAL REMOTO DE UCLINUX EN EL PUERTO JTAG UART..... | 122 |
| 4.1.2 TERMINAL REMOTO DE UCLINUX EN EL PUERTO SERIAL UART | 123 |
| 4.2 PRIMEROS PASOS EN mCLINUX..... | 130 |
| 4.2.1 ESCRIBIR Y LEER DEL BUS AVALON DESDE EL ESPACIO DE USUARIO | 130 |
| 4.2.2 CARGAR EL MÓDULO DE DRIVER PARA EL MÓDULO PWM | 133 |
| 4.2.3 INICIALIZAR EL CHIP DE ETHERNET DM9000..... | 134 |
| 4.2.4 TRANSFERIR ARCHIVOS DESDE EL HOST AL TARGET VÍA FTP SOBRE ETHERNET | 136 |

| | | |
|-----|---|-----|
| 4.3 | UTILIZAR EL HARDWARE POR MEDIO DEL DRIVER DESDE UNA APLICACIÓN DE USUARIO | 139 |
| 4.4 | DEPURAR UNA APLICACIÓN DE USUARIO DEL TARGET DESDE EL HOST POR MEDIO DE GDBSERVER Y ECLIPSE CDT.. | 141 |
| 4.5 | MONITOREO EN TIEMPO REAL DE SEÑALES CON SIGNALTAP II | 144 |
| 4.6 | LEER UNA TARJETA SD | 149 |

CONCLUSIONES Y RECOMENDACIONES

ANEXO A: HERRAMIENTAS DE DESARROLLO PARA LA CONFIGURACIÓN DEL PROCESADOR NIOS II

ANEXO B: INSTALACIÓN DEL ENTORNO DE DESARROLLO PARA NIOS II EN SCIENTIFIC LINUX 6: QUARTUS II /SOPC BUILDER Y NIOS II EDS

ANEXO C: INSTALACIÓN DE ECLIPSE IDE PARA LINUX

ANEXO D: SCRIPTS CONFIGURACIÓN DE AMBIENTE

ANEXO E: CONFIGURACIÓN DEL JTAG EN LINUX

ANEXO F: CODIGO FUENTE

ANEXO G: PROGRAMANDO UN MÓDULO DEL KERNEL

BIBLIOGRAFÍA

ÍNDICE DE FIGURAS

| | |
|--|----|
| Figura 1.1 Chip MC68328 DragonBall | 3 |
| Figure 2.1 Capas de un Sistema Embebido basado en SOPC Builder..... | 12 |
| Figura 2.2. Medios de comunicación entre el Host y el Target utilizados en el proyecto..... | 13 |
| Figura 2.3 Diagrama de funcionamiento del NIOS II..... | 16 |
| Figura 2.4 Plantilla de un componente con interface Avalon slave..... | 19 |
| Figura 2.5 Organización de los procesos en la memoria | 29 |
| Figura 2.6. Fragmentación de la memoria | 33 |
| Figura 2.7 Esquema de Diseño SOPC..... | 40 |
| Figure 2.8 Tarjeta TERASIC. | 43 |
| Figura 2.9 Conexioons a CPU_0..... | 48 |
| Figura 2.10 Construcción de la arquitectura NIOSII con el SOPC Builder..... | 56 |
| Figure 2.11 Diagrama de bloques del sistema. | 51 |
| Figura 2.12 Módulo de nivel superior en código Verilog..... | 53 |
| Figura 2.13 Diagrama de Funcionamiento PWM | 54 |
| Figura 2.14 Decodificación de la dirección y señales de control..... | 54 |

| | |
|--|----|
| Figura 2.15 Simulación del módulo en el waveform del Quartus | 55 |
| Figura 3.1. Contenido del directorio /opt/nios2 | 64 |
| Figura 3.2. Modificación del archivo .bash_profile | 65 |
| Figura 3.3 Especificaciones del compilador cruzado..... | 66 |
| Figura 3.4. Contenido del directorio /nios2-linux..... | 67 |
| Figura 3.5. Script checkout | 68 |
| Figura 3.6. Checkout de las fuentes de nios2-linux | 68 |
| Figura 3.7. Estructura de árbol del directorio de nios2–linux..... | 69 |
| Figura 3.8. Contenido del directorio /nios2-linux/linux-2.6 | 70 |
| Figura 3.9. Contenido del directorio /nios2-linux/uClinux-dist..... | 70 |
| Figura 3.10. Pantalla principal del Menú de configuración | 72 |
| Figura 3.11. Pantalla de selección del Vendedor y el producto | 73 |
| Figura 3.12. Pantalla de selección del Kernel/Library/Defaults | 73 |
| Figura 3.13 HWSELECT CPU | 74 |
| Figura 3.14. HWSELECT Memoria ejecución del Kernel. | 75 |
| Figura 3.15. HWSELECT finalizado | 75 |
| Figura 3.16. Primeros mensajes durante la compilación con make | 76 |

| | |
|---|-----|
| Figura 3.17. Código fuente del script del Makefile – Sección inicial..... | 77 |
| Figura 3.18. Algunos Targets que conforman el Makefile | 77 |
| Figura 3.19. Final de la ejecución del make. | 79 |
| Figura 3.20. Imagen comprimida del Kernel de uClinux..... | 80 |
| Figura 3.21. Opciones seleccionadas para cambiar configuración de Kernel..... | 81 |
| Figura 3.22. Ventana principal de Configuración del Kernel de Linux..... | 82 |
| Figura 3.23. Ventana de tipo e procesador y características | 86 |
| Figura 3.24. Ventana después de la compilación. | 99 |
| Figura 3.25. Contenido del directorio “images”. | 99 |
| Figura 3.26. Ventana para guardar la configuración..... | 101 |
| Figura 3.27. Modificación del archivo makeFile. | 105 |
| Figura 3.28. Ventana selección Misc devices | 107 |
| Figura 3.29. Ventana Driver de pwm avalon Ip Core..... | 108 |
| Figura 3.30. Ventana después de compilar la Imagen del Kernel..... | 109 |
| Figura 3.31. Ventana de configuración del compilador..... | 111 |
| Figura 3.32. Ventana de Sistema de configuración de ejecución. | 113 |
| Figura 3.33. Ventana de Debugger. | 114 |

| | |
|---|-----|
| Figura 3.34. Ventana de Configuraciones de TCP IP..... | 115 |
| Figura 3.35. Mensajes del Compilador al construir..... | 116 |
| Figura 3.36. Tamaño de la pila del pwmout..... | 117 |
| Figura 4.1. Código fuente pwmout, dirección de memoria switch_pio | 140 |
| Figura 4.2. Puntero declarado volatile apuntando a los switches | 141 |
| Figura 4.3 Lazo infinito de variación del ciclo de trabajo | 141 |
| Figura 4.4. Vista conjunta del Depurador de Eclipse en el host y la consola del target ejecutando el gdbserver..... | 143 |
| Figura 4.5. Estado de los registros clockdiv, duty y load antes de cargar el módulo pwm_avalon | 146 |
| Figura 4.6. Estado de los registros clockdiv, duty y load luego de cargar el módulo pwm_avalon | 146 |
| Figura 4.7. Estado de los registros clockdiv, duty y load luego de cargar 0 en la dirección del registro enable/load | 147 |
| Figura 4.8. Estado de los registros clockdiv, duty y load luego de cargar 1 en la dirección del registro enable/load | 147 |
| Figura 4.9. Estado de los registros clockdiv, duty y load luego de cargar 25000000 en la dirección del registro duty_cycle | 148 |

| | |
|--|-----|
| Figura 4.10. Estado de los registros clockdiv, duty y load luego de cargar 50000000 en la dirección del registro clock_divide | 148 |
|--|-----|

ÍNDICE DE TABLAS

| | |
|--|----|
| Tabla 2.1. Diferentes versiones de procesador Nios II | 14 |
| Tabla 2.2. Distribución de Linux compatibles con NIOS II | 22 |
| Tabla 2.3. Identificación y asignación de memoria de los componentes del System_0 | 47 |

GLOSARIO

| | |
|-----------|---|
| ASIC | Circuito Integrado de Aplicación Específica |
| ALM | Adaptative Logic Module |
| API | Application Programming Interface |
| ASSEMBLER | (Lenguaje) Ensamblador |
| BFD | Binary File Description |
| CPU | Unidad Central de Proceso |
| DMA | Direct Memory Access |
| DMIPS | Drhystone million instructions per second |
| EDS | Embebbed Development Suite |
| EIC | Enchanced Interruption Controller |
| ELA | Embbded Logic Analyzer |
| ELF | Executable and Linkeable Format |

| | |
|--------|---|
| FPGA | Field Programmable Gate Array |
| FTP | File Transfer Protocol |
| GLP | General Public License: Licencia de software libre. |
| GNU | Proyecto de sistema operativo estilo Unix totalmente libre. |
| GPIO | General purpose Input/Output |
| HAL | Hardware Abstraction Layer |
| HDL | Hardware Description Language |
| HEAP | Estructura de árbol usada en computación |
| HOST | Computador donde se desarrollar el software de un sistema embebido. |
| IDE | Integrated Development Enviroment |
| IP | Intelectual Property |
| ISA | Industry Standard Architecture |
| JTAG | Joint Test Action Group: es el nombre común utilizado para la norma IEEE 1149.1 titulada Standard Test Access Port and Boundary Scan Architecture para test access ports utilizada para testear PCBs utilizando escaneo de límites. |
| KERNEL | Núcleo. El término se asocia generalmente al kernel de Linux. Parte fundamental de un |

| | |
|------------------|---|
| | sistema operativo. |
| LE | Logic Element |
| MM | Memory Mapped |
| MMU | Unidad de manejo de memoria |
| MMU-LESS (NOMMU) | Sin unidad de manejo de memoria |
| MPU | Memory Protection Unit |
| M4K | Bloque de memoria de 4608 bytes |
| PCB | Printed circuit board |
| PCI | Peripheral Component Interconnect |
| PDA's | Asistente Personal Digital |
| PIPELINE | Segmentación |
| PORT (PORTING) | Portar. Hacer funcionar algo en |
| RISC | Reduced instruction set computer |
| RT | Real Time |
| SDK | Software Development Kit |
| SOFT-CORE | Procesador basado en software. |
| SHELL | Consola de comandos en Linux. |
| SO | Sistema Operativo |
| SOPC | System on Programmable Chip [Sistema en un chip programable. |
| SOPC HARDWARE | Soft-Hardware que se programa en el FPGA. |

| | |
|------------------|---|
| STACK, STACKSIZE | Pila, tamaño de la pila |
| TARGET | Plataforma de desarrollo de un sistema embebido. |
| THREAD | Hilo (de ejecución) |
| TLB | Translation Lookaside Buffer |
| TMPFS | Sistema de archivos temporal cargado en memoria volátil en sistemas UNIX. |
| TOOL CHAIN | Cadena de herramientas |
| UART | Universal Asynchronous Receiver-Transmitter |
| VHDL | Very High Speed Integrated Circuit y HDL |
| VM | Memoria Virtual |
| XIP | Execute In Place Kernel |

INTRODUCCIÓN

De la “Instalación y puesta a punto de una distribución de Linux en una plataforma basada en el procesador configurable Nios II” se puede desprender las siguientes palabras claves: Linux, sistema operativo, hardware configurable, SOPC, soft-core. Si nos abstraemos un poco y pensamos en algo más general, se nos ocurre Instalación del sistema operativo Linux para sistemas embebidos basados en hardware reconfigurable SOPC y soft-cores. Empezamos de esta manera para ver el proyecto desde el macro de su campo de aplicación y desarrollo. Hacemos algunas introducciones a estos temas dentro del marco de nuestro proyecto. Realizamos la implementación del tema sugerido en una tarjeta de desarrollo para FPGA de Altera.

El presente trabajo tiene la finalidad de investigar los proyectos y tecnologías existentes en dos áreas importantes en sistemas-on-chip, el desarrollo de SOPC utilizando FPGA para la etapa de prototipo en una aplicación específica y el uso de Linux como sistema operativo en sistemas embebidos en una arquitectura diferente a la x86.

En el capítulo 1, establecemos el marco de referencia para medir el éxito del proyecto. En base a los objetivos generales dictados al inicio del seminario, establecimos nuestros propios objetivos. Definimos el alcance que se le va a dar a la investigación y a la implementación y determinamos las limitaciones que tenemos que enfrentar. Finalmente hacemos un resumen de los antecedentes y parte de la historia del desarrollo de los temas involucrados en el proyecto.

En el capítulo 2, nos enfocamos en el marco teórico que rodea nuestro trabajo. Describimos técnicamente los componentes de hardware físico y lógico utilizados: Tarjeta de desarrollo DE2, SOPC Hardware, módulo PWM. El software implementado también es analizado: el módulo del kernel `pwm_avalon`, aplicación de usuario `pwmout`. Se revisa brevemente las herramientas de compilación de uClinux: el instalador de código fuente `uClinux-dist`, el compilador cruzado `nios2-linux-uClibc-gcc`, el software de desarrollo Eclipse CDT.

En el capítulo 3, revisaremos todos los pasos para realizar la implementación del proyecto utilizando las herramientas de desarrollo estudiadas en el capítulo 2.

En el capítulo 4, analizaremos los resultados de la implementación. Ejecutamos algunos comandos en μ Clinux y hacemos uso de algunos periféricos de la tarjeta, especialmente de nuestro módulo PWM. El módulo PWM lo ejecutamos desde dos variantes: desde una aplicación de usuario por medio de un módulo de driver y la otra desde una aplicación de usuario que accede directamente al hardware.

Posteriormente emitimos una serie de conclusiones y recomendaciones en base a nuestras experiencias con el proyecto.

Al final, se encuentran los anexos con información sobre ciertos temas que decidimos que no sean parte de los capítulos formales. Ahí hay información sobre el Software de diseño Quartus II, SOPC Builder, los IP Cores, scripts de configuración, código fuente y una guía de programación de drivers para Linux.

CAPÍTULO 1

1. GENERALIDADES

En este capítulo se aborda el planteamiento del proyecto, así como sus alcances y limitaciones. Se expone además la razón de esta implementación y se explica la solución propuesta al problema planteado.

1.1 ANTECEDENTES

El nombre de Linux se refiere estrictamente al kernel Linux, pero es más comúnmente utilizado para describir un sistema operativo similar a Unix pero libre, también conocido como GNU/Linux, formado por la combinación del kernel de Linux con bibliotecas y herramientas del proyecto GNU y muchos otros proyectos/grupos de software libre. La primera versión del kernel de Linux fue escrita por el finlandés Linus Torvalds para procesadores Intel i386 y liberada en 1992.

Años después el kernel de Linux fue modificado para facilitar su portabilidad hacia otras arquitecturas. Linux es ahora de hecho, uno de los núcleos de sistema

operativo más ampliamente portados, y funciona en sistemas muy diversos que van desde routers (sistemas embebidos) hasta mainframes.

Un sistema embebido es una combinación de hardware de computadora y de software, en ocasiones, partes mecánicas, diseñadas para realizar una función específica. La finalidad del diseño (generalmente hecho a medida de la aplicación) es lo que los diferencia de sus similares, los computadores personales, que son considerados de propósito general.

En un sistema embebido bien diseñado, la existencia de un procesador y software puede pasar completamente desapercibida para el usuario del dispositivo. Tal es el caso de los hornos microondas, DVD-players, o relojes de alarmas. En algunos casos, se podría incluso construir un dispositivo equivalente que no contenga un procesador y software. Este se podría hacer reemplazando la combinación con un circuito integrado hecho a la medida que desarrolle las mismas funciones en hardware. Sin embargo, se pierde mucha flexibilidad cuando un diseño es desarrollado de esta manera. Es mucho más fácil y barato, cambiar unas pocas líneas de software que rediseñar una pieza de hardware a la medida.

El Nios II de Altera es un procesador embebido versátil, con el que se puede aprovechar doblemente la ventaja del software. Ya que su arquitectura de hardware puede ser re-configurada por software, y tal como en los procesadores convencionales, el software de las aplicaciones que corre el procesador puede ser modificado tanto como se desee.

La necesidad de desarrollar sistemas embebidos cada vez más flexibles y fáciles de configurar, y el hecho que el código fuente de Linux es público, no hizo tardar los emprendimientos para implementar el kernel de Linux en arquitecturas de menor rango que la x86 y similares, y que no cuentan con MMU, que dieron como resultado el sistema operativo μ Clinux, un derivado del kernel de Linux 2.0, que carece de memoria virtual y otras características que eran provistas por la MMU. Este sistema operativo estaba dirigido a la utilización en sistemas embebidos que cuentan con procesadores de bajo rango en desempeño, pero que son adecuados para aplicaciones muy específicas como PDAs, Smartphone, etc.

El primer porting de μ Clinux fue hacia el microprocesador MC68328 DragonBall de Motorola. Actualmente soporta una amplia gama de microprocesadores embebidos, desde los tradicionales ASIC, hasta los soft-core entre los que se encuentra el Nios II. Incluso ha añadido soporte para procesadores con hardware MMU.



Figura 1.1 Chip MC68328 DragonBall

Se ha acuñado el término Linux embebido al uso del kernel Linux y software asociado en sistemas embebidos. Nosotros usaremos el término μ Clinux-nios2 para referirnos a Linux corriendo sobre un procesador Nios II.

1.2 ALCANCES Y LIMITACIONES DEL PROYECTO

1.2.1 ALCANCES

- El sistema operativo del host utilizado para la compilación del proyecto es Scientific Linux 6.1 32 bits i686 (una distribución basada en Fedora).
- Depuración solo a nivel de aplicación con Eclipse CDT desde el host vía conexión TCP.
- Monitoreo de señales a nivel de hardware con el Signal Tap II.
- Se utiliza solamente el hardware físico provisto en la tarjeta Terasic DE2.
- Se utiliza los archivos binarios del tool chain compilar cruzado uClibc para Nios II del 2008-02-03 pre-compilado y que es soportado por la comunidad de desarrolladores de AlteraWiki.
- Se utiliza las fuentes de μ Clinux soportadas por la comunidad de desarrolladores del AlteraWiki. La versión del Kernel de Linux es 2.6.30 actualizado al 2009-03-30.

1.2.2 LIMITACIONES

- Capacidad del FPGA [LE, ALM, M4K]

- La herramienta de desarrollo del SOPC hardware es Quartus II 9.1sp2 subscription Edition para Linux con licencia de desarrollo universitaria (Espol-Lab. de digitales) con limitaciones de tiempo y ejecución para procesador Nios II.
- Los módulos IP Core están limitados a: Ethernet, Serial RS-232, E/S Paralelo (Leds, Switches), Memoria SDRAM, USB, Tarjeta SD, GPIO, Display LCD 16x2, Display 7 Segmentos, Memoria FLASH.
- No se incluye módulos IP para: Entrada de A/V, VGA, Audio, IrDA, Botoneras, Memoria SRAM.
- La capacidad de la memoria SDRAM es 8 MBytes.
- La capacidad de la memoria FLASH es 4 MBytes.
- La imagen compilada del Kernel de μ Clinux será cargada directamente a la memoria SDRAM del sistema por medio del link JTAG USB desde la consola de Nios II EDS.
- La imagen compilada del Kernel de μ Clinux será cargada directamente a la memoria SDRAM del sistema por medio del link JTAG USB del SDK de Nios II.
- No se utilizarán Interrupciones en la programación de las aplicaciones.

1.3 OBJETIVOS

1.3.1 OBJETIVOS GENERALES

- Instalar y poner en marcha una distribución de Linux en una plataforma que usa la arquitectura del procesador Nios II sobre la tarjeta de Desarrollo DE2 de Terasic.
- Desarrollar un módulo de hardware propio en HDL e integrarlo al SOPC hardware del sistema Nios II.
- Desarrollar un software que permita operar el módulo de hardware en Linux.

1.3.2 OBJETIVOS ESPECÍFICOS

- Instalar el IDE de Altera en un computador distribución de Linux i386/i686. Este computador será conocido como el *Host*.
- Seleccionar una distribución de Linux que pueda correr en la arquitectura del Nios II.
- Configurar el FPGA Cyclone II de la tarjeta de desarrollo DE2 de Altera con un *SOPC hardware* basado en la arquitectura procesador Nios II sin MMU conectado mediante el bus de expansión Avalon de Altera a varios dispositivos (memorias, I/O, etc.) de la tarjeta. Nos referiremos a la tarjeta como el *Target*.
- Crear un Generador de PWM de una salida en Verilog con interface Avalon Slave y conectarlo al procesador Nios II.

- Compilar y programar el FPGA con el *SOPC hardware*.
- Generar la imagen comprimida del kernel de μ Clinux con los controladores para: USB, Red Ethernet, Serial UART, Display 7 segmentos, Display LCD 16x2.
- Cargar la imagen comprimida del kernel de μ Clinux en el Nios II SoC.
- Iniciar una sesión de consola en μ Clinux vía serial RS-232 desde el host.
- Visualizar los archivos contenidos en un pen-drive conectado en el puerto USB de la tarjeta DE2 por medio de μ Clinux.
- Configurar el Signal Tap II para monitorear en tiempo real las señales Avalon, los registros y la salida del módulo PWM.
- Visualizar los archivos contenidos en una tarjeta SD Card de 256 MB.
- Montar un servidor FTP en μ Clinux y acceder desde un cliente FTP en el host.
- Transferir una aplicación basada μ Clinux-nios2 compilada en el host por medio del enlace FTP.
- Inicializar por medio de un módulo cargable el controlador para el PWM Avalon.
- Ejecutar una aplicación que cargue valores introducidos por el teclado en los 8 displays de 7 segmentos.
- Ejecutar la aplicación que utiliza el módulo PWM Avalon.
- Aprender a desarrollar controladores para hardware dentro de arquitecturas Nios II y similares.

- Aprender a depurar aplicaciones en lenguaje C corriendo remotamente en entorno μ Clinux.

1.4 RESULTADOS ESPERADOS

- Manejar la tarjeta como un mini-computador con Linux Embebido con la posibilidad de abrir una sesión de terminal vía serial RS-232.
- Utilizar el Signal Tap II como herramienta el monitoreo y depuración de sistemas embebidos en hardware reconfigurable.
- Demostrar las ventajas de μ Clinux como plataforma de desarrollo multitarea para sistemas embebidos.
- Tener acceso al hardware I/O de la tarjeta a través de una aplicación de usuario.
- Observar que el LED conectado al PWM Core parpadee al ritmo del pulso.

1.5 APLICACIONES

- Desarrollo de un Board support package (BSP) de entorno μ Clinux para el desarrollo de aplicaciones con la tarjeta DE2.
 - Módulo de prácticas para materias como Arquitectura de computadores y Sistemas Operativos.
- Comerciales:

- Al utilizar el nivel de abstracción del entorno Linux se acelera el desarrollo y soporte de aplicaciones ya que este es muy conocido ya.
- Mayor facilidad para portar aplicaciones Linux ya existentes a μ Clinux, salvando tiempo reutilizando aplicaciones y librerías.
- Ya existen aplicaciones comerciales de ésta tecnología y todas ellas se basan en el ahorro de tiempo gracias a la portabilidad de las mismas para Linux.
- Se pueden generar aparatos mucho más configurables y con soporte para más actualizaciones.
- Aparatos que mantengan un hardware base y se pueda aumentar o cambiar su funcionalidad descargando software a base de licenciamientos.

CAPÍTULO 2

2. MARCO TEÓRICO

En este capítulo hablaremos sobre las diversas herramientas que se utilizaron para el desarrollo de nuestro proyecto y a su vez explicaremos cada una de ellas en forma detallada como se muestra a continuación:

2.1 SISTEMAS EMBEBIDOS

Un sistema embebido se conforma de varias unidades funcionales para realizar eventualmente una o varias tareas muy específicas, de acuerdo si éstas están orientadas a ser estáticas o móviles. Un sistema de este tipo deberá tener una parte de hardware capaz de procesar toda la información de acuerdo a la finalidad que se le quiera dar. Lo conforman: memoria no volátil para guardar aplicaciones y datos, memoria volátil para el procesamiento de la información, CPU para procesar los datos tanto internos como externos al sistema, además de interfaces para manejar diferentes tipos de comunicación como USB, IrDA, Ethernet, entre otros. También

debe existir una parte de software encargada, junto con el hardware, de dar vida al sistema que se desea crear. Dicho software se encargará del control completo de todo el hardware.

En sistemas embebidos complejos, es recomendable el uso de un Sistema Operativo para que haga de mediador entre las aplicaciones y el hardware como se lo muestra en la Figura 2.1. Con un SO, las aplicaciones no tienen que preocuparse de acceder directamente al hardware, operación que en algunos casos es muy complicada y tediosa de realizar. El SO será el administrador de todo el sistema y se encargará de abstraer toda la funcionalidad del hardware para que el software de aplicación pueda sacar el máximo partido de él de una manera sencilla.

Obviamente, las aplicaciones y el sistema operativo deberán estar almacenados en alguna parte, lo más conveniente en este caso, será almacenar todos los programas en memoria no volátil, como es el caso de memorias ROM, flash, CompactFlash, entre otros. Para un manejo más robusto y eficaz, los datos no pueden ser almacenados de cualquier forma, debe haber una estructura para organizar los datos que se almacenan en algún dispositivo o medio para ello. Lo anterior supone el uso de un sistema de archivos que pueda ser administrado por el sistema operativo en su totalidad y además permita que las aplicaciones puedan acceder de forma segura a los datos. Normalmente, un sistema operativo trae el soporte completo para el manejo de uno o varios sistemas de archivos.

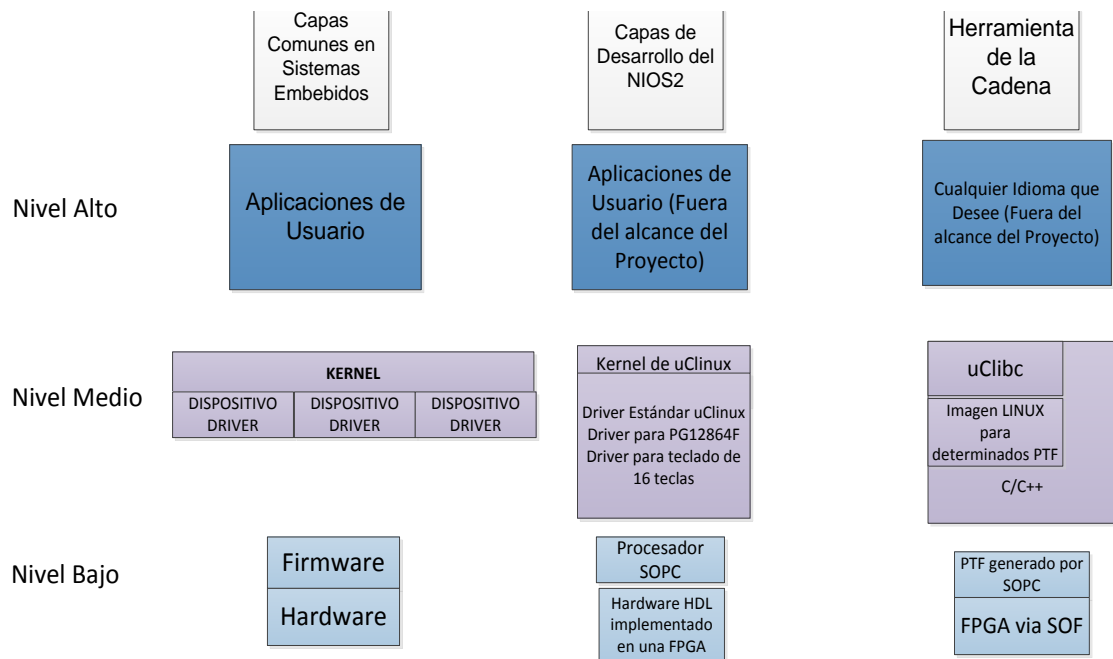


Figura 2.1 Capas de un Sistema Embebido basado en SOPC Builder

En la construcción del sistema embebido se deben tener en cuenta dos conceptos importantes: host y target. El término host se utiliza para indicar el sistema donde se realizará el desarrollo de todo el software del sistema embebido (codificación, configuración del sistema operativo, compilación, enlace, compresión, imagen del SO, construcción de las aplicaciones, entre otros). El término target hace referencia al sistema embebido propiamente. Deberá existir la forma de poder comunicar el host con el target con el fin de poder descargar todo el software del sistema en este último, además de poder hacer depuración de las aplicaciones que se instalen.

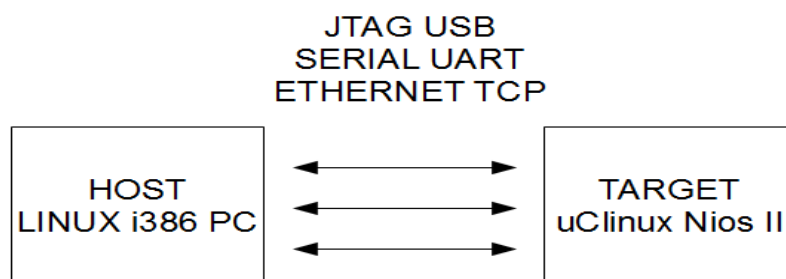


Figura 2.2 Medios de comunicación entre el Host y el Target utilizados en el proyecto.

Es de notar la gran variedad de términos que se emplean en un sistema embebido complejo. Por lo tanto, para tener mejor claridad de cada parte del sistema, tanto en su construcción como en su aplicación, se realizará una breve descripción en este capítulo de todas las partes que intervienen en éste, relacionando las diferentes unidades funcionales con el sistema operativo μ Clinux.

2.2 PROCESADOR NIOS II

Nios II de 32 bits con juego de instrucciones RISC es un microprocesador Soft-core completamente configurable y segmentado. Existe solo como código HDL hasta que esté cargado en un FPGA, a partir de lo cual es implementado en hardware completamente. El núcleo del Nios II utiliza solo una porción pequeña del total del espacio del FPGA lo cual permite que sean añadidos otros componentes estándares, así como cualquier hardware personalizado que se desee. Esto permite que el procesador, los componentes e incluso las instrucciones sean personalizados. Actualmente Altera ofrece tres versiones de Nios II para escoger:

- Nios II/f: rápido

- Nios II/s: estándar
- Nios II/e: económico

| Característica | Nios II/f | Nios II/s | Nios II/e |
|--------------------------------------|------------------|------------------|------------------|
| Segmentación | 6 niveles | 5 niveles | No |
| Multiplicación por HW | 1 ciclos | 3 ciclos | Por SW |
| Predicción de saltos | Dinámico | Estático | No |
| Cache de instrucción | Configurable | Configurable | No |
| Cache de datos | Configurable | No | No |
| Instrucciones personalizables | Hasta 256 | | |

Tabla 2.1 Diferentes versiones de procesador Nios II

Para nuestro desarrollo seleccionamos la versión f.

El núcleo rápido Nios II/f está diseñado para un rendimiento de alta ejecución. El rendimiento se gana a través del tamaño del núcleo. La base del núcleo Nios II/f, sin la unidad de administración de memoria (MMU) o unidad de protección de memoria (MPU), es aproximadamente un 25% más grande que el núcleo del Nios II/s. Altera diseñó el núcleo Nios II/f con los siguientes objetivos de diseño en mente:

- Maximizar la eficiencia en la ejecución de instrucciones por ciclo.
- Optimizar la latencia de interrupción.
- Maximizar el desarrollo f_{MAX} del núcleo del procesador.

El núcleo resultante es óptimo para aplicaciones de desarrollo crítico, así como para aplicaciones con grandes cantidades de códigos y/o datos, tales como sistemas ejecutando un sistema operativo con todas sus funciones.

Las características del Nios II/f son las siguientes:

- Tiene cache de instrucción y datos separados (opcional).
- Provee MMU para soportar sistemas operativos que requieren un MMU (opcional).
- Provee MPU para soportar sistemas operativos y entornos de ejecución que desean protección de memoria pero no necesitan administrar memoria virtual (opcional).
- Puede acceder hasta a 2GB de espacio de dirección externa cuando ningún MMU esté presente y hasta a 4GB cuando el MMU esté presente.
- Soporta controladores de interrupción externa (EIC) de interfase para proveer prioridad de interrupción personalizada (opcional).
- Soporta juegos de registros *shadow* para mejorar la latencia de interrupción (opcional).
- Soporta memoria opcional estrechamente acoplada¹ para instrucciones y datos.
- Emplea segmentación de 6 etapas² para lograr DMIPS/MHz máximos.
- Desarrolla predicción de saltos dinámico³.
- Provee multiplicadores, divisores y opciones de desplazamiento por hardware para mejorar el desarrollo aritmético (opcional).
- Soporta la adición de instrucciones personalizadas.
- Soporta el módulo de depuración JTAG.

¹ Tightly coupled memory

² 6 stage pipeline

³ Dynamic branch prediction

- Soporta mejoras opcionales para el módulo de depuración JTAG, incluyendo puntos de ruptura por hardware y rastreo en tiempo real.

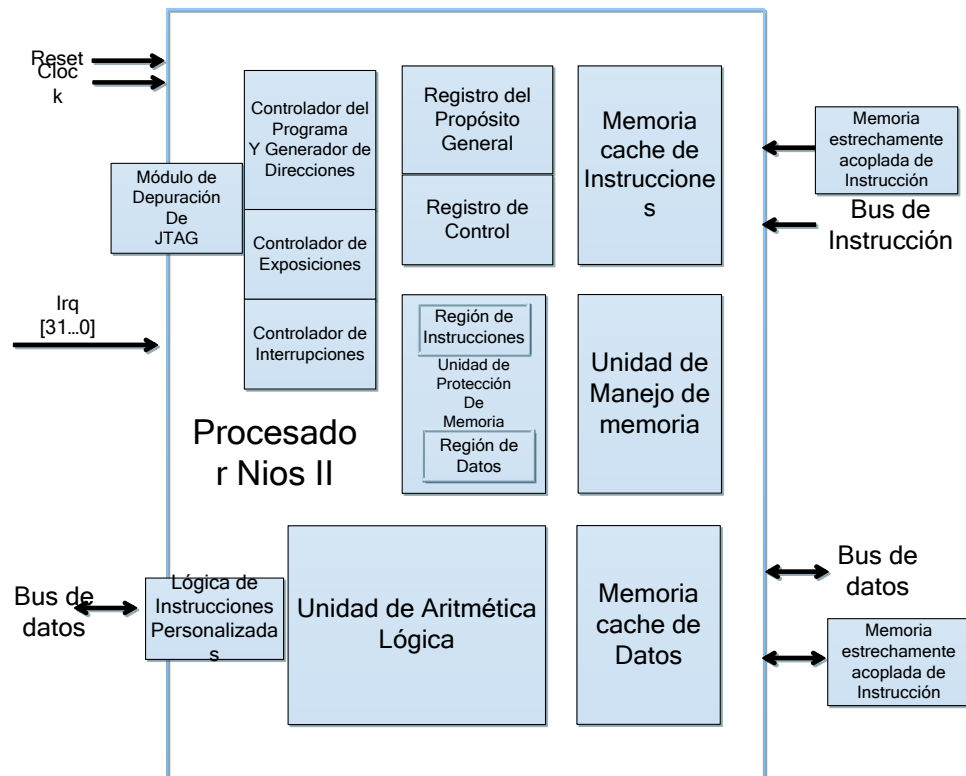


Figura 2.3 Diagrama funcional del Nios II

El núcleo Nios II/f provee cache de instrucción y datos de forma opcional. El tamaño del cache para cada uno es definido por el usuario, entre 512 bytes y 64KB. El ancho de dirección de memoria del Nios II/f depende si el MMU opcional está presente. Sin un MMU, el Nios II/f soporta el método del bit 31 para saltar el cache para acceder a E/S en el puerto maestro de datos. Por lo tanto, las direcciones son de 31 bits de ancho, reservando bit 31 para la función saltar el cache. Con un MMU, el salto del cache es una función de partición de memoria y de los contenidos de la

TLB⁴. Por lo tanto, el bit-31 de salto de cache es deshabilitado, y direcciones de 32 bits están disponibles para memoria de dirección.

La idea detrás del Nios II es que las características del soft-core del procesador deberían ser completamente transparentes ya que Altera provee un HAL API, por lo tanto el desarrollo de software para este procesador no debería ser diferente a desarrollar software para procesadores tradicionales.

2.3 AVALON SWITCH FABRIC INTERCONNECT

El sistema de interconexión Avalon Switch Fabric es la colección de recursos de interconexión y lógica que conecta los componentes en un sistema hecho en el SOPC Builder. El SOPC Builder genera el sistema de interconexión Avalon Switch Fabric para ajustarse a las necesidades específicas de los componentes.

Sus principales características son:

- Metas de diseño
 - Baja utilización de recursos del FPGA por parte del bus lógico.
 - Simplicidad.
 - Operación sincrónica.
- Interfaces que implementa
 - Avalon Memory Mapped
 - Avalon Condiut

⁴ Es una memoria cache administrada por la MMU, que contiene partes de la tabla de paginación, es decir, relaciones entre direcciones virtuales y reales.

- Avalon Tristate
- Avalon Clock
- Avalon Interrupt
- Generado a la medida para los periféricos
- Las contingencias se toman individualmente por cada periférico
- El sistema no es agobiado por la complejidad del bus.
- El SOPC Builder automáticamente genera:
 - Arbitración
 - Decodificación de la dirección
 - Ruta de datos para la multiplexación
 - Cambio de tamaño del bus
 - Generación del estado de espera
 - Interrupciones

2.3.1 AVALON MEMORY MAPPED

Las interfaces Avalon Memory-Mapped (Avalon-MM) son usados por interfaces de lectura y escritura en componentes esclavos y maestros en sistemas mapeados en memoria. Estos componentes incluyen microprocesadores, memorias, UARTs, DMAs, y temporizadores los cuales tienen interfaces maestro y esclavo. Las interfaces Avalon-MM pueden describir una gran variedad de interfaces de componentes, desde interfaces SRAM las cuales soportan transferencias de lectura y escritura simples de ciclo fijo hasta las más complejas interfaces *pipelined burst transfers*.

La figura 2.4 muestra una plantilla de una interface Avalon slave completa.

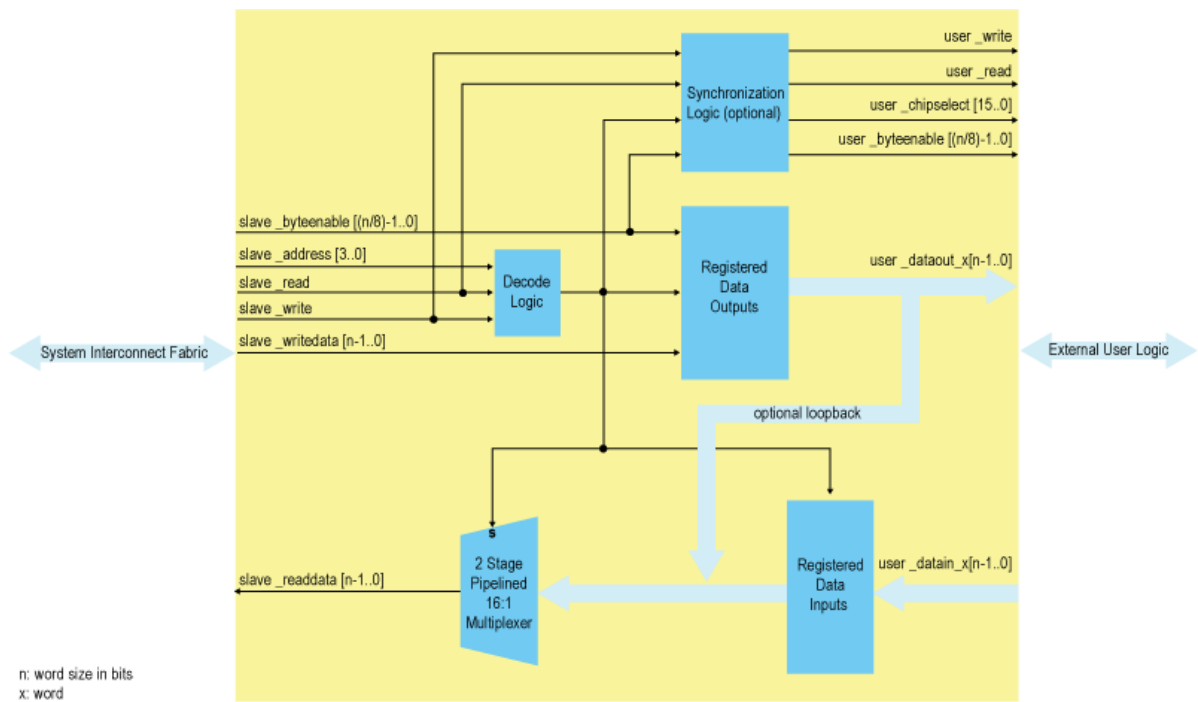


Figura 2.4 Plantilla de un componente con interface Avalon Slave.

Para los componentes Avalon Memory-Mapped (Avalon-MM), el sistema de interconexión fabric soporta transacciones emitidas por múltiples maestros Avalon hacia múltiples esclavos Avalon.

Para los componentes Avalon-MM, un puerto maestro no espera para acceder a un puerto esclavo a menos que un puerto maestro diferente intente acceder al mismo puerto esclavo en el mismo ciclo; consecuentemente, múltiples puertos maestros pueden estar activos al mismo tiempo, transfiriendo simultáneamente datos entre puertos esclavos independientes.

Se puede optimizar el sistema de interconexión fabric para componentes Avalon-MM usando puentes pipeline⁵ y puentes de cruce de reloj⁶. Al añadir manualmente puentes Avalon-MM entre los puertos Avalon-MM maestro y esclavos en un sistema, se puede controlar la topología de interconexión y hacer concesiones o canjes ente frecuencia y latencia, y entre concurrencia y uso de recursos.

Los tipos de transferencia que puede implemntar son:

- Transferencias esclavo
- Transferencias streaming
- Transferencias Latency-Aware
- Transferencias Burst

Se puede profundizar más sobre los tipos de transferencias que soporta la interface Avalon-MM en documentación de Altera: Avalon Interface Specifications.

2.3.2 INTERFACE AVALON CONDUIT

Es un tipo de interface que acomoda señales individuales o grupos de señales que no entran dentro de ninguna otro tipo de Avalon. Se puede conectar interfaces conduit dentro de un SOPC hardware o exportarlas para hacer conexiones a otros módulos en el diseño o a pines del FPGA.

⁵ Pipeline bridge

⁶ Clock-crossing bridge

2.3.3 INTERFACE AVALON TRI-STATE

Es una interface para soportar conexiones a periféricos off-chip. Múltiples periféricos pueden compartir los pines a través de la multiplexación de señales, reduciendo el conteo de pines del FPGA y el número de pistas en el PCB.

2.3.4 INTERFACE AVALON INTERRUPT

Es una interface que le permite a dispositivos los enviar interrupciones al Nios II.

Más información sobre las interfaces del bus Avalon y el sistema de interconexión Fabric de Altera, se puede encontrar en la documentación de Altera: Avalon Interface Specifications y en el Embedded Design Handbook.

2.4 DISTRIBUCIONES DE LINUX PARA NIOS II

En la tabla 2.2 se listan las distribuciones de Linux que soportan la arquitectura Nios II.

