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

Diego Jáuregui Solórzano, Sara Poveda Luna, Ing. Ronald Ponguillo Intriago
Facultad de Ingeniería en Electricidad y Computación
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
djauregu@espol.edu.ec, spoveda@espol.edu.ec, rponguill@espol.edu.ec

Resumen

En este artículo presentamos la implementación de un sistema embebido con µClinux como sistema operativo en una arquitectura de hardware basada en el procesador configurable Nios II programado en un FPGA. El Nios II se interconecta con los dispositivos del sistema embebido por medio del bus Avalon cuya especificación es creada automáticamente por la herramienta de construcción SOPC Builder de Altera. El target de la implementación es una tarjeta DE2 de Terasic. µClinux es cargado y ejecutado desde la memoria SDRAM de la tarjeta. El kernel de µClinux y el procesador Nios II se implementan sin MMU. Adicional al hardware de la tarjeta DE2 se agrega un módulo PWM escrito en código HDL e implementado en el FPGA compartiendo el espacio con el Nios II y el resto de interfaces que interconectan los periféricos externos con el procesador. El kernel de µClinux es configurado para soportar módulos cargables que pueden ser añadidos en tiempo de ejecución. El acceso al hardware es probado desde el espacio del usuario por medio de una aplicación donde se usan los dos enfoques: haciendo uso del driver y accediendo directamente a los registros mapeados en la memoria.

Palabras Claves: *FPGA, SOPC, procesador configurable, linux, uclinux, nios2, sistema embebido, driver, kernel, pwm, eclipse, compilación cruzada.*

Abstract

In this paper we show the implementation of an embedded system with µClinux as the operative system on an hardware architecture based in the Nios II configurable soft-core that is programmed on a FPGA. The Nios II is interconnected to the devices of embedded system through de Avalon bus with a specification created automatically by the Altera's SOPC Builder tool. The target of this implementation is the DE2 board by Terasic. µClinux is loaded and executed from the SDRAM memory chip of the board. µClinux kernel and Nios II core are configured without MMU. In addition to the DE2 board chip hardware a custom PWM module written in HDL code is added and implemented on the FPGA sharing the space with the Nios II core and the other interfaces that connect the off-chip devices with the processor. µClinux kernel is configured to support loadable modules that could be added into the base kernel in execution time. Hardware access is tried from the user space with a cross compiled application written from the host computer. Two hardware access approaches are used: through the kernel using the driver and by directly manipulating the memory mapped register of the devices.

Keywords*: FPGA, SOPC, soft-core, linux, uclinux, nios2, embedded system, driver, kernel, pwm, eclipse, cross-compiler.*

1. Introducción

El presente trabajo tiene la finalidad de investigar los proyectos y tecnologías existentes en dos áreas importantes de los Sistemas-on-Chip: el desarrollo de sistemas de hardware configurable utilizando FPGA y el uso de Linux como sistema operativo en sistemas embebidos.

Empezamos revisando las herramientas de software necesarias para el desarrollo del hardware configurable y el software embebido.

Luego revisaremos los pasos realizados para creación del hardware SOPC, la configuración y compilación de µClinux, la programación de aplicaciones de usuario.

Por último, cargamos µClinux en tarjeta DE2 y abrimos una sesión remota desde el host. Revisamos los métodos que tenemos para transferir archivos entre el host y el target y la ejecución de aplicaciones.

2. Desarrollo de SOPC con Linux embebido

El desarrollo de sistemas embebidos con Linux sobre hardware reconfigurable puede dividirse en dos equipos de trabajo: uno de hardware y uno de software. Se debe tener conocimiento de sistemas digitales, lenguaje HDL y arquitectura de computadores, así como estar familiarizado con Linux, compilación del kernel y la estructura de los

sistemas operativos, además del área especifica que la aplicación demande.

Este artículo servirá como una guía para el novato. Durante la marcha se deberán ir profundizando estos temas. Existe una basta bibliografía que puede ser consultada entre la que se recomienda [1] [2] [3].

2.1. Plataforma de desarrollo

La construcción de un sistema embebido tiene dos elementos importantes: el host y el target.

El host es el sistema desde 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 target es el sistema embebido propiamente.

La comunicación entre el host y el target es fundamental 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.

El target del sistema esta basado en la tarjeta de desarrollo DE2. El núcleo de la tarjeta es el FPGA Cyclone II EP2C35, que cuenta con 33,216 LEs (elementos lógicos), 105 M4K bloques de RAM, 4 PLLs. La tarjeta DE2 incluye interfaces a los periféricos, al integrar los siguientes chips de control:

- Flash (con 4MByte S29AL032D)
- SDRAM (con 8MByte IS42S16400)
- Ethernet (con DM9000A)
- USB host / client (con ISP1362)
- Audio (con WM8731)
- VGA (con ADV7123)
- SD card socket
- RS232 (con MAX232)
- IrDA (con HSDL-3201)
- PS/2
- Osciladores 50MHz, 27MHz