De ese listado, el que escogemos es el uCLinux que es soportado por la comunidad del AlteraWiki y AlteraForum. Y fue μ CLinux porque nuestro desarrollo no utiliza MMU. La distribución de la comunidad soporta las arquitecturas no-MMU y MMU además se mantiene al día con la versión actual del kernel de Linux que en la actualidad es la v3.2.1.

| Tabla de distribuciones de Linux que soportan la arquitectura Nios II | | | |
|---|------------------------------|--------------------------|--------------------------------------|
| SISTEMA OPERATIVO | NOMBRE DE LA DISTRIBUCIÓN | TIPO DE LICENCIA | SOPORTE |
| Linux | LinuxLink | Suscripcion/Licencia | Timesys |
| Linux | Wind River Linux | Suscripcion/Licencia | Wind River |
| Linux | Linux | BSP/Licencia | SLS |
| Linux | Sourcery CodeBench GNU/Linux | BSP/Licencia | Mentor Graphics (antes CodeSourcery) |
| Linux | nios2-linux | Libre/Gratis | Open Source Community |
| μ CLinux | μ CLinux | BSP//Licencia | SLS |
| μ CLinux | μ CLinux-nios2 | Libre/Sin soporte | Microtronix (outdated) |
| μ CLinux | μ CLinux-nios2 | Libre/Gratis/Con soporte | Open Source Community |
| Linux | Preempt - RT Project | Libre/Gratis/Con soporte | RTEL4I Project |
| μ CLinux | Xenomai | Libre/Gratis/Con soporte | RTEL4I Project |

Tabla 2.2. Distribución de Linux compatibles con Nios II

Las fuentes de SLS pueden ser descargadas para previo registro y se presentan como BSP de demostración de los productos de la compañía. Tienen un BSP para Linux (MMU) y otro para μ CLinux (no-MMU). Ya no se mantienen actualizadas y algunas son ofrecidas sin soporte.

El Nios II PREEMPT-RT Project ha trabajado en el desarrollo de un parche para el kernel de Linux que trata de reducir los niveles de latencia para aumentar el determinismo como en un sistema soft Real Time.

El Nios II Xenomai Project es similar al PREEEMT-RT pero para procesadores sin MMU.

2.5 μ CLINUX

El μ Clinux comenzó como un port de Linux kernel v2.x para micro-controladores, particularmente aquellos sin unidades de manejo de memoria (MMUs, por sus siglas en inglés). Fue creado por D Jeff Dionne y Kenneth Albanowski en 1998 usando el kernel de Linux 2.0.33 y desde entonces ha sido mantenido al día con los nuevos lanzamientos del kernel de Linux. Ahora es considerado un sistema operativo completo soportando la versión 2.6 del kernel de Linux y una selección de aplicaciones de usuario y herramientas de desarrollo.

Algunas de las características deseables de μ Clinux son:

- Linux: Conectividad IP incorporada, portabilidad, confiabilidad, manejo de sistema de archivos y software gratuito.
- Liviano: El kernel de Linux 2.6 completo ocupa menos de 300K y los archivos binarios son más pequeños cuando son construidos con la librería μ Clibc.
- Ejecución en sitio (XIP): Los ejecutables no tienen que ser cargados en la memoria RAM para correr. Esto causará que el desempeño del ejecutable decaiga.
- Código abierto.
- Rápido: Al no ser necesario los vaciados de cache se permiten cambios de contextos (en multi-procesos) más rápidos.
- Acceso de usuario al hardware.
- API de Linux completo: usa los mismos llamados de sistema que Linux (con algunas excepciones menores) haciendo más fácil a los desarrolladores aprenderlo.

- Kernel apropiativo⁷.
- Completamente multitarea.
- Soportado por muchos procesadores (en los que se incluye el Nios II).

El μ Clinux está basado en el kernel de Linux-2.6.x el cual maneja secciones críticas, planificación, servicio de interrupción de rutinas (ISR), multitareas⁸, y acceso al disco para mencionar algunas de sus tareas. Se comienza con un kernel mínimo como es de esperarse para un kernel embebido, conteniendo solo un intérprete de comandos⁹, servidores http y ftp, y un sistema de archivos básico. Es completamente personalizable para permitir al desarrollador añadir muchas otras características. Las características de interés para este proyecto es el soporte del kernel para cargar módulos.

Actualmente μ Clinux se puede encontrar como una distribución, es decir, no solo viene el código del kernel sino también una serie de aplicaciones ya portadas para él, además de configuraciones para una gran variedad de procesadores y vendedores de sistemas de desarrollo para μ Clinux. También existe en tal distribución librerías para μ Clinux como μ Clibc, entre otros.

Más adelante, se mostrará detalladamente como es la distribución μ Clinux incluyendo el código del kernel de éste.

⁷ Kernel preemption

⁸ Critical sections, scheduling, interruption service routine, multi-tasking.

⁹ Shell

2.6 DIFERENCIAS ENTRE LINUX Y μ CLINUX

A continuación se muestran las diferencias que existen entre Linux y μ Clinux. Se recomienda antes revisar todo lo relacionado con la arquitectura y el funcionamiento de Linux para un mejor entendimiento.

2.6.1 ADMINISTRACIÓN DE MEMORIA VIRTUAL

La diferencia definitiva y más prevaleciente entre μ Clinux y Linux, es la falta de administración de memoria virtual (VM). Bajo Linux, la administración de memoria virtual es llevada a través de la MMU que los procesadores tienen. μ Clinux ha sido creado para sistemas que no cuentan con tal unidad y por lo tanto no hace uso de memoria virtual.

Con MMU, todos los procesos pueden ejecutarse con las mismas direcciones de memoria, virtuales claro está, y la MMU se encarga de asignar memoria física para cada una de esas direcciones virtuales evitando que un proceso sobrescriba datos de otro, además de otras características.

Sin MMU, cada proceso debe estar ubicado en una zona de memoria donde pueda ejecutarse con sus propias direcciones físicas. En el caso más simple, esta área de memoria debe ser contigua llevando a que un proceso no pueda expandir su cantidad de memoria debido a que pueden haber procesos antes o después de esta área que el proceso podría corromper.

Aunque todos los programas necesitan ser reacomodados en tiempo de ejecución para que ellos se puedan ejecutar, esto es una tarea medianamente transparente para el programador, por lo que, el efecto de no tener MMU no es tan crítico en este caso. Sin embargo, no habrá ninguna clase de protección de memoria, llevando a que un proceso o el mismo kernel puedan corromper parte de los datos del sistema. Algunas arquitecturas ofrecen protección en modo usuario a algunas áreas de memoria, de registros y E/S pero no se puede garantizar. También, sin MMU es prácticamente imposible usar memoria de intercambio (swap) para aumentar la memoria física del sistema, ya que ésta se hace generalmente en grandes memorias como discos duros, sin embargo, un sistema embebido por lo general no necesita de esta clase de técnica.

2.6.2 DIFERENCIAS EN EL KERNEL

El sistema operativo μ Clinux tiene algunas diferencias en su kernel muy relacionadas principalmente por el hecho de no haber una MMU en el procesador. Por ejemplo, no se podrá hacer uso del soporte de paginación que ofrece una unidad de MMU. El sistema de archivos TMPFS no funciona en μ Clinux ya que él confía en las características que la Máquina Virtual le puede brindar. Similarmente, los formatos estándar de ejecutables (elf, a.out, etc.) no funcionan ya que estos también hacen uso de la VM que no está disponible en μ Clinux. Para solventar este problema, los archivos ejecutables deben tener un formato plano, el cual es un formato ejecutable condensado que almacena solo código ejecutable y datos, además de las reubicaciones necesarias para cargar el ejecutable dentro de memoria.

Los drivers de dispositivos que se usan en Linux y se desean usar en μ Clinux necesitan algunos cambios, no por las diferencias en el kernel, más bien por la manera en que los dispositivos son soportados por un sistema embebido. Por ejemplo, una tarjeta de red quizás no se conecta al sistema embebido a través de una ranura ISA o PCI, sino que hace parte del sistema embebido a través de un chip conectado directamente al bus o por algún controlador de dispositivo.

La implementación de mmap dentro del kernel es también completamente diferente. Aunque es a menudo transparente para el desarrollador, es necesario ser entendido con el fin de que no sea usado en formas que son particularmente ineficientes en sistemas μ Clinux. A menos que el mmap de μ Clinux pueda apuntar directamente al archivo dentro del sistema de archivos, garantizando que es secuencial y contiguo, este deberá asignar memoria y copiar los datos en ella. Los ingredientes para el uso eficiente de mmap bajo μ Clinux son muy específicos. Primero, el único sistema de archivos que actualmente garantiza que los archivos se almacenan contiguamente es ROMFS, por lo tanto este debe ser usado para evitar la asignación de memoria. Segundo, sólo los mapeos de sólo lectura pueden ser compartidos, lo cual quiere decir que un mapeo debe ser de sólo lectura para evitar la asignación de memoria. El desarrollador bajo μ Clinux no puede aprovecharse de las ventajas de Copy-on-Write por tal razón. El kernel también debe considerar que el sistema de archivos está en "memoria de sólo lectura", lo cual significa un área nominalmente de sólo lectura dentro del espacio de direcciones de la CPU. Esto es posible si el sistema de archivos está presente en alguna parte de la RAM o de la ROM, de las cuáles ambas son

direccionables directamente por la CPU. Uno no puede tener un mmap sin asignación de memoria cuando el sistema de archivos está en un disco duro, aun si se está utilizando ROMFS, ya que el contenido no es directamente direccionable por la CPU.

2.6.3 ASIGNACIÓN DE MEMORIA (KERNEL Y APLICACIONES)

El μ Clinux ofrece una elección de dos asignadores de memoria del kernel. Al principio puede no parecer obvio porque se necesita un asignador alternativo de memoria del kernel, pero en los sistemas μ Clinux la diferencia es dolorosamente aparente. El asignador predeterminado del kernel bajo Linux es un método de asignación basado en potencias de dos. Esto le ayuda a operar más rápido y poder encontrar velozmente áreas de memoria del tamaño correcto para satisfacer peticiones de asignación. Desafortunadamente, bajo μ Clinux, las aplicaciones deben ser cargadas dentro de la memoria que está fuera del alcance de este asignador. Para entender las consecuencias de esto, especialmente para asignaciones grandes, considere una aplicación requiriendo una asignación de 33 KB para ser cargada, realmente se le asigna la siguiente potencia de dos, la cual es 64 KB. Los 31 KB de espacio adicional asignado no pueden ser utilizados eficazmente. Este orden de desperdicio de memoria es inaceptable en la mayoría de sistemas μ Clinux. Para combatir este problema, se le ha creado el kernel de μ Clinux un asignador de memoria alternativo. Este es comúnmente conocido como `page_alloc2` o `kmalloc2`, dependiendo de la versión del kernel.

page_alloc2 se encarga del derroche del asignador estándar basado en potencias de dos, haciendo que solo se pueda asignar hasta el tamaño de una página (una página es de 4,096 bytes, o 4 KB). Para el ejemplo previo, una aplicación de 33 KB realmente tendrá asignado 36 KB; ahorrando 28 KB. page_alloc2() también trata de evitar la fragmentación de la memoria, ubicando todas las cantidades menores o iguales a dos páginas (8 KB) al principio de la memoria y las mayores al fin de la memoria libre (8 KB). Esto impide asignaciones transientes para los búferes de la red y otros, que fragmentan la memoria y evitan que grandes aplicaciones puedan ser ejecutadas. La figura 2.5 muestra mejor esto. page_alloc2() no es perfecto, sin embargo, funciona bien en aplicaciones embebidas que por lo general son más estáticas que dinámicas.



Figura 2.5 Organización de los procesos en la memoria

Una vez que el desarrollador supera las diferencias de asignación de memoria del kernel, los cambios verdaderos aparecen en el espacio de aplicación. Aquí es donde se da el mayor impacto por la falta de memoria virtual en μ Clinux. La primera diferencia principal más probable que puede causar que una aplicación falle bajo μ Clinux es la falta de un stack dinámico. En Linux, cuando una aplicación, trata de escribir fuera de la parte superior del stack, una excepción es activada y por lo tanto

más memoria es mapeada en la parte superior de ella permitiéndole crecer. Bajo μ Clinux, tal lujo no está disponible por lo que el stack debe ser asignado en la fase de compilación. Esto quiere decir que el desarrollador, que antes no era consciente del uso del stack dentro de su aplicación, ahora debe darse cuenta de los requerimientos que esta necesita. La primera cosa que un desarrollador debería considerar cuándo está afrontado comportamientos o bloqueos extraños en una aplicación que se ha acabado de portar, es el tamaño asignado para el stack. Por defecto, las herramientas de desarrollo (toolchains) para μ Clinux asignan 4 KB para el stack, lo cual no se acerca en nada a las aplicaciones modernas. El desarrollador debería probar aumentar el tamaño del stack con uno de los siguientes métodos:

1. Añadir `FLTFLLAGS = -s <stacksize>` y `export FLTFLLAGS` al archivo `Makefile` de la aplicación antes de construirla.
2. Ejecutar `flthdr -s <stacksize> executable`, después de que la aplicación se ha construido.

La segunda diferencia principal que afecta a un desarrollador de μ Clinux es la falta de *heap* dinámico, está corresponde al área usada para satisfacer asignaciones de memoria con `malloc` y funciones C relacionadas. En Linux con memoria virtual, una aplicación puede aumentar su tamaño de proceso, permitiéndole tener un *heap* dinámico. Esto tradicionalmente es implementado a bajo nivel usando las llamadas al sistema `sbrk()/brk()`, las cuales incrementan/cambian el tamaño del espacio de direcciones de un proceso. La administración del *heap* por funciones de librerías tales

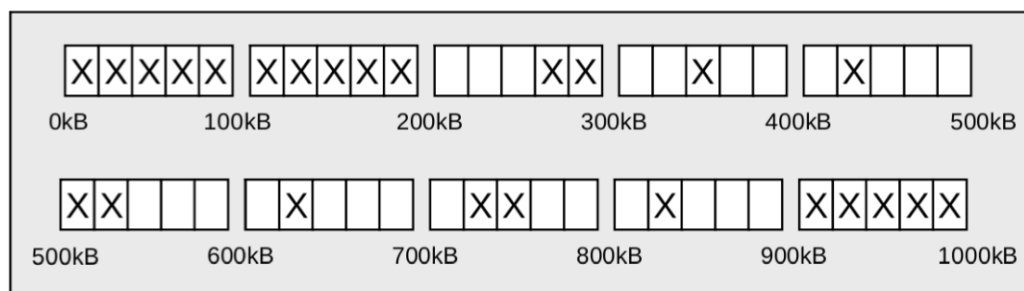
como malloc es realizada con memoria extra que se obtiene cuando se llama a sbrk() en nombre de la aplicación. Si una aplicación necesita más memoria en cualquier punto, puede obtener más simplemente llamando a sbrk() otra vez; También puede disminuir la memoria usando a brk(). Sbrk() trabaja adicionando más memoria al fin de un proceso (aumentando su tamaño). El brk() arbitrariamente puede establecer el fin del proceso para estar más cerca al inicio del proceso (reducir el tamaño de proceso) o más lejos (aumentar el tamaño de proceso).

Debido a que μ Clinux no puede implementar la funcionalidad de brk () y sbrk (), en lugar de eso implementa un pool de memoria global que básicamente es un pool de memoria libre del kernel. Hay dificultades con este método, por ejemplo, un proceso descontrolado puede usar la memoria disponible de todo el sistema. La asignación desde el pool no es compatible con sbrk () y brk (), ya que ellos requieren que la memoria sea adicionada al final del espacio de direcciones de un proceso. Así, una implementación normal del malloc no es buena, y por lo tanto se hace necesario una nueva implementación.

Un método de pool global tiene algunas ventajas. Primero, solo la cantidad de memoria actualmente requerida es usada, a diferencia del heap preasignado del sistema que algunos sistemas embebidos usan. Esto es sumamente importante en sistemas μ Clinux, los cuales generalmente corren con poca memoria. Otra ventaja es que la memoria puede ser devuelta al pool global tan pronto como su uso se finalice,

y la implementación puede tomar ventaja del asignador existente dentro del kernel para gestionar esta memoria, reduciendo el tamaño del código de aplicación.

Uno de los problemas comunes que los nuevos usuarios encuentran es el problema de falta de memoria. El sistema muestra una cantidad grande de memoria libre, pero una aplicación no puede ubicar un buffer de tamaño X y el problema aquí es la fragmentación de memoria, y todas las soluciones μ Clinux disponibles en este momento padecen de ella. Debido a la falta de memoria virtual en el ambiente μ Clinux, es casi imposible utilizar la memoria completamente debido a la fragmentación. Esto se explica mejor con un ejemplo. Suponga que un sistema tiene 500 KB de memoria libre y uno tiene el deseo de ubicar 100 KB para cargar una aplicación. Es fácil de pensar que esto sería posible. Sin embargo, es importante recordar que uno debe tener un bloque contiguo de 100 KB de memoria para satisfacer tal asignación. Suponga que el mapa de memoria tiene el aspecto mostrado en la figura 2.6. Cada bloque representa 20 KB, y X señala áreas asignadas o en uso por otros programas o por el kernel:



Memoria Total: 1000 kB
 Memoria Libre: 500 kB
 Tamaño Bloque: 20kB
 Max. Bloque Contiguo: 80kB

Figura 2.6 Fragmentación de la memoria

En este caso, los 500 KB están libres, pero el bloque contiguo más grande es de sólo 80 KB. Hay muchas formas para lograr semejante situación. La causa más probable puede estar relacionada con un programa que ubica algo de memoria y luego libera la mayor parte de ella, dejando una pequeña asignación en la mitad de un gran bloque libre. Los programas transientes bajo μ Clinux también pueden afectar dónde y cómo la memoria es asignada. El asignador del kernel de μ Clinux `page_alloc2` tiene una opción de configuración que puede ayudar a identificar este problema.

Habilitando esto, aparece una nueva entrada en `/proc`, `/proc/mem_map` que muestra las páginas y sus asignaciones por grupo. Documentar esto está más allá del alcance de este trabajo, pero más información puede ser encontrada en el código fuente del kernel para `page_alloc2.c`.

La pregunta realizada a menudo es, ¿por qué la memoria no puede ser desfragmentada para que sea posible cargar una aplicación de 100 KB? El problema es que no tenemos memoria virtual y por lo tanto no podemos mover la memoria que

está siendo usada por otros programas. Los programas usualmente tienen referencias a direcciones que están dentro de las regiones asignadas, y sin memoria virtual para hacer que la memoria parezca tener las direcciones correctas, el programa podría fallar o bloquearse si intentamos mover su memoria. No hay solución para este problema bajo μ Clinux. El desarrollador necesita darse cuenta del problema y, en lo posible, intentar utilizar pequeños bloques de asignación.

2.6.4 APLICACIONES Y PROCESOS

Otra diferencia entre Linux y μ Clinux es la falta de la llamada de sistema `fork()`. Esto puede requerir bastante trabajo de parte del desarrollador cuando está portando aplicaciones que usan `fork()`. La única opción bajo μ Clinux es usar `vfork()`.

Aunque `vfork()` comparte muchas propiedades con `fork()`, las diferencias están en lo más importante. `fork()` y `vfork()` le permiten a un proceso dividirse en dos, un padre y un hijo. Un proceso puede dividirse muchas veces para crear múltiples hijos. Cuando un proceso llama a `fork()`, el hijo es un duplicado del padre en todas las formas, pero no comparte nada con él y puede funcionar independientemente, tal como puede hacerlo el padre. Con `vfork()` esto no es el caso. Primero, el padre es suspendido y no puede continuar ejecutándose hasta que el hijo termine o llame a `exec()`, la llamada de sistema usada para iniciar una nueva aplicación. El hijo, directamente después del llamado a `vfork()`, se ejecuta con el stack del padre y además está usando la memoria y los datos del mismo. Esto quiere decir que el hijo puede corromper las estructuras de datos o el stack en el padre, dando como resultado una falla. Esto se evita

asegurando que el hijo nunca retorne de la ejecución actual hasta que haga uso de `exit()` o `exec()`. El hijo también debe evitar cambiar cualquier información en las estructuras globales de datos o las variables, ya que tales cambios pueden parar la ejecución del padre.

El formato plano de ejecutables en μ Clinux, aunque no afecta directamente a las aplicaciones y sus operaciones, permite algunas opciones que usualmente los ejecutables de formato ELF no pueden realizar bajo Linux. El formato plano de ejecutables viene en dos clases básicas, completamente reubicable y una variación del Position Independent Code (PIC). La versión completamente reubicable tiene reacomodación para el código y los datos, mientras la versión PIC generalmente necesita sólo algunas reubicaciones para sus datos.

Una de las características más ventajosas para el desarrollador embebido es la de *execute-in-place* (XIP). La aplicación puede ejecutarse directamente desde flash o ROM, requiriendo solo un mínimo de memoria para los datos que la aplicación necesita. Esto le permite a la porción de código ser compartido entre múltiples instancias de la aplicación. No todas las plataformas μ Clinux son capaces de soportar XIP, ya que este requiere que el compilador lo soporte y el formato plano de ejecutable sea con versión PIC. Así es que a menos que la toolchain para una plataforma dada pueda hacer uso de PIC, no se puede hacer un XIP. Actualmente, sólo las toolchains de M68k y de ARM proveen el nivel requerido de soporte para formato plano XIP. Además, ROMFS es el único sistema de archivos para mantener a

XIP bajo μ Clinux, porque la aplicación debe ser almacenada contiguamente dentro del sistema de archivos para que XIP sea posible.

El formato plano también define el tamaño del stack para una aplicación como un campo en el encabezado plano. Aumentar el stack asignado a una aplicación, lleva realizar un pequeño cambio en esta parte. Esto puede ser hecho con el comando `flthdr`, así:

```
flthdr -s flat-executable .
```

El formato plano también permite dos opciones de compresión. La primera es comprimir el ejecutable completamente, proporcionando un ahorro máximo en la ROM. También le ofrece el efecto secundario, a menudo útil, que es cargar la aplicación enteramente en un bloque contiguo de RAM. La segunda es comprimir sólo el segmento de datos. Esto es importante si usted quiere ahorrar espacio de la memoria de sólo lectura y seguir utilizando XIP. El siguiente comando crea un ejecutable comprimido completamente:

```
flthdr -z flat-executable
```

Y la siguiente comprime sólo el segmento de datos:

```
flthdr -d flat-executable
```

2.6.5 LIBRERÍAS COMPARTIDAS

Aunque una discusión completa de librerías compartidas está más allá del alcance de este trabajo, ellas son realmente diferentes bajo μ Clinux. Las soluciones actualmente disponibles requieren cambios del compilador y cuidado de parte del desarrollador. La mejor forma para crear librerías compartidas debe comenzar con un ejemplo. Las distribuciones actuales de μ Clinux proveen las librerías compartidas para μ C-libc y μ Clibc. El método para crear una librería compartida no es difícil, y estas librerías proveen un buen ejemplo de como se hace. Para establecer expectativas apropiadas, la opción `-shared` de GCC no es parte del proceso de creación de la librería compartida. Las librerías compartidas bajo μ Clinux son de formato plano, justo como las aplicaciones, y para ser verdaderamente compartidas deben ser compiladas para XIP. Sin XIP, las librerías compartidas dan como resultado una copia completa de ella por cada aplicación que la usa, lo cual es peor que enlazar las aplicaciones estáticamente.

2.6.6 LIBRERÍA C USADA CON μ CLINUX

Como vimos antes, Linux hace bastante uso de la librería C ya que allí están depositadas las funciones C que sirven para realizar llamadas al sistema y poder acceder tanto al hardware como a los recursos del kernel. Dicha librería es de gran tamaño y normalmente contiene un centenar de funciones que le sirven a las aplicaciones para realizar algunas tareas. Sin embargo, en un sistema embebido que

por lo general cuenta con baja cantidad de memoria, tanto de RAM como de ROM, implementar la librería estándar de Linux en estos sistemas resulta casi imposible.

Cuando el proyecto μ Clinux nació, prontamente contó con el proyecto μ Clibc el cual generó una librería de nombre similar que tenía toda la funcionalidad de la librería estándar de Linux pero con un tamaño bastante menor, apta para poderse instalar y adecuar en un sistema embebido junto con μ Clinux. μ Clibc es una librería ligera de C para sistemas embebidos, no obstante, con la mayoría de características de las librerías principales.

El proyecto μ Clibc ahora es independiente aunque μ Clinux normalmente sigue soportando tal librería. Ya existe una versión de Nios II portada con dicha librería.

2.6.7 DISTRIBUCIÓN DE μ CLINUX

Actualmente μ Clinux se puede encontrar como una distribución, es decir, no solo viene el código del kernel sino también una serie de aplicaciones ya portadas para él, además de configuraciones para una gran variedad de procesadores y fabricantes de sistemas de desarrollo μ Clinux como μ Clibc, entre otros. Más adelante, se mostrará detalladamente como es la distribución μ Clinux incluyendo del código kernel de este.

La distribución oficial uClinux-dist es mantenida en el sitio web uclinux.org. En el proyecto utilizamos la distribución del Altera Wiki [nios2-linux](http://www.altera.com/wiki/nios2-linux) que esta tiene partes de uClinux-dist.

2.7 DISEÑO DEL SISTEMA EN CHIP PROGRAMABLE (SOPC)

El desarrollo basado en SOPC ofrece nuevas alternativas de espacio de diseño. Es posible explorar opciones de diseño que usan software, hardware dedicado, o una mezcla de ambos. Las soluciones de hardware ofrecen cálculos más rápidos, pero ofrecen menos flexibilidad y puede requerir una FPGA de mayores dimensiones. Las soluciones implementadas usando software facilitan el diseño de algoritmos más complejos.

Es posible también considerar una combinación de ambos enfoques. Algunos kernels de procesadores permiten al usuario añadir instrucciones personalizadas. Si un programa de aplicación requiere el mismo cálculo repetidamente en bucles, añadir una instrucción personalizada usando hardware extra para acelerar el código del lazo interno puede acelerar la aplicación.

Una infraestructura flexible puede también ser beneficioso para extender la vida (por lo tanto reducir el costo) del hardware de un producto. Con flexibilidad y lógica reconfigurable, frecuentemente una tarjeta de circuito impreso puede ser diseñada para que pueda ser usada en múltiples líneas de productos y en múltiples versiones de un mismo producto. Usando lógica reconfigurable como el corazón del diseño, le permite ser reprogramado para implementar un amplio rango de sistemas y diseños. Extendiendo la vida de la placa diseñada, incluso una generación puede resultar en ahorros significativos y puede reducir el incremento del precio por unidad de equipos reconfigurables.

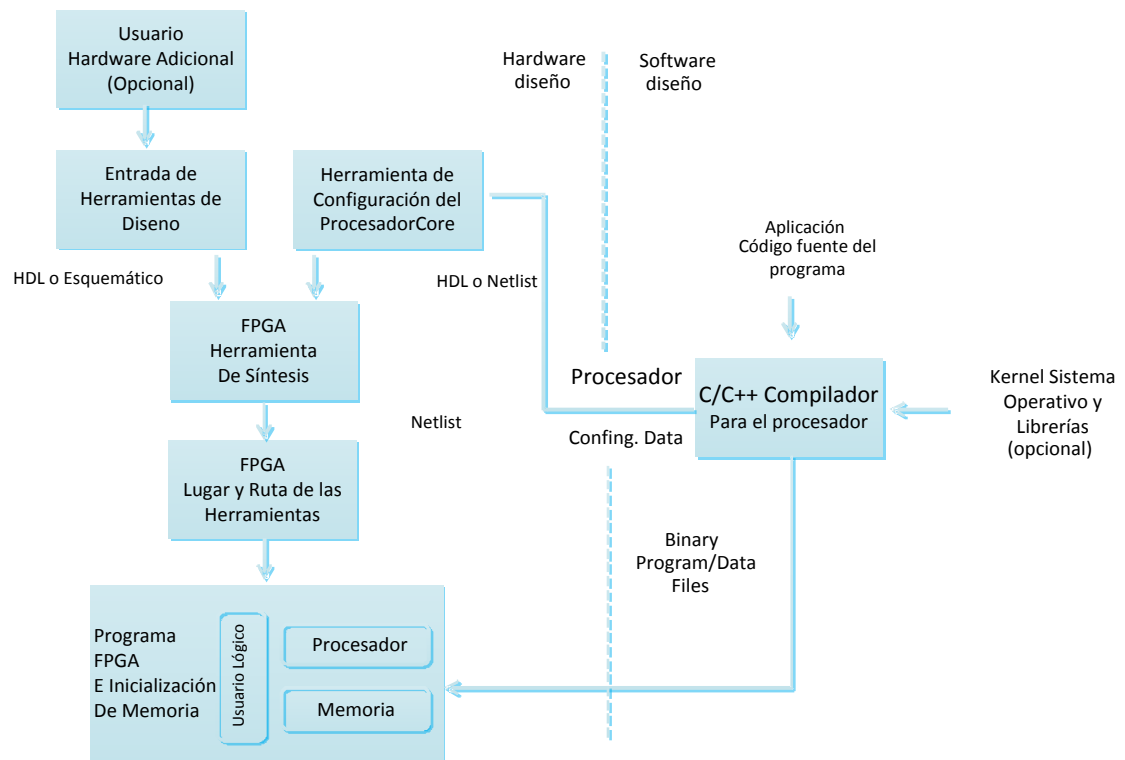


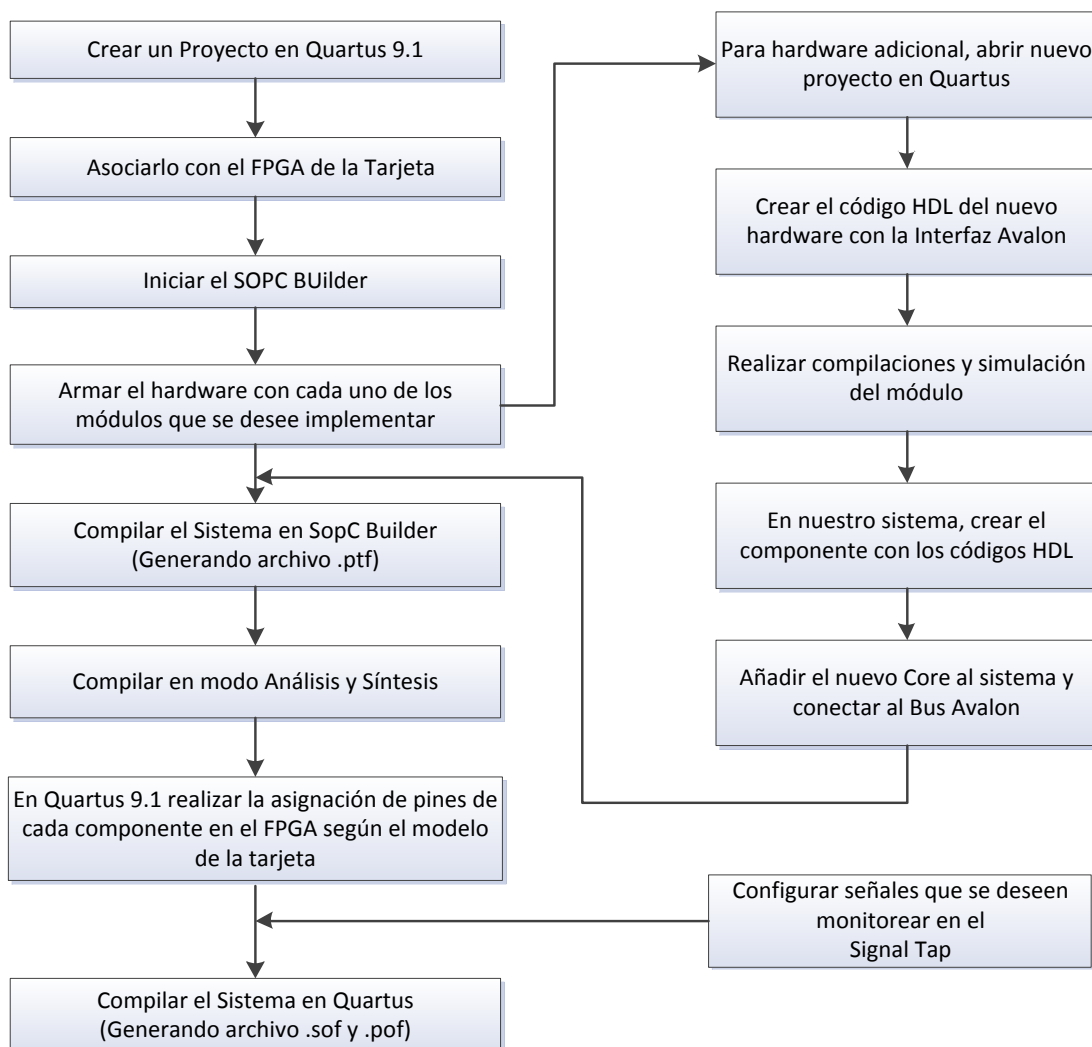
Figura 2.7 Esquema de Diseño SOPC

En el diagrama de bloques de la figura 2.7 se esquematiza el proceso de diseño en hardware y software bajo SOPC.

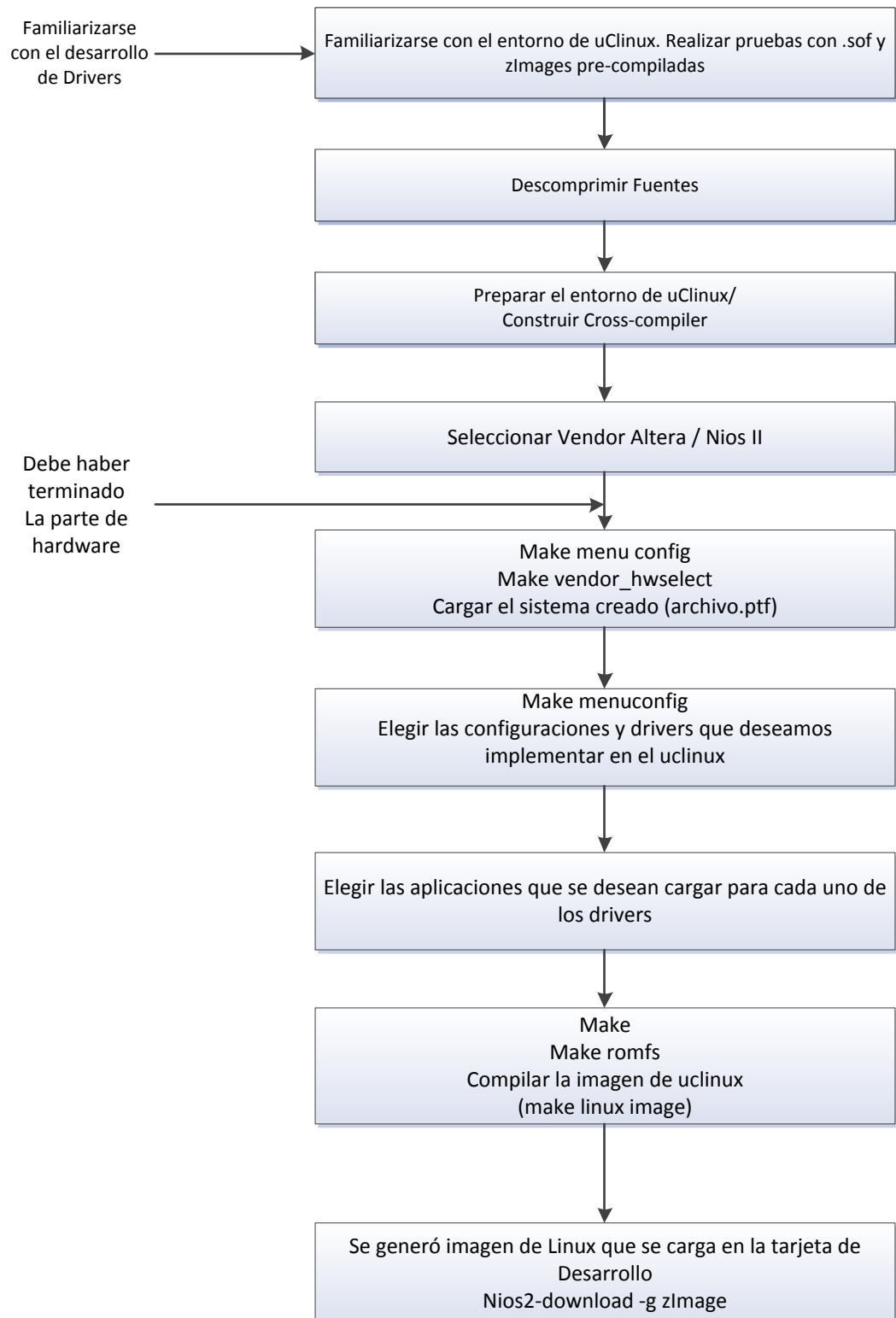
2.8 FLUJOGRAMA DEL PROCESO DE DESARROLLO

Para entender el desarrollo del proyecto se lo separa en tres flujos, el primero se refiere como seguir los pasos para el desarrollo del hardware mientras que el segundo explica el desarrollo del software paso a paso y un tercero que indicara como se unen hardware y software para el resultado final del proyecto.

FLUJO DEL HARDWARE



FLUJO DEL SOFTWARE



2.9 HARDWARE Y TOOLCHAIN DE COMPILACION DE UCLINUX

2.9.1 HARDWARE

2.9.1.1 TARJETA TERAASIC ALTERA DE2

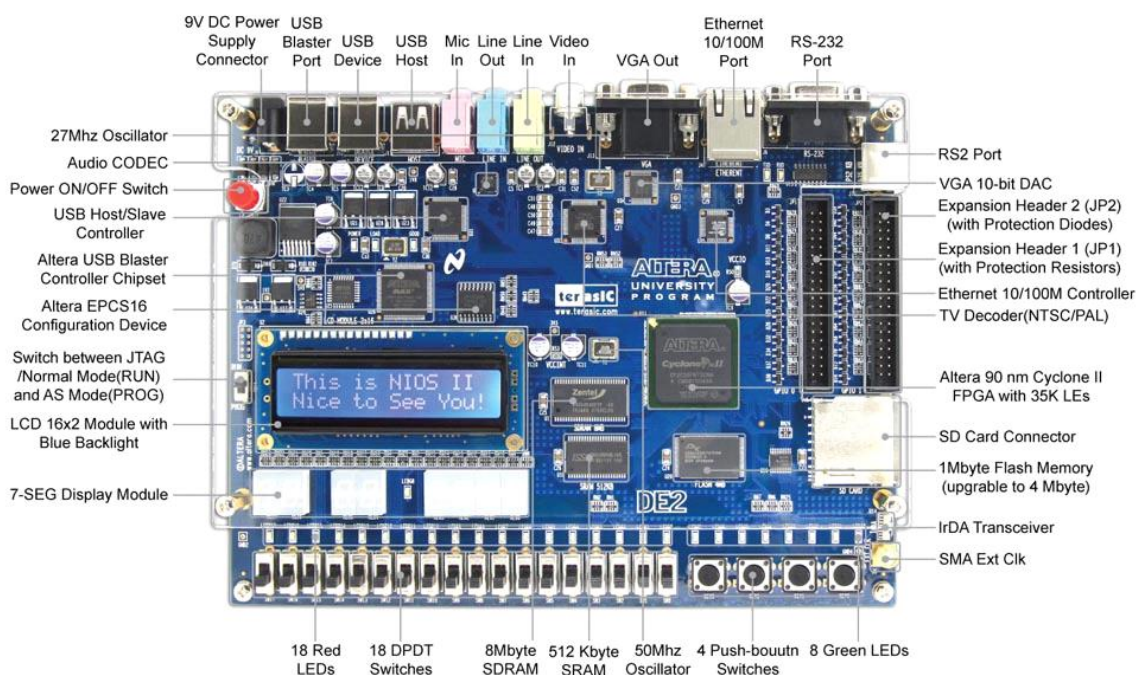


Figura 2.8 Tarjeta Terasic DE2

La tarjeta de desarrollo y educación Altera DE2 figura como parte del Programa Universitario de Altera como una plataforma de desarrollo orientada a cursos de pregrado y de grado.

Fue lanzada al mercado en el año de 2006 (aproximadamente) y es mantenida como una solución de bajo costo para competir con dispositivos ASIC de similares prestaciones.

Aún es utilizada en muchas universidades en las áreas de sistemas digitales y arquitectura de computadoras. En el sitio web de Terasic se pueden encontrar enlaces

hacia los cursos y proyectos que usan la tarjeta DE2 en varias universidades a nivel mundial.

Las especificaciones de la tarjeta son las siguientes:

- FPGA Altera Cyclone II 2C35 con 35000 LEs (elementos lógicos)
- Dispositivo de configuración serial de Altera (EPCS16) para Cyclone II 2C35
- USB Blaster incorporado en la tarjeta para programación y control desde los API de usuario
- Modo JTAG y AS son soportados
- 8Mbyte (1M x 4 x 16) de memoria SDRAM
- 512K byte(256K X16) de memoria SRAM
- 4Mbyte de memoria Flash
- Ranura para Tarjeta SD
- 4 Botoneras con anti-rebote
- 18 interruptores
- 9 LEDs verde de usuario
- 18 LEDs rojo de usuario
- Módulo LCD de 16x2
- Oscilador de 50MHz, 27MHz y una entrada SMA para oscilador externo
- CODEC de Audio 24-bit con jacks para Entrada de línea, salida de línea y entrada de micrófono
- VGA DAC 10-bit con conector de salida VGA

- TV Decoder (NTSC/PAL) y conector para entrada de TV
- Controlador Ethernet 10/100 con socket RJ-45
- Controlador USB Host/Slave con conectores USB tipo A y tipo B
- Transceptor RS-232 y conector de 9 pines
- Conector PS/2 para mouse/teclado
- Transceptor IrDA
- Ranuras de expansión (dos ranuras de 40-pin) con diodo de protección

2.9.1.2 **HARDWARE SOPC**

En este apartado revisaremos la arquitectura del hardware implementada para el sistema de la tarjeta. Los programas de diseño que intervienen aquí son el SOPC Builder para la creación del sistema con sus componentes modulares y el Quartus II para la integración en un archivo top level en código HDL y la creación de la imagen de hardware con la que se programará en el FPGA.

Para completar el diseño del hardware con las herramientas de Altera, tenemos que hacer lo siguiente:

- Crear o abrir un proyecto de Quartus II.
- Iniciar el SOPC Builder y crear el sistema, añadir componentes, etc.
- Generar el SOPC Hardware. Se generan los archivos extensión PTF, SOPC, HTML.
- Crear una instancia del módulo generado por el SOPC Builder en el archivo de jerarquía superior del proyecto Quartus II.

- Compilar primero en modo “Análisis y Síntesis”.
- Asignar los pines de FPGA.
- Ejecutar una compilación completa. Se genera el archivo SOF.
- Programar el FPGA con el SOF generado.

Después de haber configurado el hardware necesario en el dispositivo FPGA, ahora es necesario crear y ejecutar una aplicación-programa que realice la operación deseada.

2.9.1.2.1 REQUERIMIENTO MÍNIMOS DE UCLINUX

Los requisitos mínimos del SOPC hardware para μ Clinux son (24):

- Nios II/f o Nios II/s, con multiplicador por hardware (f es sugerido, s es más lento)
- SDRAM (mínimo requerimiento 8MB)
- Un temporizador completamente funcional
- Un jtag UART o un serial UART.

En Linux, el IRQ 0 es detectado automáticamente, por lo que no se debe usar IRQ 0 para ningún dispositivo, excepto para el temporizador.

2.9.1.2.2 SYSTEM_0

SYSTEM_0 es el nombre que le hemos dado a todo nuestro sistema de hardware reconfigurable. Una vez generado por el SOPC Builder, lo podemos llamar como modulo o bloque dentro de un proyecto de Quartus II.

En la tabla 2.3 listamos los componentes modulares utilizados en nuestra arquitectura Nios II con los nombres, tipos de componentes y la dirección base en el Bus Avalon. También asociamos cada componente con su identificación en el archivo nios2.h que es generado por los scripts de configuración del compilador de μ Clinux.

| Nombre de módulo SOPC | Tipo de componente | Dirección de memoria base | #define en nios2.h |
|-----------------------|------------------------------|---------------------------|----------------------|
| cfi_flash_0 | Flash Memory Interface (CFI) | 0x01000000 | na_cfi_flash_0 |
| s dram_0 | SDRAM Controller | 0x02800000 | na_s dram_0 |
| e pcs_controller | EPCS Serial Flash Controller | 0x01400800 | na_e pcs_controller |
| jtag_uart_0 | JTAG UART | 0x01401150 | na_jtag_uart |
| uart_0 | UART (RS-232 Serial Port) | 0x014010a0 | na_nasys_printf_uart |
| timer_0 | Interval Timer | 0x01401080 | na_timer0 |
| lcd_16207_0 | Character LCD | 0x014010e0 | na_lcd_16207_0 |
| ps2_0 | PS2 Serial Port | 0x01401158 | na_ps2_0 |
| pipeline_bridge | AvalonMM Pipeline Bridge | 0x01000000 | na_pipeline_bridge |
| onchip_memory2_0 | On-Chip Memory (RAM or ROM) | 0x03000000 | na_onchip_memory2_0 |
| dm9000 | DM9000A | 0x01401160 | na_dm9000 |
| sysid | System ID Peripheral | 0x01401168 | na_sysid |
| gpio_0 | Gpio | 0x01401000 | na_gpio_0 |
| ISP1362 | ISP1362_IF | 0x014010c0 | na_mmc_spi |
| mmc_spi | SPI (3 Wire Serial) | 0x014010f0 | na_ISP1362 |
| led_pio | PIO (Paralell I/O) | 0x01401100 | na_led_pio |
| button_pio | PIO (Paralell I/O) | 0x01401110 | na_button_pio |

| | | | |
|---------------|--------------------------|------------|------------------|
| switch_pio | PIO (Paralell I/O) | 0x01401120 | na_switch_pio |
| pwm_avalon_0 | pwm_avalon_interf ace | 0x01401130 | na_pwm_avalon_0 |
| altpll_0 | Avalon ALTPLL | 0x01401140 | na_altpll_0 |
| seven_seg_pio | SEG7_LUT_8 | 0x01401178 | na_seven_seg_pio |

Tabla 2.3 Identificación y asignación de memoria de los componentes del System_0

La figura 2.9 esquematiza las conexiones del cpu_0, el procesador Nios II/f que fue configurado sin MMU, memoria cache de datos e instrucción, memoria estrechamente acoplada para los puertos maestros de datos e instrucción (onchip_memory2_0). El cpu_0 se conecta por medio de un Puente de segmentación (bridge_pipeline) con el resto de periféricos de la tarjeta. Los beneficios de esta arquitectura pueden ser profundizados en las documentación de Altera.

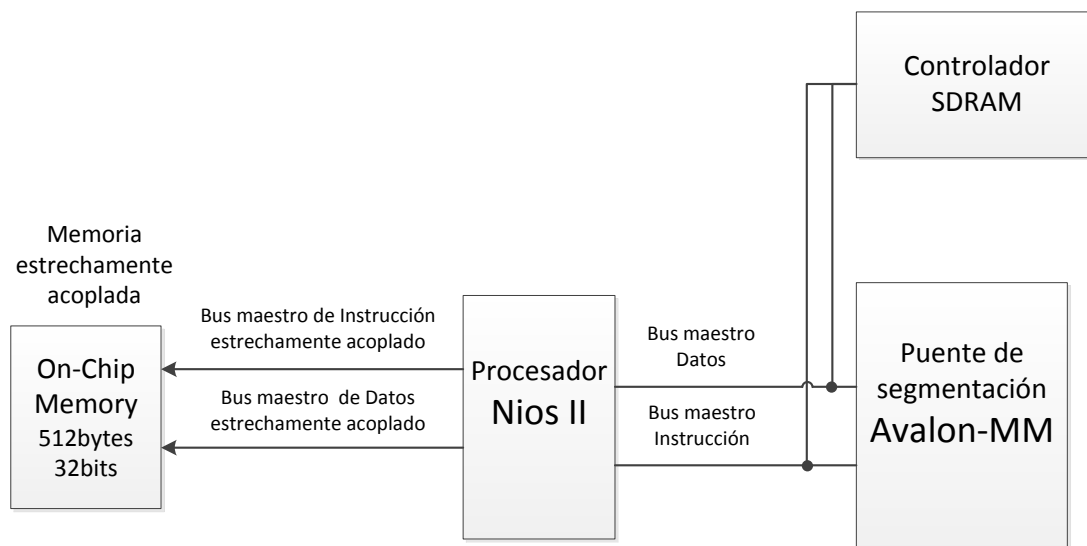


Figura 2.9 Conexiones a cpu_0

La figura 2.10 es una captura de pantalla del SOPC Builder que muestra todo el sistema construido con las direcciones de memoria asignadas, y las asignaciones de número de interrupción.

El archivo nio2.h es generado por μ Clinux-dist durante el make HWSELECT por medio de unos scripts. Por eso es importante respetar los nombres para los componentes que utilizamos ya que son los recomendados por los desarrolladores del Altera Wiki.

En la figura 2.11 se muestra un diagrama de bloques de la arquitectura de hardware configurable del system_0.

Altera SOPC Builder - system_0.sopc* (/home/diegxi/projects_q2/ae2_vga_nommu2/dezvg_nommu/system_0.sopc)

File Edit Module System View Tools Nios II Help

System Contents System Generation

Component Library

- New component...
- Library
- pio_eigen
- Avalon Verification Suite
- Bridges and Adapters
- Memory Mapped
- Streaming
- Display
- Interface Protocols
- ISP 362_IF
- ASI
- Ethernet
- High Speed
- PCI
- Serial
- Legacy Components
- Active Serial Memory
- AMD 29LV128M Flash
- Legacy AMD 29LV06
- Legacy IDT71V416 S
- Memories and Memory Control
- QDR II and QDR II+ SF
- RLDRAM II Controller
- Traffic Generator and DMA
- Flash
- On-Chip SRAM

Target: Device Family: Cyclone II

Module Name: onchip_memory2_0

Use: Con...

| Use | Con... | Module Name | Description (RAM or ROM) | Clock | Base | End | Tags | IRQ |
|-------------------------------------|-------------------------------------|--------------------|------------------------------|----------|------------|------------|------|-----|
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | onchip_memory2_0 | On-Chip Memory (RAM or ROM) | multiple | 0x00400000 | 0x004007ff | | 0 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | cpu_0 | Nios II Processor | sys_clk | 0x00800000 | 0x007fffff | | 1 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | sdram_0 | SDRAM Controller | sys_clk | 0x00000000 | 0x007fffff | | 2 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | pipeline_bridge_0 | Avalon-MM Pipeline Bridge | sys_clk | 0x00000000 | 0x007fffff | | 3 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | tri_state_bridge_0 | Avalon-MM Tristate Bridge | sys_clk | 0x00000000 | 0x007fffff | | 4 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | cfi_flash_0 | Flash Memory Interface (CFI) | sys_clk | 0x00000000 | 0x003fffff | | 5 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | timer_0 | Interval Timer | sys_clk | 0x00401080 | 0x0040109f | | 6 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | epcs_controller | EPCS Serial Flash Controller | sys_clk | 0x00400800 | 0x00400fff | | 7 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | jtag_uart_0 | JTAG UART | sys_clk | 0x00401150 | 0x00401157 | | 8 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | uart_0 | UART (RS-232 Serial Port) | sys_clk | 0x004010a0 | 0x004010bf | | 9 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | led_16207_0 | Character LCD | sys_clk | 0x00401178 | 0x0040117b | | 10 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | seven_seg_pio | PS2 Serial Port | sys_clk | 0x00401158 | 0x0040115f | | 11 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | ps2_0 | SE7 LUT_8 | sys_clk | 0x00401160 | 0x00401167 | | 12 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | dm9000 | DM9000A | sys_clk | 0x00401168 | 0x0040116f | | 13 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | sysid | System ID Peripheral | sys_clk | 0x00401000 | 0x0040107f | | 14 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | gpio_0 | gpio | sys_clk | 0x004010c0 | 0x004010df | | 15 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | mmc_spi | SPI (3 Wire Serial) | sys_clk | 0x00401000 | 0x0040107f | | 16 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | isp1362 | ISP 362_IF | sys_clk | 0x00401100 | 0x0040110f | | 17 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | led_pio | PIO (Parallel I/O) | sys_clk | 0x00401110 | 0x0040111f | | 18 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | button_pio | PIO (Parallel I/O) | sys_clk | 0x00401110 | 0x0040111f | | 19 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | switch_pio | PIO (Parallel I/O) | sys_clk | 0x00401120 | 0x0040112f | | 20 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | pwm_avalon_0 | pwm_avalon_interface | sys_clk | 0x00401130 | 0x0040113f | | 21 |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | altpll_0 | Avalon ALTPLL | clk_50 | 0x00401140 | 0x0040114f | | 22 |

Filter: Default

Remove Edit... Address Map... Filter... Next... Generate

Info: cfi_flash_0: Flash memory capacity: 4.0 MBytes (4194304 bytes).

Info: button_pio: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Info: switch_pio: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figura 2.10 Construcción de la arquitectura Nios II con el SOPC Builder

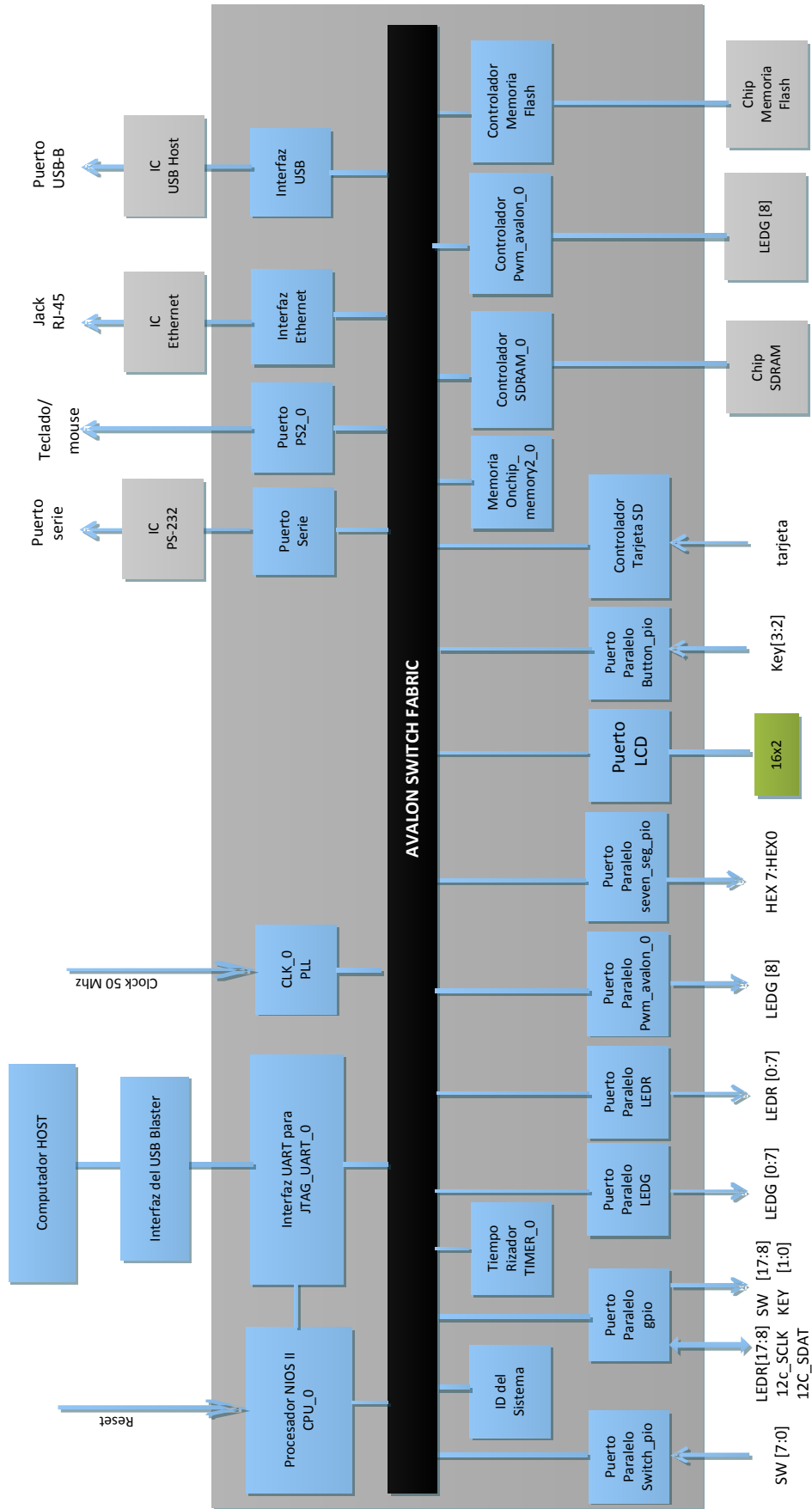


Figura 2.11 Diagrama de Bloques del Sistema

2.9.1.2.3 MODULO PWM

Este es el módulo de ejemplo que ofrece la guía de Altera para el Desarrollo de periféricos para SOPC Builder y se distribuye gratuitamente en Internet. Lo hemos usado para nuestra implementación en la instalación de Linux.

Se compone de tres archivos de código Verilog HDL:

- pwm_task_logic.v
- pwm_register_file.v
- pwm_avalon_interface.v

Características del Módulo de Hardware:

- Operación sincrónica con un solo clock maestro.
- Usa valores de 32 bits para proveer un rango apropiado a periodos y ancho de pulso del PWM.
- Un procesador anfitrión es el responsable de asignar el valor del periodo y el ancho de pulso del PWM. Esto implica la necesidad de una interface de lectura/escritura para la lógica de control.
- Se definen registros para sostener el valor del periodo y el ancho de pulso del PWM.

Las señales de interface de este módulo son las siguientes y la convención de nombres de las señales y buses de datos es tomando como referencia al Maestro.

- address: offset interno de la dirección.
- chipselect: señal que habilita el dispositivo.

- clk: señal de reloj para sincronización de la interface Avalon. Todas las señales son síncronas al clk.
- resetn: resetea el dispositivo. El desarrollado decide que pasa en ese estado.
- write: habilita la escritura de información desde el registro writedata a otro registro del dispositivo.
- read: habilita la escritura de datos desde un registro del dispositivo hacia el registro readdata.
- writedata: registro de datos para las transferencias de escritura.
- pwm_out: línea de salida que conecta el dispositivo con un I/O pin del FPGA.
- readdata: registro de datos para las transferencias de lectura.

```

/*****
/* File: pwm_avalon_interface.v                                     */
/* Description: Top level module.  Instantiates pwm_task_logic and */
/*    pwm_register_file modules and adds Avalon slave interface.  */
/*****

module pwm_avalon_interface
(
    clk,
    resetn,
    avalon_chip_select,
    address,
    write,
    write_data,
    read,
    read_data,
    pwm_out
);

```

Figura 2.12 Módulo de nivel superior en código Verilog

A continuación en la figura 2.13 se muestra el diagrama de bloque funcional y su interface con el bus Avalon.

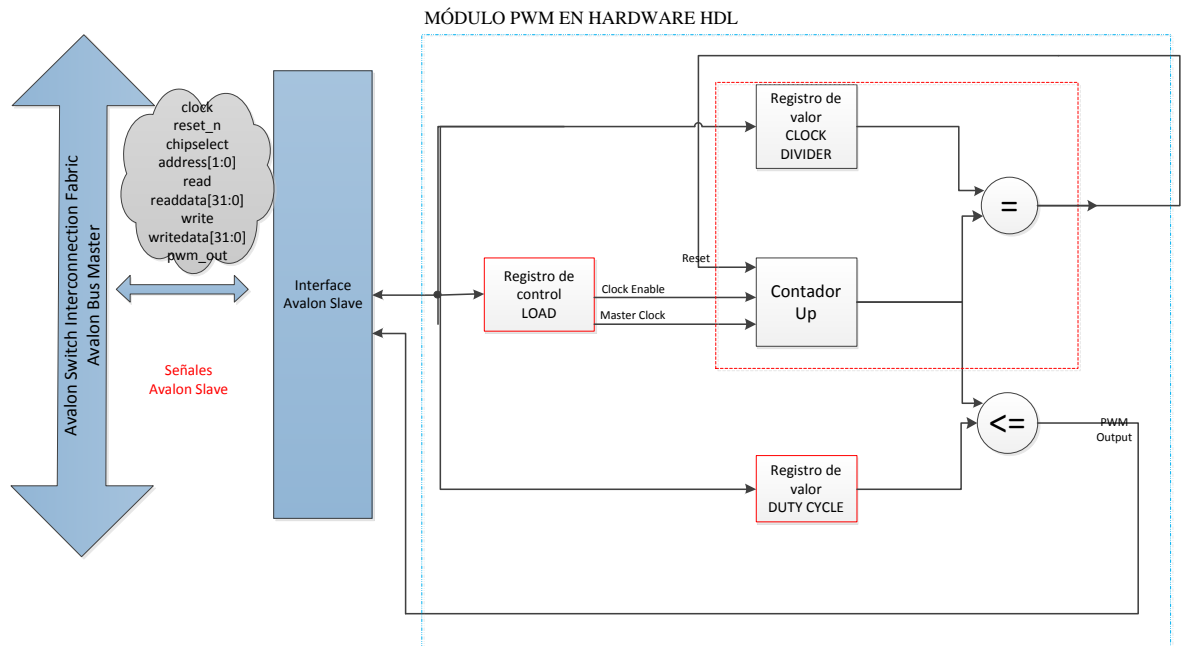


Figura 2.13 Diagrama de Funcionamiento PWM

```
//Inicio Código Principal
//Decodificación de la dirección
assign clock_divide_reg_selected = !address[1] & !address[0]; //Dirección offset 00
assign duty_cycle_reg_selected  = !address[1] & address[0]; //Dirección offset 01
assign enable_reg_selected      = address[1] & !address[0]; //Dirección offset 10

//Determinar si una transacción valida se inició
assign valid_write = chip_select & write;
assign valid_read  = chip_select & read;

//Determinar si una operación de escritura se produjo a una dirección específica
assign write_to_clock_divide = valid_write & clock_divide_reg_selected;
assign write_to_duty_cycle   = valid_write & duty_cycle_reg_selected;
assign write_to_enable       = valid_write & enable_reg_selected;

//Determinar si se produjo una lectura a una dirección específica
assign read_to_clock_divide = valid_read & clock_divide_reg_selected;
assign read_to_duty_cycle   = valid_read & duty_cycle_reg_selected;
assign read_to_enable       = valid_read & enable_reg_selected;
```

Figura 2.14 Decodificación de la dirección y señales de control.

La figura 2.14 muestra una porción del código HDL encargada de decodificar la dirección y las señales de control y habilitación enviadas por el procesador hacia este dispositivo. Esto se podrá apreciar más claramente en el capítulo 4.

Antes de agregar el módulo PWM como componente en el SOPC hardware system_0, es importante verificar su correcto funcionamiento. Por eso se le realizó una compilación como un proyecto aparte y se lo simuló en el Editor de Forma de Onda de Quartus II 9.1. Una parte de los resultados se puede apreciar en la figura 2.15.

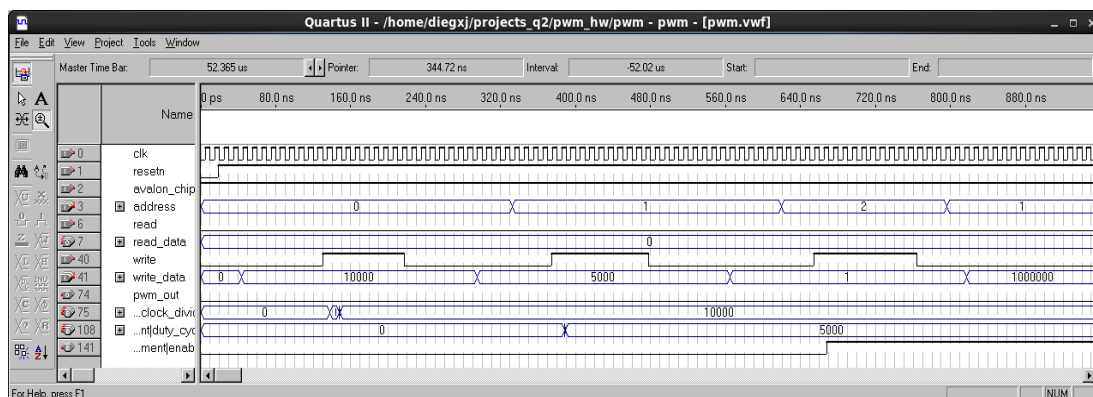


Figura 2.15 Simulación del módulo en el waveform del Quartus.

2.9.2 ENTORNO DE DESARROLLO PARA LA COMPILACIÓN DEL KERNEL DE μ CLINUX

2.9.2.1 MAQUINA HOST CON LINUX

La máquina host donde se realizará el desarrollo debe tener instalada alguna distribución de Linux. También puede ser una máquina virtual sobre sistema operativo Windows o similar.

El Altera-Wiki alienta a usar un IDE en Linux en lugar de Windows por las siguientes razones:

- El entorno de desarrollo en Windows no es bueno para el desarrollo del kernel y de aplicaciones μ Clinux. Los programas y librerías Cygwin y coLinux no garantizan compatibilidad nativa.
- El compilador por defecto del IDE en Window es newlib o glib (librerías para sistemas embebidos de RedHat), mientras nosotros preferimos usar μ Clibc en μ Clinux. Construir y portar software será mucho más fácil con el compilador cruzado creado desde Buildroot.
- Si se quiere desarrollar Nios II μ Clinux, se debería trabajar en Linux y aprender Linux. Se puede aprender mucho cuando se trabaja en Linux, y el "Linux Know-How" es la clave del éxito en Nios II μ Clinux.

2.9.2.2 TOOLCHAIN DEL CROSS-COMPILER DE NIOS II

Un compilador cruzado (más conocido como cross compiler) es un compilador capaz de crear código ejecutable para una plataforma diferente en la cual el compilador es ejecutado. Sus herramientas son usadas para crear ejecutables para sistemas embebidos o múltiples plataformas. Son usados para compilar para una plataforma en la cual no es factible hacer la compilación, como es el caso de nuestro target con μ Clinux.

Hay dos opciones para el compilador cruzado

- Usar uno pre-compilado.
- Compilar uno desde cero.

Se pueden encontrar en sitios web especializados sobre uClinux algunas versiones de compiladores cruzados pre-compilado. Se debe tener en cuenta que las opciones con las que fue compilado ya que de estas depende su efectividad como compilador en una aplicación dada. En este proyecto utilizamos el compilador cruzado pre-compilador provisto y recomendado por el Altera Wiki para proyectos de familiarización y aprendizaje.

Lo recomendable para aplicaciones específicas es compilar el compilador cruzado propio con la librería de C deseada. uClibc usa una herramienta llamada Buildroot para ello.

Buildroot es un conjunto de Makefiles, parches y herramientas que ayudan a construir el compilador cruzado, las bibliotecas, las aplicaciones del kernel de Linux y el sistema de ficheros raíz. Es decir, la construcción de casi todo, desde las fuentes a un sistema Linux.

Para ejecutar buildroot, debe disponer de un escritorio Linux con paquetes de desarrollo de software. Inicie una sesión como root o usando sudo para instalar estos paquetes.

En Fedora, Red Hat Enterprise Linux, CentOS

sudo yum install subversion make gcc ncurses-devel bison flex gawk (paquete que se desea actualizar).

En Debian / Ubuntu,

*sudo apt-get update sudo apt-get install subversion make gcc ncurses-dev bison flex
gawk gettext*

2.9.2.3 UCLIBC

El μ Clibc (también conocido como uClibc/pronunciado micro-see-lib-see) es una librería de C para el desarrollo de sistemas embebidos Linux. Es mucho más pequeña que la librería C de GNU, pero casi todas las aplicaciones soportadas por glibc también trabajan perfectamente en μ Clibc. Portar aplicaciones de glibc a μ Clibctípicamente involucra solamente recompilar el código fuente. μ Clibc incluso soporta librerías compartidas y threading. Actualmente corre en sistemas Linux estándar y sin-MMU (μ Clinux) que soportan alpha, amd64, ARM, Blackfin, cris, h8300, hppa, i386, i960, ia64, m68k, mips/mipsel, PowerPC, procesadores v850 y soft-core Nios II.

Para usar μ Clibc, necesitamos tener un toolchain. El toolchain consiste en GNU binutils, la colección del compilador GNU GCC y μ Clibc, todos construidos para producir parches de Nios II enlazado con μ Clibc. En el sitio web recomiendan usar Buildroot.

2.9.2.4 GNU BINUTILS

Las GNU Binutils son una colección de herramientas binarias. La mayoría de los programas que lo contienen usan BFD, la librería de Descripción de Archivo Binario, para hacer manipulaciones a bajo-nivel. Muchos de ellos también usan la

librería de *opcodes* (*código de operación*) para ensamblar y desensamblar las instrucciones de máquina.

Los principales programas son: ld, enlazador¹⁰ GNU y as, ensamblador GNU.

2.9.2.5 μCLINUX-DIST -MENÚ DE CONFIGURACIÓN KCONFIG

El mecanismo de Kconfig es hoy el mecanismo estándar de configuración y es usado por los proyectos open source líderes, como Kernel de Linux, Busybox y μClibc. El Kconfig tiene una sintaxis de configuración básica y permite añadir opciones de configuración de varios tipos, crear dependencias y escribir algunas líneas de descripción. . Las fuentes del kernel de Linux tienen un archivo Kconfig casi por cada directorio. Cada archivo Kconfig configura su propio nivel. Por ejemplo, se encontrarán un Kconfig en cada directorio de *driver* y de *arch*. Cada uno de ellos configura un driver o una arquitectura e incluye otros archivos Kconfig en los niveles debajo de este.

2.9.2.6 PROGRAMACIÓN DE UN MÓDULO DE KERNEL PARA EL DRIVER DEL MÓDULO PWM

Para este apartado estudiamos la documentación que adjuntamos en el anexo 7.

Realizamos el driver de la misma manera, considerando en este caso que son 3 los registros.

¹⁰ Linker

También le sumamos 0x80000000 a las direcciones de memoria de los registros ya que estamos usando el Nios II/f con cache de datos y sin MMU, y para poder acceder rápidamente a la memoria mapeada de los periféricos debemos saltarnos la cache lo que se logra haciendo el bit 31 igual 1.

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN

3.1 INSTALACIÓN DE μ CLINUX EN LA TARJETA DE2

Una vez ya instalado Linux, en una sesión de terminal ejecutamos el siguiente comando como root:

```
#yum install git-all git-gui make gcc ncurses-devel bison byacc flex gawk gettext  
ccache zlib-devel gtk2-devel lzo-devel paxutils libglade2-devel
```

```
#yum upgrade git-all git-gui make gcc ncurses-devel bison byacc flex gawk gettext  
ccache zlib-devel gtk2-devel lzo-devel pax-utils libglade2-devel
```

Este comando instalar o actualizará todo esos paquetes que son necesarios para la correcta compilación del kernel de uClinux.

3.1.1 CÓDIGO FUENTE DE μ CLINUX PARA ALTERA NIOS II

El código fuente de μ CLinux base o general se lo puede descargar desde la dirección www.uclinux.org/. Con estas fuentes se puede hacer el porting de uCLinux hacia cualquier microcontrolador o microprocesador o instalando en uno que ya se tenga soporte.

Es recomendable trabajar con las fuentes que se encuentran en <http://www.niosftp.com/pub/> ya que son las auspiciadas por Altera y la comunidad de usuarios del Altera Wiki <http://www.alterawiki.com>.

En el directorio */pub/uclinux* podemos descargar las fuentes para trabajar con la versión sin MMU del Nios II. En el */pub/linux* encontramos la última línea de desarrollo que incluye la versión MMU-LESS y la versión con MMU.

Actualmente podemos descargar del directorio */pub/uclinux* el archivo

<http://www.niosftp.com/pub/uclinux/nios2-linux-20090730.tar>

Dicho archivo contiene las fuentes de μ CLinux que trabajan con nuestro Nios II. El tamaño aproximado del archivo es 1.9 GB.

En el directorio */pub/gnutools* podemos descargar el archivo <http://www.niosftp.com/pub/gnutools/nios2gcc-20080203.tar.bz2>

Este archivo contiene el compilador cruzado de μ Clinux pre-compilado. Estas son las herramientas para compilar el kernel de μ Clinux en un computador anfitrión que tenga instalado una distribución de Linux para PC. El tamaño aproximado del archivo es 69 MB.

Una vez descargados, se procede a descomprimir los archivos. Las fuentes es recomendable descomprimirlas en el directorio raíz del usuario (\$HOME), en nuestro caso */home/diegxj*.

Y el ToolChain del compilador cruzado en la carpeta */home/diegxj/opt*.

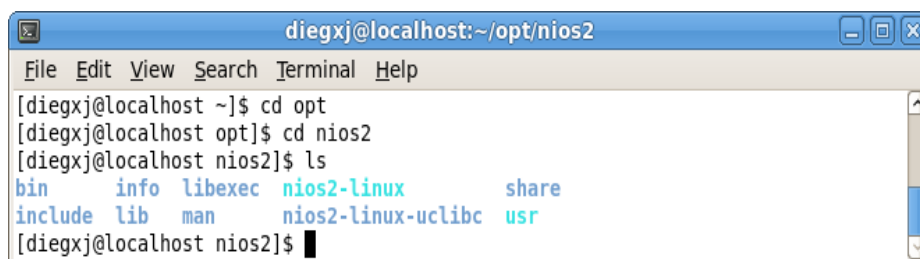
Para descomprimir los archivos podemos usar algún programa con entorno gráfico apuntando a la ruta indicada. También podemos ejecutar los siguientes comandos en una ventana de terminal:

```
$tar -C /home/diegxj -xvf nios2-linux-200900730.tar
```

```
$tar -C /home/diegxj/opt -xvf nios2gcc-20080203.tar
```

3.1.2 ESTRUCTURA DEL DIRECTORIO DEL TOOLCHAIN DEL COMPILADOR CRUZADO NIOS2GCC-20080203

El ToolChain del compilador cruzado se descomprime en el directorio *nios2* dentro de *home/diegxj/opt*.

A terminal window titled 'diegxj@localhost:~/opt/nios2' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
[diegxj@localhost ~]$ cd opt
[diegxj@localhost opt]$ cd nios2
[diegxj@localhost nios2]$ ls
bin      info  libexec  nios2-linux      share
include  lib   man      nios2-linux-uclibc  usr
[diegxj@localhost nios2]$
```

Figura 3.1 Contenido del directorio /opt/nios2

El subdirectorio *bin* es el que más nos interesa ya que en él se encuentran los archivos ejecutables del compilador.

De hecho antes de continuar debemos añadir ese directorio en la variable de entorno PATH.

Para ver el contenido actual de la variable PATH, en la terminal ejecutamos de manera específica el comando

```
$ echo $PATH
```

o de manera general

```
$ export
```

Con *export* podemos ver todas las variables de entorno con su valor respectivo.

Para modificar el valor de la variable de entorno PATH, debemos añadir una línea en el archivo `.bash_profile`. Este archivo lo encontramos en nuestro directorio `$HOME`.

Modificamos el archivo con el editor de texto de nuestra preferencia. Nosotros usaremos gedit.

Ejecutamos el siguiente comando en /\$HOME

```
$ gedit .bash_profile
```

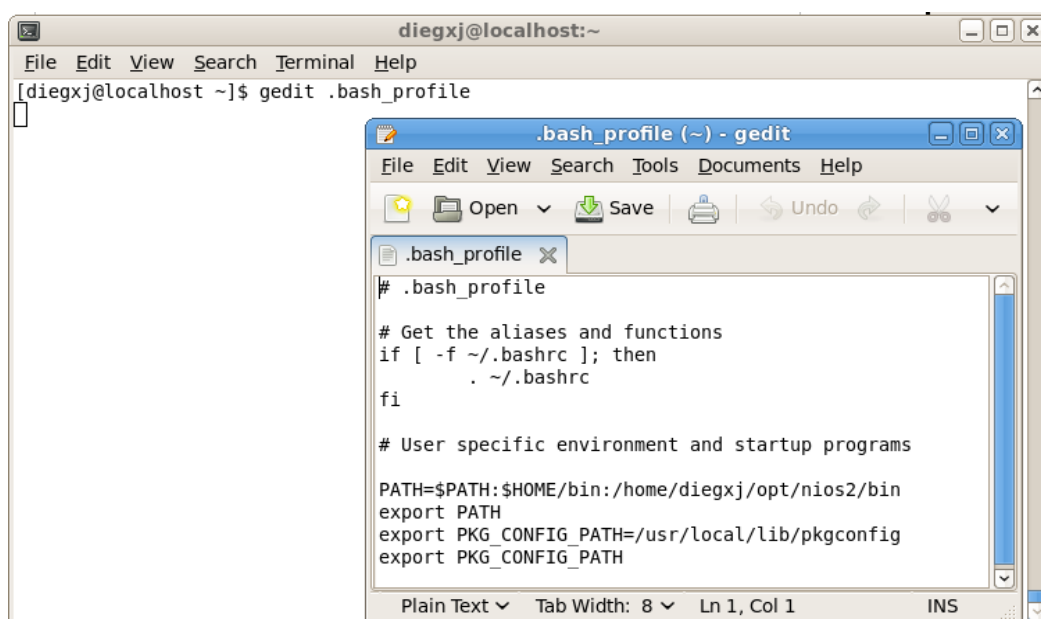


Figura 3.2 Modificación del archivo .bash_profile

La línea modificada debe verse algo así:

```
PATH=$PATH:$HOME/bin:/home/diegxj/opt/nios2/bin
```

Los dos puntos (:) se usan para separar las rutas que se agregan al path y el \$PATH sirve para no perder el path del sistema, ya que esto lo estamos añadiendo a nivel de usuario.

Terminada la modificación, guardamos los cambios y cerramos el archivo.

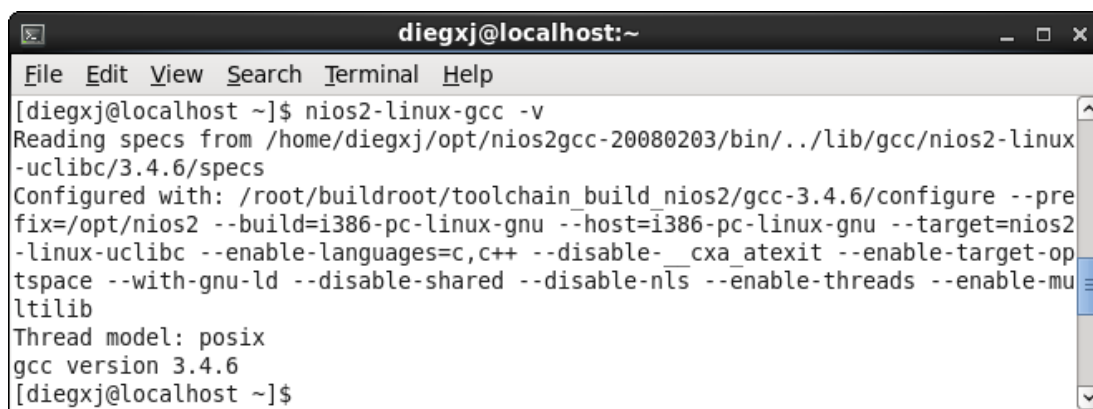
Para hacer efectivo el nuevo PATH, debemos ejecutar el siguiente comando:

```
$ source .bash_profile
```

Podemos comprobar el path ejecutando nuevamente *echo \$PATH*.

Para comprobar que el nuevo PATH está habilitado, ejecutamos el siguiente comando en cualquier directorio>

```
$ nios2-linux-gcc -v
```



```
diegxj@localhost:~
File Edit View Search Terminal Help
[diegxj@localhost ~]$ nios2-linux-gcc -v
Reading specs from /home/diegxj/opt/nios2gcc-20080203/bin/../lib/gcc/nios2-linux
-uclibc/3.4.6/specs
Configured with: /root/buildroot/toolchain_build_nios2/gcc-3.4.6/configure --pre
fix=/opt/nios2 --build=i386-pc-linux-gnu --host=i386-pc-linux-gnu --target=nios2
-linux-uclibc --enable-languages=c,c++ --disable-__cxa_atexit --enable-target-op
tspace --with-gnu-ld --disable-shared --disable-nls --enable-threads --enable-mu
ltilib
Thread model: posix
gcc version 3.4.6
[diegxj@localhost ~]$
```

Figura 3.3 Especificaciones del compilador cruzado.

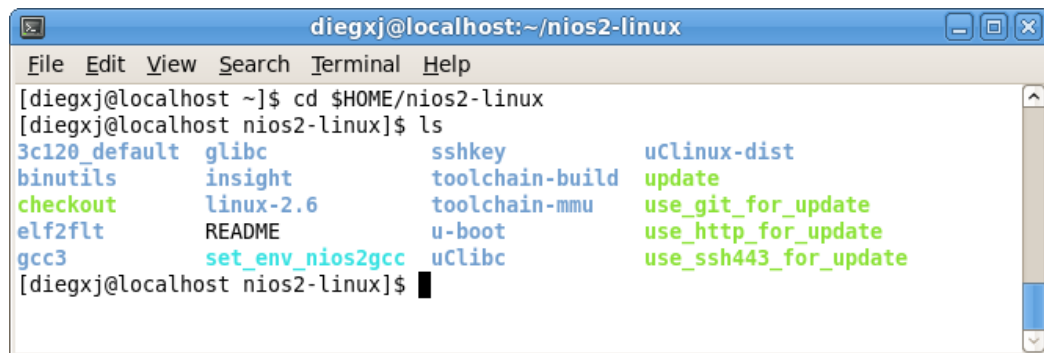
3.1.3 ESTRUCTURA DEL DIRECTORIO DE LAS FUENTES NIOS2-LINUX-20090730

Las fuentes se descomprimen en un directorio llamado *nios2-linux*.

Desde un terminal, ejecutamos los comandos

```
$ cd $HOME/nios2-linux
```

```
$ ls
```



```

diegxj@localhost:~/nios2-linux
File Edit View Search Terminal Help
[diegxj@localhost ~]$ cd $HOME/nios2-linux
[diegxj@localhost nios2-linux]$ ls
3c120_default  glibc          sshkey          uClinux-dist
binutils       insight        toolchain-build update
checkout      linux-2.6     toolchain-mmu  use_git_for_update
elf2flt       README        u-boot         use_http_for_update
gcc3          set_env_nios2gcc uClibc         use_ssh443_for_update
[diegxj@localhost nios2-linux]$

```

Figura 3.4 Contenido del directorio /nios2-linux

El contenido de los subdirectorios y archivos de la carpeta nios2-linux lo podemos clasificar de la siguiente manera:

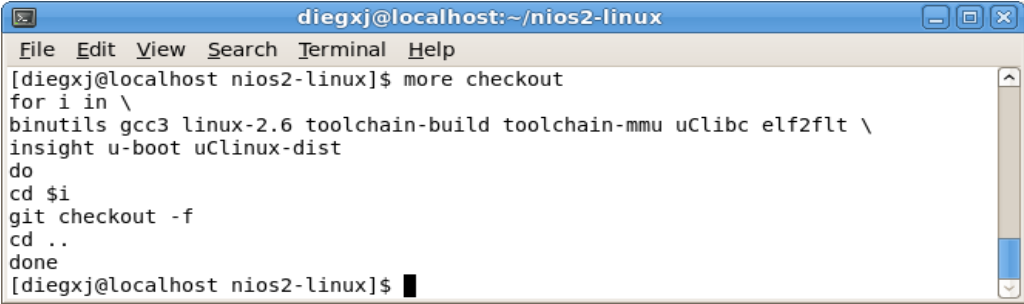
- Herramientas auxiliares para la compilación
- ToolChain del compilador cruzado de μ Clinux
- Librerías de μ Clibc
- Kernel de Linux (linux-2.6)
- Distribución de μ Clinux (μ Clinux-dist)

En esta etapa del desarrollo de μ Clinux/Linux para Nios II, los desarrolladores están utilizando la herramienta GIT para administrar y organizar las diferentes versiones que se van originando con el esfuerzo combinado de programadores aficionados y profesionales.

Si revisamos el directorio μ Clinux-dist notaremos que no hay ningún archivo a no ser el .git. Esto es porque no hemos “activado” ningún ramal de los que vienen pre-cargados en la distribución.

Para eso las fuentes vienen con un archivo llamado *checkout* que al ejecutarlo selecciona y habilita los ramales por defecto de cada carpeta de la distribución.

El contenido del script ejecutable *checkout* es el siguiente:



```

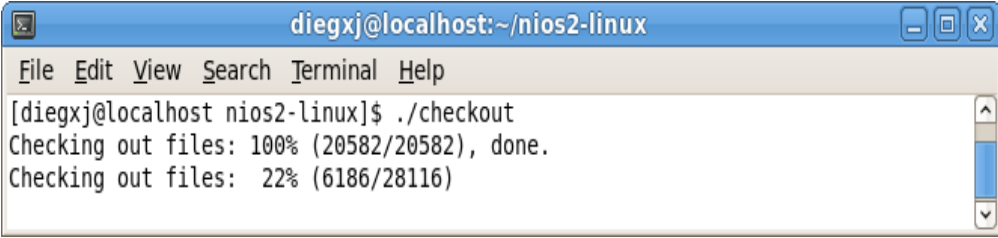
diegxxj@localhost:~/nios2-linux
File Edit View Search Terminal Help
[diegxxj@localhost nios2-linux]$ more checkout
for i in \
binutils gcc3 linux-2.6 toolchain-build toolchain-mmu uClibc elf2flt \
insight u-boot uclinux-dist
do
cd $i
git checkout -f
cd ..
done
[diegxxj@localhost nios2-linux]$ █

```

Figura 3.5 Script checkout

Como se puede apreciar el script realiza un `git checkout -f` en cada uno de los directorios de *nios2-linux*. La sintaxis para ejecutar el scripts es:

`./checkout`



```

diegxxj@localhost:~/nios2-linux
File Edit View Search Terminal Help
[diegxxj@localhost nios2-linux]$ ./checkout
Checking out files: 100% (20582/20582), done.
Checking out files: 22% (6186/28116)

```

Figura 3.6 Checkout de las fuentes de nios2-linux

La ejecución del script tomará un tiempo. Una vez terminada podemos empezar a trabajar con los directorios. A continuación se presenta en resumen toda la estructura:

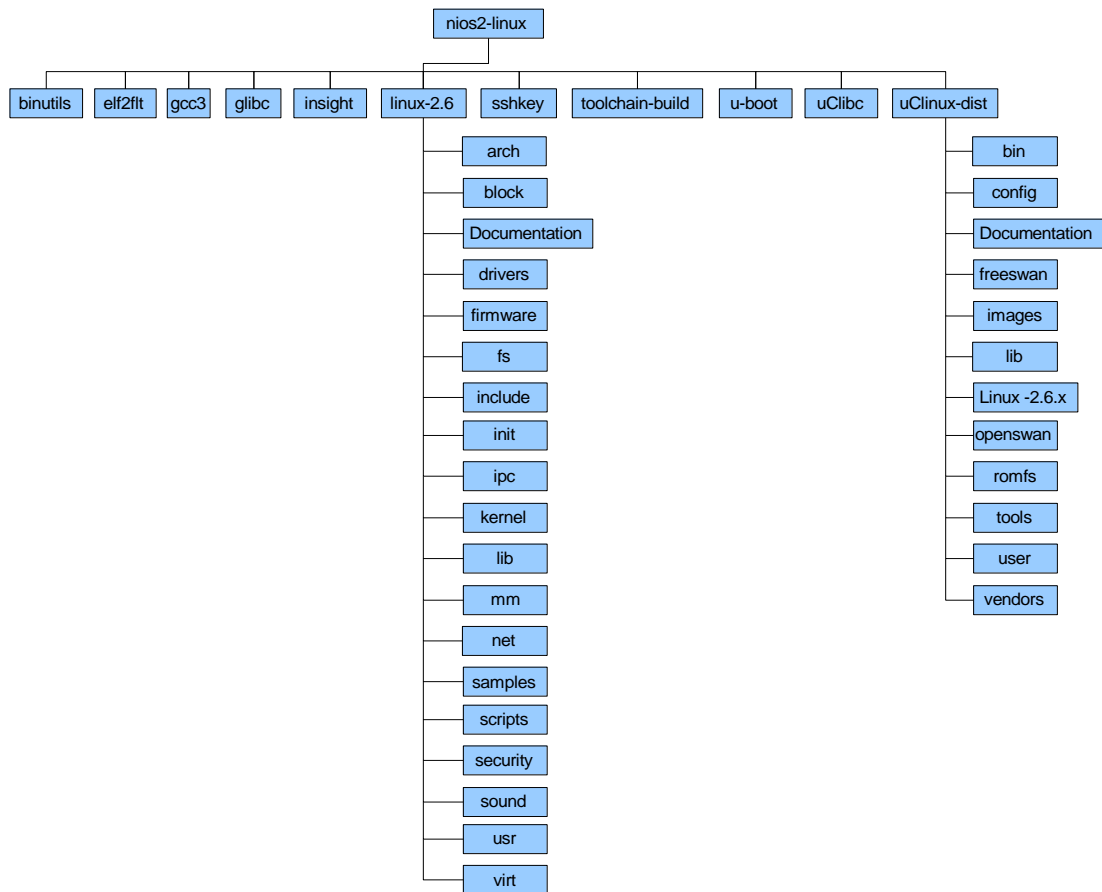
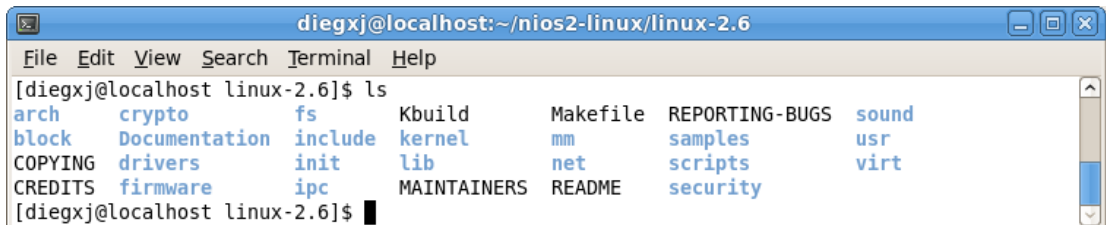


Figura 3.7 Estructura de árbol del directorio de Nios II – Linux.

3.1.4 SELECCIÓN DE LOS RAMALES DE DESARROLLO

Los directorios más importantes para nuestros fines son el de μ Clinux-dist y linux-2.6.

En el directorio *linux-2.6* están las fuentes del kernel de μ Clinux con los archivos de base para diferentes arquitecturas además del Nios II.



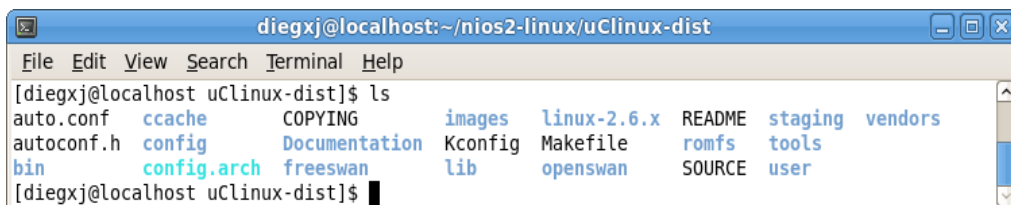
```

diegxxj@localhost:~/nios2-linux/linux-2.6
File Edit View Search Terminal Help
[diegxxj@localhost linux-2.6]$ ls
arch      crypto    fs        Kbuild    Makefile  REPORTING-BUGS  sound
block    Documentation  include  kernel    mm        samples         usr
COPYING  drivers   init      lib        net       scripts         virt
CREDITS  firmware  ipc      MAINTAINERS  README   security
[diegxxj@localhost linux-2.6]$

```

Figura 3.8 Contenido del directorio /nios2-linux/linux-2.6

En el directorio *μ Clinux-dist* están los archivos base para ejecutar la compilación del kernel así como los archivos de configuración de opciones para el kernel, aplicaciones de usuario, selección del “vendedor” (dispositivo desde donde el cual se corre el kernel compilado, en nuestro caso Altera Nios II).