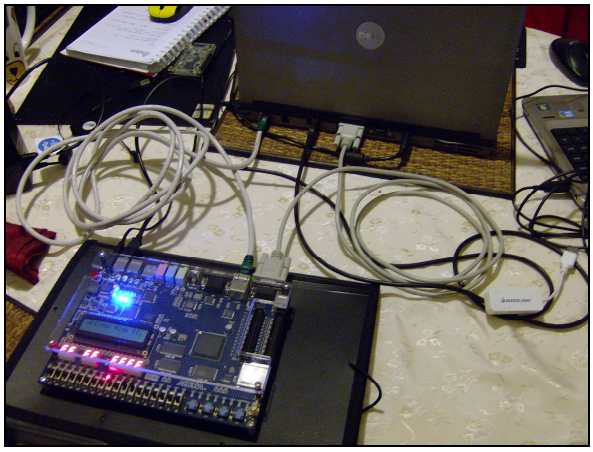

Figura 1. Tarjeta DE2 conectada PC-portátil.

El host del sistema es un PC-portátil basado en Intel con una distribución de Linux de 32 bits instalada.

La conexión entre el host y el target se da a través de los siguientes medios: serie rs-232, Ethernet TCP/IP y serial JTAG por USB.

2.2. Herramientas de diseño, programación, configuración y compilación

Se han utilizado las herramientas de diseño del fabricante del FPGA y las que son recomendadas por los desarrolladores de la comunidad de Nios II:

- Suite de diseño Altera Quartus II 9.1 con service pack 2.
- Eclipse IDE para desarrolladores de C/C++ en Linux.
- Toolchain de compilador cruzado (cross compiler) y librería uClibc.
- Fuentes de µClinux con soporte para Nios II.

2.1.1. Quartus II

Quartus II es la suite de herramientas para el diseño de circuitos digitales sobre dispositivos FPGA de Altera. Provee aplicaciones para la entrada de diseño, síntesis lógica, simulación de la lógica, ubicación y ruteo, análisis de tiempos, administración de potencia y programación de dispositivos, junto con una variedad de utilitarios y aplicaciones adicionales para el diseño y depuración de la lógica programable, entre ellos el SOPC Builder y el SignalTap II.

2.1.2. Eclipse

Eclipse es una plataforma de desarrollo abierta compuesta de entornos de trabajo, herramientas y ejecutables para construir, desarrollar y manejar software. Lo utilizamos para compilar y depurar las aplicaciones de usuario ya que se puede configurar para que utilice el compilador cruzado GCC para uClinux-nios2. Puede ser configurado para la depuración remota de las aplicaciones cargadas en el target a través de gdbserver.

2.1.3. nios2-linux

El µClinux comenzó como un port de Linux kernel v2.x para microcontroladores, particularmente aquellos sin unidad de manejo de memoria (MMU). Fue creado 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.

Aquí usamos la versión nios2-linux-20090730 con el kernel linux-2.6.30. Ya que no hay MMU en µClinux, una aplicación de espacio de usuario tiene acceso directo al hardware y puede por lo tanto saltarse la intermediación del kernel.

El toolchain que utilizamos es el pre-compilado suministrado por la comunidad del Altera Wiki. Está configurado para sistemas Linux 32 bits y es

específico para sistema µClinux sin MMU. Incluye la librería uClibc, una versión micro de C compatible con glibc y que permite generar ejecutables de menor tamaño [4].

Las principales diferencias entre µClinux y Linux pueden ser consultadas en [5].

3. Diseño e implementación

El diseño comprende tres capas:

- Nivel bajo: hardware y SOPC hardware
- Nivel medio: kernel de µClinux
- Nivel alto: aplicaciones de usuario

Para diseñar y ejecutar el sistema se requiere el software de Altera, Quartus II v9.1 y la suite de diseño embebido (EDS) Nios II v9.1. Ya que Altera ofrece documentación detallada sobre como usar su software de diseño, eso no será discutido [6] [7].

3.1. Hardware SOPC

El componente principal de la arquitectura de hardware es el soft-core Nios II [8] [9]. La mayoría de los IP cores utilizados forman parte del CD que acompaña a la tarjeta DE2 o que se pueden descargar de la web. También se utilizó módulos del Programa Universitario de Altera. Existen sitios en la web donde pueden descargarse IP cores libres (opencores) que pueden ser fácilmente adaptados como componentes SOPC Builder [10] [11].

Los periféricos de la tarjeta se conectan con el procesador a través de la interface de bus *Avalon-Memory Mapped*. Se utiliza un *Pipeline Brigde*, para optimizar la cantidad de bloques lógicos que se utilizan para la interconexión de los componentes en el FPGA [12] [13].

La frecuencia del sistema es 100MHz que son obtenidos a través de un multiplicador de frecuencia basado en PLL aplicado al oscilador de 50MHz de la tarjeta DE2.