```

diegxxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
[diegxxj@localhost uClinux-dist]$ ls
auto.conf  ccache    COPYING    images  linux-2.6.x  README  staging  vendors
autoconf.h config    Documentation  Kconfig  Makefile    romfs   tools
bin        config.arch  freeswan    lib      openswan    SOURCE  user
[diegxxj@localhost uClinux-dist]$

```

Figura 3.9 Contenido del directorio /nios2-linux/ μ Clinux-dist

En cada uno de estos directorios debemos ejecutar los siguientes comandos:

```
$ git branch -a
```

```
* test-nios2
```



```
trunk
```

```
remotes/origin/HEAD -> origin/test-nios2
```

```
remotes/origin/test-nios2
```

```
remotes/origin/trunk
```

```
remotes/origin/unstable
```

```
$ git checkout test-nios2
```

Con el comando *git branch -a* listamos los diferentes ramales de desarrollo, cada uno tiene una forma particular de ejecutarse y características especiales. Con el comando *git checkout test-nios2* seleccionamos ese ramal (branch).

Hacemos lo mismo en el directorio de linux-2.6. Seleccionamos también el ramal `test-nios2`.

Nota: Para cambiar de ramal, los cambios realizados en el ramal actual deben haberse guardados en un nuevo ramal (en terminología git es commit) o bien deben limpiarse.

Para limpiar los cambios realizados en el ramal se usa el siguiente comando:

```
git clean -f -x -d .
```

Una vez finalizados estos pasos podemos empezar con la compilación del kernel.

3.1.5 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE UCLINUX CON LOS PARÁMETROS POR DEFECTO

Nos ubicamos en el directorio de `/nios2-linux/μClinux-dist`. Desde ahí ejecutamos el comando:

```
$ make menuconfig
```

Con un tamaño de ventana de terminal adecuado, debe aparecer el siguiente recuadro:

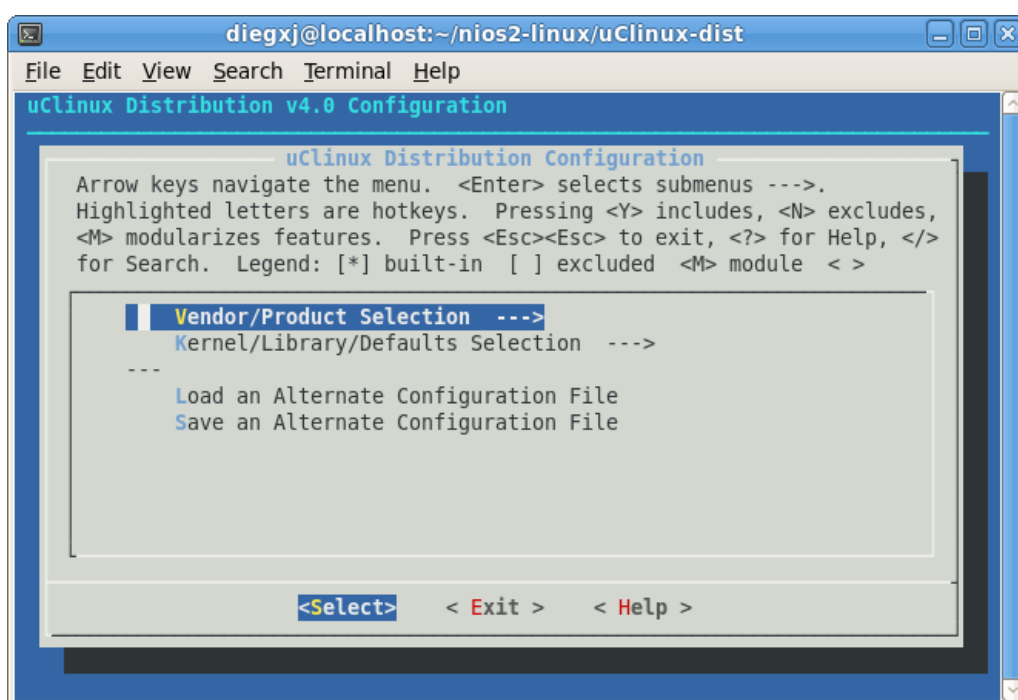


Figura 3.10 Pantalla principal del Menú de configuración

Seguimos las instrucciones de manejo del menú de configuración de la distribución de `μClinux`.

- Entramos a la sección de *Vendor/Product Selection* --->
- Ahí seleccionamos como fabricante a *Altera* y como producto a *nios2*
- Salimos con *Exit* al menú principal.
- Vamos ahora a la sección *Kernel/Library/Defaults Selection* --->
- Aquí seleccionamos en *Libc Version (None)* y marcamos (con la barra espaciadora) la opción *[*] Default all settings (lose changes)*

```

-- Select the Vendor you wish to target
Vendor (Altera) --->
--- Select the Product you wish to target
Altera Products (nios2) --->

```

Figura 3.11 Pantalla de selección del Vendedor y el producto

```

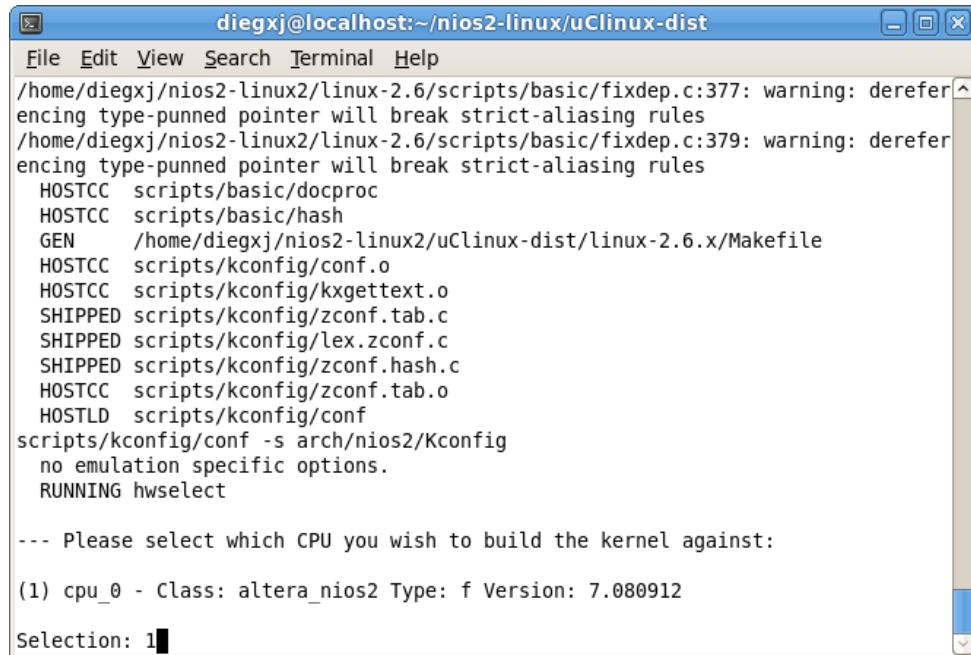
-- Kernel is linux-2.6.x
Libc Version (None) --->
[*] Default all settings (lose changes)
[ ] Customize Kernel Settings (NEW)
[ ] Customize Application/Library Settings (NEW)
[ ] Update Default Vendor Settings (NEW)

```

Figura 3.12 Pantalla de selección del Kernel/Library/Defaults

- Salimos de vuelta al menú principal con <Exit> y nuevamente presionamos <Exit> para salir de menú de configuración, respondemos <Yes> a la pregunta sobre si queremos grabar los cambios.
- Nuevamente en la línea de comandos introducimos el comando:
- `$make vendor_hwselect SYSPTF=system_0.ptf`

- Durante este paso el kernel de μ Clinux reconoce el hardware con el que interactuará. El archivo **system_0.ptf** es generado por el SOPC Builder.



```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
/home/diegxj/nios2-linux2/linux-2.6/scripts/basic/fixdep.c:377: warning: derefer
encing type-punned pointer will break strict-aliasing rules
/home/diegxj/nios2-linux2/linux-2.6/scripts/basic/fixdep.c:379: warning: derefer
encing type-punned pointer will break strict-aliasing rules
HOSTCC scripts/basic/docproc
HOSTCC scripts/basic/hash
GEN /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/Makefile
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/kxgettext.o
SHIPPED scripts/kconfig/zconf.tab.c
SHIPPED scripts/kconfig/lex.zconf.c
SHIPPED scripts/kconfig/zconf.hash.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTLD scripts/kconfig/conf
scripts/kconfig/conf -s arch/nios2/Kconfig
no emulation specific options.
RUNNING hwselect

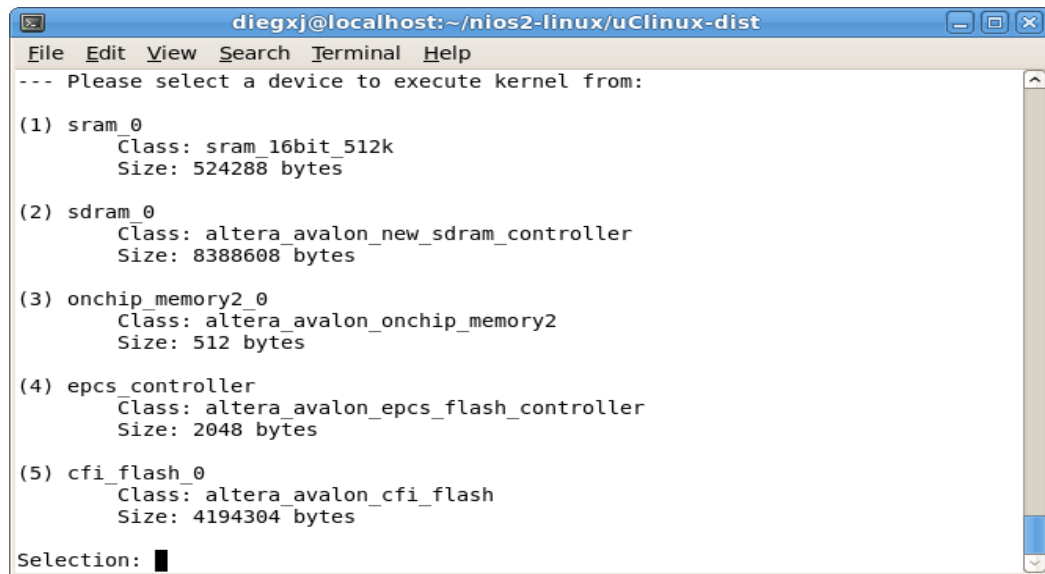
--- Please select which CPU you wish to build the kernel against:

(1) cpu_0 - Class: altera_nios2 Type: f Version: 7.080912
Selection: 1

```

Figura 3.13 HWSELECT CPU

El script `make hwselect` escribirá la configuración y cargará archivos de acuerdo a lo que seleccionemos para CPU y para MEMORIA.



```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
--- Please select a device to execute kernel from:

(1) sram_0
    Class: sram_16bit_512k
    Size: 524288 bytes

(2) sdram_0
    Class: altera_avalon_new_sdram_controller
    Size: 8388608 bytes

(3) onchip_memory2_0
    Class: altera_avalon_onchip_memory2
    Size: 512 bytes

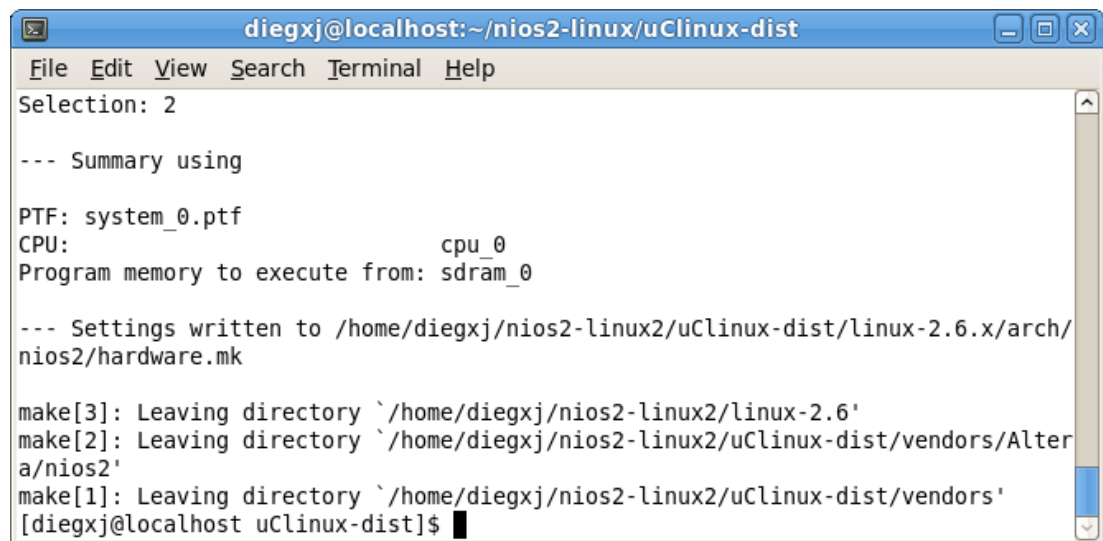
(4) epcs_controller
    Class: altera_avalon_epcs_flash_controller
    Size: 2048 bytes

(5) cfi_flash_0
    Class: altera_avalon_cfi_flash
    Size: 4194304 bytes

Selection: █

```

Figura 3.14 HWSELECT Memoria ejecución del Kernel



```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
Selection: 2

--- Summary using

PTF: system_0.ptf
CPU:                               cpu_0
Program memory to execute from: sdram_0

--- Settings written to /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/arch/
nios2/hardware.mk

make[3]: Leaving directory `/home/diegxj/nios2-linux2/linux-2.6'
make[2]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors/Alter
a/nios2'
make[1]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors'
[diegxj@localhost uClinux-dist]$ █

```

Figura 3.15 HWSELECT finalizado

Ya en este punto estamos listos para compilar el kernel.

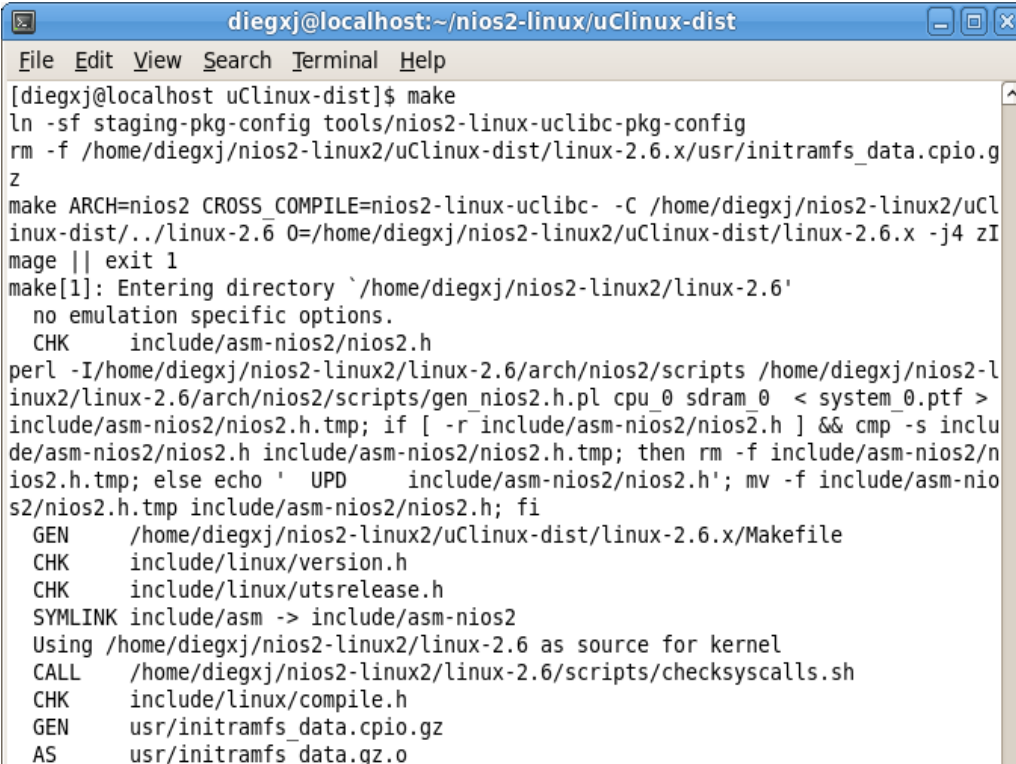
Para proceder con la compilación ejecutamos el comando make.

```
$ make
```

El comando `make` ejecuta todo lo que se encuentra en el archivo script **Makefile** del directorio base `/nios2-linux/μClinux-dist` y las llamadas a los **Makefile** de cada subdirectorio.

El **Makefile** puede ser invocado completo o por etapas dependiendo de la necesidad.

Las diferentes etapas que conforman el **Makefile** pueden ser revisadas en los archivos. Las etapas son conocidas como **targets**. Se recomienda un editor de textos como `gedit` para visualizar el archivo resaltando la sintaxis ya que es un estándar de Linux.

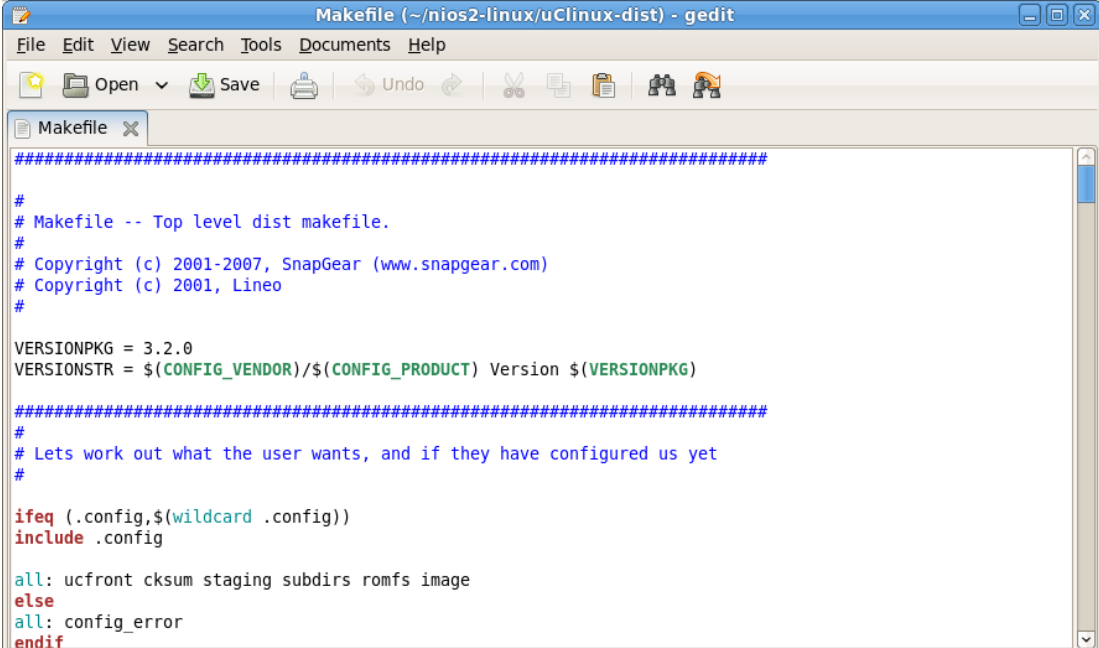


```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
[diegxj@localhost uClinux-dist]$ make
ln -sf staging-pkg-config tools/nios2-linux-uclibc-pkg-config
rm -f /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/usr/initramfs_data.cpio.g
z
make ARCH=nios2 CROSS_COMPILE=nios2-linux-uclibc- -C /home/diegxj/nios2-linux2/uCl
inux-dist/./linux-2.6 0=/home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x -j4 zI
mage || exit 1
make[1]: Entering directory `/home/diegxj/nios2-linux2/linux-2.6'
  no emulation specific options.
  CHK    include/asm-nios2/nios2.h
perl -I/home/diegxj/nios2-linux2/linux-2.6/arch/nios2/scripts /home/diegxj/nios2-l
inux2/linux-2.6/arch/nios2/scripts/gen_nios2.h.pl cpu_0 sdram_0 < system_0.ptf >
include/asm-nios2/nios2.h.tmp; if [ -r include/asm-nios2/nios2.h ] && cmp -s inclu
de/asm-nios2/nios2.h include/asm-nios2/nios2.h.tmp; then rm -f include/asm-nios2/n
ios2.h.tmp; else echo '  UPD    include/asm-nios2/nios2.h'; mv -f include/asm-nio
s2/nios2.h.tmp include/asm-nios2/nios2.h; fi
  GEN    /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/Makefile
  CHK    include/linux/version.h
  CHK    include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-nios2
Using /home/diegxj/nios2-linux2/linux-2.6 as source for kernel
  CALL   /home/diegxj/nios2-linux2/linux-2.6/scripts/checksyscalls.sh
  CHK    include/linux/compile.h
  GEN    usr/initramfs_data.cpio.gz
  AS     usr/initramfs_data.gz.o

```

Figura 3.16 Primeros mensajes durante la compilación con `make`



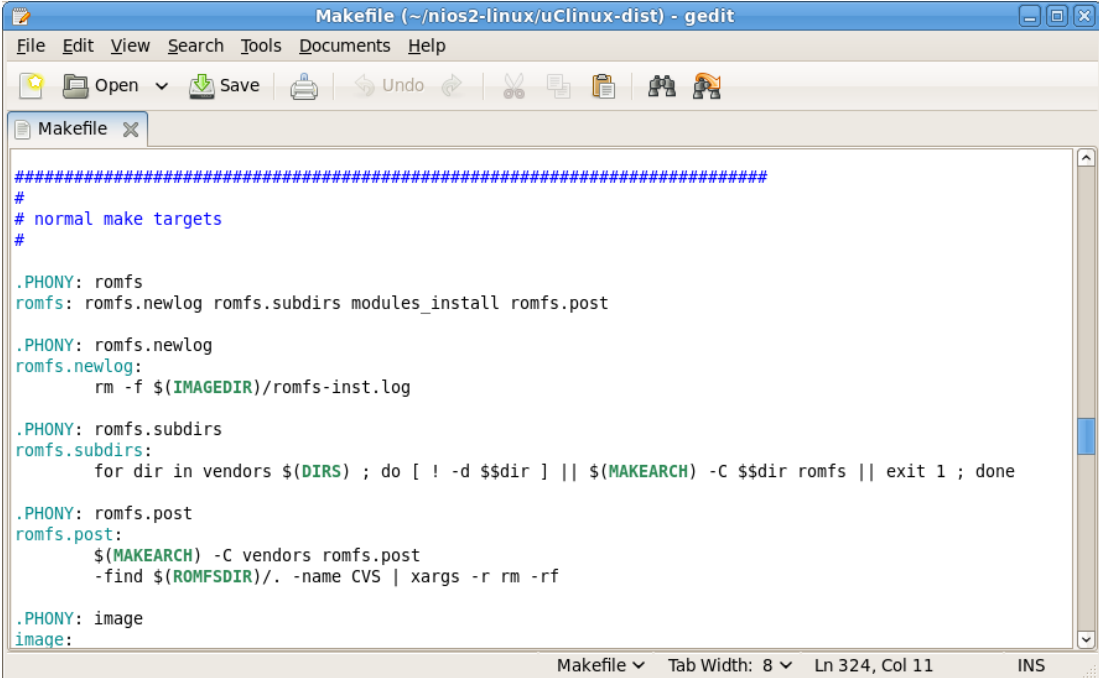
```

#####
#
# Makefile -- Top level dist makefile.
#
# Copyright (c) 2001-2007, SnapGear (www.snapgear.com)
# Copyright (c) 2001, Lineo
#
VERSIONPKG = 3.2.0
VERSIONSTR = $(CONFIG_VENDOR)/$(CONFIG_PRODUCT) Version $(VERSIONPKG)
#####
#
# Lets work out what the user wants, and if they have configured us yet
#
ifeq (.config,$(wildcard .config))
include .config

all: ucfront cksum staging subdirs romfs image
else
all: config_error
endif

```

Figura 3.17 Código fuente del script del Makefile – Sección inicial



```

#####
#
# normal make targets
#
.PHONY: romfs
romfs: romfs.newlog romfs.subdirs modules_install romfs.post

.PHONY: romfs.newlog
romfs.newlog:
    rm -f $(IMAGEDIR)/romfs-inst.log

.PHONY: romfs.subdirs
romfs.subdirs:
    for dir in vendors $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH) -C $$dir romfs || exit 1 ; done

.PHONY: romfs.post
romfs.post:
    $(MAKEARCH) -C vendors romfs.post
    -find $(ROMFSDIR)/. -name CVS | xargs -r rm -rf

.PHONY: image
image:

```

Figura 3.18 Algunos Targets que conforman el Makefile

Los targets más usados son:

➤ **make:**

Ejecuta el script completo. Termina con la generación de la imagen del μ Clinux.

➤ **make romfs:**

Crea el contenido de la carpeta romfs (la que contiene el sistema de archivos del μ Clinux) y termina generando la imagen comprimida de μ Clinux.

➤ **make linux image:**

Genera la imagen comprimida de μ Clinux. Una copia de la imagen se guarda en la carpeta nios2-linux/ μ Clinux-dist/images

➤ **make clean:**

Elimina todo rastro de una compilación anterior. Borra todo los archivos objetos y ejecutables creados.

Una vez terminada la ejecución y solucionados todos los errores que se puedan presentar, tendremos en nuestro terminal algo similar a esto.


```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
CALL /home/diegxj/nios2-linux2/linux-2.6/scripts/checksyscalls.sh
CHK include/linux/compile.h
GEN usr/initramfs_data.cpio.gz
AS usr/initramfs_data.gz.o
LD usr/built-in.o
LD vmlinux.o
MODPOST vmlinux.o
WARNING: modpost: Found 2 section mismatch(es).
To see full details build your kernel with:
'make CONFIG_DEBUG_SECTION_MISMATCH=y'
GEN .version
CHK include/linux/compile.h
UPD include/linux/compile.h
CC init/version.o
LD init/built-in.o
LD vmlinux
SYSMAP System.map
OBJCOPY arch/nios2/boot/vmlinux.bin
GZIP arch/nios2/boot/vmlinux.gz
LD arch/nios2/boot/compressed/piggy.o
LD arch/nios2/boot/compressed/vmlinux
OBJCOPY arch/nios2/boot/zImage
Kernel: arch/nios2/boot/zImage is ready
make[3]: Leaving directory `/home/diegxj/nios2-linux2/linux-2.6'
cp /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/arch/nios2/boot/zImage /home
/diegxj/nios2-linux2/uClinux-dist/images/zImage
make[2]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors/Altera/
nios2'
make[1]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors'
[diegxj@localhost uClinux-dist]$ █

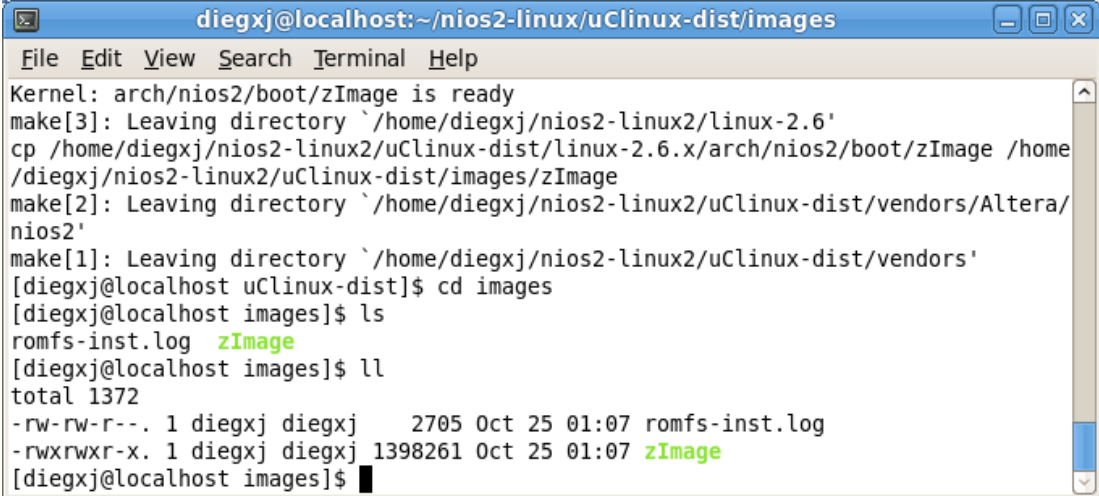
```

Figura 3.19 Final de la ejecución del make.

Podemos notar de la figura lo siguiente:

Se genera una imagen comprimida del Kernel (Kernel: arch/nios2/boot/zImage is ready).

La zImage es copiada al directorio ../nios2-linux/μClinux-dist/images/zImage.



```

diegxj@localhost:~/nios2-linux/uClinux-dist/images
File Edit View Search Terminal Help
Kernel: arch/nios2/boot/zImage is ready
make[3]: Leaving directory `/home/diegxj/nios2-linux2/linux-2.6'
cp /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/arch/nios2/boot/zImage /home
/diegxj/nios2-linux2/uClinux-dist/images/zImage
make[2]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors/Altera/
nios2'
make[1]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors'
[diegxj@localhost uClinux-dist]$ cd images
[diegxj@localhost images]$ ls
romfs-inst.log  zImage
[diegxj@localhost images]$ ll
total 1372
-rw-rw-r--. 1 diegxj diegxj   2705 Oct 25 01:07 romfs-inst.log
-rwxrwxr-x. 1 diegxj diegxj 1398261 Oct 25 01:07 zImage
[diegxj@localhost images]$ █

```

Figura 3.20 Imagen comprimida del Kernel de μ Clinux

Así hemos terminado la compilación del kernel de μ Clinux para el procesador Nios II con los parámetros por defecto, o sea lo básico para funcionar.

Este paso es necesario para tener una base para el control de errores y verificar que el código fuente no está corrupto o desactualizado.

3.1.6 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE μ CLINUX CON PARÁMETROS PERSONALIZADOS

Si necesitamos o deseamos añadir dispositivos adicionales debemos tener en cuenta lo siguiente:

- La configuración del sistema en el SOPC Builder incluye los periféricos deseados y la generación del sistema se dio sin errores.

- Tener los controladores para los dispositivos que se desean usar con μ Clinux (opcional con μ Clinux-nommu).
- Tener las aplicaciones a nivel de usuario para controlar los dispositivos.

Las fuentes que estamos usando vienen ya con algunos controladores pre-cargados que pueden ser usados con los dispositivos de la tarjeta DE2.

Para proceder a modificar la configuración del kernel hay que volver a ingresar al menú de configuración del Kernel.

Nuevamente ejecutamos el comando `make menuconfig` en la terminal.

Vamos a la sección `Kernel/Library/Defaults Selection` --->

Ahí seleccionamos:

```
--- Kernel is linux-2.6.x
    Libc Version (None) --->
[ ] Default all settings (lose changes)
[*] Customize Kernel Settings
[*] Customize Application/Library Settings
[ ] Update Default Vendor Settings
```

Figura 3.21 Opciones seleccionadas para cambiar configuración de Kernel

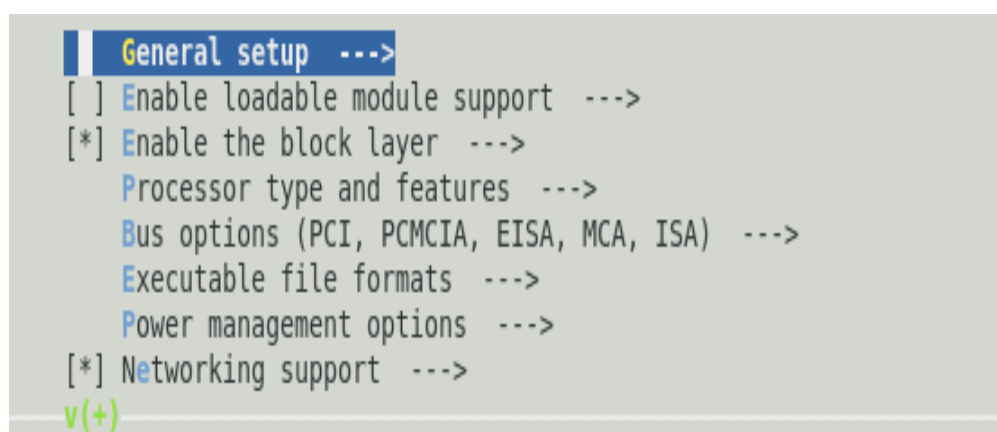
Salimos del menú de configuración del Kernel con doble <Exit> y <Yes>.

Inmediatamente se muestra la ventana de configuración del kernel de Linux (Linux Kernel Configuration).

De una rápida inspección a la ventana destacamos que en la esquina superior izquierda nos indica la versión del kernel (en este caso la 2.6.30) y también se muestran instrucciones para el uso de la ventana de configuración.

Podemos usar las teclas <Y>, <N>, <M>, <Esc>, <?>, </> como indican las instrucciones, o podemos usar la tecla <Space Bar> para seleccionar o deseleccionar las opciones.

Las opciones de configuración del kernel están distribuidas en grupos. En el menú principal tenemos los siguientes grupos de opciones:



```

General setup --->
[ ] Enable loadable module support --->
[*] Enable the block layer --->
    Processor type and features --->
    Bus options (PCI, PCMCIA, EISA, MCA, ISA) --->
    Executable file formats --->
    Power management options --->
[*] Networking support --->
v(+)
```

Figura 3.22 Ventana principal de Configuración del Kernel de Linux

General setup --->

Enable loadable module support --->

Enable the block layer --->

Processor type and features --->

Bus options (PCI, PCMCIA, EISA, MCA, ISA) --->

Executable file formats --->

Power management options --->

Networking support --->

Device Drivers --->

File systems --->

Kernel hacking --->

Security options --->

Cryptographic API --->

Library routines --->

Revisaremos los grupos de opciones donde se realizaron cambios a la configuración por defecto y anotaremos las opciones que deben estar marcadas.

General setup --->

[*] Prompt for development and/or incomplete code/drivers

(14) Kernel log buffer size (16 => 64KB, 17 => 128KB)

[*] Group CPU scheduler

[*] Group scheduling for SCHED_OTHER

- [*] Create deprecated sysfs layout for older userspace tools

- [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support

(../romfs ../vendors/Altera/nios2/romfs_list) Initramfs source file(s)

- (*500) User ID to map to 0 (user root) (* debe ser el mismo del usuario host)

- (500) Group ID to map to 0 (group root)

- [*] Support initial ramdisks compressed using gzip

- [*] Configure standard kernel features (for small systems) --->

- [*] Enable support for printk

- [*] BUG() support

- [*] Enable full-sized data structures for core

- [*] Enable futex support

- [*] Enable signalfd() system call

- [*] Enable timerfd() system call

- [*] Enable eventfd() system call

- [*] Enable AIO support

Disable heap randomization

Choose SLAB allocator (SLAB) --->

SLAB

Enable the block layer --->

IO Schedulers --->

Deadline I/O scheduler

Default I/O scheduler (Deadline) --->

Deadline

Processor type and features --->

Platform (Altera DE2 Development board support) --->

Nios II Hardware Multiply Support (Enable mul instruction) --->

*** Platform drivers Options ***

GPIO interface

*** Miscellaneous Options ***

Include breakpoint trap on kernel startup

Kernel executes from (RAM) --->

Preemption Model (Preemptible Kernel (Low-Latency Desktop))

Timer frequency (100 HZ) --->

[*] Passed kernel command line from u-boot

Memory model (Flat Memory) --->

[*] Add LRU list to track non-evictable pages

(1) Turn on mmap() excess space trimming before booting

(0x00500000) Link address offset for booting

```

*** Platform dependant setup ***
CPU (NIO2) --->
[ ] MMU support
Platform (Altera DE2 Development board support) --->
Nios II Hardware Multiply Support (Enable mul instruction)
*** Platform drivers Options ***
[*] GPIO interface
[ ] Support of DMA controller with Avalon interface
[ ] Altera PCI host bridge
[ ] Remote update support
v(+)

```

Figura 3.23 Ventana de tipo e procesador y características

Executable file formats --->

[*] Kernel support for flat binaries

[*] Enable ZFLAT support

[*] Networking support --->

Networking options --->

[*] Unix domain sockets

[*] TCP/IP networking

Device Drivers --->

Generic Driver Options --->

[*] Select only drivers that don't need compile-time external firmware

[*] Prevent firmware from being built

[*] Enable loadable module support --->

--- Enable loadable module support

[*] Module unloading

[*] Memory Technology Device (MTD) support --->(*Soporte para Flash como disco Duro*)

--- Memory Technology Device (MTD) support

[*] MTD partitioning support

[*] Command line partition table parsing

*** User Modules And Translation Layers ***

[*] Direct char device access to MTD devices

-*- Common interface to block layer for MTD 'translation layers

[*] Caching block device access to MTD devices

RAM/ROM/Flash chip drivers --->

[*] Detect flash chips by Common Flash Interface (CFI) probe

[*] Support for AMD/Fujitsu/Spansion flash chips

Mapping drivers for chip access --->

[*] Flash device in physical memory map

Self-contained MTD device drivers --->

[*] Support most SPI Flash chips (AT26DF, M25P, W25X, ...)

[*] Use FAST_READ OPCode allowing SPI CLK <= 50MHz

[*] Block devices --->

SCSI device support --->

[*] SCSI device support

[*] legacy /proc/scsi/ support

*** SCSI support type (disk, tape, CD-ROM) ***

[*] SCSI disk support

[*] SCSI low-level drivers --->

[*] Network device support --->

[*] Enable older network device API compatibility

[*] Ethernet (10 or 100Mbit) --->

-*- Generic Media Independent Interface device support

[*] DM9000 support

4. DM9000 maximum debug level

Input device support --->

-*- Generic input layer (needed for keyboard, mouse, ...)

[*] Mouse interface

[*] Provide legacy /dev/psaux device

(1024) Horizontal screen resolution

(768) Vertical screen resolution

[*] Event interface

*** Input Device Drivers ***

[*] Keyboards --->

--- Keyboards

[*] AT keyboard

[*] PS/2 mouse

[*] ALPS PS/2 mouse protocol extension

[*] Logitech PS/2++ mouse protocol extension

[*] Synaptics PS/2 mouse protocol extension

[*] IBM Trackpoint PS/2 mouse protocol extension

Hardware I/O ports --->

-*- Serial I/O support

[*] Altera UP PS2 controller

-*- PS/2 driver library

Character devices --->

[*] Virtual terminal

Serial drivers --->

[*] Altera JTAG UART support

[*] Altera JTAG UART console support

[*] Altera UART support

[*] Altera UART console support

(4) Maximum number of Altera UART ports

(115200) Default baudrate for Altera UART ports

[*] Unix98 PTY support

[*] Legacy (BSD) PTY support

10. Maximum number of legacy PTY in use

[*] **SPI support --->**

*** SPI Master Controller Drivers ***

[*] Altera SPI Controller

-*- Utilities for Bitbanging SPI masters

Graphics support --->

[*] Support for frame buffer devices --->

[*] Altera framebuffer support

[*] **HID Devices --->**

-*- Generic HID support

[*] HID debugging support

*** USB Input Devices ***

[*] USB Human Interface Device (full HID) support

[*] **USB support --->**

[*] Support for Host-side USB

*** Miscellaneous USB options ***

[*] USB device filesystem

*** USB Host Controller Drivers ***

[*] ISP1362 HCD support

[*] USB Mass Storage support

[*] MMC/SD/SDIO card support --->

*** MMC/SD/SDIO Card Drivers ***

[*] MMC block device driver

[*] Use bounce buffer for simple hosts

*** MMC/SD/SDIO Host Controller Drivers ***

[*] MMC/SD/SDIO over SPI

[*] LED Support --->

[*] LED Class Support

*** LED drivers ***

[*] LED Support for GPIO connected LEDs

[*] Platform device bindings for GPIO LEDs

*** LED Triggers ***

[*] LED Trigger support

[*] LED Heartbeat Trigger

File systems --->

[*] Enable POSIX file locking API

DOS/FAT/NT Filesystems --->

[*] VFAT (Windows-95) fs support

(437) Default codepage for FAT

(iso8859-1) Default iocharset for FAT

Pseudo filesystems --->

[*] /proc file system support

[*] sysfs file system support

[*] Miscellaneous filesystems --->

[*] Journalling Flash File System v2 (JFFS2) support

(0) JFFS2 debugging verbosity (0 = quiet, 2 = noisy)

[*] JFFS2 write-buffering support

[*] Network File Systems --->

[*] NFS client support

[*] NFS client support for NFS version 3

-*- Native language support --->

--- Native language support

(iso8859-1) Default NLS Option

[*] Codepage 437 (United States, Canada)

[*] NLS ISO 8859-1 (Latin 1; Western European Languages)

Kernel hacking --->

[*] Enable __deprecated logic

(1024) Warn for stack frames larger than (needs gcc 4.4)

[*] Kernel debugging

[*] Detect Soft Lockups

[*] Detect Hung Tasks

[*] Collect scheduler debugging info

[*] Compile the kernel with debug info

Library routines --->

.*- CRC ITU-T V.41 functions

.*- CRC32 functions

.*- CRC7 functions

Al terminar, salimos con <Exit>, escogemos <Yes> en el cuadro que nos solicita guardar los cambios.

La nueva configuración es guarda en un archivo para uso del script Makefile.

Inmediatamente después de salir de la ventana de configuración del kernel, se abre la ventana de configuración de usuario o de librerías/aplicaciones/Misceláneos (siempre que se haya marcado la opción respectiva en el menú de configuración principal).

El menú principal de esta sección tiene los siguientes grupos de opciones:

Library Configuration --->

Core Applications --->

Flash Tools --->

Filesystem Applications --->

Network Applications --->

Miscellaneous Applications --->

BusyBox --->

Tinylogin --->

MicroWindows --->

Games --->

Miscellaneous Configuration --->

Debug Builds --->

Estas son las opciones que deben estar marcadas en el grupo respectivo.

Core Applications --->

[*] init

[*] enable console shell

Shell Program (sash) --->

(X) sash

[*] simple (sash) history

[*] sash ps

Network Applications --->

[*] boa

[*] dhcpd-new (2.0/2.4)

[*] ftp

[*] ftpd

inetd

telnetd

BusyBox --->

BusyBox

--- BusyBox Configuration

Busybox Settings --->

General Configuration --->

Buffer allocation policy (Allocate with Malloc) --->

Allocate with Malloc

Show terse applet usage messages

Store applet usage messages in compressed form

Support for --long-options

Use the devpts filesystem for Unix98 PTYs

--- Support for SUID/SGID handling

(/proc/self/exe) Path to BusyBox executable

Build Options --->

Build BusyBox as a static binary (no shared libs)

Build with Large File Support (for accessing files >
2 GB)

Debugging Options --->

Additional debugging library (None) --->

None

Enable obsolete features removed before SUSv3?

Installation Options --->

Applets links (as soft-links) --->

as soft-links

(./_install) BusyBox installation prefix

Busybox Library Tuning --->

(6) Minimum password length

(2) MD5: Trade Bytes for Speed

(4) Copy buffer size, in kilobytes

Use clock_gettime(CLOCK_MONOTONIC) syscall

Use ioctl names rather than hex values in error messages

--- Applets

Coreutils --->

cp

cut

date

dd

Enable DD signal handling for status reporting

rm

Login/Password Management Utilities --->

Use internal crypt functions

[*] login

[*] passwd

Linux Module Utilities --->

[*] insmod

[*] rmmod

[*] lsmod

[] Pretty output

[*] modprobe

[*] Support tainted module checking with new kernels

(/lib/modules) Default directory containing modules

(modules.dep) Default name of modules.dep

Linux System Utilities --->

[*] dmesg

[*] Pretty dmesg output

[*] mount

[*] Support mounting NFS file systems

[*] Support lots of -o flags in mount

[*] Support lots of -o flags in mount

[*] Support /etc/fstab and -a

Networking Utilities --->

[*] ifconfig

[*] Enable status reporting output (+7k)

[*] Enable option "hw" (ether only)

[*] netstat

[*] ping

[*] Enable fancy ping output

[*] route

[*] wget

[*] Enable a nifty process meter (+2k)

[*] Enable HTTP authentication

Miscellaneous Configuration --->

RAMFS Image (none) --->

(X) none

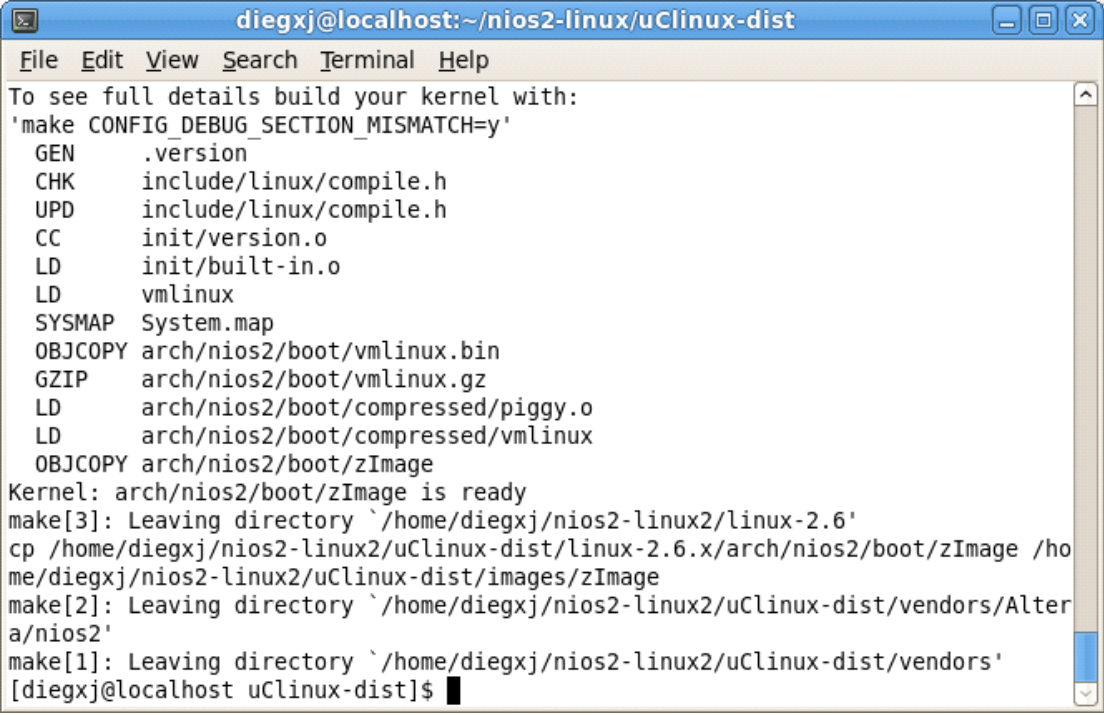
[*] generic cgi

Luego de haber seleccionado todas estas opciones, compilamos el kernel con el comando:

\$make

.

Al final de la compilación, deberíamos obtener una pantalla así:



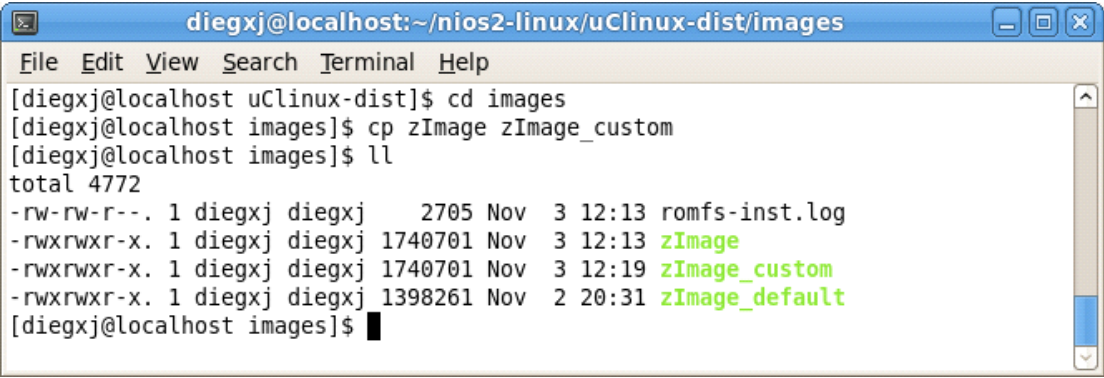
```

diegxj@localhost:~/nios2-linux/uClinux-dist
File Edit View Search Terminal Help
To see full details build your kernel with:
'make CONFIG_DEBUG_SECTION_MISMATCH=y'
GEN    .version
CHK    include/linux/compile.h
UPD    include/linux/compile.h
CC     init/version.o
LD     init/built-in.o
LD     vmlinux
SYSMAP System.map
OBJCOPY arch/nios2/boot/vmlinux.bin
GZIP   arch/nios2/boot/vmlinux.gz
LD     arch/nios2/boot/compressed/piggy.o
LD     arch/nios2/boot/compressed/vmlinux
OBJCOPY arch/nios2/boot/zImage
Kernel: arch/nios2/boot/zImage is ready
make[3]: Leaving directory `/home/diegxj/nios2-linux2/linux-2.6'
cp /home/diegxj/nios2-linux2/uClinux-dist/linux-2.6.x/arch/nios2/boot/zImage /home/diegxj/nios2-linux2/uClinux-dist/images/zImage
make[2]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors/Altera/nios2'
make[1]: Leaving directory `/home/diegxj/nios2-linux2/uClinux-dist/vendors'
[diegxj@localhost uClinux-dist]$

```

Figura 3.24 Ventana después de la compilación.

En la carpeta `uClinux-dist/images` realizamos una copia al nuevo archivo `zImage`, y la renombramos a `zImage_custom`.



```

diegxj@localhost:~/nios2-linux/uClinux-dist/images
File Edit View Search Terminal Help
[diegxj@localhost uClinux-dist]$ cd images
[diegxj@localhost images]$ cp zImage zImage_custom
[diegxj@localhost images]$ ll
total 4772
-rw-rw-r--. 1 diegxj diegxj  2705 Nov  3 12:13 romfs-inst.log
-rwxrwxr-x. 1 diegxj diegxj 1740701 Nov  3 12:13 zImage
-rwxrwxr-x. 1 diegxj diegxj 1740701 Nov  3 12:19 zImage_custom
-rwxrwxr-x. 1 diegxj diegxj 1398261 Nov  2 20:31 zImage_default
[diegxj@localhost images]$

```

Figura 3.25 Contenido del directorio “images”.

Al comparar los tamaños de las imágenes comprimidas del kernel nos damos cuenta del peso de la configuración seleccionada.

La imagen comprimida del kernel no debe pasar de 1.8 a 1.9 MB ya que se carga en la memoria SDRAM de la tarjeta DE2 que es de 4 MB.

La carga de parámetros se puede simplificar manejando los archivos de configuración que se encuentran en la carpeta *nios2-linux/μClinix-dist/vendors/Altera/nios2*.

El archivo *config.linux-2.6.x* guarda las configuraciones del kernel.

El archivo *config.vendor* guarda las configuraciones de las aplicaciones de usuario y librerías.

Estos dos archivos guardan las configuraciones que se cargan con la opción *Default all settings (lose changes)*.

Una vez que hemos terminado de configurar el μClinix a la medida de nuestro hardware podemos salvar nuestra configuración con las opciones que está al final de cada menú principal.

Save an Alternate Configuration File

De la misma manera podemos cargar esta configuración nuevamente con

Load an Alternate Configuration File

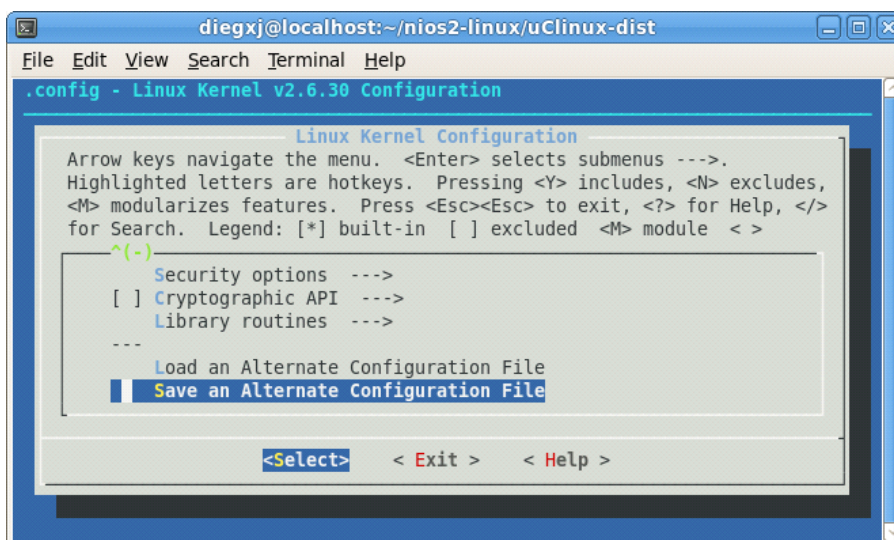


Figura 3.26 Ventada para guardar la configuración

3.1.7 CONFIGURACIÓN, COMPILACIÓN Y GENERACIÓN DE LA IMAGEN DEL KERNEL DE μ CLINUX CON PARÁMETROS PERSONALIZADOS PARA INCLUIR HARDWARE ADICIONAL Y FUNCIONES DE DEPURACIÓN DE APLICACIÓN DE USUARIO

Para integrar módulos de controladores de hardware adicionales a los que vienen en la distribución de μ Clinux para Altera Nios II hay que conocer la forma en que han sido empaquetadas las fuentes y los archivos clave del menú de configuración.

El directorio linux-2.6

Contiene la versión del kernel de Linux modificado para ejecutarse en el Nios II. Destacamos el subdirectorio drivers donde se encuentran los módulos de controladores de hardware agrupados en carpeta por afinidad.

El directorio μ Clinux-dist

Contiene los archivos de configuración de la distribución actual y el kconfig.

Dentro de los subdirectorios más importantes tenemos:

images

Aquí se guardan las imágenes que son generadas luego de una compilación exitosa. El nombre por defecto es zImage.

linux-2.6.x

En este directorio se guardan los archivos objeto que resultan de la compilación de los archivos fuentes individuales.

romfs

Este directorio contiene la estructura de árbol de los archivos de μ Clinux antes de ser comprimidos en el archivo imagen.

users

Este directorio contiene los archivos fuentes de las aplicaciones, librerías, etc. que se encuentran en la sección respectiva del menú de configuración de μ Clinux.

vendors

Este directorio contiene información de configuración específica de cada fabricante y de su producto respectivo. En teoría podríamos hacer un porting de μ Clinux para cada uno de los productos que ahí se encuentran.

Make menuconfig: Sistema de configuración kconfig.

El instalador de código fuente μ Clinux-dist utiliza el sistema de configuración make menuconfig.

En gran parte de directorios se encuentran archivos sin extensión de nombres *Kconfig* y *Makefile*.

En los *Kconfig* se encuentra la base de cada sección del menú de configuración. Los textos de las ayudas, las descripciones, y los nombres de variables que relacionan a cada ítem de los menús.

En los *Makefile* se encuentran los parámetros de compilación y el nombre del archivo objeto (extensión .o) que debe ser igual al nombre del archivo del código fuente de lenguaje C (extensión .c).

Para los propósitos de nuestro proyecto, necesitamos incluir el driver de nuestro Core PWM Avalon Slave.

3.1.8 AÑADIR UN MÓDULO CARGABLE PARA EL CONTROLADOR DEL PWM CORE.

Los archivos necesarios para controlar y utilizar el PWM IP Core desde μ Clinux son los siguientes:

Módulo de Driver:

`pwm_avalon.c`

Aplicación de usuario:

`pwmout.c`

`pwmavalon.h`

`altera_avalon_pwm_routines.h`

El código fuente de cada uno de estos archivos puede ser revisado en los anexos. Copiamos el archivo `pwm_avalon.c` en la carpeta `nios2-linux/linux-2.6/drivers/misc`. Con la ayuda de un editor de texto modificamos el archivo `Kconfig`, añadiendo las siguientes líneas al final del archivo:

```
config PWM_AVALON
```

```
    tristate "Driver de PWM avalon IP core"
```

```
    help
```

```
    Habilita el módulo para acceder al IP core PWM avalon.
```

Así mismo modificamos el archivo Makefile añadiendo la siguiente línea como se indica en la figura.

```

#
# Makefile for misc devices that really don't fit anywhere else.
#

obj-$(CONFIG_IBM_ASM)           += ibmasm/
obj-$(CONFIG_HDPU_FEATURES)     += hdpuftrs/
obj-$(CONFIG_ATMEL_PWM)         += atmel_pwm.o
obj-$(CONFIG_ATMEL_SSC)        += atmel-ssc.o
obj-$(CONFIG_ATMEL_TCLIB)      += atmel_tclib.o
obj-$(CONFIG_ICS932S401)       += ics932s401.o
obj-$(CONFIG_LKDTM)            += lkdtm.o
obj-$(CONFIG_TIFM_CORE)        += tifm_core.o
obj-$(CONFIG_TIFM_7XX1)        += tifm_7xx1.o
obj-$(CONFIG_PHANTOM)          += phantom.o
obj-$(CONFIG_SGI_IOC4)         += ioc4.o
obj-$(CONFIG_ENCLOSURE_SERVICES) += enclosure.o
obj-$(CONFIG_KGDB_TESTS)       += kgdbts.o
obj-$(CONFIG_SGI_XP)           += sgi-xp/
obj-$(CONFIG_SGI_GRU)          += sgi-gru/
obj-$(CONFIG_HP_ILO)           += hpilo.o
obj-$(CONFIG_ISL29003)         += isl29003.o
obj-$(CONFIG_PWM_AVALON)       += pwm_avalon.o
obj-$(CONFIG_C2PORT)           += c2port/
obj-y                           += eeprom/

```

Figura 3.27 Modificación del archivo Makefile.

Notamos que PWM_AVALON escrito así con mayúsculas es común para el Kconfig y el Makefile.

En el Makefile debemos asegurarnos que el nombre del archivo objeto (.o) sea igual al nombre del archivo fuente (.c).

Para más información sobre los archivos Makefile revisar el archivo *linux-2.6/Documentation/kbuild/makefiles.txt*

Estos pasos son suficientes para agregar el módulo PWM al menú de configuración de μ Clinux.

Para comprobar que se añadió correctamente el módulo entramos al menú correspondiente en la plataforma de configuración de μ Clinux

```
$ cd  $\mu$ Clinux-dist
```

```
$ make menuconfig
```

```
...
```

```
Kernel/Library/Defaults Selection --->
```

```
[*] Customize Kernel Settings
```

```
<Exit><Exit><Yes>
```

```
...
```

```
Device Drivers --->
```

```
[*] Misc devices --->
```

```
[*] Driver de PWM avalon IP core
```

```
<Exit><Exit><Exit><Yes>
```

```

-- Misc devices
< > Enclosure Services
<M> Driver de PWM avalon IP core
< > Silicon Labs C2 port support (EXPERIMENTAL) --->
    EEPROM support --->
    DE2 PIO DEVICES DRIVERS --->

```

Figura 3.28 Ventana selección Misc devices

```

Driver de PWM avalon IP core
CONFIG_PWM_AVALON:
Habilita el modulo para acceder al IP core PWM avalon.
Symbol: PWM_AVALON [=y]
Prompt: Driver de PWM avalon IP core
Defined at drivers/misc/Kconfig:236
Depends on: MISC_DEVICES
Location:
-> Device Drivers
-> Misc devices (MISC_DEVICES [=y])

```

Figura 3.29 Ventana Driver de pwm avalon Ip Core.

Una vez que salimos del menú, ejecutamos el comando *make* dar inicio a la compilación.

Si seguimos con atención los resultados (verbose) de la compilación, podremos ver que se hace referencia a la compilación del `pwm_avalon` con la siguiente línea:

```
CC [M] drivers/misc/pwm_avalon.o
```

```
Building modules, stage 2.
```

```
MODPOST 2 modules
```

```
LD [M] drivers/misc/pwm_avalon.ko
```

Los errores y advertencias que se presenten durante la compilación también aparecerán en este espacio.

Ahora necesitamos incluir una herramienta que servirá para depurar la aplicación de usuario.

3.1.8.1 DEPURACIÓN REMOTA DE APLICACIÓN DEL TARGET DESDE EL HOST

```
$ make menuconfig
```

```
...
```

```
Kernel/Library/Defaults Selection --->
```

```
  [*] Customize Application/Library Settings
```

```
<Exit><Exit><Yes>
```

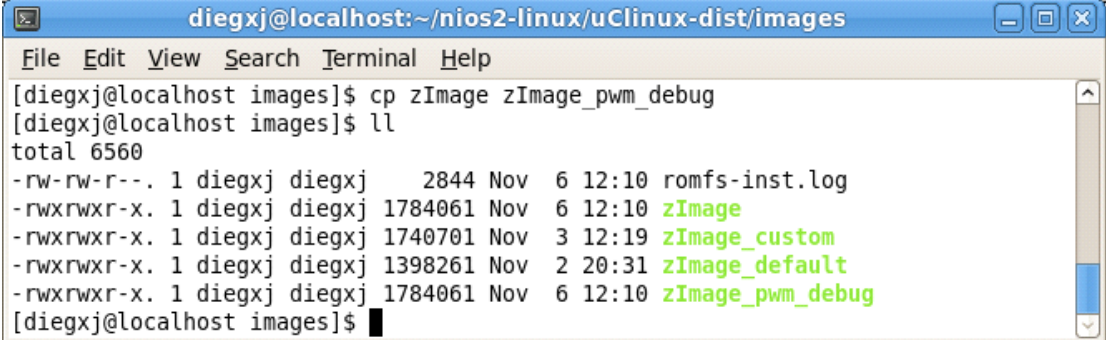
```
...
```

```
Miscellaneous Applications --->
```

```
  [*] gdbserver (old)
```

```
<Exit><Exit><Yes>
```

Volvemos a compilar la imagen del kernel y la renombramos como `zImage_pwm_debug`.



```

diegxm@localhost:~/nios2-linux/uClinux-dist/images
File Edit View Search Terminal Help
[diegxm@localhost images]$ cp zImage zImage_pwm_debug
[diegxm@localhost images]$ ll
total 6560
-rw-rw-r--. 1 diegxm diegxm  2844 Nov  6 12:10 romfs-inst.log
-rwxrwxr-x. 1 diegxm diegxm 1784061 Nov  6 12:10 zImage
-rwxrwxr-x. 1 diegxm diegxm 1740701 Nov  3 12:19 zImage_custom
-rwxrwxr-x. 1 diegxm diegxm 1398261 Nov  2 20:31 zImage_default
-rwxrwxr-x. 1 diegxm diegxm 1784061 Nov  6 12:10 zImage_pwm_debug
[diegxm@localhost images]$

```

Figura 3.30 Ventana después de compilar la Imagen del Kernel.

3.1.8.2 COMPILACIÓN Y DEPURACIÓN DE LA APLICACIÓN

Podemos compilar la aplicación de dos maneras:

- Por línea de comando en la consola.
- Utilizando un Entorno integrado de Desarrollo como Eclipse

En ambas utilizaremos las herramientas del compilador cruzado de μ Clinux.

3.1.8.3 COMPILACIÓN POR LÍNEA DE COMANDO EN CONSOLA

Debemos asegurarnos que en el PATH se encuentra la ruta del compilador cruzado de μ Clinux.

Colocamos todos los archivos fuentes de la aplicación (archivos `.c` y archivos `.h`) en una misma carpeta.

```
$ nios2-linux-uclibc-gcc pwmout.c -o pwmout -elf2flt="-s 16000" -Wall
```

Este comando genera el archivo *pwmavalon* (sin extensión) con un tamaño de pila (stack) de 16000 con todos los warning habilitados.

El archivo generado lo copiamos en la carpeta *romfs/bin* del directorio *μClinix-dist*.

Luego ejecutamos los siguientes comandos:

```
$make linux image
```

Esto generará la nueva imagen con la aplicación de usuario **pwmout** en el directorio */bin* de nuestra distribución de *μClinix*.

3.1.8.4 COMPILACIÓN Y DEPURACIÓN UTILIZANDO UN ENTORNO INTEGRADO DE DESARROLLO COMO ECLIPSE IDE

3.1.8.4.1 CONFIGURACIÓN DE ECLIPSE CDT PARA UTILIZAR EL COMPILADOR CRUZADO DE UCLINUX

A continuación vamos a listar una serie de pasos para configurar Eclipse CDT para compilar y depurar nuestra aplicación de *μClinix*:

- Ejecutar el Eclipse CDT

- Seleccionar un directorio para que sea el área de trabajo (workspace)
- Crear un nuevo proyecto C
- En el cuadro de proyecto C, colocar lo siguiente:
 - Nombre de proyecto: pwmout
 - Tipo de proyecto: Executable - Cross-Compile Project
 - Toolchains: Cross GCC
 - Darle click en Next,
- En el cuadro de Cross compile command, colocar lo siguiente:
 - Tool command prefix: nios2-linux-
 - Tool command path: /home/diegxj/opt/nios2
 - Darle click en Next,
- Finish.

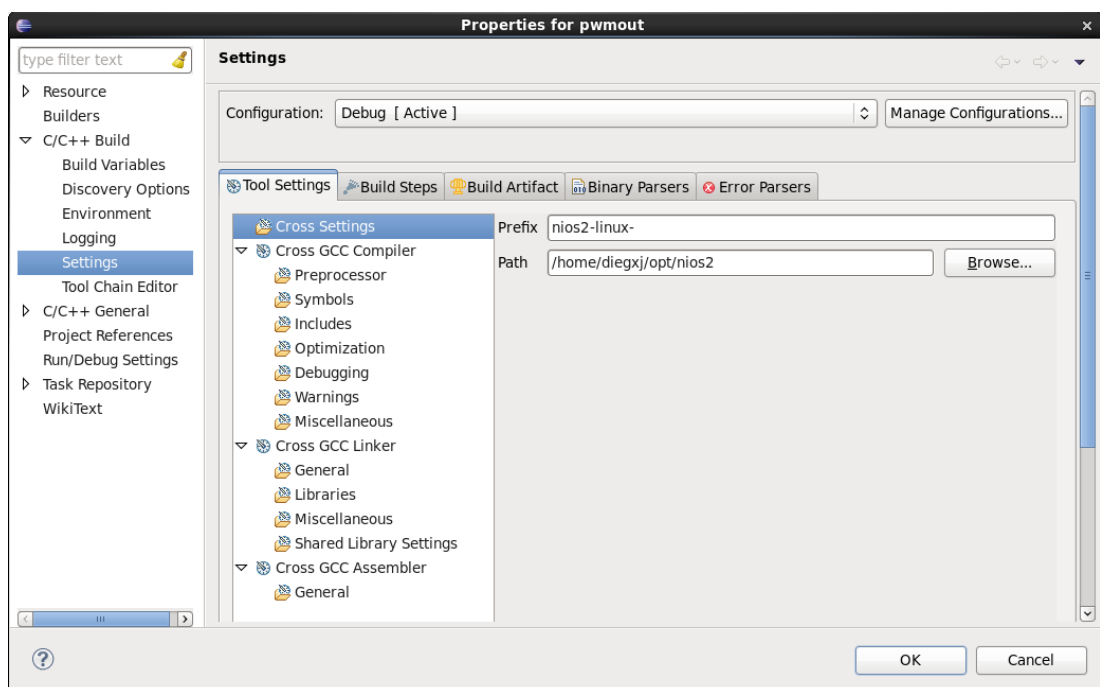


Figura 3.31 Ventana de configuración del compilador.

- Damos OK para guardar los cambios y salir.

3.1.8.4.2 CONFIGURACIÓN DEL DEPURADOR (DEBUGGER)

Esta configuración sirve para que el Eclipse CDT funcione como cliente remoto del gdbserver (servidor de depuración de aplicaciones remoto) instalado en la distribución de μ Clinux de la tarjeta DE2. Por defecto el gdbserver utiliza el puerto 9999. Utilizamos una conexión TCP por facilidad ya que tenemos un puerto Ethernet en la tarjeta DE2 y se puede encontrar uno en cualquier computador.

- En el menú “Run”, damos click en “Debug configurations...”
- Damos doble click en “C/C++ Application”
- Escribimos un nombre (Name) para nuestra configuración de depuración. Por ejemplo: pwmout Debug
- En la pestaña de “Main”, en el cuadro de texto de “C/C++ Application” escribimos Debug/pwmout.gdb.
- En “Project”: pwmout (o el nombre de nuestro proyecto)
- En la parte inferior, junto a los botones “Apply” y “Revert”, damos click en “Select other...”
- En el cuadro de dialogo que aparece, seleccionamos “Use Estándar Create Process Laucher”, y damos click en OK.

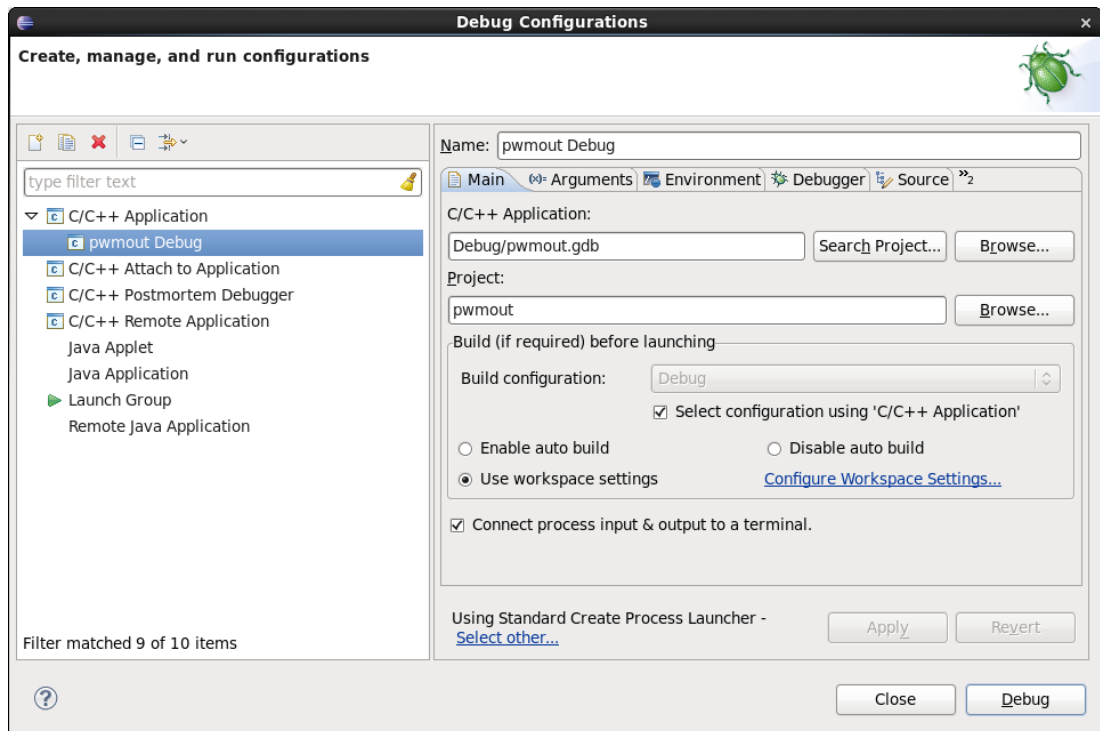


Figura 3.32. Ventana de Sistema de configuración de ejecución.

- En la pestaña de “Debugger”, en el recuadro de “Debugger Options”, en la pestaña “Main”, tecleamos *nios2-linux-gdb* en el cuadro de texto de “GDB Debugger”.

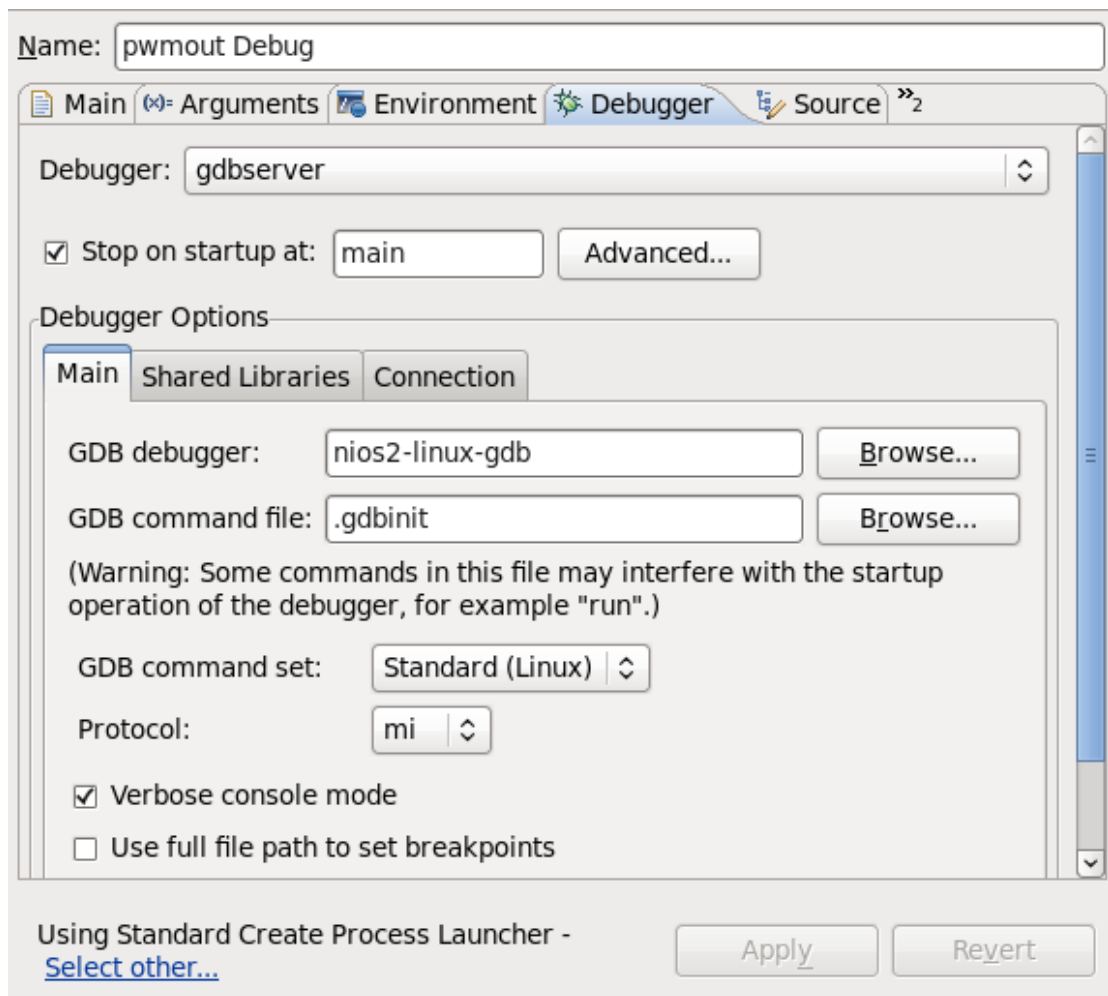


Figura 3.33 Ventana de Debugger.

- En la pestaña de “Connection”, seleccionamos *TCP* en “Type” y en “Host or IP address” escribimos una dirección IP cualquiera, en nuestro caso usamos la *192.168.1.2*. Esta dirección IP deberá ser luego asignada al módulo de Ethernet de la tarjeta DE2. En “Port” escribimos *9999*.

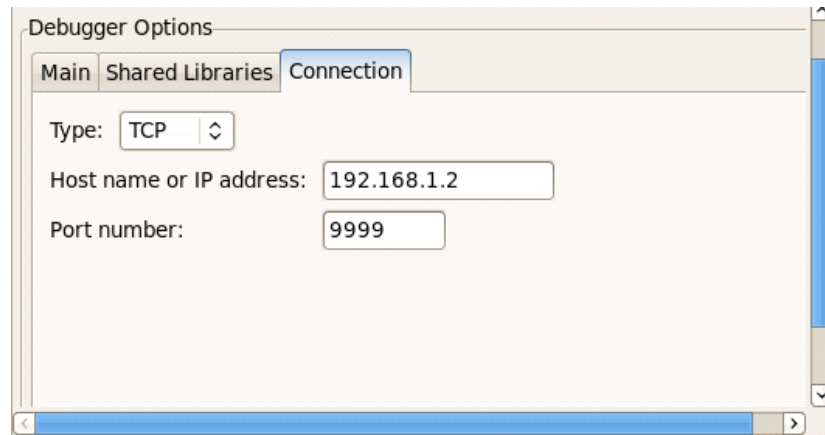


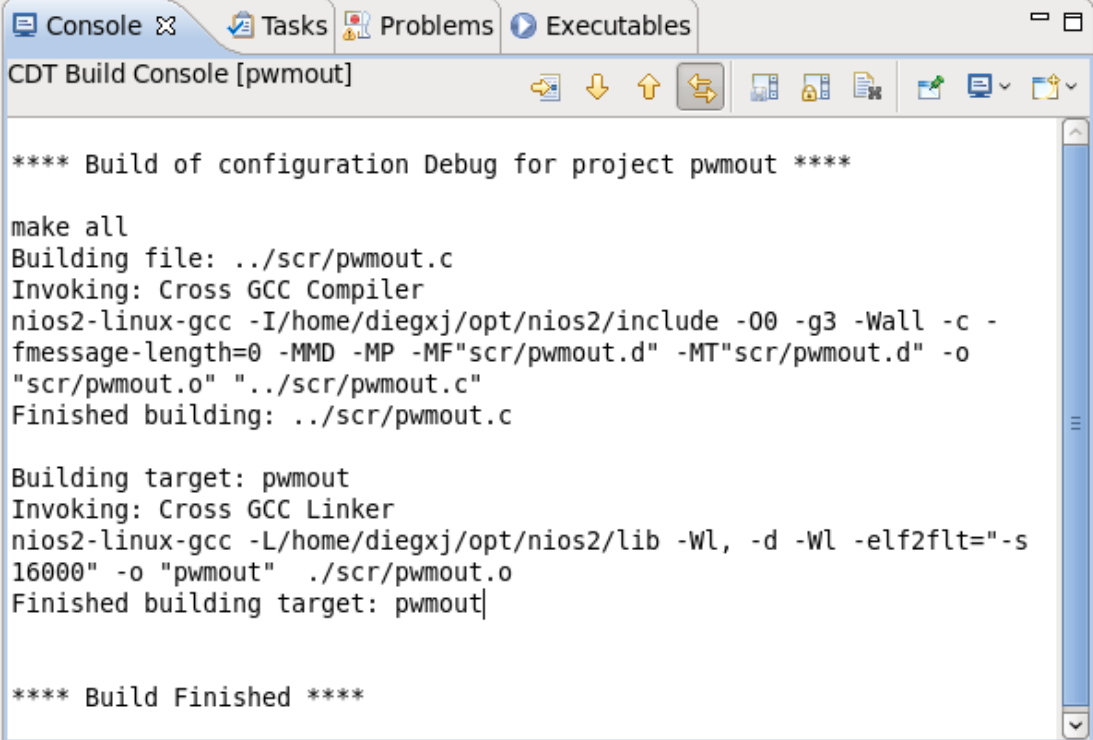
Figura 3.34 Ventana de Configuraciones de TCP IP

- Damos click en “Apply”
- Damos click en OK.

3.1.8.5 COMPILACIÓN DE LA APLICACIÓN

En el menú Project, damos click en Build All. Corregimos los errores y warnings que se presenten hasta que la compilación sea exitosa.

Se deben haber generado dos archivos: *pwmout* y *pwmout.gdb*.



```

CDT Build Console [pwmout]

**** Build of configuration Debug for project pwmout ****

make all
Building file: ../scr/pwmout.c
Invoking: Cross GCC Compiler
nios2-linux-gcc -I/home/diegxj/opt/nios2/include -O0 -g3 -Wall -c -
fmessage-length=0 -MMD -MP -MF"scr/pwmout.d" -MT"scr/pwmout.d" -o
"scr/pwmout.o" "../scr/pwmout.c"
Finished building: ../scr/pwmout.c

Building target: pwmout
Invoking: Cross GCC Linker
nios2-linux-gcc -L/home/diegxj/opt/nios2/lib -Wl, -d -Wl -elf2flt="-s
16000" -o "pwmout" ../scr/pwmout.o
Finished building target: pwmout

**** Build Finished ****

```

Figura 3.35 Mensajes del Compilador al construir.

Comprobamos el archivo generado con el siguiente comando en el directorio donde se crea el programa *pwmout*:

```
$nios2-linux-uclibc-flthdr pwmout .
```

Este comando nos da una lectura del header del archivo binary flat (formato de ejecutable de μ Clinux). El resultado debe ser similar a este:

```

Magic:      bFLT
Rev:        4
Build Date: Mon Jan 16 03:17:08 2012
Entry:      0x44

```

Data Start: 0x8940
Data End: 0xa460
BSS End: 0xc4c0
Stack Size: 0x3e80
Reloc Start: 0xa460
Reloc Count: 0x282
Flags: 0x1 (Load-to-Ram)

Con esto podemos verificar el tamaño de la pila (stack) que debe ser de 16 KB (0x3e80 = 16000) tal como lo indicamos en las opciones del Linker.

```
/home/diegxj/workspace/pwmout/Debug/pwmout
Magic:      bFLT
Rev:        4
Build Date: Mon Jan 16 03:17:08 2012
Entry:      0x44
Data Start: 0x8940
Data End:   0xa460
BSS End:    0xc4c0
Stack Size: 0x3e80
Reloc Start: 0xa460
Reloc Count: 0x282
Flaas:      0x1 ( Load-to-Ram )
```

Figura 3.36 Tamaño de la pila del pwmout

CAPÍTULO 4

4. RESULTADOS

4.1 EJECUCIÓN DE μ CLINUX-NIOS2

Para poder descargar los archivos necesarios para ejecutar μ Clinux en la tarjeta, necesitamos utilizar la consola de Nios II EDS.

Hemos preparado un script para ejecutarlo en la consola de Linux. El contenido del script puede revisarse en el Anexo IV.

```
$/nios2shell
```

```
-----
```

```
Welcome To Altera SOPC Builder
```

```
Version 9.1, Built Wed Mar 24 20:50:27 PDT 2010
```

```
-----
```

```
-----
```


Welcome to the Nios II Embedded Design Suite

Version 9.1, Built Wed Mar 24 22:04:58 PDT 2010

Example designs can be found in

/home/diegxj/opt/altera/9.1/nios2eds/examples

(You may add a startup script: /home/diegxj/opt/altera/9.1/nios2eds/user.bashrc)

~/opt/altera/9.1/nios2eds

[NiosII EDS]\$

Debemos tener conectado el cable USB que vino junto con la tarjeta DE2 y realizar la configuración contenida en el Anexo Configuración de JTAG.

Finalizada la configuración, ejecutamos el siguiente comando en el shell de Nios II EDS.

[NiosII EDS]\$ jtagconfig

1) USB-Blaster [USB 1-1.6.1]

020B40DD EP2C35

Para configurar el FPGA con el archivo imagen SOF:

[NiosII EDS]\$ nios2-configure-sof

/home/diegxj/projects_q2/de2_vga_nommu2/de2vga_nommu/DE2_NIOS_HOST_M
OUSE_VGA_time_limited.sof

File

/home/diegxj/projects_q2/de2_vga_nommu2/de2vga_nommu/DE2_NIOS_HOST_M
OUSE_VGA_time_limited.sof contains one or more time-limited megafunctions that
support the OpenCore Plus feature that will not work after the hardware evaluation
time expires. Refer to the Messages window for evaluation time details.

Info: SRAM Object File

/home/diegxj/projects_q2/de2_vga_nommu2/de2vga_nommu/DE2_NIOS_HOST_M
OUSE_VGA_time_limited.sof contains time-limited megafunction that supports
OpenCore Plus feature -- Vendor: 0x6AF7, Product: 0x00A2

Info:

Info: Running Quartus II Programmer

Info: Command: quartus_pgm --no_banner --mode=jtag -o

p;/home/diegxj/projects_q2/de2_vga_nommu2/de2vga_nommu/DE2_NIOS_HOST_
MOUSE_VGA_time_limited.sof

Info: Using programming cable "USB-Blaster [USB 1-1.6.1]"

Info: Started Programmer operation at Sun Jan 15 13:31:38 2012

Info: Configuring device index 1

Info: Device 1 contains JTAG ID code 0x020B40DD

Info: Configuration succeeded -- 1 device(s) configured

Info: Successfully performed operation(s)

Info: Ended Programmer operation at Sun Jan 15 13:31:40 2012

Please enter i for info and q to quit:

Debemos dejar esta ventana abierta por restricciones de la licencia.

Abrimos otra sesión de shell Nios II EDS ejecutando nuevamente el script.

Para descargar el archivo zImage de la imagen del Kernel de μ Clinux, ejecutamos lo siguiente:

```
[NiosII EDS]$ nios2-download -g /home/diegxj/nios2-linux/ $\mu$ Clinux-  
dist/images/zImage
```

```
Using cable "USB-Blaster [USB 1-1.6.1]", device 1, instance 0x00
```

```
Pausing target processor: OK
```

```
Initializing CPU cache (if present)
```

```
OK
```

```
Downloaded 1815KB in 9.7s (187.1KB/s)
```

```
Verified OK
```

```
Starting processor at address 0x02D00000
```

```
~/opt/altera/9.1/nios2eds
```

```
[NiosII EDS]$
```

Aquí podemos notar lo siguiente: la dirección a la que el procesador es iniciado es la 0x02D00000, esto se debe a que la dirección de la memoria SDRAM en el mapeo de dirección de la interface Avalon es la 0x02800000 y a que en la configuración del kernel indicamos que se inicie con un offset de 0x00500000.

Ya esta descargada la imagen de μ Clinux en el target Tarjeta DE2.

Para iniciar μ Clinux podemos configurar dos alternativas: Terminal en el puerto JTAG UART y Terminal en el puerto SERIAL UART.

En el capítulo 3 vimos donde configurar estas opciones. Solo es posible usar una de ellas a la vez.

4.1.1 TERMINAL REMOTO DE UCLINUX EN EL PUERTO JTAG UART

Luego descargada la imagen del kernel, en el shell de Nios II EDS escribimos:

```
[NiosII EDS]$ nios2-terminal
```

En unos instantes debería aparecer el siguiente mensaje, seguido del *verbose* de la inicialización de μ Clinux.