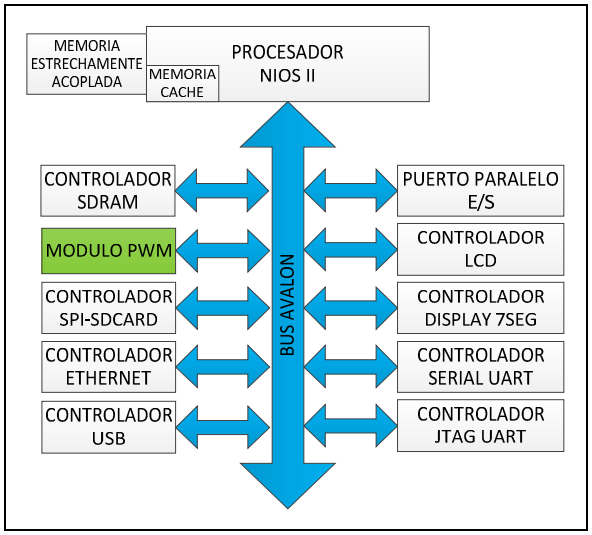

Figura 2. Diagrama de bloques de la arquitectura de hardware SOPC.

El módulo de hardware PWM tiene tres entradas (periodo, ciclo de trabajo, habilitador) y una salida (señal PWM). El módulo está escrito en lenguaje Verilog y es usado como ejemplo por Altera en una nota de aplicación sobre el desarrollo de periféricos usando SOPC Builder [14]. Allí se explica cómo crear un nuevo componente o módulo para SOPC Builder a partir un código HDL personalizado.

Los requerimientos mínimos de hardware para cargar µClinux son [15]:

- Procesador Nios II versión *fast* o *standard*, con multiplicador por hardware
- SDRAM (mínimo requerimiento 8MB)
- Un temporizador completamente funcional
- Un jtag UART o un serial UART.

La configuración de hardware SOPC esta contenida en el archivo PTF generado por el SOPC Builder de Quartus II. Los módulos que componen el sistema son especificados en ese archivo.

cpu_0	Nios II Processor	**sys_clk**	**0x00400000**
sdram_0	SDRAM Controller	**sdram_clk**	**0x02800000**
pipeline_bridge	Avalon-MM Pipeline Bridge	**sys_clk**	**0x01000000**
tri_state_bridge_0	Avalon-MM Tristate Bridge	**sys_clk**	
cfi_flash_0	Flash Memory Interface (CFI)	**sys_clk**	**0x00000000**
timer_0	Interval Timer	**sys_clk**	**0x00401080**
epcs_controller	EPCS Serial Flash Controller	**sys_clk**	**0x00400800**
jtag_uart_0	JTAG UART	**sys_clk**	**0x00401150**
uart_0	UART (RS-232 Serial Port)	**sys_clk**	**0x004010a0**
lcd_16207_0	Character LCD	**sys_clk**	**0x004010e0**
seven_seg_pio	SEG7_LUT_8	**sys_clk**	**0x00401178**
ps2_0	PS2 Serial Port	**sys_clk**	**0x00401158**
dm9000	DM9000A	**sys_clk**	**0x00401160**
sysid	System ID Peripheral	**sys_clk**	**0x00401168**
gpio_0	gpio	**sys_clk**	**0x00401000**
mmc_spi	SPI (3 Wire Serial)	**sys_clk**	**0x004010c0**
ISP1362	ISP1362_IF	**sys_clk**	*multiple*
led_pio	PIO (Parallel I/O)	**sys_clk**	**0x00401100**
button_pio	PIO (Parallel I/O)	**sys_clk**	**0x00401110**
switch_pio	PIO (Parallel I/O)	**sys_clk**	**0x00401120**
pwm_avalon_0	pwm_avalon_interface	**sys_clk**	**0x00401130**

Figura 3. Módulos que componen el hardware SOPC

3.2. Construcción de µClinux

En este apartado se describe como instalar el toolchain de µClinux y como usarlo para construir la imagen del kernel.

µClinux no es la única opción que se tiene para implementar Linux en Nios II. Otras alternativas pueden encontrase en [16] [17].

La distribución de µClinux versión 20090730 es descargada de http://www.niosftp.com/pub/uclinux/nios2-linux-20090730.tar y el toolchain de http://www.niosftp.com/pub/gnutools/nios2gcc-20080203.tar.bz2. El Altera Wiki (http://www.alterawiki.com) enseña como construir µClinux y como escribir módulos, manejar interrupciones y aplicaciones de usuario.

Otros sitios de consulta son el Altera Forum (http://www.alteraforum.com) y la lista de correos nios2-dev (http://sopc.et.ntust.edu.tw/cgi-bin/mailman/listinfo/nios2-dev).

Antes de construir µClinux hay que instalar y actualizar el conjunto de herramientas que son usadas durante la construcción:

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

3.2.1. Instalación del Toolchain

La construcción del kernel de µClinux require que el toolchain este instalado.

Este incluye el compilador cruzado GCC. Extraer el toolchain desde el shell o por medio del entorno gráfico en la carpeta $HOME/opt/nios2.

El directorio del compilador cruzado GCC debe ser agregado al PATH. Esto puede ser realizado añadiendo al archivo $HOME/.bash_profile la línea

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

Para verificar el compilador cruzado ejecutamos siguiente comando en el shell:

```
$ nios2-linux-uclibc-gcc –v
```

Esto debería mostrar

```
Reading specs from
/opt/nios2/lib/gcc/nios2-linux-
uclibc/3.4.6/specs
Configured with:
/root/buildroot/toolchain_build_nios2/gcc-
3.4.6/configure --prefix=/opt/nios2
--build=i386-pc-linux-gnu
--host=i386-pc-linux-gnu
--target=nios2-linux-uclibc
--enable-languages=c
--enable-shared
--disable-__cxa_atexit
--enable-target-optspace
--with-gnu-ld
--disable-nls
--enable-threads
--disable-multilib
--enable-cxx-flags=-static
Thread model: posix
gcc version 3.4.6
```

Si no se desea utilizar el toolchain pre-compilado, se puede encontrar instrucción de como compilar un toolchain personalizado en [18] [19] [20] [21].

3.2.2. Construcción del kernel de µClinux

Una vez que el toolchain esta instalado, el kernel de µClinux puede ser construido.

Es recomendable descomprimir las fuentes en el directorio $HOME por medio comandos en el shell o por una aplicación del entorno gráfico.

El directorio donde quedarían instaladas las fuentes de µClinux sería $HOME/nios2-linux.

Luego que hayamos descomprimidos el archivo con las fuentes, debemos ejecutar un script dentro del directorio nios2-linux

```
$ ./checkout
```

Luego en ~/nios2-linux/uClinux-dist

```
$ git branch –a
$ git checkout test-nios2
```

Y en ~/nios2-linux/linux-2.6

```
$ git branch –a
$ git checkout test-nios2
```

Con eso estamos listos para empezar.

El instalador de la distribución esta basado e}n menús de configuración kconfig [22].

Iniciamos la configuración con el comando

```
$ make menuconfig
```

Se carga la pantalla inicial con las opciones

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

Seleccionamos la primera opción y nos aparece el menú de la figura 4.

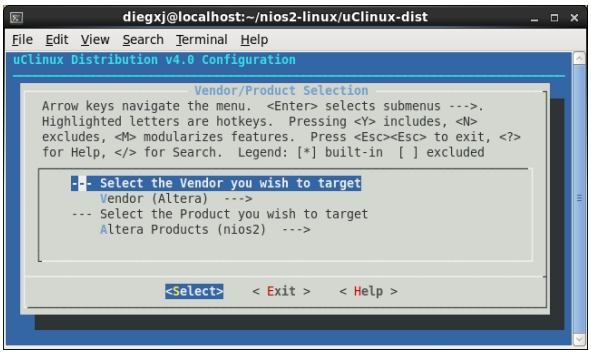

Figura 4. Selección del fabricante y producto del target

En este menú seleccionamos Altera para el vendedor y nios2 para el producto. Luego salimos y guardamos los cambios con <exit> <yes>.

Ahora en la otra opción del menú

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

Seleccionamos como se indica abajo.

```
(linux2.6.x) Kernel Version
(None) Libc Version
[*] Default all settings (lose changes)
[ ] Customize Kernel Settings
[ ] Customize Vendor/User Settings
[ ] Update Default Vendor Settings
```

Luego salimos y guardamos los cambios con <exit> <exit> <yes>.

Es recomendable no cambiar otro parámetro hasta el primer arranque exitoso.

Luego, configuramos y definimos las características del SOPC hardware (procesador, direcciones de memoria y E/S) para la sistema cargado en la tarjeta DE2.

```
$ make vendor_hwselect SYSPTF=system_0.ptf
```

Previo a este comando el archivo system_0.ptf debe haber sido copiado o vinculado a la carpeta $HOME/nios2-linux/uClinux-dist/linux-2.6.x.

El script HWSELECT prepara el archivo de cabecera (.h) que contiene las direcciones de memoria y de los dispositivos de E/S de nuestra tarjeta, el archivo de cabecera “nios2.h”. HWSELECT debe ser ejecutado antes de compilar el kernel. Si estas direcciones son cambiadas luego, es necesario hacer

una limpieza por medio de “make clean” para luego volver a ejecutar “make hwselect” otra vez. Ese es el resultado de la ejecución de HWSELECT:

```
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

--- Please select a device to execute kernel
from:

(1) sdram_0
       Class:
       altera_avalon_new_sdram_controller
       Size: 8388608 bytes

(2) onchip_memory2_0
       Class: altera_avalon_onchip_memory2
       Size: 512 bytes

(3) epcs_controller
       Class:
       altera_avalon_epcs_flash_controller
       Size: 2048 bytes

(4) cfi_flash_0
       Class: altera_avalon_cfi_flash
       Size: 4194304 bytes

Selection: 1

--- Summary using

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

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

Esto genera el archivo ~/nios2-linux/uClinux-dist/linux-2.6.x/include/asm/nios2.h.