```
Uncompressing Linux... Ok, booting the kernel.
```

4.1.2 TERMINAL REMOTO DE UCLINUX EN EL PUERTO SERIAL

UART

Para abrir una Terminal por el puerto serie, primero necesitamos conectar un cable serial DB9 Macho – DB9 Hembra entre el target y el host.

En el host debemos abrir una sesión de un emulador de terminal como HyperTerminal en Windows, o minicom en Linux.

En Linux, configuramos minicom para transmitir y recibir por el puerto serie `/dev/ttyS0`.

Iniciamos una sesión de minicom con el siguiente comando y esperamos. Es posible que sea necesario iniciarlo como root.

```
#minicom
```

```
Welcome to minicom 2.3
```

```
OPTIONS: I18n
```

```
Compiled on Nov 23 2010, 13:27:26.
```

```
Port /dev/ttyS0
```

Press CTRL-A Z for help on special keys

Si μ Clinux fue configurado para iniciar sesion de Terminal via serial uart, el proceso de inicialización debería empezar inmediatamente despues de descargar la imagen del kernel en el Nios II.

Uncompressing Linux... Ok, booting the kernel.

Linux version 2.6.30 (diegxj@localhost.localdomain) (gcc version 3.4.6) #340 PR2

μ Clinux/Nios II

Built 1 zonelists in Zone order, mobility grouping off. Total pages: 2032

Kernel command line:

NR_IRQS:32

PID hash table entries: 32 (order: 5, 128 bytes)

Console: colour dummy device 80x25

Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)

Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)

Memory available: 4512k/3394k RAM, 0k/0k ROM (2256k kernel code, 1137k data)

Calibrating delay loop... 47.10 BogoMIPS (lpj=235520)

Mount-cache hash table entries: 512

net_namespace: 264 bytes

NET: Registered protocol family 16

init_BSP(): registering device resources

bio: create slab <bio-0> at 0

SCSI subsystem initialized

usbcore: registered new interface driver usbfs

usbcore: registered new interface driver hub

usbcore: registered new device driver usb

NET: Registered protocol family 2

IP route cache hash table entries: 1024 (order: 0, 4096 bytes)

TCP established hash table entries: 512 (order: 0, 4096 bytes)

TCP bind hash table entries: 512 (order: -1, 2048 bytes)

TCP: Hash tables configured (established 512 bind 512)

TCP reno registered

NET: Registered protocol family 1

JFFS2 version 2.2. (NAND)  © 2001-2006 Red Hat, Inc.

io scheduler noop registered

io scheduler deadline registered (default)

Device /dev/lcd16207 registered

ttyJ0 at MMIO 0x1401150 (irq = 2) is a Altera JTAG UART

ttyS0 at MMIO 0x14010a0 (irq = 3) is a Altera UART

console [ttyS0] enabled

El dispositivo PWM Avalon ha sido registrado con 1 salida en LEDG8.

Lo conforman los registros CLOCK_DIV 0x0081401130 DUTY 0x0081401134

LOAD 0x00818

Registration is a success The pwmclockdiv major device number is 242.

Registration is a success The pwmduty major device number is 243.

Registration is a success The pwmload major device number is 244.

Espere unos segundos y observe el movimiento del LEDG8 en la tarjeta DE2.

Ha sido cargado inicialmente con los siguientes valores: Periodo 100000000 Duty.

Eso nos da un periodo de 1 segundo aproximadamente a 100MHz del clock del siste.

Módulo cargado exitosamente.

Driver 'sd' needs updating - please use bus_type methods

dm9000 Ethernet Driver, V1.31

dm9000 dm9000.0: eth%d: Invalid ethernet MAC address. Please set using ifconfig

eth0 (dm9000): not using net_device_ops yet

eth0: dm9000a at 81401160,81401164 IRQ 5 MAC: 00:00:00:00:00:00 (chip)

physmap platform flash device: 00400000 at 01000000

physmap-flash.0: Found 1 x16 devices at 0x0 in 8-bit bank

Amd/Fujitsu Extended Query Table at 0x0040

number of CFI chips: 1

cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.

RedBoot partition parsing not available

Using physmap partition information

Creating 2 MTD partitions on "physmap-flash.0":

0x000000200000-0x000000400000 : "romfs/jffs2"

0x000000000000-0x000000200000 : "loader/kernel"

m25p80 spi0.0: found m25p64, expected m25p16

driver isp1362-hcd, 2005-04-04

isp1362-hcd isp1362-hcd: ISP1362 Host Controller

isp1362-hcd isp1362-hcd: new USB bus registered, assigned bus number 1

isp1362_hc_reset:

isp1362-hcd isp1362-hcd: irq 7, io mem 0x014010f0

isp1362_hc_start:

isp1362-hcd isp1362-hcd: ISP1362 Memory usage:

isp1362-hcd isp1362-hcd: ISTL: 2 * 256: 512 @ \$0000:\$0100

isp1362-hcd isp1362-hcd: INTL: 16 * (64+8): 1152 @ \$0200

isp1362-hcd isp1362-hcd: ATL : 32 * (64+8): 2304 @ \$0680

isp1362-hcd isp1362-hcd: USED/FREE: 3968 128

usb usb1: configuration #1 chosen from 1 choice

hub 1-0:1.0: USB hub found

hub 1-0:1.0: 1 port detected

ISP1362 Host Controller, irq 7

Initializing USB Mass Storage driver...

usbcore: registered new interface driver usb-storage

USB Mass Storage support registered.

altps2 : base 81401158 irq 4

mice: PS/2 mouse device common for all mice

mmc_spi spi2.0: ASSUMING SPI bus stays unshared!

mmc_spi spi2.0: ASSUMING 3.2-3.4 V slot power

mmc_spi spi2.0: SD/MMC host mmc0, no DMA, no WP, no poweroff

Registered led device: led0

usbcore: registered new interface driver usbhid

usbhid: v2.6:USB HID core driver

TCP cubic registered

NET: Registered protocol family 17

RPC: Registered udp transport module.

RPC: Registered tcp transport module.

Freeing unused kernel memory: 752k freed (0x2a94000 - 0x2b4f000)

Shell invoked to run file: /etc/rc

Command: hostname μ Clinux

Command: mount -t proc proc /proc -o noexec,nosuid,nodev

atkbd.c: keyboard reset failed on altps2.0

Command: mount -t sysfs sysfs /sys -o noexec,nosuid,nodev

Command: mount -t devpts devpts /dev/pts -o noexec,nosuid

Command: mount -t usbfs none /proc/bus/usb

Command: mkdir /var/tmp

Command: mkdir /var/log

Command: mkdir /var/run

Command: mkdir /var/lock

Command: mkdir /var/empty

Command: ifconfig lo 127.0.0.1


```
/>
```

4.2 PRIMEROS PASOS EN mCLINUX

4.2.1 ESCRIBIR Y LEER DEL BUS AVALON DESDE EL ESPACIO DE USUARIO

En el directorio /bin se encuentra la aplicación nios2io, que viene con las fuentes de μ Clinux y se la puede escoger durante la configuración de las aplicaciones.

Es una aplicación de usuario con funciones de acceso directo a las direcciones físicas mapeadas en memoria de los periféricos I/O.

```
/>nios2io
```

```
/> nios2io
```

```
nios [type] [address] [value]
```

```
[type]
```

rd - read the value from address and return decimal string

wd - write the decimal string of [value] to the address

Rd - same as parameter rh but with verbose output

Wd - same as parameter wh but with verbose output

rh - read the value from address and return hex string

wh - write the hex string of [value] to the address

Rh - same as parameter rh but with verbose output

Wh - same as parameter wh but with verbose output

dd - wait for micro seconds of decimal value [address]

Dd - same as parameter dh but with verbose output

dh - wait for micro seconds of hex value [address]

Dh - same as parameter dh but with verbose output

[address]

address (hex) of the component on the avalon bus

[value]

This parameter is only used for writing.

Examples:

```
nios2io wd 80681070 127
```

```
nios2io wh 80681070 F
```

```
nios2io rd 806810A0
```

```
nios2io rh 806810A0
```

```
nios2io dd 120
```

```
nios2io dh ABCDEF
```

Las direcciones deben ser escritas con el bit 31 = 1 para saltarse el cache de datos.

Esto se logra sumándole 0x80000000 a la dirección del dispositivo.

Por ejemplo para encender todos los leds verdes, escribimos

```
/> nios2io wh 81401100 FF
```

Para leer los 8 switches que estan bajo la interface switch_pio, ejecutamos lo siguiente:

```
/> nios2io rh 81401120
```

```
0000000A
```

```
/> nios2io rh 81401120
```

```
0000007F
```

```
/> nios2io rh 81401120
```

```
000000FF
```

Siendo las posiciones de los switches las equivalentes en formato binario.

Incluso podemos cargar valores en nuestro controlador PWM. Para setear una señal

PWM de Periodo = 0.5 Hz, Ancho de pulso = 50%, ejecutamos:

```
/> nios2io wd 81401130 50000000
```

```
/> nios2io wd 81401134 25000000
```

```
/> nios2io wd 81401138 1
```

El código fuente de esta aplicación (nios2io.c) se encuentra disponible en las fuentes de nios2-linux (μ Clinux-dist/usr/nios2io/nios2io.c).

4.2.2 CARGAR EL MÓDULO DE DRIVER PARA EL MÓDULO PWM

```
/> modprobe pwm_avalon
```

El dispositivo PWM Avalon ha sido registrado con 1 salida en LEDG8.

```
Lo conforman los registros CLOCK_DIV 0x0081401130 DUTY 0x0081401134  
LOAD 0x00818
```

Registration is a success The pwmclockdiv major device number is 242.

Registration is a success The pwmduty major device number is 243.

Registration is a success The pwmload major device number is 244.

Espere unos segundos y observe el movimiento del LEDG8 en la tarjeta DE2.

Ha sido cargado inicialmente con los siguientes valores: Periodo 100000000 Duty cycle: 50000000.

Eso nos da un periodo de 1 segundo aproximadamente a 100MHz del clock del sistema.

Módulo cargado exitosamente.

```
/>
```

Observamos en la tarjeta el parpadeo del LEDG8.

Si lo queremos descargar, utilizamos el comando

```
/>rmmod pwm_avalon
```

PWM OFF!! Módulo descargado exitosamente.

Se resalta el hecho de que al descargar el módulo se apaga el led. Esto fue completamente intencional, ya que la descarga del módulo no implica la desconexión del hardware sino solo que queda sin control del sistema operativo. El valor de los registros del PWM Avalon core se sostiene en su último estado.

El código fuente documentado del driver puede encontrarse en los anexos.

4.2.3 INICIALIZAR EL CHIP DE ETHERNET DM9000

En uno de los scripts de inicio se añadieron unos comandos para inicializar el hardware de red.

El archivo script de inicio rc, que en la fuentes se encuentra en el


```
directorio ../vendors/Altera/nios2
```

Podemos escribir estos comandos en la consola de μ Clinux, o añadir estas líneas en el archivo rc antes de la compilación del kernel de μ Clinux.

```
ifconfig eth0 hw ether 00:07:ed:0a:03:29
```

```
ifconfig eth0 192.168.1.2
```

```
route add default gw 192.168.1.254
```

```
ifconfig eth0 up
```

Si hay un cable de red cruzado conectado en el target y el host, aparece el siguiente mensaje en el terminal:

```
/> eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
```

Verificamos las interfaces de red

```
/>ifconfig
```

```
eth0  Link encap:Ethernet HWaddr 00:07:ED:0A:03:29
```

```
inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0
```

```
UP BROADCAST MULTICAST MTU:1500 Metric:1
```

```
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:1000
```

```
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

```
Interrupt:5 Base address:0x1160
```

```
lo    Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0

        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0

        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

4.2.4 TRANSFERIR ARCHIVOS DESDE EL HOST AL TARGET VÍA FTP SOBRE ETHERNET

Para levantar el servidor FTP de μ Clinux, ejecutamos

```
/>inetd&
```

```
[37]
```

```
/>
```

inetd es un script binario que se compila junto con las demás aplicaciones que conforman el μ Clinux-dist. Carga el servidor ftpd y el httpd.

El número que se genera es el PID se le asigno a la aplicación. El & después del nombre permite ejecutar la aplicación en segundo plano.

Desde el computador host, abrimos cualquier cliente FTP. Nosotros usamos un cliente

FTP para consola.

```
$ ftp 192.168.1.2
```

```
Connected to 192.168.1.2 (192.168.1.2).
```

```
220- Welcome to the µClinix ftpd!
```

```
220 µClinix FTP server (GNU inetutils 1.4.1) ready.
```

```
Name (192.168.1.2:diegxj): ftp
```

```
331 Guest login ok, type your name as password.
```

```
Password:
```

```
230 Guest login ok, access restrictions apply.
```

```
Remote system type is UNIX.
```

```
Using binary mode to transfer files.
```

```
ftp>
```

La transferencia de archivos se realiza en modo binario. Utilizaremos el comando put.

Transferiremos la aplicación de usuario generada en el capítulo 3 con el Eclipse CDT.

```
ftp> put /home/diegxj/workspace/pwmout/Debug/pwmout pwmout
```

```
local: /home/diegxj/workspace/pwmout/Debug/pwmout remote: pwmout
```

```
227 Entering Passive Mode (192,168,1,2,206,104)
```

```
150 Opening BINARY mode data connection for 'pwmout'.
```

```
226 Transfer complete.
```

```
44648 bytes sent in 0.0657 secs (679.26 Kbytes/sec)
```

```
ftp>
```

El archivo se copia en el target en la carpeta /home/ftp.

El archivo al inicio no tiene permisos de ejecución, deben ser añadidos con el comando chmod.

```
/home/ftp> ls -l
```

```
-rw-r----- 1 14 50 44648 Nov 29 16:01 pwmout
```

```
/home/ftp>
```

```
/home/ftp> chmod 777 pwmout
```

```
/home/ftp> ls -l
```

```
-rwxrwxrwx 1 14 50 44648 Nov 29 16:01 pwmout
```

```
/home/ftp>
```

4.3 UTILIZAR EL HARDWARE POR MEDIO DEL DRIVER DESDE UNA APLICACIÓN DE USUARIO

El driver es de tipo char device. Implementa 4 operaciones para el dispositivo: abrir, cerrar, leer, escribir.

Una vez compilado y cargado el driver, las cuatro operaciones se manejan como si se tratase de un fichero. Se implemento una función para lectura `read_dev()` y una función para escritura `write_dev()`.

El código fuente del driver se encuentra en los anexos.

La aplicación `pwmout` hace uso de las funciones de lectura y escritura, así de como cierto nivel de abstracción tomado de los archivos fuentes Altera para el IDE de Windows.

```
/> ./home/ftp/pwmout &
```

Periodo = 100000000

Ancho de pulso = 50000000

Enable bit = 1

Modo de uso:

El valor del ancho de pulso debe ser menor que el del periodo.

Para el Periodo y el ciclo de trabajo solo ingresar numeros enteros de hasta 4294967295.

El Enable bit solo puede ser de dos valores: 0 - PWM desactivado, 1 - PWM

activado.

Ingrese valor de periodo: 50000000

Ingrese valor de ancho de pulso: 25000000

Note el LEDG8 pulsando en la tarjeta de desarrollo.

En este punto, el programa entra en un lazo while infinito que empezará a variar el valor del ancho de pulso desde su valor actual hacia el valor del periodo y luego hacia 1 y luego de nuevo hacia el periodo y así sucesivamente.

Esto provocará que la aplicación consuma casi todos los recursos del CPU.

A la aplicación se le añadió unas instrucciones para controlar el estado de Habilitado/desabilitado del modulo PWM. El control se lo realiza a través del switch SW0 de la tarjeta DE2. El acceso al valor de este switch se lo realiza directamente desde la aplicación por medio de un puntero hacia la dirección de memoria del registro de interface Avalon del switch_pio.

```
//Includes
#include <stdio.h>
#include <unistd.h>
#include "headers/altera_avalon_pwm_routines.h"
#include "headers/pwmavalon.h"

//Dirección de memoria para acceso directo a E/S desde aplicación de usuario
#define na_switch_pio 0x81401120
```

Figura 4.1 Código fuente pwmout, dirección de memoria switch_pio

Se hace uso del atributo volatile para indicarle al compilador que no haga ciertas

optimizaciones al code que asume valores que no pueden cambiar su cuenta dentro de una función sino que recibe información externa, como es el caso de los dispositivos I/O.

```
int main(void)
{
// El uso de volatile asegura que el compilador siempre lleve a cabo el acceso a la memoria,
// en lugar de optimizarlos (por ejemplo, si el acceso es en un lazo).
volatile unsigned *switches = ((volatile unsigned *) (na_switch_pio));
unsigned long period = 0;
unsigned long duty = 0;
int return_code = ALTERA_AVALON_PWM_OK;
```

Figura 4.2 Puntero declarado volatile apuntando a los switches.

```
// Variación infinita del ciclo de trabajo entre 1 y el valor del periodo.
// El switch SW0 puede habilitar o deshabilitar el PWM
while(1)
{
while(duty++ < IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER)
{
IOWR_ALTERA_AVALON_PWM_LOAD(switches[0]);
altera_avalon_pwm_change_duty_cycle(duty);
check_return_code(duty_file, return_code);
}
while(--duty > 1)
{
IOWR_ALTERA_AVALON_PWM_LOAD(switches[0]);
altera_avalon_pwm_change_duty_cycle(duty);
check_return_code(duty_file, return_code);
}
}
return 0;
}
```

Figura 4.3 Lazo infinito de variación del ciclo de trabajo

4.4 DEPURAR UNA APLICACIÓN DE USUARIO DEL TARGET DESDE EL HOST POR MEDIO DE GDBSERVER Y ECLIPSE CDT

Ejecutamos la aplicación gdbserver

```
gdbserver 192.168.1.1:9999 /home/ftp/pwmout
```

```
Process /home/ftp/pwmout created; pid = 42
```

```
Listening on port 9999
```

En Eclipse, debemos tener abierto el proyecto donde fue compilada la aplicación y ejecutar Debug, desde el menu Run o desde el icono en la barra de herramienta. Previo a esto hay que configurar la instancia de Depuración remota tal como se lo vio en el capítulo 3.

Antes de comenzar la depuración, Eclipse nos pregunta si queremos cambiar al Entorno de depuración. Aceptamos sin problema. Ahí encontramos las opciones típicas de todo depurador: ejecución continua, detener, step into, step over, step return, etc.

Remote debugging from host 192.168.1.1

Hello from the PWM test program.

The starting values in the PWM registers are:

Periodo = 100000000

Ancho de pulso = 50000000

Enable bit = 1

Modo de uso:

El valor del ancho de pulso debe ser menor que el del periodo.

Para el Periodo y el ciclo de trabajo solo ingresar numeros enteros de hasta 4294967295.

El Enable bit solo puede ser de dos valores: 0 - PWM desactivado, 1 - PWM activ.

Ingrese valor de periodo: 2000

Ingrese valor de ancho de pulso: 1000

Note el LEDG8 pulsando en la tarjeta de desarrollo.

A medida que vamos ejecutando el programa, en la consola de μ Clinux se van imprimiendo los printf y nos solicita los datos en los scanf.

En la siguiente figura nos hacemos una idea más clara del depurador de Eclipse.

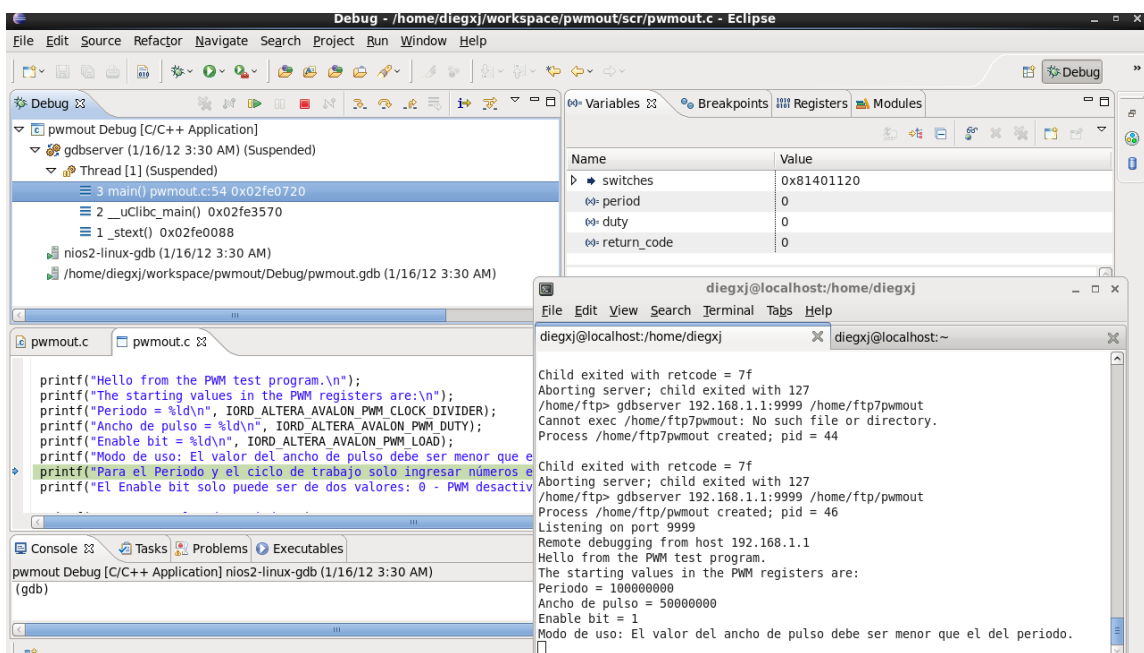


Figura 4.4. Vista conjunta del Depurador de Eclipse en el host y la consola del target ejecutando el gdbserver.

4.5 MONITOREO EN TIEMPO REAL DE SEÑALES CON SIGNALTAP II

Ruta de ejecución del Signal Tap II: Quartus II – Menú Tools – Signal Tap II Logic Analyzer

Debemos configurado lo siguiente en el Signal Tap:

- Señales que deseamos monitorear
- La imagen SOF con la que va a trabajar
- El hardware de comunicación a usar
- Condiciones de disparo (triggers)
- Señal Reloj de barrido

El Signal Tap II se agrega a nuestro diseño y consume parte de los recursos del FPGA. Dependiendo de las señales a medir necesita una señal de reloj de frecuencia igual a la del sistema o mayor (de preferencia en múltiplos enteros de la del sistema).

Ya que el Signal Tap II consume recursos del FPGA, una vez configurado hay que volver a compilar totalmente el proyecto de Quartus II.

Una vez cargado el SOF en el FPGA, dejamos al Signal Tap en un Autorun Analysis. Es preciso recalcar que el Signal Tap esperará hasta que la condición o las condiciones de disparo se cumplan para adquirir un dato. Si no hay seteada ninguna condición de disparo, el Signal Tap adquirirá todo y es probable que por la rapidez del sistema perdamos lo que deseamos.

Para hacer más gráfica la demostración del Signal Tap, vamos a capturar unas pantallas, utilizando la aplicación nios2io para enviar y recibir datos de los registros del PWM Avalon.

Primero se carga el módulo de pwm_avalon al kernel.

```
/>modprobe pwm_avalon
```

El dispositivo PWM Avalon ha sido registrado con 1 salida en LEDG8.

Lo conforman los registros CLOCK_DIV 0x81401130 DUTY 0x81401134 LOAD 0x81401138.

Registration is a success The pwmclockdiv major device number is 242.

Registration is a success The pwmduty major device number is 243.

Registration is a success The pwmload major device number is 244.

Espere unos segundos y observe el movimiento del LEDG8 en la tarjeta DE2.

Ha sido cargado inicialmente con los siguientes valores: Periodo 100000000 Duty: 50000000.

Eso nos da un periodo de 1 segundo aproximadamente a 100MHz del clock del sistema.

Módulo cargado exitosamente.

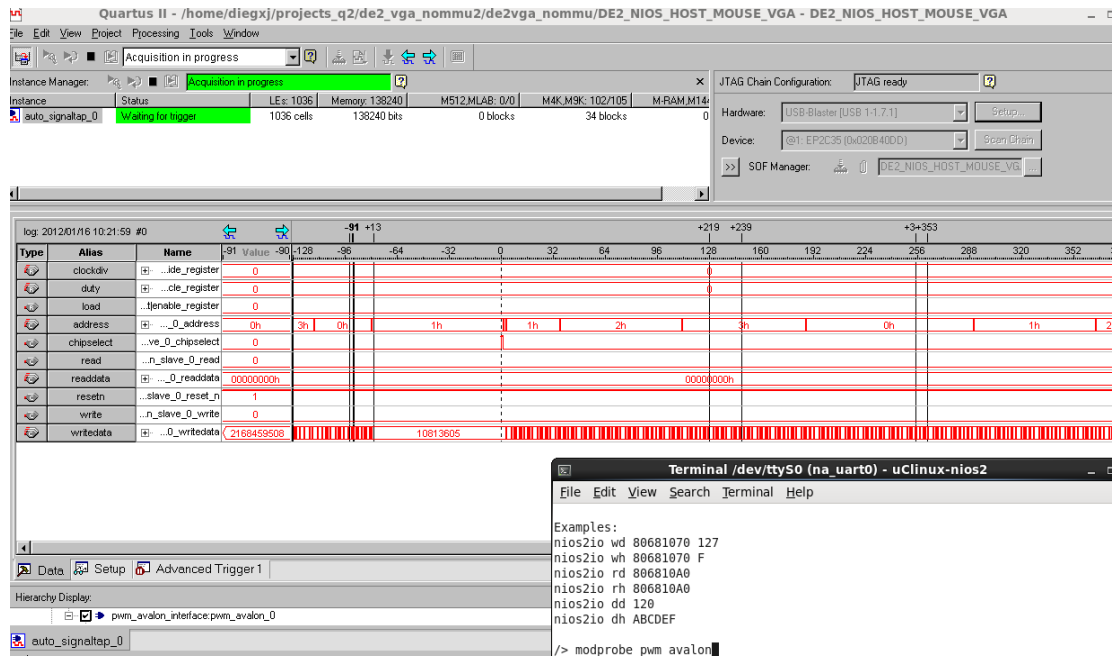


Figura 4.5 Estado de los registros clockdiv, duty y load antes de cargar el módulo `pwm_avalon`

En este estado el LEDG8 se encuentra apagado.

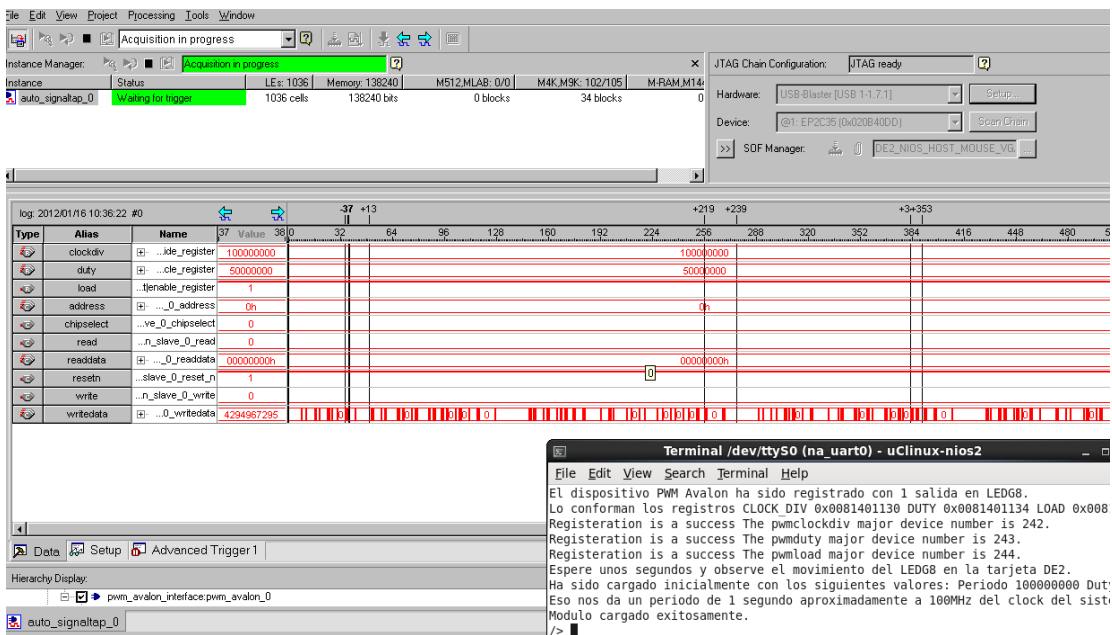


Figura 4.6 Estado de los registros clockdiv, duty y load luego de cargar el módulo `pwm_avalon`

Ahora el LEDG8 debería estar parpadeando con un período de 1 segundo y un ancho de pulso de 50%.

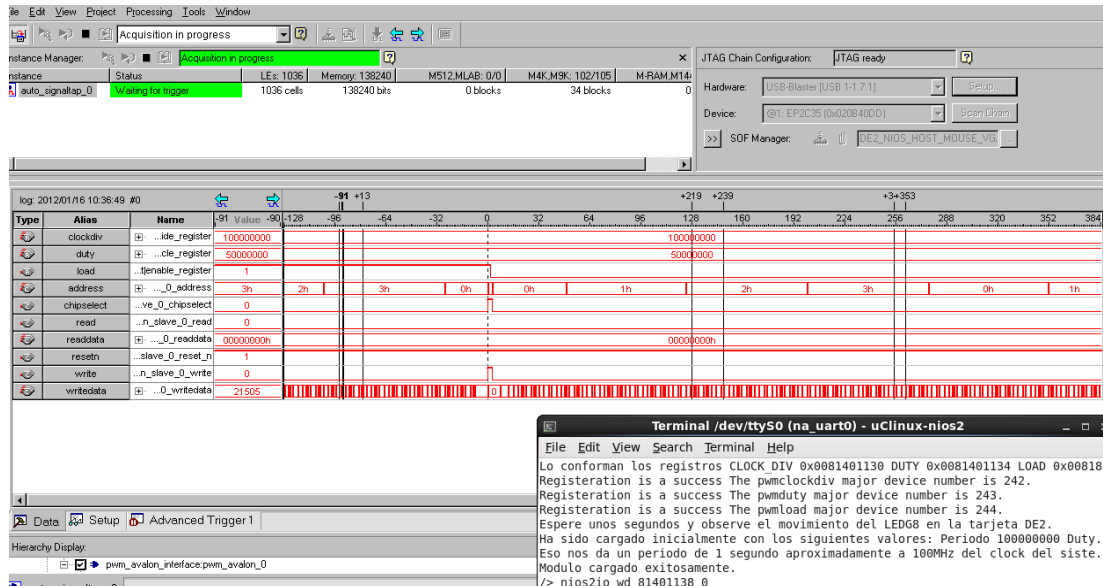


Figura 4.7 Estado de los registros clockdiv, duty y load luego de cargar 0 en la dirección del registro enable/load.

LEDG8 debería mostrarse apagado.

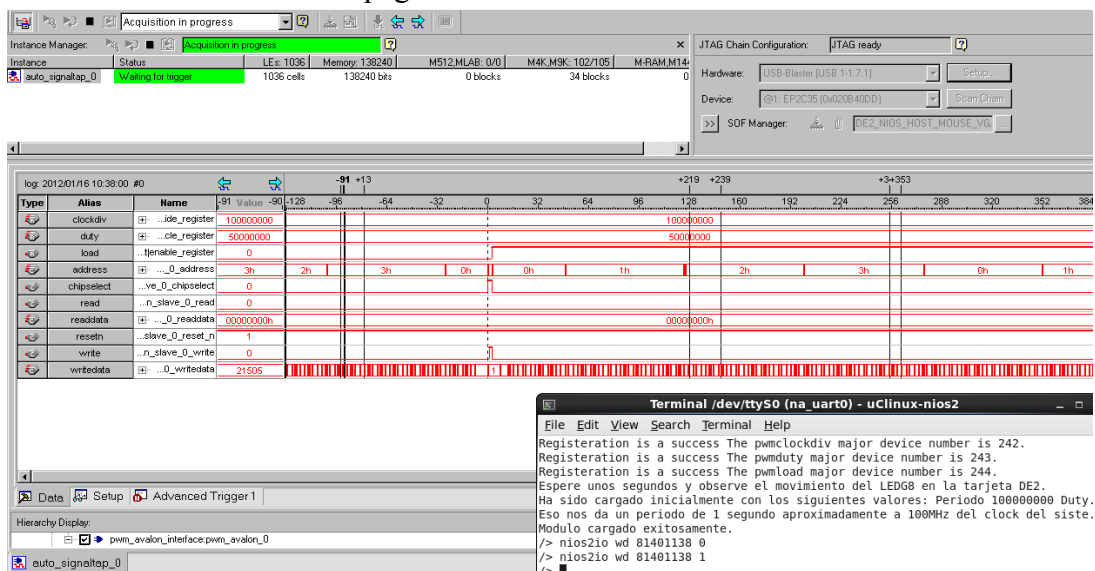


Figura 4.8 Estado de los registros clockdiv, duty y load luego de cargar 1 en la dirección del registro enable/load

LEDG8 debería volver a parpadear bajo los mismo parámetros anteriores.

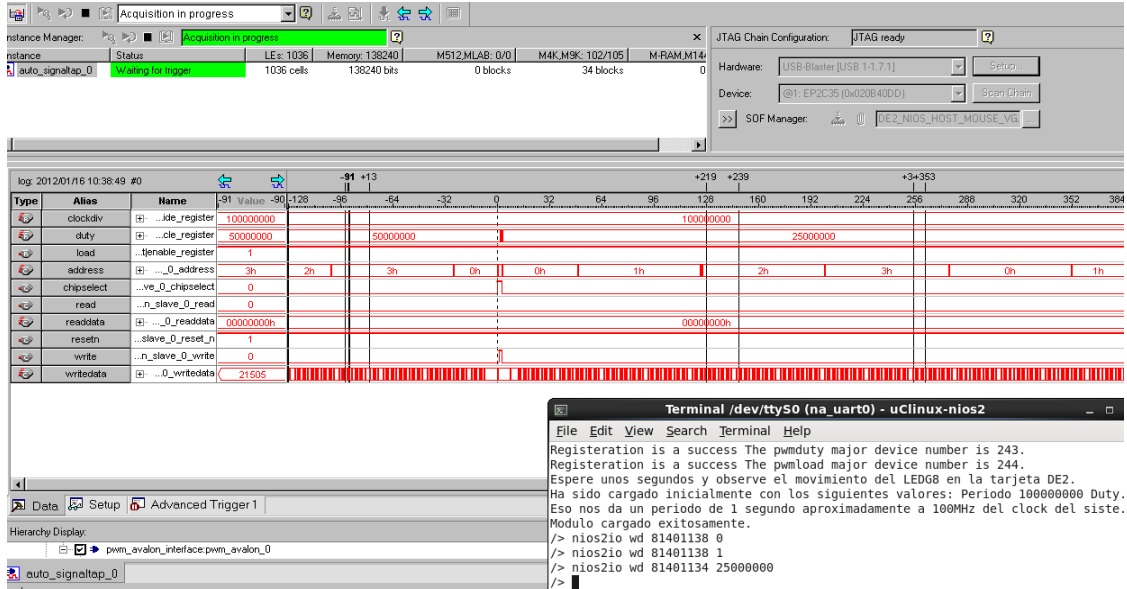


Figura 4.9 Estado de los registros clockdiv, duty y load luego de cargar 25000000 en la dirección del registro duty_cycle

El led LEDG8 debería parpadear más lento.

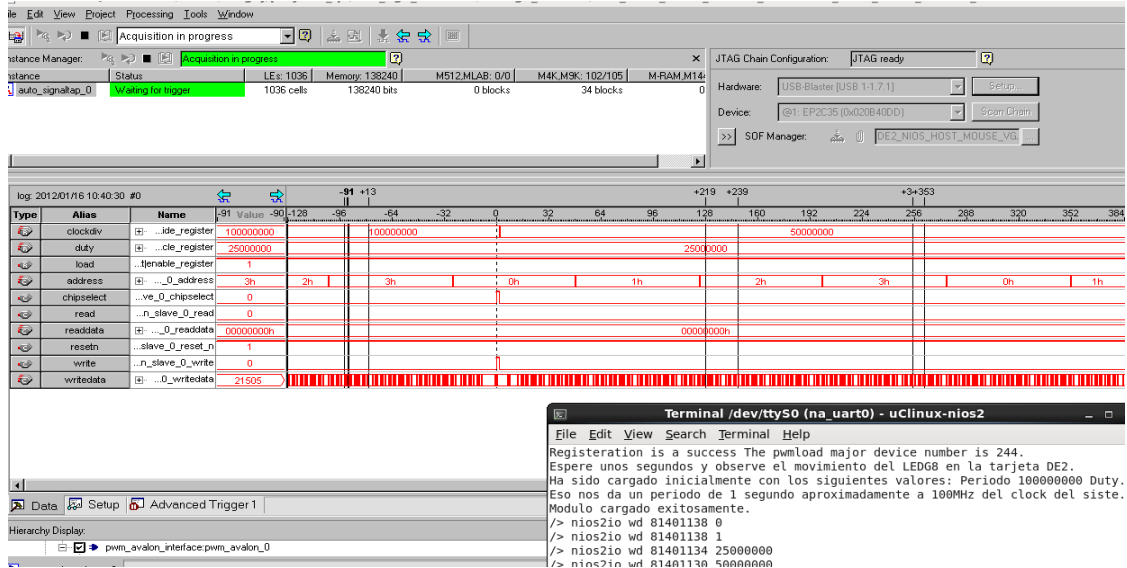


Figura 4.10 Estado de los registros clockdiv, duty y load luego de cargar 50000000 en la dirección del registro clock_divide

El LEDG8 debería parpadear con un periodo de 0.5 segundos con un ancho de pulso del 50%.

En las figuras 4.6 a 4.10 podemos notar las señales que intervienen de en la escritura de los datos en el registro.

La transferencia de escritura es de la siguiente manera:

- El procesador envía el dato a través del bus writedata de la interface Avalon-MM. Este dato le llega a todos los dispositivos conectados al bus.
- El procesador cuando esta listo para escribir en el registro del dispositivo, envia la señal write, la señal chipselect al dispositivo y la señal address.
- El hardware del dispositivo decodifica estas señales y dirige el dato del writedata hacia uno de sus registros. [Esta información puede ser verificada en el código HDL del dispositivo que esta disponible como anexo.]
- Como la transferencia es de latencia fija, el procesador no espera recibir ninguna señal de confirmación proveniente explícitamente del dispositivo.
- El hardware del dispositivo mantendrá el valor de la señal hasta que un nuevo valor sea enviado o el dispositivo sea reseteado.

4.6 LEER UNA TARJETA SD

Conectamos una tarjeta SD en la ranura de la tarjeta DE2. La compatibilidad del controlador Host SD Card con la especificación completa del hardware es limitada.

Puede que no lea todas las tarjetas que se le inserten. Lo hemos probado con tarjetas de hasta 2 GB con sistema de archivo FAT.

El hardware controlador de la tarjeta esta basado en un controlador SPI en lugar de un controlador MMC/SD/SDIO nativo. Este tiene la desventaja de ser relativamente pesado.

La tarjeta debe estar insertada en la ranura antes de cargar μ Clinux para que pueda ser detectada.

```
mmc_spi spi2.0: ASSUMING SPI bus stays unshared!
```

```
mmc_spi spi2.0: ASSUMING 3.2-3.4 V slot power
```

```
mmc_spi spi2.0: SD/MMC host mmc0, no DMA, no WP, no poweroff
```

```
.
```

```
.
```

```
.
```

```
mmc0: host does not support reading read-only switch. assuming write-enable.
```

```
mmc0: new SD card on SPI
```

```
mmcblk0: mmc0:0000 SU02G 1.84 GiB
```

```
mmcblk0: p1
```

μ Clinux carga el driver y luego detecta la tarjeta y le asigna un `/dev/mmcblk0`.

Para proceder a montar el sistema de archivo de la tarjeta en μ Clinux, ejecutamos

```
/mnt> mkdir sd
```



```
/mnt> ls
```

```
sd
```

```
/mnt> mount -t vfat /dev/mmcblk0p1 /mnt/sd
```

Si no hubo ningún mensaje de error entonces la tarjeta fue montada con éxito.

Para acceder a los archivos, vamos al directorio de montaje y ejecutamos el comando

ls.