```
$ make
```

Si hay algún error, ejecutamos “make” nuevamente. Con “make” se compila todo hasta generar la imagen. Si deseamos hacer una compilación especifica podemos usar: “make romfs” para generar la estructura del directorio romfs que va a contener el sistema de archivos de µClinux; “make linux image” que va a generar la imagen comprimida de µClinux.

Esto construye el kernel comprimido en ~/nios2-linux/uClinux-dist/images/zImage en formato FLT.

3.2.3. Personalizando el Kernel y añadiendo aplicaciones del sistema

El procedimiento de construcción del kernel permite su personalización y la selección de aplicaciones de sistema como boa (servidor web). Los siguientes pasos ilustran la construcción del kernel con algunas funcionalidades soportadas por el hardware de la tarjeta DE2 y por el SOPC hardware del FPGA.

En la consola tecleamos

```
$ make menuconfig
```

Luego en el menú

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

(linux2.6.x) Kernel Version
(None) Libc Version
[ ] Default all settings (lose changes)
[*] Customize Kernel Settings
[*] Customize Vendor/User Settings
[ ] Update Default Vendor Settings
```

Luego salimos y guardamos los cambios con <exit> <exit> <yes>.

Con esto se entrará primero a la configuración del kernel, luego entrará a la configuración de las aplicaciones de usuario.

El menú de la figura 5 es el de la configuración del kernel.

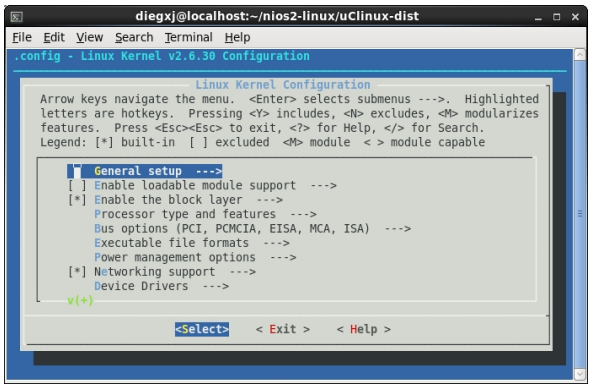

Figura 5. Configuración del kernel

A continuación revisamos las opciones adicionales a las de la configuración por defecto. Para estas y otras configuraciones es recomendable revisar las indicaciones del UClinux en el Altera Wiki [15].

```
[*] Enable loadable module support --->
       [*] Module unloading

[*] Enable the block layer  --->
    Processor type and features --->
       Platform (Altera DE2 Development board
       support) --->
       *** Platform drivers Options ***
       [*] GPIO interface

    Device Drivers --->
       Generic Driver Options --->
       [*] 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
       [*] 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

[*] Block devices --->
    SCSI device support --->
        [*] SCSI device support
        [*] legacy /proc/scsi/ support
        [*] SCSI disk support
        [*] SCSI low-level drivers --->

[*] Network device support --->
        [*] Enable older network device API
        compatibility
        [*] Ethernet (10 or 100Mbit) --->
        [*] DM9000 support
        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
        [*] Keyboards --->
        [*] Mice (NEW)  --->
        <Hardware I/O ports --->
        [*] Altera UP PS2 controller

        Character devices --->
        [ ] Virtual terminal
        <*> Nios LCD 16207 device support
        Serial drivers --->
        [*] Altera JTAG UART support
        [ ] Altera JTAG UART console  support
        [*] Altera UART support
        (4) Maximum number of Altera UART
        ports
        (115200) Default baudrate for
        Altera UART ports
        [*] Altera UART console support

[*] SPI support --->
        [*] Altera SPI Controller

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

[*] USB support --->
        <*> Support for Host-side USB
        [*] USB announce new devices
        [*] USB device filesystem
        [*] ISP1362 HCD support
        [*] USB Mass Storage support

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

[*] LED Support --->
        [*] LED Class Support
        [*] LED Support for GPIO connected
        LEDs
        [*] LED Trigger support
        [*] LED Heartbeat Trigger

File systems --->
    DOS/FAT/NT Filesystems --->
        <*> VFAT (Windows-95) fs support
        (437) Default codepage for FAT
        (iso8859-1) Default iocharset for FAT
    [*] Miscellaneous filesystems --->
        [*] Journalling Flash File System v2
        (JFFS2) support
        (0) JFFS2 debugging verbosity (0 =
        quiet, 2 = noisy)
        [*] JFFS2 write-buffering support
    -*- Native language support --->
        (iso8859-1) Default NLS Option
        [*] Codepage 437 (United States,
        Canada)
        [*] NLS ISO 8859-1 (Latin 1; Western
        European Languages)
```

Salimos y guardamos los cambios <exit> <exit> <yes>.

Esta configuración se guarda en un archivo ~/nios2-linux/uClinux-dist/linux-2.6.x/.config.

El segundo menú es la configuración de las aplicaciones.

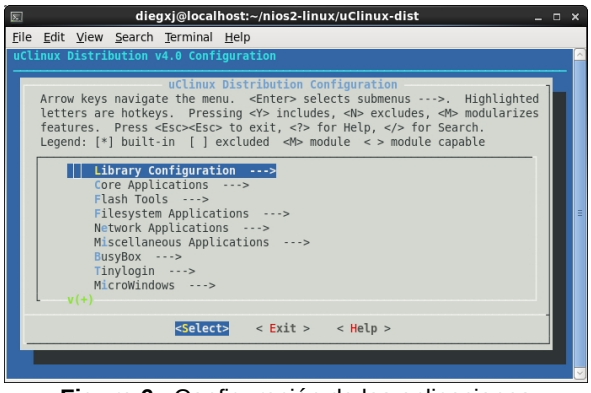

Figura 6. Configuración de las aplicaciones

A continuación marcamos las aplicaciones adicionales a las de la configuración por defecto.

```
Flash Tools  --->
        --- MTD utils
        [*] mtd-utils
        [*] eraseall

Miscellaneous Applications  --->
        [*] gdbserver (old)

Busybox  --->
        Coreutils --->
        [*] chmod
        Process Utilities  --->
        [*] free
        [*] kill
        [*] ps
        [*] Enable argument for wide
        output (-w)
        [*] top
        [*] Show CPU per-process usage
        percentage (NEW)
        [*] Show CPU global usage
        percentage (NEW)
        [*] SMP CPU usage display ('c' key)
        [*] Show 1/10th of a  percent in
        CPU/mem statistics
        [*] Show CPU process runs on  ('j'
        field)
        [*] Topmem command ('s' key)

Miscellaneous Configuration  --->
        [*] Access the NIOS2 Avalon bus
```

Salimos y guardamos los cambios <exit> <exit> <yes>.

Esta configuración se guarda en un archivo ~/nios2-linux/uClinux-dist/config/.config.

Una vez concluida con las configuraciones, ejecutamos nuevamente

```
$ make
```

Al final, una copia de la imagen se guarda en ~/nios2-linux/uClinux-dist/images. El tamaño de la imagen ahora es mayor que con las opciones por defecto.

3.3. Añadiendo hardware nuevo al kernel

Copiamos los archivos fuentes del driver en la carpeta ~/nios2-linux/linux/2.6/drivers/misc. Allí modificamos los archivos *kconfig* y *Makefile* siguiendo el esquema usado para los drivers ya contenidos en la carpeta.

En el archivo ~/nios2-linux/uClinux-dist/vendors/Altera/nios2/romfs_list, agregamos los nombre de nodo que identifican al driver en µClinux.

```
nod /dev/pwmclockdiv 666 0 0 c 242 0
nod /dev/pwmduty 666 0 0 c 243 0
nod /dev/pwmload 666 0 0 c 244 0
```

Figura 7. Información del driver pwm_avalon.c para el romfs_list

Una vez realizadas estas modificaciones, entramos nuevamente al menuconfig y buscamos en la sección correspondiente el nuevo driver. Lo agregamos como módulo <M> o integrado <*>, y salimos guardando los cambios.

Si el driver es agregado como módulo, se deben instalar las aplicaciones *modprobe, rmmod* para poder cargarlo y descargarlo del kernel en modo de ejecución. Estas aplicaciones se pueden encontrar en el paquete de Busybox.

Compilamos con "make" la nueva imagen de µClinux.

La compilación del driver puede hacerse también fuera del configurador del kernel utilizando el compilador cruzado desde la consola [23].

3.4. Programación del target

Para empezar a programar en la tarjeta DE2 se debe tener configurado el driver para el JTAG USB en Linux [24].

Copiamos el archivo .SOF y el zImage en una misma carpeta.

En una sesión del shell de [NiosII EDS] ejecutamos estos comandos desde la carpeta que contiene los archivos.

Para cargar el SOF

```
[NiosII EDS]$ nios2-configure-sof
 DE2_NIOS_HOST_MOUSE_VGA_time_limited.sof
```

Para descargar la imagen del kernel

```
[NiosII EDS]$ nios2-download -g zImage
```

Luego de dar enter, la descarga de la imagen empieza.

3.5. Aplicaciones de usuario

Programar aplicaciones para µClinux es similar a programar para Linux o para Windows, solo hay que tener en cuenta las limitaciones de hardware y de software propias de la configuración y arquitecturas utilizadas.

En Eclipse creamos un proyecto nuevo basado en un Cross GCC. El prefijo en nuestro caso sería "nios2-linux-". El nombre del proyecto es *pwmout*. El linker debe recibir un parámetro que defina el tamaño del stack, ya que sin MMU este no puede crecer automáticamente y dependiendo de la aplicación el tamaño asignado por defecto puede no ser suficiente [28].

```
**** 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 8. *Verbose* de construcción de aplicación pwmout en Eclipse.

3.5.1. Acceso al hardware

El acceso al hardware desde una aplicación de usuario se puede hacer aplicando el atributo *volatile* a un puntero que apunte a la dirección de memoria del registro [25]. Este enfoque es recomendable cuando se diseña el hardware y el software al vez ya que así se garantiza conocer la dirección de memoria a la que esta mapeado el registro del dispositivo de donde se quiere leer y/o escribir.