```
/mnt/sd> ls
```

```
ANEXOS.odt
```

```
B2100.samsung
```

```
INDICE DE FIGURAS.docx
```

```
Images
```

```
Music
```

```
Other files
```

```
Sounds
```

```
TEMARIO_0201.docx
```

```
TEMARIO_0201.odt
```

```
TESIS DIEGO v2.docx
```

```
TESIS DIEGO v3.docx
```

```
TESIS DIEGO v4.docx
```

```
TESIS DIEGO v5.docx
```

TESIS DIEGO v6.docx

TESIS DIEGO v7.docx

TESIS DIEGO.docx

Videos

bookmarks-2012-01-10.json

codigo fuente

dcim

graficos por hacer

linux

misc

quit_2

tablas tesis.xls

tablas tesis.xlsx

tesisi

ultima tesis

CONCLUSIONES Y RECOMENDACIONES

Podemos concluir lo siguiente:

1. Para realizar la instalación y puesta a punto de uClinux en la tarjeta DE2 se tiene que revisar muy a fondo la documentación existente en el Altera Wiki ya que las fuentes están en constante actualización.
2. Se debe tener cuidado al añadir funciones del kernel y aplicaciones ya que puede que no estén soportadas todas. Por eso es importante unirse a la lista de mails de los desarrolladores y revisar el historial de cambios.

3. El acceso al hardware a través de un driver toma más tiempo que hacer por medio de funciones de acceso directo. El método de acceso a utilizar debe ser escogido considerando los requerimientos de la aplicación.
4. Programar el hardware y software de un sistema al mismo tiempo permite una mayor versatilidad de la aplicación y mayor flexibilidad al diseñador. Se puede llegar a un nivel de optimización mayor que con los métodos tradicionales.

Podemos recomendar lo siguiente:

1. Formar un proyecto para analizar a fondo como fue portado uClinux al Nios II.
2. Realizar la puesta en marcha de uClinux en una tarjeta similar a la DE2 pero con la licencia completa de Nios II para poder realizar un booteo directamente desde la tarjeta.
3. Actualizar las fuentes de uClinux-nios2 a la última versión y usar la última versión del Quartus II para obtener el archivo de configuración del hardware.

4. Realizar pruebas de benchmarking para comparar las diferentes configuraciones de Nios II con μ Clinux. Las pruebas también pueden ser entre μ Clinux y otros sistemas operativos para Nios II.
5. Se recomienda armar un BSP de μ Clinux para la tarjeta DE2 o cualquier otra tarjeta con aplicaciones para realizar prácticas y demostraciones a los alumnos de materias como Sistemas Operativos y Organización y arquitectura de computadoras.
6. Se recomienda tomar este trabajo como referencia para realizar la instalación en otras tarjetas con más recursos que la DE2 de Linux con MMU.
7. Se recomienda probar los parches para soft-real time en Linux que provee el proyecto RTEI4. Ambos proyectos están portados al Nios II. Son el proyecto PREEMPT-RT y el proyecto XENOMAI.
8. Se recomienda convertir a hardware el bucle infinito de la aplicación pwmout. Luego con medir el uso que hace la aplicación del CPU. Comparar ambos enfoques.

9. Probar el módulo Performance Counter Unit de SOPC Builder para realizar optimizaciones a aplicaciones que busquen reducir sus tiempos de ejecución y respuesta.

ANEXO A

**HERRAMIENTAS DE DESARROLLO PARA LA
CONFIGURACIÓN DEL PROCESADOR NIOS II**

Las herramientas que ayudan en el desarrollo de un dispositivo de hardware desde la especificación en un HDL (Hardware Description Language) hasta la evaluación con la medición de las señales de entrada y salida del dispositivo implementado en FPGA con un analizador lógico son las siguientes.

ALTERA QUARTUS II

El sistema de desarrollo Quartus II es una plataforma de herramientas para el diseño de circuitos digitales sobre dispositivos FPGA y CPLD de Altera. Quartus II provee aplicaciones para la entrada de diseño, síntesis lógica, simulación lógica, ubicación y conexionado, análisis temporal, administración de potencia y programación de dispositivos, junto con una variedad de utilitarios y aplicaciones adicionales para el diseño lógico programable.

- Entorno de desarrollo: Quartus II de Altera específicamente para :
 - Compilar: verificar sintaxis de la especificación.
 - Simular lógicamente: generar un netlist de compuertas independiente de la implementación, aceptar un test de prueba (señales de entrada del dispositivo) y generar las salidas que el dispositivo generaría de acuerdo con la especificación y asumiendo que las compuertas funcionan de manera "instantánea".
 - Grabar/Programar la FPGA: implementar sobre la FPGA el dispositivo que se ha especificado. Es necesaria, además, al menos una placa de desarrollo de Altera, que tenga una FPGA que sea posible de ser

grabada/programada. A partir de la grabación de la FPGA, ésta funciona como el dispositivo especificado.

SOPC BUILDER

El SOPC Builder es una herramienta utilizada en conjunción con el software CAD Quartus II. Que permite al usuario fácilmente crear un sistema basado en el procesador Nios II, con sólo seleccionar las unidades funcionales deseadas (IP Cores) y con indicación de sus parámetros al system_0. Para implementar el sistema, tenemos que crear instancias de las siguientes unidades funcionales:

Usamos SOPC Builder para diseñar el hardware requerido y así poder realizar las conexiones entre las interfaces paralelas, los interruptores, los LED que actúan como dispositivos I/O, los puertos generales y el procesador Nios II.

IP CORE

Un bloque o módulo de propiedad intelectual (IP), o simplemente un IP Core, es un módulo prediseñado que puede ser usado en otros diseños. Los IP Cores son para el diseño de hardware lo que las librerías son para la programación de computadoras. Pueden ayudar a reducir el tiempo de diseño de una solución ya que son comercialmente ofertados por Altera y por terceros (también los hay gratuitos pero no vienen con soporte técnico). Su beneficio se entiende en el hecho uno puede concentrarse en su hardware específico y obtener las otras partes al sistema de terceros.

Los IPCore propietarios no se pueden modificar, sólo parametrizar. Se venden con algún tipo de licenciamiento.

Su contraparte son los OpenCores Org que son de código abierto y con licencias de software libre, claro está el soporte proviene de una comunidad de Usuarios.

NIOS II EDS

La suite de desarrollo embebido de Nios II es una colección de herramientas de software, utilidades, librerías y controladores última tecnología que ayuda acelerar el tiempo de desarrollo de un diseño.

La suite incluye:

- Herramientas para construir software Nios II para Eclipse
- Herramientas para construir software Nios II
- Software embebido
- Controladores para dispositivos Altera IP Cores y HAL API
- Nios II IDE (para soporte de dispositivos discontinuado)

De estas herramientas la que nosotros vamos a usar es el Eclipse, aunque no necesariamente el Eclipse que viene en el Nios II EDS ya que la versión 9.1 para Linux presenta cierta incompatibilidad.

SIGNALTAP II

El analizador lógico embebido (ELA) Signal-Tap II es una herramienta de depuración a nivel de sistema que captura y muestra las señales en tiempo real en un diseño SOPC. Al usar SignalTap II ELA en sistemas generados por SOPC Builder, los diseñadores pueden observar el comportamiento del hardware (como registros de periféricos, buses de memoria, y otros componentes on-chip) en respuesta a la ejecución del software.

Se debe desactivar la opción de Compilación Incremental, en el menú Assignments - Settings - Compilation Process Settings - Incremental Compilation - Off.

En la etapa de síntesis de la lógica, Quartus II podría optimizar sacando señales que se quieren analizar con el SignalTap II.

Si esto ocurre, saldrá un error de compilación. Se puede forzar a Quartus II a preservar estas señales al añadir el atributo keep o preserve en el código fuente de HDL a las señales que se quiere monitorear. El atributo keep es usado para un nodo wire o net. Por ejemplo:

En Verilog:

```
wire my_wire /* synthesis keep = 1 */;
```

En VHDL:

```
signal my_signal: bit;
```

```
attribute syn_keep : boolean;
```

```
attribute syn_keep of my_signal: signal is true;
```

The preserve attribute is used for a register. For example:

El atributo preserve es usado para los registros. Por ejemplo:

En Verilog:

```
reg my_reg /* synthesis preserve = 1 */;
```

En VHDL:

```
signal my_reg: stdlogic;
```

```
attribute preserve : boolean;
```

```
attribute preserve of my_signal: signal is true;
```

Se puede encontrar información sobre el uso y configuración de esta herramienta en la web de Altera.

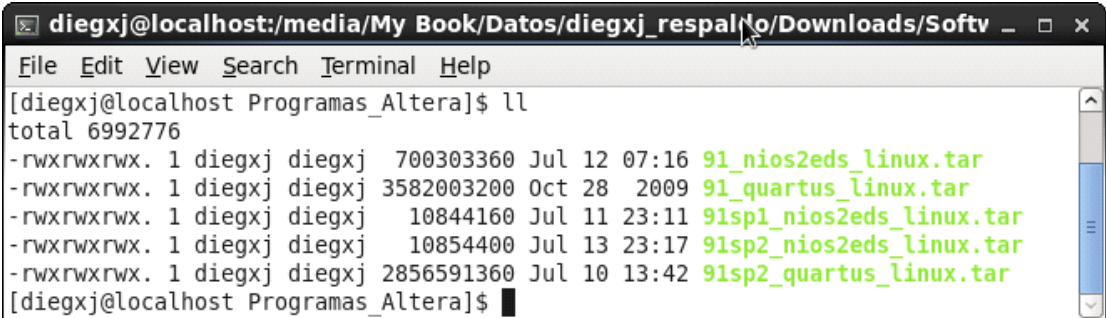
ANEXO B

INSTALACIÓN DEL ENTORNO DE DESARROLLO
PARA NIOS II EN SCIENTIFIC LINUX 6: QUARTUS II
/SOPC BUILDER Y NIOS II EDS

Para proceder a instalar Quartus II versión 9.1 Service Pack 2 primero descargamos las fuentes en la pagina web de Altera [<http://www.altera.com>]. Añadir actualización programas antes de instalar.

Al final de las descargas colocamos todos los archivos en un mismo directorio.

Deberíamos tener algo así:



```
diegxj@localhost:/media/My Book/Datos/diegxj_respaldo/Downloads/Softv
File Edit View Search Terminal Help
[diegxj@localhost Programas_Altera]$ ll
total 6992776
-rwxrwxrwx. 1 diegxj diegxj 700303360 Jul 12 07:16 91_nios2eds_linux.tar
-rwxrwxrwx. 1 diegxj diegxj 3582003200 Oct 28 2009 91_quartus_linux.tar
-rwxrwxrwx. 1 diegxj diegxj 10844160 Jul 11 23:11 91sp1_nios2eds_linux.tar
-rwxrwxrwx. 1 diegxj diegxj 10854400 Jul 13 23:17 91sp2_nios2eds_linux.tar
-rwxrwxrwx. 1 diegxj diegxj 2856591360 Jul 10 13:42 91sp2_quartus_linux.tar
[diegxj@localhost Programas_Altera]$
```

Figura B.1. Ventana de Archivos almacenados en un solo directorio.

Los paquetes de instalacion hacen referencia a la versión_nombre_sistema operativo.

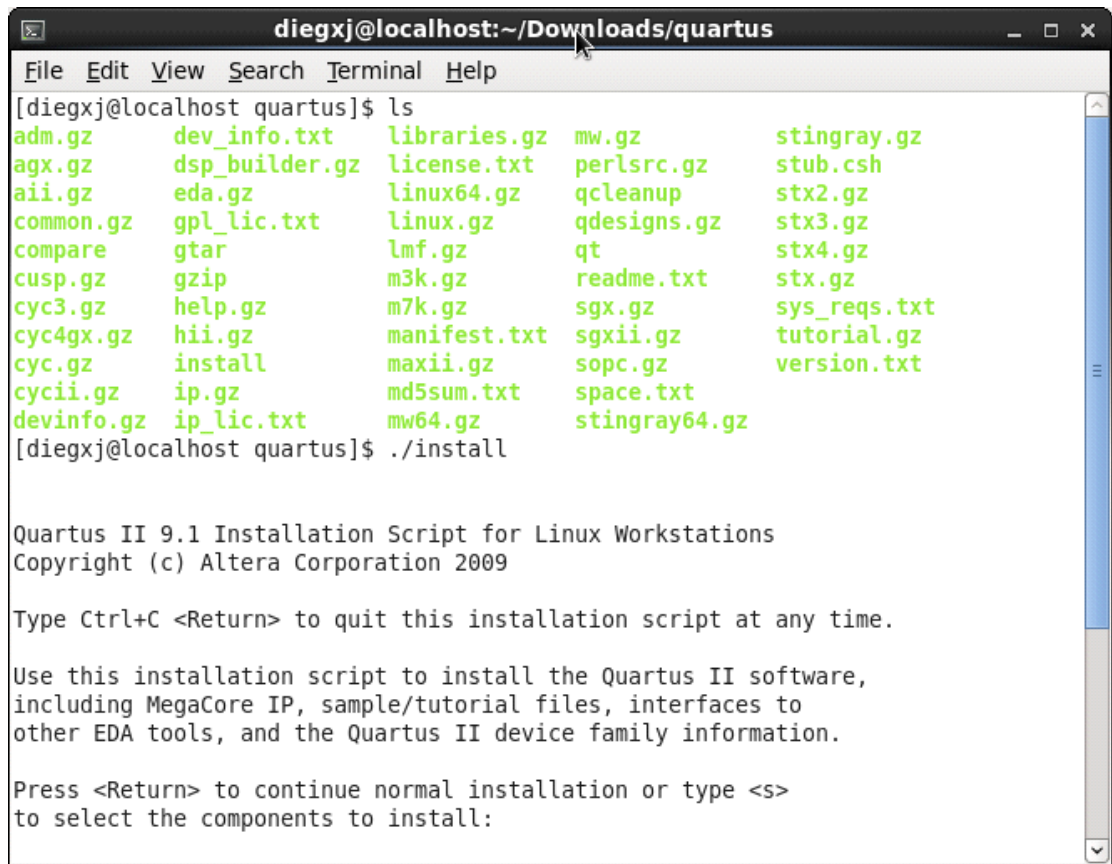
Los paquetes *nios2eds* contienen las fuentes para el entorno de desarrollo del Nios II.

Se procede a descomprimir todos los paquetes en el mismo directorio con los siguientes comandos:

```
tar -xvf 91_quartus_linux.tar
tar -xvf 91sp2_quartus_linux.tar
tar -xvf 91_nios2eds_linux_linux.tar
tar -xvf 91sp2_nios2eds_linux.tar
```

Empezamos con el instalador del directorio quartus. Iniciamos una sesion de la consola dentro del directorio quartus y ejecutamos el siguiente comando:

```
$ ./install
```



```
diegxj@localhost:~/Downloads/quartus
File Edit View Search Terminal Help
[diegxj@localhost quartus]$ ls
adm.gz      dev_info.txt  libraries.gz  mw.gz        stingray.gz
agx.gz      dsp_builder.gz license.txt   perlsrc.gz   stub.csh
aai.gz      eda.gz        linux64.gz   qcleanup     stx2.gz
common.gz   gpl_lic.txt   linux.gz     qdesigns.gz  stx3.gz
compare     gtar          lmf.gz       qt           stx4.gz
cusp.gz     gzip          m3k.gz       readme.txt   stx.gz
cyc3.gz     help.gz       m7k.gz       sgx.gz       sys_reqs.txt
cyc4gx.gz   hii.gz        manifest.txt  sgxii.gz    tutorial.gz
cyc.gz      install       maxii.gz     sopc.gz     version.txt
cycii.gz    ip.gz         md5sum.txt   space.txt
devinfo.gz  ip_lic.txt    mw64.gz      stingray64.gz
[diegxj@localhost quartus]$ ./install

Quartus II 9.1 Installation Script for Linux Workstations
Copyright (c) Altera Corporation 2009

Type Ctrl+C <Return> to quit this installation script at any time.

Use this installation script to install the Quartus II software,
including MegaCore IP, sample/tutorial files, interfaces to
other EDA tools, and the Quartus II device family information.

Press <Return> to continue normal installation or type <s>
to select the components to install:
```

Figura B.2. Ventana que muestra la instalación del directorio.

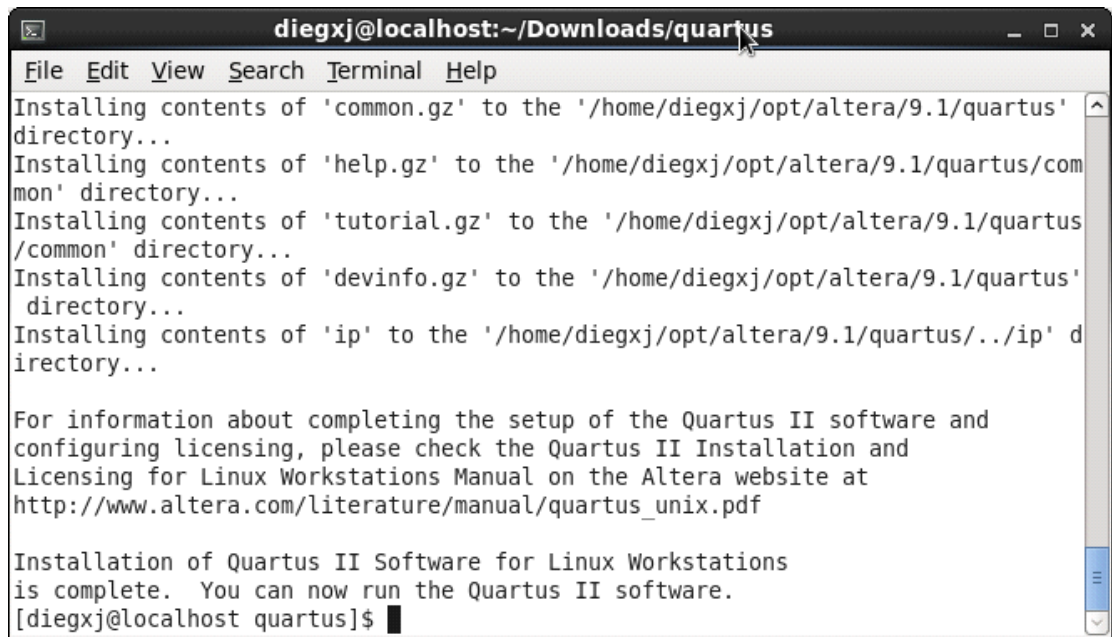
Damos <Enter> para proceder con una instalación completa. Cuando el instalador nos solicite el directorio de instalación escribimos:

```
(default: /opt/altera9.1): /home/diegxj/opt/altera/9.1
```

y damos <Enter>.

Una vez que el programa verifica que tenemos espacio en el disco, nos permite continuar con la instalación dando una vez mas<Enter>.

Al finalizar la instalación el programa nos invita a revisar la documentación de instalación y licenciamiento de Quartus en Linux [].



```
diegxj@localhost:~/Downloads/quartus
File Edit View Search Terminal Help
Installing contents of 'common.gz' to the '/home/diegxj/opt/altera/9.1/quartus'
directory...
Installing contents of 'help.gz' to the '/home/diegxj/opt/altera/9.1/quartus/com
mon' directory...
Installing contents of 'tutorial.gz' to the '/home/diegxj/opt/altera/9.1/quartus
/common' directory...
Installing contents of 'devinfo.gz' to the '/home/diegxj/opt/altera/9.1/quartus'
directory...
Installing contents of 'ip' to the '/home/diegxj/opt/altera/9.1/quartus/./ip' d
irectory...

For information about completing the setup of the Quartus II software and
configuring licensing, please check the Quartus II Installation and
Licensing for Linux Workstations Manual on the Altera website at
http://www.altera.com/literature/manual/quartus_unix.pdf

Installation of Quartus II Software for Linux Workstations
is complete. You can now run the Quartus II software.
[diegxj@localhost quartus]$
```

Figura B.3. Instalación Finalizada

De manera similar instalamos el Nios II EDS y los *service packs*.

Luego de instalados los programas, configuramos las variables de entorno y la licencia tal como indica el manual de instalación.

Esto debe ser agregado a las variables de entorno del usuario por medio del comando *source*.

Se puede crear un archivo de texto que contenga el texto arriba indicado y luego ejecutar el siguiente comando en la consola:

```
$ #Asumiendo que el archivo sea var_entorno_quartus2
```

```
$ source var_entorno_quartus2
```


Comprobamos que se hayan añadido las nuevas variables de entorno el siguiente comando:

```
$ export
```



The screenshot shows a terminal window titled "diegxj@localhost:~/scripts". The terminal displays a list of environment variables being declared with the 'declare -x' command. The variables include file extensions, mail directory, home directory, temporary directory, PATH, PWD, Qt directories, Quartus root directory, session manager, shell, shell level, Sopc builder path, Sopc kit path, SSH askpass, SSH auth socket, and terminal type.

```
File Edit View Search Terminal Help
:*.au=01;36:*.flac=01;36:*.mid=01;36:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=
01;36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:*.axa=01;36:*.oga=01;36:*.spx=01;36:*.x
spf=01;36:"
declare -x MAIL="/var/spool/mail/diegxj"
declare -x OLDPWD="/home/diegxj"
declare -x ORBIT_SOCKETDIR="/tmp/orbit-diegxj"
declare -x PATH="/usr/lib/qt-3.3/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbi
n:/usr/sbin:/sbin:/home/diegxj/bin:/home/diegxj/opt/altera/9.1/quartus/bin"
declare -x PWD="/home/diegxj/scripts"
declare -x QTDIR="/usr/lib/qt-3.3"
declare -x QTINC="/usr/lib/qt-3.3/include"
declare -x QTLIB="/usr/lib/qt-3.3/lib"
declare -x QT_IM_MODULE="xim"
declare -x QUARTUS_ROOTDIR="/home/diegxj/opt/altera/9.1/quartus"
declare -x SESSION_MANAGER="local/unix:@/tmp/.ICE-unix/1941,unix/unix:/tmp/.ICE-
unix/1941"
declare -x SHELL="/bin/bash"
declare -x SHLVL="2"
declare -x SOPC_BUILDER_PATH="/home/diegxj/opt/altera/9.1/nios2eds+/home/diegxj/
opt/altera/9.1/ip"
declare -x SOPC_KIT_NIOS2="/home/diegxj/opt/altera/9.1/nios2eds"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x SSH_AUTH_SOCK="/tmp/keyring-fYqGPx/socket.ssh"
declare -x TERM="xterm"
```

Figura B.4. Comprobación de nuevas variables de entorno añadidas.

El licenciamiento debe ser resuelto antes de empezar a desarrollar los proyectos. Se puede usar la versión WEB de Quartus II que incluye licenciamiento básico gratis. En nuestro caso usamos la licencia universitaria de la ESPOL.

Para esto hemos creado unos scripts útiles. [Ver Shell scripts útiles en el segundo anexo]

El archivo de la licencia debe estar almacenado en la siguiente ruta:

/usr/local/flexlm/licenses/license.dat.

El archivo de licencia de red empieza con las siguiente líneas:

```
SERVER <hostname><dirección MAC><puerto>
```

```
VENDOR alterad "<directorio raiz de Altera>/quartus/linux/alterad"
```

El hostname debe ser el la máquina, la dirección MAC de la máquina debe ser la misma de la licencia. El puerto puede ser obviado. El programa de registro de la licencia asignará un puerto libre.

El servidor de licenciamiento debe ser activado mediante los siguientes comandos:

```
$/lmgrd -c /usr/local/flexlm/licenses/license.dat -l  
log_license.txt
```

```
$/lmutil lmstat -a -c @localhost
```

Estos comandos deben ser ejeuctados antes de iniciar Quartus.

Para iniciar Quartus, ejecutamos el siguiente script en una sesión de terminal o directamente dando doble click en el archivo desde un administrador de archivos de entorno gráfico:

```
$ quartus_start91
```

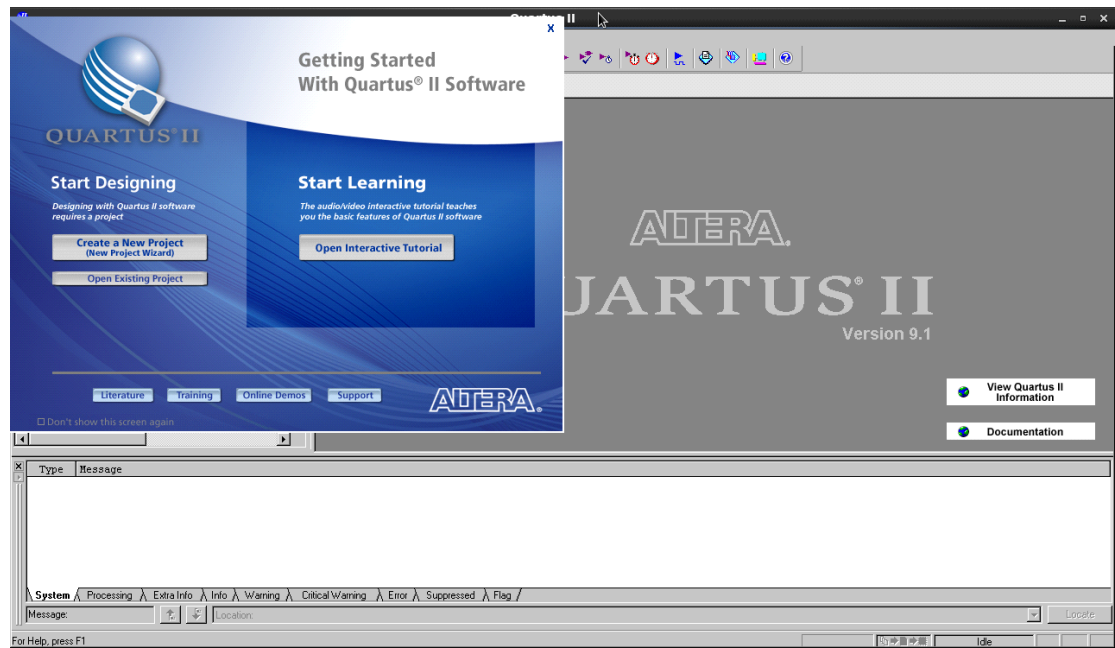


Figura B.5. Ventana Inicial para la ejecución de Quartus II

Para ejecutar el Nios II EDS corremos el script *nios2-shell*.

```
./nios2_shell
```

```
Nios II EDS 9.1
File Edit View Search Terminal Help
Version 9.1, Built Wed Mar 24 20:50:27 PDT 2010
-----
Welcome to the Nios II Embedded Design Suite
Version 9.1, Built Wed Mar 24 22:04:58 PDT 2010

Example designs can be found in
  /home/diegxj/opt/altera/9.1/nios2eds/examples

-----
(You may add a startup script: /home/diegxj/opt/altera/9.1/nios2eds/user.bashrc)

-----
Welcome to the Nios II Embedded Design Suite
Version 9.1, Built Wed Mar 24 22:04:58 PDT 2010

Example designs can be found in
  /home/diegxj/opt/altera/9.1/nios2eds/examples

-----
(You may add a startup script: /home/diegxj/opt/altera/9.1/nios2eds/user.bashrc)
~/opt/altera/9.1/nios2eds
[NiosII EDS]$
```

Figura B.6. NiosII EDS Shell

ANEXO C

INSTALACIÓN DE ECLIPSE IDE PARA LINUX

El programa puede ser descargado de <http://www.eclipse.org/downloads/>,

Eclipse IDE for C/C++ Linux Developers (Eclipse IDE para desarrolladores de C/C++ en Linux). Debe ser instalado de acuerdo a las instrucciones contenidas en el paquete.

Para nuestro proyecto hemos utilizado la versión de Eclipse IDE que viene pre-instalada en la distribución de Scientific Linux 6, que es la 3.6.1 y le descargamos las CDT (C development Tools).

ANEXO D

SCRIPTS CONFIGURACIÓN DE AMBIENTE

Los scripts están creados con la finalidad de setear correctamente las variables de entorno y la licencia teniendo varias versiones de Quartus II instaladas en un computador con Linux.

El archivo del shell script debe tener la propiedad de ejecutable marcada.

Shell script: script_set_env_quartus_91

```
PATH=$PATH:$HOME/opt/altera/9.1/quartus/bin
export PATH

LM_LICENSE_FILE=/usr/local/flexlm/licenses/license.dat
export LM_LICENSE_FILE

QUARTUS_ROOTDIR=/home/diegxj/opt/altera/9.1/quartus
export QUARTUS_ROOTDIR

SOPC_KIT_NIOS2=/home/diegxj/opt/altera/9.1/nios2eds
export SOPC_KIT_NIOS2

unset GCC_EXEC_PREFIX
```

Shell script: set_license_q2_91

```
cd /home/diegxj/opt/altera/9.1/quartus/linux

./lmutil lmdown

./lmgrd -c /usr/local/flexlm/licenses/license.dat -l log_license.txt

./lmutil lmstat -a -c @localhost

cd ~
```

Shell script: nios2shell

```
source /home/diegxj/Desktop/Tesis/scripts/script_set_env_quartus_91
```



```
export
```

```
bash /home/diegxj/Desktop/Tesis/scripts/set_license_q2_91
```

```
cd /home/diegxj/opt/altera/9.1/nios2eds/
```

```
./sdk_shell
```

Shell script: quartus_start

```
rm -rf /usr/local/flexlm/licenses/license.dat
```

```
ln -s /usr/local/flexlm/licenses/license91.dat  
/usr/local/flexlm/licenses/license.dat
```

```
source /home/diegxj/scripts/script_set_env_quartus_91
```

```
export
```

```
bash -xv /home/diegxj/scripts/set_license_q2_91
```

```
cd /home/diegxj/opt/altera/9.1/quartus/bin
```

```
./quartus
```

ANEXO E

CONFIGURACIÓN DEL JTAG EN LINUX

El puerto jtag USB Blaster necesita usbfs para funcionar.

Como root hay que añadir o editar la siguiente línea en el archivo /etc/fstab

```
usbfs /proc/bus/usb usbfs devmode=0666 0 0
```

Esto montará automáticamente el usbfs en la localización /proc/bus/usb durante el inicio de Linux.

El modo se seteará en 0666 lo que significa que cualquiera puede leer y escribir los archivos que son creados ahí.

Reiniciamos el sistema o montamos el sistema de archivos con el comando:

Como root

```
# mount /proc/bus/usb
```

Luego preparamos el jtag server. Como root

```
#mkdir/etc/jtagd cp  
$HOME/opt/altera/9.1/quartus/linux/pgm_parts.txt  
/etc/jtagd/jtagd.pgm_parts
```

Si el directorio /etc/jtag no existe, lo creamos.

Añadir estas líneas a el archivo /etc/rc.local para iniciar el jtag server. Como root

```
echo 356 40000 32 32000 > /proc/sys/kernel/sem  
/home/diegxj/opt/altera/9.1/quartus/bin/jtagd
```

Listo. Antes de continuar es recomendable reiniciar el computador.

ANEXO F

CODIGO FUENTE

pwm_avalon_interface.v

pwm_register_files.v

pwm_task_logic.v

pwm_avalon.c

seven_seg_pio.c

ANEXO G

PROGRAMANDO UN MÓDULO PARA EL KERNEL

BIBLIOGRAFÍA

1. Hamblen-Tyson-Furman, Rapid prototyping of digital systems SOPC Edition, Springer, Primera edición, 2008
2. Michael Barr, Programming Embedded Systems in C and C++, O'Reilly, Primera edición, Enero 1999
3. Doug Abbott, Newnes, Linux for Embedded and Real-time Applications, Primera edición, 2003
4. Nios II Core Implementation Details, Nios II Processor Reference Handbook, Altera Corporation, Mayo 2011
5. Processor Architecture, Nios II Processor Reference Handbook, Altera Corporation, Mayo 2011

6. Cache and Tightly-Coupled Memory, Nios II Software Developer's Handbook, Altera Corporation, Mayo 2011
7. Bhavya Daya, RAPID PROTOTYPING OF EMBEDDED SYSTEMS USING FIELD PROGRAMMABLE GATE ARRAYS, University of Florida, Spring, 2009
8. Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, LINUX DEVICE DRIVERS, O'REILLY, Tercera Edición, 2005
9. Avalon Interface Specifications , Altera Corporation, Mayo 2011
10. SOPC Builder Design Optimizations, Embedded Design Handbook, Altera Corporation, Julio 2011
11. Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems , Application Note 323, versión 1.1, Altera Corporation , Noviembre 2007
12. Microtronix v1.4 μ Clinux Development,
http://www.alterawiki.com/wiki/Microtronix_v1.4_\muClinux_Development
13. Mc Cullough, David, μ Clinux for Linux programmers.
www.linuxjournal.com/article.php?sid=7221, Julio 2004

14. The Linux Kernel Archives, <http://www.kernel.org/>

15. Embedded Linux for the Nios II Processor,
<http://www.altera.com/devices/processor/nios2/tools/embed-partners/ni2-linux-partners.html>

16. Welcome to the NIOS II PREEMPT-RT Project,
<http://uuu.enseirb.fr/~kadionik/nios2-preempt-rt/>

17. A C library for embedded Linux, <http://uclibc.org/about.html>

18. Buildroot: making Embedded Linux easy, <http://buildroot.uclibc.org/>

19. Building embedded Linux systems with Buildroot,
<http://2009.rml.info/IMG/pdf/buildroot-RMLL09.pdf>

20. GNU Binutils, <http://www.gnu.org/software/binutils/>

21. Working with Kconfig, <http://www.rt-embedded.com/blog/archives/working-with-kconfig/>

22. Programmer united development net, <http://en.pudn.com>

23. Opencores, <http://www.opencores.org>

24. UCLinux, <http://www.alterawiki.com/wiki/UCLinux>

25. ARM Technical Support Knowledge Articles, Placing C variables at specific addresses to access memory-mapped peripherals,
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka3750.html>, 2011-09-20

```

/*****
/*      Copyright © 2004 Altera Corporation. All rights reserved.      */
/* Altera products are protected under numerous U.S. and foreign patents, */
/* maskwork rights, copyrights and other intellectual property laws. This */
/* reference design file, and your use thereof, *is subject to and */
/* governed by the terms and conditions of the applicable Altera Reference */
/* Design License Agreement (either as signed by you or found at */
/* www.altera.com). By using this reference design file, you indicate */
/* your acceptance of such terms and conditions between you and Altera */
/* Corporation. In the event that you do not agree with such terms and */
/* conditions, you may not use the reference design file and please */
/* promptly destroy any copies you have made. This reference design file */
/* is being provided on an "as-is" basis and as an accommodation and */
/* therefore all warranties, representations or guarantees of any kind */
/* (whether express, implied or statutory) including, without limitation, */
/* warranties of merchantability, non-infringement, or fitness for a */
/* particular purpose, are specifically disclaimed. By making this */
/* reference design file available, Altera expressly does not recommend, */
/* suggest or require that this reference design file be used in */
/* combination with any other product not provided by Altera. */
*****/

/*****
/* File: pwm_avalon_interface.v */
/* Description: Top level module. Instantiates pwm_task_logic and */
/*      pwm_register_file modules and adds Avalon slave interface. */
*****/
/*COMENTARIO TRADUCIDOS POR DIEGO JAUREGUI Y SARA POVEDA*/
module pwm_avalon_interface
(
    clk,
    resetn,
    avalon_chip_select,
    address,
    write,
    write_data,
    read,
    read_data,
    pwm_out
);

//Valores de los parámetros para pasar a la instancia pwm_register_file
parameter clock_divide_reg_init = 32'h0000_0000;
parameter duty_cycle_reg_init = 32'h0000_0000;

//Avalon_Slave_PWM Avalon I/O
input clk; //Clock del sistema- vinculado a todos los bloques
input resetn; //Reset del sistema - Vinculado a todos los bloques
input avalon_chip_select; //Avalon Chip select
input [1:0]address; //Bus de direccionamiento Avalon
input write; //Señales de escritura Avalon
input [31:0]write_data; //Bus de escritura de Datos Avalon
input read; //Señal de escritura Avalon

output [31:0]read_data; //Bus de lectura de datos Avalon

//Avalon_Slave_PWM exportados de E / S
output pwm_out; //Señal de salida PWM

//Avalon_Slave_PWM Nodos Internos
wire [31:0] pwm_clock_divide; // Señal del Clock divide desde el registro hacia el bloque lógico pwm
wire [31:0] pwm_duty_cycle; //Señal del duty_cycle desde el registro hacia el bloque lógico pwm
wire pwm_enable; //Señal de enable del PWM desde el registro hacia el bloque lógico pwm

//Instancias PWM
pwm_task_logic task_logic
(

```

```
.clk          (clk ),
.pwm_enable   (pwm_enable),
.resetn       (resetn),
.clock_divide (pwm_clock_divide),
.duty_cycle   (pwm_duty_cycle),
.pwm_out      (pwm_out)
);

//Instancia del registro de archivos
pwm_register_file #(clock_divide_reg_init, duty_cycle_reg_init) memory_element
(
    .clk          (clk),
    .resetn       (resetn),
    .chip_select  (avalon_chip_select),
    .address      (address),
    .write        (write),
    .write_data   (write_data),
    .read         (read),
    .read_data    (read_data),
    .pwm_clock_divide (pwm_clock_divide),
    .pwm_duty_cycle (pwm_duty_cycle),
    .pwm_enable   (pwm_enable)
);

endmodule
```

```

/*****
/*      Copyright © 2004 Altera Corporation. All rights reserved.      */
/* Altera products are protected under numerous U.S. and foreign patents, */
/* maskwork rights, copyrights and other intellectual property laws. This */
/* reference design file, and your use thereof, *is subject to and */
/* governed by the terms and conditions of the applicable Altera Reference */
/* Design License Agreement (either as signed by you or found at */
/* www.altera.com). By using this reference design file, you indicate */
/* your acceptance of such terms and conditions between you and Altera */
/* Corporation. In the event that you do not agree with such terms and */
/* conditions, you may not use the reference design file and please */
/* promptly destroy any copies you have made. This reference design file */
/* is being provided on an "as-is" basis and as an accommodation and */
/* therefore all warranties, representations or guarantees of any kind */
/* (whether express, implied or statutory) including, without limitation, */
/* warranties of merchantability, non-infringement, or fitness for a */
/* particular purpose, are specifically disclaimed. By making this */
/* reference design file available, Altera expressly does not recommend, */
/* suggest or require that this reference design file be used in */
/* combination with any other product not provided by Altera. */
*****/

/*****
/* File: pwm_register_file */
/* Description: Register interface for PWM. Contains logic for reading */
/* and writing to PWM registers. */
*****/

module pwm_register_file
(
    //Señales Avalon
    clk,
    resetn,
    chip_select,
    address,
    write,
    write_data,
    read,
    read_data,

    //Señales de salida de PWM
    pwm_clock_divide,
    pwm_duty_cycle,
    pwm_enable
);

//Parametros
parameter clock_divide_reg_init = 32'h0000_0000;
parameter duty_cycle_reg_init   = 32'h0000_0000;

//Entradas

input clk; //Clock del sistema- vinculado a todos los bloques
input resetn; //Reset del sistema - Vinculado a todos los bloques
input chip_select; //Avalon Chip select
input [1:0] address; //Bus de direccionamiento Avalon
input write; //Señales de escritura Avalon
input [31:0] write_data; //Bus de escritura de Datos Avalon
input read; //Señal de escritura Avalon

//Outputs
output [31:0] read_data; //Avalon read data bus
output [31:0] pwm_clock_divide; //PWM clock divide drive signals
output [31:0] pwm_duty_cycle; //PWM duty cycle drive signals
output pwm_enable; //PWM enable drive signals

//Signal Declarations

```

```

reg [31:0] clock_divide_register;      //Clock divider register
reg [31:0] duty_cycle_register;       //Duty Cycle Register
reg      enable_register;             //Enable Bit
reg [31:0] read_data;                 //Read_data bus

//Los nodos utilizados para decodificación de direcciones
wire clock_divide_reg_selected, duty_cycle_reg_selected, enable_reg_selected;
//Nodos para determinar si una escritura válida ocurrió a una dirección específica
wire write_to_clock_divide, write_to_duty_cycle, write_to_enable;
//Nodos para determinar si una lectura válida ocurrió a una dirección específica
wire read_to_clock_divide, read_to_duty_cycle, read_to_enable;
//Los nodos utilizados para determinar si un acceso válido se ha producido
wire valid_write, valid_read;

//Inicio Código Principal

//address decode
assign clock_divide_reg_selected = !address[1] & !address[0]; //address 00
assign duty_cycle_reg_selected  = !address[1] & address[0];  //address 01
assign enable_reg_selected      = address[1] & !address[0];   //address 10

//determinar si una transacción valida se inició
assign valid_write = chip_select & write;
assign valid_read  = chip_select & read;

//determinar si una operación de escritura se produjo a una dirección específica
assign write_to_clock_divide = valid_write & clock_divide_reg_selected;
assign write_to_duty_cycle   = valid_write & duty_cycle_reg_selected;
assign write_to_enable       = valid_write & enable_reg_selected;

//determinar si se produjo una lectura a una dirección específica
assign read_to_clock_divide = valid_read & clock_divide_reg_selected;
assign read_to_duty_cycle   = valid_read & duty_cycle_reg_selected;
assign read_to_enable       = valid_read & enable_reg_selected;

//Escritura al Registro clock_divide
always@(posedge clk or negedge resetn)
begin
    if(~resetn)begin // Reset Asíncronico
        clock_divide_register <= clock_divide_reg_init; //32'h0000_0000;
    end
    else begin
        if(write_to_clock_divide) begin
            clock_divide_register <= write_data;
        end
        else begin
            clock_divide_register <= clock_divide_register;
        end
    end
end

//Escritura al Registro duty_cycle
always@(posedge clk or negedge resetn)
begin
    if(~resetn)begin //Reset Asíncronico
        duty_cycle_register <= duty_cycle_reg_init; //32'h0000_0000;
    end
    else begin
        if(write_to_duty_cycle) begin
            duty_cycle_register <= write_data;
        end
        else begin
            duty_cycle_register <= duty_cycle_register;
        end
    end
end
end

```

```
//Escritura para habilitar registro
always@(posedge clk or negedge resetn)
begin
    if(~resetn)begin //Reset Asincrónico
        enable_register <= 1'b0;
    end
    else begin
        if(write_to_enable)begin
            enable_register <= write_data[0];
        end
        else begin
            enable_register <= enable_register;
        end
    end
end

//Lectura del Bus de datos
always@(read_to_clock_divide or read_to_duty_cycle or read_to_enable or clock_divide_register or
duty_cycle_register or enable_register)
begin
    if(read_to_clock_divide) begin
        read_data = clock_divide_register;
    end
    else if(read_to_duty_cycle) begin
        read_data = duty_cycle_register;
    end
    else if(read_to_enable) begin
        read_data = {31'd0,enable_register};
    end
    else begin
        read_data = 32'h0000_0000;
    end
end

//asignar valores de registro para registrar salidas de archivo en el PWM
assign pwm_clock_divide = clock_divide_register;
assign pwm_duty_cycle = duty_cycle_register;
assign pwm_enable = enable_register;

endmodule
```

```

/*****
/*      Copyright © 2004 Altera Corporation. All rights reserved.      */
/* Altera products are protected under numerous U.S. and foreign patents, */
/* maskwork rights, copyrights and other intellectual property laws. This */
/* reference design file, and your use thereof, *is subject to and      */
/* governed by the terms and conditions of the applicable Altera Reference */
/* Design License Agreement (either as signed by you or found at      */
/* www.altera.com). By using this reference design file, you indicate   */
/* your acceptance of such terms and conditions between you and Altera  */
/* Corporation. In the event that you do not agree with such terms and   */
/* conditions, you may not use the reference design file and please     */
/* promptly destroy any copies you have made. This reference design file */
/* is being provided on an "as-is" basis and as an accommodation and     */
/* therefore all warranties, representations or guarantees of any kind   */
/* (whether express, implied or statutory) including, without limitation, */
/* warranties of merchantability, non-infringement, or fitness for a    */
/* particular purpose, are specifically disclaimed. By making this     */
/* reference design file available, Altera expressly does not recommend, */
/* suggest or require that this reference design file be used in       */
/* combination with any other product not provided by Altera.         */
*****/

/*****
/* File: pwm_task_logic.v                                             */
/* Description: This module contains the core of the pwm functionality. */
/* The clock_divide and duty_cycle inputs are used in conjunction with  */
/* a counter to determine how long the pwm output stays high and low.   */
/* The output is 1 bit.                                               */
*****/

module pwm_task_logic
(
    clk,
    pwm_enable,
    resetn,
    clock_divide,
    duty_cycle,
    pwm_out
);

//Inputs
input clk; //Entrada de Clock para ser dividida
input [31:0] clock_divide; //Clock Divide value
input [31:0] duty_cycle; //Duty Cycle value
input pwm_enable; //Señal Enable
input resetn; //Reset

//Salidas
output pwm_out; //Salidas PWM

//Declaración de Señales
reg [31:0] counter; // Contador interno PWM
reg pwm_out; // Salida PWM

//Inicio del Programa Principal
always @(posedge clk or negedge resetn) //Contador de Procesos PWM
begin
    if (~resetn)begin
        counter <= 0;
    end
    else if(pwm_enable)begin
        if (counter >= clock_divide)begin
            counter <= 0;
        end
        else begin
            counter <= counter + 1;
        end
    end
end

```

```
        end
    else begin
        counter <= counter;
    end
end

always @(posedge clk or negedge resetn) // Comparador PWM

begin
    if (~resetn)begin
        pwm_out <= 0;
    end
    else if(pwm_enable)begin
        if (counter >= duty_cycle)begin
            pwm_out <= 1'b1;
        end
        else begin
            if (counter == 0)
                pwm_out <= 0;
            else
                pwm_out <= pwm_out;
            end
        end
    else begin
        pwm_out <= 1'b0;
    end
end

endmodule
```



```
/* Copyright (C) 2011 Diego Jáuregui, Sara Rafaela Poveda Luna, Ronald Ponguillo */
/* This program is free software; you can redistribute it and/or modify */
/* it under the terms of the GNU General Public License as published by */
/* the Free Software Foundation; either version 2 of the License, or */
/* (at your option) any later version. */
/* This program is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. */
/* GNU General Public License for more details. */