```
#define na_switch_pio 0x81401120

main() {
...
volatile unsigned *switches = ((volatile
unsigned *)(na_switch_pio));
...
}
```

Figura 9. Extracto de código fuente pwmout.c [29]

Para acceder al hardware a través de un driver no se requiere conocer la dirección de memoria (el driver ya la conoce). La implementación del driver normalmente incluye funciones de lectura y escritura como las que se muestran en la tabla 1.

Tabla 1. Funciones comunes implementadas en drivers del tipo *char device* a nivel de kernel y de usuario

Eventos	Funciones de usuarios	Funciones del kernel
Carga de módulo	modprobe	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo	fread	file operations: read
Escribir dispositivo	fwrite	file operations: write
Cerrar dispositivo	fclose	file operations: release
Quitar módulo	rmmod	cleanup_module

Esto podría ser útil si disponemos de un archivo .SOF, una imagen del kernel y el módulo del driver pre-compilados. Así solo desarrollaríamos la aplicación.

Los drivers para Linux deben ser escritos en C/C++, las aplicaciones pueden ser escritas en cualquier lenguaje.

La documentación necesaria para empezar a desarrollar drivers para Linux/µClinux puede ser encontrada en [26] [27].

3.5.2. Depuración remota de aplicaciones

Se puede configurar Eclipse para depurar aplicaciones por medio del gdbserver por medio de un enlace serie o TCP [28] [29].

Para levantar el gdbserver en el target ejecutamos el siguiente comando:

```
> gdbserver 192.168.1.1:9999 pwmout
```

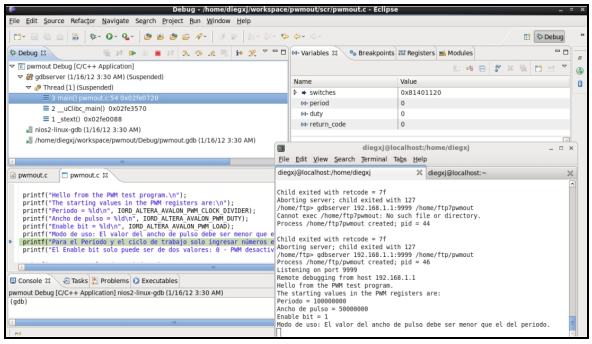

Figura 10. Depuración de la aplicación con Eclipse y gdbserver

4. Resultados

Una vez finalizada la descarga de la imagen, el arranque de µClinux es inmediato si se seleccionó el soporte para consola por Altera UART, para lo cual se debe tener abierta una sesión de un terminal como *minicom* en Linux o *Hyperterminal* en Windows. Por el contrario, si se seleccionó el soporte para consola Altera JTAG UART en el shell de Nios II EDS se debe ejecutar el comando

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

Terminada la secuencia de arranque deberíamos obtener un cuadro similar a la figura 11.

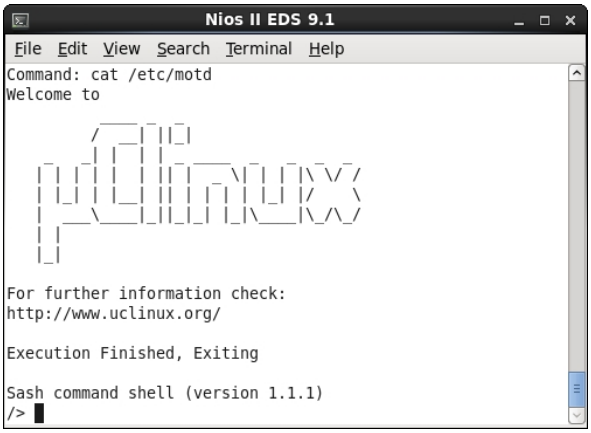

Figura 11. µClinux iniciado en un terminal

4.1. Transferencia de archivos entre el host y el target

La transferencia de archivos puede ser realizada por medio de un enlace FTP o al montar una unidad de red NFS desde µClinux si contamos con un puerto de red en el target. Las aplicaciones que se transfieran a µClinux deben haber sido construidos por medio del compilador cruzado para poder ser ejecutadas.

Para poder usar el servidor FTP en µClinux primero se debe configurar los parámetros de red. Esto se puede hacer agregando al archivo ~/nios2-linux/uClinux-dist/vendors/Altera/nios2/rc las siguientes líneas:

```
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
```

Con esto se cargaran los parámetros durante el arranque de µClinux. Para estar en red con el target el host debe pertenecer a la misma red.

El comando para iniciar el servidor FTP en el target es

```
> inetd & # ejecución en segundo plano
```

El cliente FTP es ejecutado desde el host es

```
$ ftp 192.168.1.2
Connected to 192.168.1.2 (192.168.1.2).
220- Welcome to the µClinux ftpd!
220 µClinux 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>
```

```
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)
```

4.2. Ejecución de aplicaciones

Al archivo deben asignarsele permiso de ejecución por lo que es necesario tener instalada la aplicación chmod. Para ejecutar el archivo tecleamos