/* uncomment following line to use the kernelmodule without */
/* io (inl / outb) functions, needed for debug purposes on host */
/* machine. */

// #define DEBUG
#ifdef DEBUG
#define DEBUG_PRINT(string)      printk(KERN_ALERT "===pwm_avalon.c:%3d: %s\n", __LINE__, string)
#define DEBUG_FUNC_ENTER()      printk(KERN_ALERT "===pwm_avalon.c:%3d: ENTER %s()\n", __LINE__,
    __FUNCTION__)
#define DEBUG_FUNC_EXIT()       printk(KERN_ALERT "===pwm_avalon.c:%3d: EXIT %s()\n", __LINE__,
    __FUNCTION__)
#else
#define DEBUG_PRINT(string)
#define DEBUG_FUNC_ENTER()
#define DEBUG_FUNC_EXIT()
#endif

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/ioctl.h>
#include <linux/ioport.h>
#include <linux/types.h>
#include <linux/delay.h>

/* números importantes para la identificación de los ficheros de dispositivo */
/* mknod /dev/pwmclockdiv c 242 0 */
#define CLOCK_DIV_MAJOR 242
/* mknod /dev/pwmduty c 243 0 */
#define DUTY_MAJOR 243
/* mknod /dev/pwmload c 244 0 */
#define LOAD_MAJOR 244

// na_pwm_avalon_0 0x01401130
/* Direcciones de registros tomados desde la dirección
PWM_AVALON_BASE desplazado cada 4 bits */
#define PWM_AVALON_BASE      na_pwm_avalon_0 + 0x80000000 // Bit 31 = 1 para bypassar el cache de
datos del Nios II/f
#define CLOCK_DIV_REGISTER   PWM_AVALON_BASE + 0
#define DUTY_REGISTER        PWM_AVALON_BASE + 4
#define LOAD_REGISTER        PWM_AVALON_BASE + 8

// Longitud de la cadena buffer
#define BUFFER_LEN 11

/* Definiendo buffers para las cadenas enviadas por el usuario */
char clkdiv_buf[BUFFER_LEN]="0\n";
char duty_buf[BUFFER_LEN]="0\n";
char load_buf[BUFFER_LEN]="0\n";
char* clkdiv_ptr=clkdiv_buf;
char* duty_ptr=duty_buf;
```

```

char* load_ptr=load_buf;

/* Variables para definicion de apertura de archivo */

int duty_is_open = 0;
int clkdiv_is_open = 0;
int load_is_open = 0;

/* Convierte string a Entero */
static long atoi(const char *name)
{
    long val = 0;
    for (;;)name++
    {
        switch (*name)
        {
            case '0'...'9':
                val = 10 * val + (*name - '0');
                break;
            default:
                return val;
        }
    }
}

/* COnvierte byte a string */

void itoa(char *name, unsigned long value)
{
    long i,j=0;
    for(i=100000000;i>0;i/=10)
    {
        name[j++] = '0' + value/i;
        value = value - (value/i) * i;
        if(name[0]=='0' && i!=1)
            j=0;
    }
    name[j++]='\n';
    name[j]=0;
}

////////////////////////////////////
// Dispositivo controlador de archivo abierto
////////////////////////////////////

/* Funcion llamada cuando el archivo de dispositivo clock_divisor es abierto */

static int clkdiv_open(struct inode *inode, struct file *file)
{
    unsigned long value;
    DEBUG_FUNC_ENTER(); ] //Imprimiendo mensaje en el kernel
    if (clkdiv_is_open++) { // Valida que no haya una instancia del archivo de dispositivo
        printk(KERN_ALERT "Device busy...\n");
        return -EBUSY;
    }
    /* capturar la nueva información de registro de clock_divisor */
    #ifndef DEBUG
    value=inl(CLOCK_DIV_REGISTER); //Acceso a puerto de 32 bits para obtener valor
    itoa(clkdiv_buf,value);
    #endif
    clkdiv_ptr=clkdiv_buf;
    try_module_get(THIS_MODULE); // Función que remueve módulo.
    return 0;
}

////////////////////////////////////
// Dispositivo controlador de archivo

```

```
////////////////////////////////////
/* Llamado cuando archivo de dispositivo clock_divisor es cerrado */
static int clkdiv_release(struct inode *inode, struct file *file)
{
    DEBUG_FUNC_ENTER();
    --clkdiv_is_open;
    module_put(THIS_MODULE); // función encargada de cerrar módulo
    DEBUG_FUNC_EXIT();
    return 0;
}

////////////////////////////////////
// Controlador de dispositivo en lectura
////////////////////////////////////

/* Llamado cuando archivo de dispositivo clock_divisor esta siendo leído*/
static ssize_t clkdiv_read(struct file *file, char *buffer, size_t length, loff_t *offset)
{
    /* Numero de bytes actualmente escritos en el buffer */
    int bytes_read = 0;
    /* Si esta en el final del mensaje, retorna 0 significa final del archivo */
    if (*clkdiv_ptr == 0)
        return 0;
    while (length-- && *clkdiv_ptr)
    {
        /* The buffer is in the user clock_divisor segment, not the kernel */
        /* segment so "*" assignment won't work. */
        /* put user which copies data from the kernel data segment to */
        /* the user data segment. */
        put_user(*(clkdiv_ptr++), buffer++);
        bytes_read++;
    }
    /* retorna contado de bytes, enivados al usuario */
    return bytes_read;
}

/* Controlador de dispositivo en escritura*/
static ssize_t clkdiv_write(struct file *file, const char *buff, size_t len, loff_t *off)
{
    int not_copied=0;
    if(len > (BUFFER_LEN-1))
        len=BUFFER_LEN-1;
    /* copiar datos del usuario dentro del buffer de cadena clkdiv*/
    not_copied=copy_from_user(clkdiv_buf, buff, len);
    clkdiv_buf[len]='\0';

    /* Salida el valor numérico del buffer clkdiv */
    /* A la dirección del registro */
    #ifndef DEBUG
    outl((unsigned long) atoi(clkdiv_buf),CLOCK_DIV_REGISTER);
    #endif
    return len-not_copied;
}

/* llamado cuando archivo de dispositivo duty_cycle se abre*/
static int duty_open(struct inode *inode, struct file *file)
{
    unsigned long value;

    if (duty_is_open++) // Valido que archivo de dispositivo no este en uso
        return -EBUSY;
    /* capturar la nueva información del registro de duty_cycle */
}
```

```

    #ifndef DEBUG
    value=inl(DUTY_REGISTER);
    itoa(duty_buf,value);
    #endif
    duty_ptr=duty_buf;

    try_module_get(THIS_MODULE);// Función que remueve módulo.
    return 0;
}

/* llamado cuando archivo de dispositivo duty_cycle se cierra */
static int duty_release(struct inode *inode, struct file *file)
{
    --duty_is_open;
    module_put(THIS_MODULE); // Función encargada de cerrar el módulo
    return 0;
}

/* llamado cuando archivo de dispositivo duty_cycle es leído */
static ssize_t duty_read(struct file *file, char *buffer, size_t length, loff_t * offset)
{
    /* Número de bytes actualmente escritos en el buffer*/
    int bytes_read = 0;

    /* Si esta en el final del mensaje, retorna 0 significa final del archivo */
    if (*duty_ptr == 0)
        return 0;
    while (length-- && *duty_ptr)
    {
        /* The buffer is in the user data segment, not the kernel */
        /* segment so "*" assignment won't work. */
        /* put user which copies data from the kernel data segment to */
        /* the user data segment. */
        put_user(*(duty_ptr++), buffer++);

        bytes_read++;
    }
    /* Retorna contador de bytes, eviados al usuario */
    return bytes_read;
}

/* llama cuando un proceso escribe en archivo duty_cycle */
static ssize_t duty_write(struct file *file, const char *buff, size_t len, loff_t * off)
{
    int not_copied=0;
    if(len > (BUFFER_LEN-1))
        len=BUFFER_LEN-1;
    /* copiar dato de usuario dentro de buffer duty_buf */
    not_copied=copy_from_user(duty_buf,buff,len);
    duty_buf[len]='\0';

    /* salida del valor numerico en buffer duty_buf */
    /* a la dirección del registro */
    #ifndef DEBUG
    outl((unsigned long) atoi(duty_buf),DUTY_REGISTER);
    #endif

    return len-not_copied;
}

/*Llamado cuando el archivo de dispositivo de activación es abierto */
static int load_open(struct inode *inode, struct file *file)
{
    unsigned long value;
    if (load_is_open++)
        return -EBUSY;
}

```

```

    /* Obtener información desde el registro de activación */
    #ifndef DEBUG
    value=inl(LOAD_REGISTER);
    itoa(load_buf,value);
    #endif
    load_ptr=load_buf;
    try_module_get(THIS_MODULE);
    return 0;
}

/* Llamado cuando el archivo de dispositivo de activación es cerrado */
static int load_release(struct inode *inode, struct file *file)
{
    --load_is_open;
    module_put(THIS_MODULE);
    return 0;
}

/* Llamado cuando el archivo de dispositivo de activación es leído */
static ssize_t load_read(struct file *file, char *buffer, size_t length, loff_t * offset)
{
    /* Numero de bytes actualmente escritos en el buffer*/
    int bytes_read = 0;
    /* Si esta en el final del mensaje, retorna 0 significa final del archivo */
    if (*load_ptr == 0)
        return 0;
    while (length-- && *load_ptr)
    {
        /* The buffer is in the user enable segment, not the kernel */
        /* segment so "*" assignment won't work. */
        /* put user which copies data from the kernel data segment to */
        /* the user data segment. */
        put_user>(*load_ptr++, buffer++);
        bytes_read++;
    }
    /* retorna contador de bytes, enviado al usuario */
    return bytes_read;
}

/* llama cuando un proceso escribe el archivo*/
static ssize_t load_write(struct file *file, const char *buff, size_t len, loff_t * off)
{
    int not_copied=0;
    if(len > (BUFFER_LEN-1))
        len=BUFFER_LEN-1;
    /* Copiar dato de usuario dentro de clkdiv string */
    not_copied=copy_from_user(load_buf,buff,len);
    load_buf[len]='\0';

    /* output the value of the number in buffer enable string */
    /* to address of the direction register */
    #ifndef DEBUG
    outl((unsigned long)atoi(load_buf),LOAD_REGISTER);
    #endif
    return len-not_copied;
}

/* Definiciones, que funciones son llamadas para /dev/pwmclockdiv */
static struct file_operations fops_clkdiv =
{
//Manejo de estructura
    .read= clkdiv_read,
    .write= clkdiv_write,
    .open= clkdiv_open,
    .release= clkdiv_release
};

```

```

/* Definiciones, que funciones son llamadas para /dev/pwmduty */
static struct file_operations fops_duty =
{
    .read= duty_read,
    .write= duty_write,
    .open= duty_open,
    .release= duty_release
};

/* Definiciones, que funciones son llamadas para /dev/pwmload */
static struct file_operations fops_load =
{
    .read= load_read,
    .write= load_write,
    .open= load_open,
    .release= load_release
};

/* Inicializando el módulo */
static int __init mod_init(void)
{
    if(register_chrdev(CLOCK_DIV_MAJOR, "pwmclockdiv", &fops_clkdiv))
    {
        printk("register_chrdev of pwmclockdiv failed!\n");
        return -EIO;
    }
    if(register_chrdev(DUTY_MAJOR, "pwmduty", &fops_duty))
    {
        printk("register_chrdev of pwmduty failed!\n");
        return -EIO;
    }
    if(register_chrdev(LOAD_MAJOR, "pwmload", &fops_load))
    {
        printk("register_chrdev of pwmload failed!\n");
        return -EIO;
    }
    /*Presentación de mensajes en el Kernel*/
    printk(KERN_INFO "El dispositivo PWM Avalon ha sido registrado con 1 salida en LEDG8.\n");
    printk(KERN_INFO "Lo conforman los registros CLOCK_DIV 0x00%8X DUTY 0x00%8X LOAD 0x00%8X\n",
    CLOCK_DIV_REGISTER, DUTY_REGISTER, LOAD_REGISTER);
    printk(KERN_INFO "%s The pwmclockdiv major device number is %d.\n", "Registration is a success",
    CLOCK_DIV_MAJOR);
    printk(KERN_INFO "%s The pwmduty major device number is %d.\n", "Registration is a success",
    DUTY_MAJOR);
    printk(KERN_INFO "%s The pwmload major device number is %d.\n", "Registration is a success",
    LOAD_MAJOR);
    printk(KERN_INFO "Espere unos segundos y observe el movimiento del LEDG8 en la tarjeta DE2.\n");
    printk(KERN_INFO "Ha sido cargado inicialmente con los siguientes valores: Periodo 100000000 Duty
    cycle: 50000000.\n");
    printk(KERN_INFO "Eso nos da un periodo de 1 segundo aproximadamente a 100MHz del clock del
    sistema.\n");
    outl(100000000,CLOCK_DIV_REGISTER);
    udelay(2000);
    outl(50000000,DUTY_REGISTER);
    udelay(2000);
    outl(1,LOAD_REGISTER);
    udelay(2000);
    printk(KERN_INFO "Modulo cargado exitosamente.\n");

    return 0;
}

/* Saliendo del módulo */

```

```
static void __exit mod_exit(void)
{
    outl(0,LOAD_REGISTER);
    udelay(2000);
    unregister_chrdev(CLOCK_DIV_MAJOR,"pwmclockdiv");
    unregister_chrdev(DUTY_MAJOR,"pwmduty");
    unregister_chrdev(LOAD_MAJOR,"pwmload");
    printk(KERN_INFO "\nPWM OFF!! Modulo descargado exitosamente.\n");
}

/* what are the module init/exit functions */
module_init(mod_init);
module_exit(mod_exit);

MODULE_AUTHOR("Diego Jáuregui, Sara Rafaela Poveda Luna, Ronald Ponguillo");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Module which creates device handler for /dev/pwmclockdiv, /dev/pwmduty and /dev/
pwmload for Hard-PWM pin");
MODULE_SUPPORTED_DEVICE("none");
```

```
#
# Misc strange devices
#

menuconfig MISC_DEVICES
    bool "Misc devices"
    default y
    ---help---
    Say Y here to get to see options for device drivers from various
    different categories. This option alone does not add any kernel code.

    If you say N, all options in this submenu will be skipped and disabled.

if MISC_DEVICES

config ATMEL_PWM
    tristate "Atmel AT32/AT91 PWM support"
    depends on AVR32 || ARCH_AT91SAM9263 || ARCH_AT91SAM9RL || ARCH_AT91CAP9
    help
    This option enables device driver support for the PWM channels
    on certain Atmel processors. Pulse Width Modulation is used for
    purposes including software controlled power-efficient backlights
    on LCD displays, motor control, and waveform generation.

config ATMEL_TCLIB
    bool "Atmel AT32/AT91 Timer/Counter Library"
    depends on (AVR32 || ARCH_AT91)
    help
    Select this if you want a library to allocate the Timer/Counter
    blocks found on many Atmel processors. This facilitates using
    these blocks by different drivers despite processor differences.

config ATMEL_TCB_CLKSRC
    bool "TC Block Clocksource"
    depends on ATMEL_TCLIB && GENERIC_TIME
    default y
    help
    Select this to get a high precision clocksource based on a
    TC block with a 5+ MHz base clock rate. Two timer channels
    are combined to make a single 32-bit timer.

    When GENERIC_CLOCKEVENTS is defined, the third timer channel
    may be used as a clock event device supporting oneshot mode
    (delays of up to two seconds) based on the 32 KiHz clock.

config ATMEL_TCB_CLKSRC_BLOCK
    int
    depends on ATMEL_TCB_CLKSRC
    prompt "TC Block" if ARCH_AT91RM9200 || ARCH_AT91SAM9260 || CPU_AT32AP700X
    default 0
    range 0 1
    help
    Some chips provide more than one TC block, so you have the
    choice of which one to use for the clock framework. The other
    TC can be used for other purposes, such as PWM generation and
    interval timing.

config IBM_ASM
    tristate "Device driver for IBM RSA service processor"
    depends on X86 && PCI && INPUT && EXPERIMENTAL
    ---help---
    This option enables device driver support for in-band access to the
    IBM RSA (Condor) service processor in eServer xSeries systems.
    The ibmasm device driver allows user space application to access
    ASM (Advanced Systems Management) functions on the service
    processor. The driver is meant to be used in conjunction with
    a user space API.
```


The `ibmasm` driver also enables the OS to use the UART on the service processor board as a regular serial port. To make use of this feature serial driver support (`CONFIG_SERIAL_8250`) must be enabled.

WARNING: This software may not be supported or function correctly on your IBM server. Please consult the IBM ServerProven website <<http://www.pc.ibm.com/ww/eserver/xseries/serverproven>> for information on the specific driver level and support statement for your IBM server.

config PHANTOM

tristate "Sensable PHANToM (PCI)"
depends on PCI
help

Say Y here if you want to build a driver for Sensable PHANToM device.

This driver is only for PCI PHANToMs.

If you choose to build module, its name will be `phantom`. If unsure, say N here.

config SGI_IOC4

tristate "SGI IOC4 Base IO support"
depends on PCI
---help---

This option enables basic support for the IOC4 chip on certain SGI IO controller cards (IO9, IO10, and PCI-RT). This option does not enable any specific functions on such a card, but provides necessary infrastructure for other drivers to utilize.

If you have an SGI Altix with an IOC4-based card say Y. Otherwise say N.

config TIFM_CORE

tristate "TI Flash Media interface support (EXPERIMENTAL)"
depends on EXPERIMENTAL && PCI
help

If you want support for Texas Instruments(R) Flash Media adapters you should select this option and then also choose an appropriate host adapter, such as 'TI Flash Media PCI74xx/PCI76xx host adapter support', if you have a TI PCI74xx compatible card reader, for example.

You will also have to select some flash card format drivers. MMC/SD cards are supported via 'MMC/SD Card support: TI Flash Media MMC/SD Interface support (MMC_TIFM_SD)'.

To compile this driver as a module, choose M here: the module will be called `tifm_core`.

config TIFM_7XX1

tristate "TI Flash Media PCI74xx/PCI76xx host adapter support (EXPERIMENTAL)"
depends on PCI && TIFM_CORE && EXPERIMENTAL
default TIFM_CORE
help

This option enables support for Texas Instruments(R) PCI74xx and PCI76xx families of Flash Media adapters, found in many laptops. To make actual use of the device, you will have to select some flash card format drivers, as outlined in the TIFM_CORE Help.

To compile this driver as a module, choose M here: the module will be called `tifm_7xx1`.

config ICS932S401

tristate "Integrated Circuits ICS932S401"
depends on I2C && EXPERIMENTAL
help

If you say yes here you get support for the Integrated Circuits ICS932S401 clock control chips.

This driver can also be built as a module. If so, the module will be called ics932s401.

config ATMEL_SSC

tristate "Device driver for Atmel SSC peripheral"
depends on AVR32 || ARCH_AT91

---help---

This option enables device driver support for Atmel Synchronized Serial Communication peripheral (SSC).

The SSC peripheral supports a wide variety of serial frame based communications, i.e. I2S, SPI, etc.

If unsure, say N.

config ENCLOSURE_SERVICES

tristate "Enclosure Services"
default n

help

Provides support for intelligent enclosures (bays which contain storage devices). You also need either a host driver (SCSI/ATA) which supports enclosures or a SCSI enclosure device (SES) to use these services.

config SGI_XP

tristate "Support communication between SGI SSIs"
depends on NET

depends on (IA64_GENERIC || IA64_SGI_SN2 || IA64_SGI_UV || X86_UV) && SMP

select IA64_UNCACHED_ALLOCATOR if IA64_GENERIC || IA64_SGI_SN2

select GENERIC_ALLOCATOR if IA64_GENERIC || IA64_SGI_SN2

select SGI_GRU if X86_64 && SMP

---help---

An SGI machine can be divided into multiple Single System Images which act independently of each other and have hardware based memory protection from the others. Enabling this feature will allow for direct communication between SSIs based on a network adapter and DMA messaging.

config HP_ILO

tristate "Channel interface driver for HP iLO/iLO2 processor"
depends on PCI

default n

help

The channel interface driver allows applications to communicate with iLO/iLO2 management processors present on HP ProLiant servers. Upon loading, the driver creates /dev/hpilo/dXccbN files, which can be used to gather data from the management processor, via read and write system calls.

To compile this driver as a module, choose M here: the module will be called hpilo.

config SGI_GRU

tristate "SGI GRU driver"
depends on X86_UV && SMP

default n

select MMU_NOTIFIER

---help---

The GRU is a hardware resource located in the system chipset. The GRU contains memory that can be mmaped into the user address space. This memory is used to communicate with the GRU to perform functions such as load/store, scatter/gather, bcopy, AMOs, etc. The GRU is directly accessed by user instructions using user virtual addresses. GRU instructions (ex., bcopy) use user virtual addresses for operands.

If you are not running on a SGI UV system, say N.

```
config SGI_GRU_DEBUG
    bool "SGI GRU driver debug"
    depends on SGI_GRU
    default n
    ---help---
    This option enables addition debugging code for the SGI GRU driver. If
    you are unsure, say N.

config DELL_LAPTOP
    tristate "Dell Laptop Extras (EXPERIMENTAL)"
    depends on X86
    depends on DCDBAS
    depends on EXPERIMENTAL
    depends on BACKLIGHT_CLASS_DEVICE
    depends on RFKILL
    depends on POWER_SUPPLY
    default n
    ---help---
    This driver adds support for rfkill and backlight control to Dell
    laptops.

config ISL29003
    tristate "Intersil ISL29003 ambient light sensor"
    depends on I2C && SYSFS
    help
    If you say yes here you get support for the Intersil ISL29003
    ambient light sensor.

    This driver can also be built as a module. If so, the module
    will be called isl29003.

config PWM_AVALON
    tristate "Driver de PWM avalon IP core"
    help
    Habilita el modulo para acceder al IP core PWM avalon.

source "drivers/misc/c2port/Kconfig"
source "drivers/misc/eeprom/Kconfig"
source "drivers/misc/de2_pio/Kconfig"

endif # MISC_DEVICES
```

```
#
# Makefile for misc devices that really don't fit anywhere else.
#
obj-$(CONFIG_IBM_ASM) += ibmasm/
obj-$(CONFIG_HDPU_FEATURES) += hdpuftrs/
obj-$(CONFIG_ATMEL_PWM) += atmel_pwm.o
obj-$(CONFIG_ATMEL_SSC) += atmel-ssc.o
obj-$(CONFIG_ATMEL_TCLIB) += atmel_tclib.o
obj-$(CONFIG_ICS932S401) += ics932s401.o
obj-$(CONFIG_LKDTM) += lkdtm.o
obj-$(CONFIG_TIFM_CORE) += tifm_core.o
obj-$(CONFIG_TIFM_7XX1) += tifm_7xx1.o
obj-$(CONFIG_PHANTOM) += phantom.o
obj-$(CONFIG_SGI_IOC4) += ioc4.o
obj-$(CONFIG_ENCLOSURE_SERVICES) += enclosure.o
obj-$(CONFIG_KGDB_TESTS) += kgdbts.o
obj-$(CONFIG_SGI_XP) += sgi-xp/
obj-$(CONFIG_SGI_GRU) += sgi-gru/
obj-$(CONFIG_HP_ILO) += hpilo.o
obj-$(CONFIG_ISL29003) += isl29003.o
obj-$(CONFIG_PWM_AVALON) += pwm_avalon.o
obj-y += de2_pio/
obj-$(CONFIG_C2PORT) += c2port/
obj-y += eeprom/
```

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <asm/io.h>
#include <asm/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Diego Jauregui");
MODULE_DESCRIPTION("Driver for the eight seven segment displays on the DE2 dev kit - SEG7_LUT_8 IP Core.");

#ifdef DEBUG
#define DEBUG_PRINT(string) printk(KERN_ALERT "===seven_seg_pio.c:%3d: %s\n", __LINE__, string)
#define DEBUG_FUNC_ENTER(__FUNCTION__) printk(KERN_ALERT "===seven_seg_pio.c:%3d: ENTER %s()\n", __LINE__, __FUNCTION__)
#define DEBUG_FUNC_EXIT(__FUNCTION__) printk(KERN_ALERT "===seven_seg_pio.c:%3d: EXIT %s()\n", __LINE__, __FUNCTION__)
#else
#define DEBUG_PRINT(string)
#define DEBUG_FUNC_ENTER()
#define DEBUG_FUNC_EXIT()
#endif

#undef na_seven_seg_pio
#define na_seven_seg_pio ((np_pio*) 0x81401178)

// PIO write helper
#define WRITE_SEGS(val) outl(~((unsigned short) val), &(na_seven_seg_pio->np_piodata))

// Major number (/dev/sevensseg)
#define SEVEN_SEG_PIO_MAJOR 71

// Length of buffer_string
#define BUFFER_LEN 11

// Misc
char seven_seg_pio_buf[BUFFER_LEN] = "\0";
int seven_seg_pio_device_open = 0;

/* convert Hex-string to integer */
static long hexstr2int(const char *name)
{
    unsigned long val = 0;
    for (;name++)
    {
        switch (*name)
        {
            case '0'...'9':
                val = 16 * val + (*name - '0');
                break;
            case 'A':
            case 'B':
            case 'C':
            case 'D':
            case 'E':
            case 'F':
                val = 16 * val + (10 + *name - 'A');
                break;
            case 'a':
            case 'b':
```

```

        case 'c':
        case 'd':
        case 'e':
        case 'f':
            val = 16 * val + (10 + *name - 'a');
            break;

        default:
            return val;
    }
}

////////////////////////////////////
// Device File Open Handler
////////////////////////////////////

static int seven_seg_pio_open(struct inode *inode, struct file *file)
{
    DEBUG_FUNC_ENTER();
    if (seven_seg_pio_device_open++) {
        printk(KERN_ALERT "Device busy...\n");
        return -EBUSY;
    }
    try_module_get(THIS_MODULE);
    DEBUG_FUNC_EXIT();
    return 0;
}

////////////////////////////////////
// Device File Close Handler
////////////////////////////////////

static int seven_seg_pio_release(struct inode *inode, struct file *file)
{
    DEBUG_FUNC_ENTER();
    --seven_seg_pio_device_open;
    module_put(THIS_MODULE);
    DEBUG_FUNC_EXIT();
    return 0;
}

////////////////////////////////////
// Device File Write Handler
////////////////////////////////////

static ssize_t seven_seg_pio_write(struct file *file, const char *buff, size_t len, loff_t *off)
{
    DEBUG_FUNC_ENTER();

    if (len > (BUFFER_LEN-1))
        len = BUFFER_LEN-1;

    // Copy data from user into buffer
    copy_from_user(seven_seg_pio_buf, buff, len);
    seven_seg_pio_buf[len] = '\0';

    // Output to 7seg
    outl((unsigned long) hexstr2int(seven_seg_pio_buf), &(na_seven_seg_pio->np_piodata));

#ifdef DEBUG
    printk(KERN_ALERT "Writing 0x%02X to Seven Seg Displays\n", ((unsigned char) seven_seg_pio_buf
[0]));
#endif
}

```

```
    DEBUG_FUNC_EXIT();
    return len;
}

////////////////////////////////////
// Device File Operations Structure
////////////////////////////////////

static struct file_operations fops_seven_seg_pio =
{
    .write    = seven_seg_pio_write,
    .open    = seven_seg_pio_open,
    .release = seven_seg_pio_release
};

////////////////////////////////////
// Module Init
////////////////////////////////////

static int seven_seg_pio_init(void)
{
    DEBUG_FUNC_ENTER();

    // Reserve memory region for seven_seg_pio port
    if(!request_mem_region((unsigned long) na_seven_seg_pio, sizeof(np_pio), "seven_seg_pio"))
        return -1;

    // Register seven_seg_pio port as a character device to the kernel
    if(register_chrdev(SEVEN_SEG_PIO_MAJOR, "seven_seg_pio", &fops_seven_seg_pio))
    {
        DEBUG_PRINT("register_chrdev() failed!");
        release_mem_region((unsigned long) na_seven_seg_pio, sizeof(np_pio));
        return -EIO;
    }

    // Init 7 seg
    outl((unsigned long) 4294967295, &(na_seven_seg_pio->np_piodata));

    DEBUG_FUNC_EXIT();
    return 0;
}

////////////////////////////////////
// Module Exit
////////////////////////////////////

static void seven_seg_pio_exit(void)
{
    DEBUG_FUNC_ENTER();
    // Init 7 seg
    outl((unsigned long) 0, &(na_seven_seg_pio->np_piodata));
    // Unregister character device
    unregister_chrdev(SEVEN_SEG_PIO_MAJOR, "seven_seg_pio");

    // Release memory region
    release_mem_region((unsigned long) na_seven_seg_pio, sizeof(np_pio));

    DEBUG_FUNC_EXIT();
}

// Pass init/exit functions to kernel
module_init(seven_seg_pio_init);
module_exit(seven_seg_pio_exit);
```

```
menu "DE2 PIO DEVICES DRIVERS"
```

```
config SEVEN_SEG_PIO
```

```
    tristate "8 Seven segments displays"
```

```
    help
```

```
    This is a driver for the 8 Seven segment displays in the DE2 board.  
    Works with the SEG7_LUT_8 Avalon slave Controller. The module name in  
    SOPC Builder must be "seven_seg_pio".
```

```
endmenu
```



```
obj-$(CONFIG_SEVEN_SEG_PIO) += seven_seg_pio.o
```

```
/*
 * pwmavalon.h
 *
 * Created on: Sep 19, 2011
 * Author: diegxj
 */

#ifndef PWMAVALON_H_
#define PWMAVALON_H_

#ifndef NULL
#define NULL ((void *) 0)
#endif

#define CLOCK_DIV_FILE "/dev/pwmclockdiv"
#define DUTY_FILE "/dev/pwmduty"
#define LOAD_FILE "/dev/pwmload"

typedef enum{clock_div_file, duty_file, load_file} files;

/* Función que convierte un string a un entero */

static long atoi(const char *name)
{
    long val = 0;
    for (;name++)
    {
        switch (*name)
        {
            case '0' ... '9':
                val = 10*val+(*name-'0');
                break;
            default:
                return val;
        }
    }
}

/* Función que convierte byte a string*/

void itoa(char *name, unsigned long value)
{
    long i,j=0;
    for(i=100000000;i>0;i/=10)
    {
        name[j++]='0'+value/i;
        value=value-(value/i)*i;
        if(name[0]=='0' && i!=1)
            j=0;
    }
    name[j++]='\n';
    name[j]=0;
}

/* Función que lee el archivo del dispositivo y
retorna el valor almacenado recibiendo como parámetro
el archivo*/
unsigned long read_dev(files file)
{
    char string[11]={0};
    unsigned long ret;
    FILE *f=NULL;
    int i=0;
    /* Abriendo archivo requerido para
lectura del mismo*/
    switch(file)
```

```

{
    case clock_div_file:
        f=fopen(CLOCK_DIV_FILE, "r");
    break;
    case duty_file:
        f=fopen(DUTY_FILE, "r");
    break;
    case load_file:
        f=fopen(LOAD_FILE, "r");
    break;
}
/* leyendo lineas de archivo */
while((i<10) && (string[i++]=fgetc(f)));
fclose(f);
/* retornando valor */
ret = atoi(string); // Función que convierte string a entero
return ret;
}

/* Funcion que escribe bytes al archivo del dispositivo
recibe como parámetro el valor a escribirse*/
void write_dev(files file, unsigned long val)
{
    //Declaración de Variables
    long valor = 0;
    char string[11]={0};
    FILE *f=NULL;
    valor = val;
    /* Abriendo archivo requerido para la escritura */
    switch(file)
    {
        case clock_div_file:
            f=fopen(CLOCK_DIV_FILE, "r+");
        break;
        case duty_file:
            f=fopen(DUTY_FILE, "r+");
        break;
        case load_file:
            f=fopen(LOAD_FILE, "r+");
        break;
    }
    /* escribiendo archivo*/
    itoa(string,valor); //función para convertir a string
    fprintf(f, string);
    fclose(f);
}

/*Definición de constantes*/
#define IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER                read_dev(clock_div_file) /
/*función para leer dirección */
#define IOWR_ALTERA_AVALON_PWM_CLOCK_DIVIDER(val)          write_dev(clock_div_file,
val) /*de memoria*/
#define ALTERA_AVALON_PWM_CLOCK_DIVIDER_MSK                0xFFFFFFFF
#define ALTERA_AVALON_PWM_CLOCK_DIVIDER_OFST              0

#define IORD_ALTERA_AVALON_PWM_DUTY                        read_dev
(duty_file)
#define IOWR_ALTERA_AVALON_PWM_DUTY(val)                  write_dev(duty_file,
val)
#define ALTERA_AVALON_PWM_DUTY_MSK                        0xFFFFFFFF
#define ALTERA_AVALON_PWM_DUTY_OFST                       0

#define IORD_ALTERA_AVALON_PWM_LOAD                        read_dev(load_file)
#define IOWR_ALTERA_AVALON_PWM_LOAD(val)                  write_dev(load_file, val)
#define ALTERA_AVALON_PWM_LOAD_MSK                        0x1
#define ALTERA_AVALON_PWM_LOAD_OFST                       0

```

```
#endif /* PWMAVALON_H_ */
```

```

/*****
/* Copyright © 2004 Altera Corporation, San Jose, California, USA. All rights reserved. */
/* Permission is hereby granted, free of charge, to any person obtaining a copy of this */
/* software and associated documentation files (the "Software"), to deal in the Software */
/* without restriction, including without limitation the rights to use, copy, modify, */
/* merge, publish, distribute, sublicense, and/or sell copies of the Software, and to */
/* permit persons to whom the Software is furnished to do so, subject to the following */
/* conditions: The above copyright notice and this permission notice shall be included */
/* in all copies or substantial portions of the Software. */
/* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, */
/* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A */
/* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT */
/* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF */
/* CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE */
/* OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. */
/* This agreement shall be governed in all respects by the laws of the State of */
/* California and by the laws of the United States of America. Altera does not */
/* recommend, suggest or require that this reference design file be used in conjunction */
/* or combination with any other product. */
*****/

//Includes de librerias que usaremos
#include <stdio.h>
#include <unistd.h>
#include "headers/altera_avalon_pwm_routines.h"
#include "headers/pwmavalon.h"

#define na_switch_pio 0x81401120 // declarando constante con direcci3 de memoria de switches

void print_error(files file, int return_code); //Presenta motivos del error
void check_return_code(files file, int return_code); //funcion para chequear estado del dispositivo

/*Programa Principal*/
int main(void)
{
    /*Rutina que permite acceder a hardware desde usuario*/
    volatile unsigned *switches = ((volatile unsigned *) (na_switch_pio));
    unsigned long period = 0;
    unsigned long duty = 0;
    int return_code = ALTERA_AVALON_PWM_OK;

    printf("Hello from the PWM test program.\n");
    printf("The starting values in the PWM registers are:\n");
    /*Mostrando valores almacenados en registros*/
    printf("Periodo = %ld\n", IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER);
    printf("Ancho de pulso = %ld\n", IORD_ALTERA_AVALON_PWM_DUTY);
    printf("Enable bit = %ld\n", IORD_ALTERA_AVALON_PWM_LOAD);
    /*-----*/
    printf("Modo de uso: El valor del ancho de pulso debe ser menor que el del periodo.\n");
    printf("Para el Periodo y el ciclo de trabajo solo ingresar n3meros enteros de hasta 4294967295.\n");
    printf("El Enable bit solo puede ser de dos valores: 0 - PWM desactivado, 1 - PWM activado.\n");
    /*-----Solicitando valores al usuario-----*/
    printf("\nIngrese valor de periodo: ");
    scanf("%ld",&period); //Almacenando valor ingresado en variable period
    printf("\nIngrese valor de ancho de pulso: ");
    scanf("%ld",&duty); //Almacenando valor ingresado en variable duty

    printf("\n Observe pulsaciones de LED en la tarjeta.\n");
}

```

```

/*Iniciando registros con valores de periodo y duty*/
return_code = altera_avalon_pwm_init(period, duty);

/*Verificando códigos de retorno*/
check_return_code(clock_div_file, return_code);
check_return_code(duty_file, return_code);

//Habilitando PWM and Check Return Code
return_code = altera_avalon_pwm_load();
check_return_code(load_file, return_code);

/*Igualando la variable duty con el valor escrito en el registro de duty_sycle durante
la inicialización*/
duty = IORD_ALTERA_AVALON_PWM_DUTY;

/*Rutina que varia vaores del duty*/
while(1)
{
    while(duty++ < IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER)
    {
        IOWR_ALTERA_AVALON_PWM_LOAD(switches[0]);
        altera_avalon_pwm_change_duty_cycle(duty);
        check_return_code(duty_file, return_code);
    }
    while(--duty > 1)
    {
        IOWR_ALTERA_AVALON_PWM_LOAD(switches[0]);
        altera_avalon_pwm_change_duty_cycle(duty);
        check_return_code(duty_file, return_code);
    }
}
return 0;
}

/*Verificación de códigos de retorno*/
void check_return_code(files file, int return_code)
{
    if(return_code != ALTERA_AVALON_PWM_OK) //el return_code debe ser OK
        print_error(file, return_code);
}

void print_error(files file, int return_code)
{
    printf("Programa terminó debido a un error con Avalon PWM situado en el fichero de dispositivo");
    switch(return_code)
    {
        /*Validando diferentes tipos de causa de errores*/
        case ALTERA_AVALON_PWM_DUTY_GREATER_THAN_CLOCK_CYCLE_ERROR:
            printf("The value in the clock cycle register must be greater than the value in
the duty cycle register\n");
            printf("Value in the Clock Divide Register: 0x%ld\n",
IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER);
            printf("Value in the Duty Cycle Register: 0x%ld\n", IORD_ALTERA_AVALON_PWM_DUTY);
            break;
        case ALTERA_AVALON_PWM_LOADED_CONFIRMATION_ERROR:
            printf("Unable to confirm that the PWM is enabled\n");
            printf("Value in the Enable Register: 0x%ld\n", IORD_ALTERA_AVALON_PWM_LOAD);
            break;
        case ALTERA_AVALON_PWM_DISABLED_CONFIRMATION_ERROR:
            printf("Unable to confirm that the PWM is disabled\n");
            printf("Value in the Enable Register: 0x%ld\n", IORD_ALTERA_AVALON_PWM_LOAD);
            break;
        default:
            break;
    }
    while(1);
}

```

}

5 Programming kernel driver

5.1 Implementation of the driver

5.1.1 General

This section of the tutorial shows, how to program a driver for linux kernel 2.6. We recommend to print out the kernel module source (see appendix B), which is attached to this how to and to read the source simultaneously with this section.

First of all you should know, that a kernel module is in principle just a c program with at least two functions, `static int __init mod_init(void)` and `static void __exit mod_exit(void)`. (The names of the functions are specified by `module_init(mod_init);` and `module_exit(mod_exit);`; where function pointers are given to the kernel)

5.1.2 Defines

- We define the major numbers for linking between file-system and driver:
`#define DATA_MAJOR 240` - data device file gets major number 240
`#define DIR_MAJOR 241` - direction device file gets major 241
- We need the addresses of the data and direction register in the avalon bus. This address is assigned when adding the `pio_eigen.vhd` to the hardware specification in SOPC-builder (see section 5).
`#define DATA_REGISTER 0x00801090` - data register is at address 0x00801090
`#define DIR_REGISTER 0x00801091` - direction register is at address 0x00801091
- We need buffers to hold the data read from registers for copying into user-space.
`#define STRING_LEN 5` - buffers are 5 bytes long

5.1.3 Function calls

The `init` function is called when the module is loaded into the kernel, e.g. with `insmod modulename` and the `exit` function is executed when unloading the module from the kernel, e.g. with `rmmod modulename`.

This way it is possible to run a program in kernel-space, but it is still not possible to access the hardware from the user-space. To make that possible, it is necessary to extend the module to a driver which links the hardware registers into the linux filesystem.

To get such a driver we need to register all device files we want to use to the IO-management of the kernel. This is done by `register_chrdev(DATA_MAJOR, "iodata", &fops_dat);` for the data register and by `register_chrdev(DIR_MAJOR, "iodir", &fops_dir);` for the direction register in module's `init` function. The first parameter of these function calls

(DATA_MAJOR and DIR_MAJOR) are just integer values for linking the device files of the filesystem to the driver. The second parameter is the name, which identifies the device file in kernel IO-management. The last parameter is a struct, including pointers to functions which are called when accessing the device files.

```
static struct file_operations fops_dir =
{
.read= dir_read,
.write= dir_write,
.open= dir_open,
.release= dir_release
};
```

- Function **int dir_open(..)** is called when a process tries to open the device file. It checks, whether another process uses this file already, if so, it will return a busy signal to the process, which tries to open. If a process got successfully access to the device file, the data of the direction register is read with `inb(DIR_REGISTER)` and stored as string in a buffer.
- Function **size_t dir_read(..)** is called when a process reads from device file. It copies every byte of the buffer (containing the data of the direction register) into user-space using the command `put_user(..)`. Normally the reason why we cannot just copy from a kernel space buffer into a user space buffer using `memcpy` would be, that the buffers are in different virtual address blocks. But due to the fact that we don't have a memory management unit (mmu), those buffers are in the same address block. Nevertheless we decided to program it like if we had a mmu, to allow debugging on a host-machine with mmu.
- Function **size_t dir_write(..)** is called when writing to the device file. It copies data from an user space buffer into a kernel space buffer using `copy_from_user(..)`. The reason for not using `memcpy` is the same like for function `dir_read`. Data is stored into direction register using `outb(buffer, DIR_REGISTER)` after copying.
- Function **int dir_release(..)** is called when a process closes the file. It unlocks the device file and the module.

Same functions exist for the data register too.

When unloading the driver the device files have to be unregistered from IO-management. To accomplish that, those two functions must be called in the module's exit function:

```
unregister_chrdev(DATA_MAJOR, "iodata"); for unregistering the data device file
and unregister_chrdev(DIR_MAJOR, "iodir"); for unregistering the direction de-
vice file.
```

5.2 Adding device files to rootfs

Now we have to take care, that our device files are generated when booting the linux image on the target. Therefor you can add these two lines to `~/download/rootfs_list`:

```
nod /dev/iodata 666 0 0 c 240 0
nod /dev/iodir 666 0 0 c 241 0
```

The lines say, that 2 device files are generated, `/dev/iodata` and `/dev/iodir` both with the Unix permissions 666, and the major numbers for associating the files with the driver, in this case 240 and 241.

When building a new kernel, it will automatically create all device files like configured in `~/download/rootfs_list`.

5.3 Adding driver to kernel

To add the driver to the kernel sources, copy the source code to `linux-source/drivers/misc/`. Now we need to tell menuconfig something about the new kernel driver. Therefor the line `obj-$(CONFIG_IO) += io.o` has to be added to the Makefile in `linux-source/drivers/misc/`. And following lines to the Kconfig file in `linux-source/drivers/misc/` right above the "endmenu" statement:

```
config IO
tristate "IO module for custom IO"
help
Enable module to access custom IO.
```

5.4 Building the kernel

Change into the directory of the linux source and type:

```
make ARCH=nios2nommu CROSS_COMPILE=nios2-linux-uclibc- menuconfig
```

Now the kernel configuration interface shows up. To enable the custom kernel driver change to

```
Device Drivers --> Misc devices
```

and press 'y' when highlighting `IO module for custom IO`. Then press `exit` and save the configuration.

To cross-compile the kernel type:

```
make ARCH=nios2nommu CROSS_COMPILE=nios2-linux-uclibc- zImage
```

After kernel was successfully generated, the zImage including kernel and rootfs can be copied from: `linux-source/arch/nios2nommu/boot/zImage`.