```
> ./pwmout
```

Usamos la aplicación gdbserver si queremos ejecutarlo por pasos o depurarlo.

Existe una aplicación que nos permite leer y escribir a direcciones de memoria mapeadas de los dispositivos conectados al bus Avalon. Se llama *nios2io* y se encuentra en el directorio */bin.*

4.3. Monitoreo a nivel de hardware

El uso de SignalTap II como herramienta de monitoreo en tiempo real del hardware es muy útil para la depuración y solución de problemas ya que nos permite verificar si la comunicación entre el hardware y el software se esta dando [30].

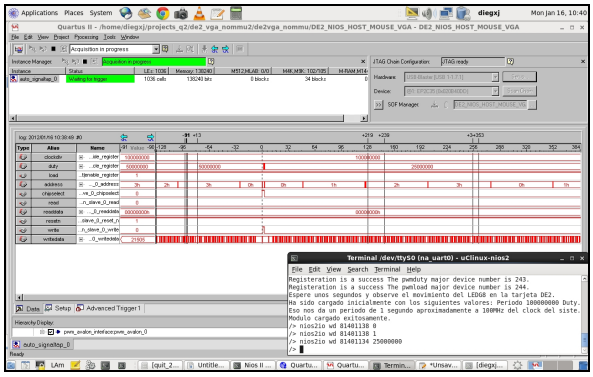

Figura 12. Monitoreo desde el SignalTap II durante el envió de datos por medio de la aplicación nios2io

5. Conclusiones

Al programar el hardware y software de un sistema al mismo tiempo se puede obtener una mayor versatilidad de la aplicación y se da mayor flexibilidad al diseñador. Se puede llegar a un nivel de optimización mayor que con los métodos tradicionales.

En Linux sin MMU se puede aminorar la latencia de la aplicación al acceder directamente al hardware desde el espacio del usuario.

El acceso al hardware a través de un driver toma más tiempo que hacerlo por medio de funciones de acceso directo. El método de acceso a utilizar debe ser escogido considerando los requerimientos de la aplicación.

Se puede salvar tiempo de programación reutilizando aplicaciones y drivers ya existentes para Linux portándolos hacia uClinux al recompilar su código fuente con el compilador cruzado y si se quiere la librería uClibc.

6. Referencias

[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] A C library for embedded Linux, http://uclibc.org/about.html

[5] Mc Cullough, David, µClinux for Linux programmers. www.linuxjournal.com/article.php?sid=7221, Julio 2004

[6] Quartus II Software Support, http://www.altera.com/support/software/sof-quartus.html

[7] Nios II Embedded Design Suite Support, http://www.altera.com/support/ip/processors/nios2/ips-nios2_support.html

[8] Nios II Core Implementation Details, Nios II Processor Reference Handbook, Altera Corporation, Mayo 2011

[9] Processor Architecture, Nios II Processor Reference Handbook, Altera Corporation, Mayo 2011

[10] Programmer united development net, http://en.pudn.com

[11] Opencores, http://www.opencores.org

[12] Avalon Interface Specifications , Altera Corporation, Mayo 2011

[13] SOPC Builder Design Optimizations, Embedded Design Handbook, Altera Corporation, Julio 2011

[14] Altera Corporation, Developing Peripherals for SOPC Builder , Application Note 333 , Marzo 2004

[15] UClinux, http://www.alterawiki.com/wiki/UClinux

[16] Embedded Linux for the Nios II Processor, http://www.altera.com/devices/processor/nios2/tools/embed-partners/ni2-linux-partners.html

[17] Welcome to the NIOS II PREEMPT-RT Project, http://uuu.enseirb.fr/~kadionik/nios2-preempt-rt/

[18] Buildroot: making Embedded Linux easy, http://buildroot.uclibc.org/

[19] Building embedded Linux systems with Buildroot, http://2009.rmll.info/IMG/pdf/buildroot-RMLL09.pdf

[20] Buildroot Guide, www.alterawiki.com/wiki/BuildrootGuide

[21] Install Nios II Linux, http://www.alterawiki.com/wiki/InstallNios2Linux#Toolchain

[22] Working with Kconfig, http://www.rt-embedded.com/blog/archivos/working-with-kconfig/

[23] Module Programming, www.alterawiki.com/wiki/ModuleProgramming

[24] Quartus_for_Linux, http://www.alterawiki.com/wiki/Quartus_for_Linux

[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

[26] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, *Linux Device Drivers*, O'REILLY, Tercera Edición, 2005

[27] Peter Feuerer, *nios2 uclinux and additional hardware*, http://piie.net/index.php?section=nios

[28] Eclipse CDT, http://www.alterawiki.com/wiki/Eclipse_CDT

[29] Diego Jáuregui, Sara Poveda, *Instalación y puesta a punto de una distribución de Linux en una plataforma basada en el procesador configurable Nios II*, ESPOL-FIEC, Enero 2012.

[30] Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems , Aplication Note 323, versión 1.1, Altera Corporation , Noviembre 2007