Arquitectura y Entornos de desarrollo para videoconsolas

Práctica 2
Acceso a información en el hardware de la videoconsola NDS

M. Agustí
Febrero 2017

La práctica se podrá realizar en entorno GNU/Linux en el laboratorio. Vamos a abordar el acceso a la información básica de la consola:como personalización, gestión del tiempo y gestión de la entrada de datos del usuario.

Recuerde que ha de guardar los resultados de sus acciones a lo largo de las prácticas para confeccionar el porfolio de prácticas, que es la forma en que evaluará el trabajo tareas y las prácticas de la asignatura. No descuide pues hacer copias de lo que necesite del laboratorio en su espacio del servidor de la asignatura.

1 Introducción

Trabajaremos a partir de unos ejemplos de código disponibles en la documentación del SDK; *Libnds Documentation* [1]. Para tener disponible el contenido de la documentación en local, descargue desde el sitio web de *devKitPro*[1] el fichero (en el momento de redactar este boletín se llama `libnds-docs-20110214.tar.bz2`).

Instale su contenido en `${DEVKITPRO}/documentacio`, p. ej., asumiendo que se ha descargado el fichero en `/tmp`, con las órdenes:

```
$ mkdir ${DEVKITPRO}/documentacio
```

```
$ (cd ${DEVKITPRO}/documentacio; tar xjvf /tmp/libnds-docs-20110214.tar.bz2 )
```

Acceda a *Libnds Documentation* [1] y compruebe que tiene la misma estructura que lo que acaba de instalar en `${DEVKITPRO}/documentacio.` En concreto, en esta práctica nos vamos a referir a los ejemplos relativos a los apartados de la documentación del SDK de:*System y User Input/Output.*

En el apartado *Examples* de la documentación se referencia a los mismos ejemplos que ya tendrá instalados en su instalación local en `${DEVKITPRO}/examples`. Vamos a ver de estos los relativos a los apartados mencionados

Asumiremos que ya ha instalado las herramientas necesarias en la sesión previa: *devkitPro* para NDS, *libnds* y emuladores (desmume y no$gba).

En el segundo apartado de esta práctica, lo dedicamos a revisar algunas propiedades de la plataforma y utilizaremos algunos ejemplos del API para probar de forma rápida algunas cuestiones relativas a la temática que nos ocupa cuyo uso tiene una gran complejidad. También nos servirán para ir recogiendo en el camino, elementos que nos pueden ayudar en otros desarrollos.

El tercero se centrará en revisar los ejemplos disponibles de cómo se accede al uso de los controles del dispositivo, que ya habrá aparecido al revisar los ejemplos en el segundo apartado.

2 Cuestiones básicas

Veamos aquí cómo acceder a informaciones muy básicas como parámetros de configuración del dispositivo y gestión del tiempo.

2.1.1 Acceso al sistema

En este apartado revisamos parte de la sección *System* de la documentación. Nos vamos a centrar sólo en cuestiones relativas a los parámetros de configuración ("Hardware Initilization") y al uso de temporizador como cuestiones de bajo nivel relativas a este sistema. Además, en este apartado de la

1En Sourceforge: <http://sourceforge.net/projects/devkitpro/files/libnds/>.

documentación, podemos encontrar algunas definiciones de interés relativas al contenido de la NDS hardware de la consola: definiciones de tipos de datos, acceso a funciones optimizadas (BIOS), interrupciones , operaciones para gestionar la caché del ARM9 y comunicación entre los procesadores (FIFO).

Respecto a la configuración de la máquina, podemos acceder a una serie de datos básicos de identificación que se guardan en la memoria no volátil de la misma y que pueden utilizar las aplicaciones que se ejecutan en ella. Se recogen en la estructura de datos denominada tPERSONAL_DATA descrito en `system.h`:

struct tPERSONAL_DATA __attribute__ ((packed)) PERSONAL_DATA

User's DS settings. Defines the structure the DS firmware uses for transfer of the user's settings to the booted program.

((PERSONAL_DATA)0x2FFFC80) Default location for the user's personal data*

#define PersonalData

Para acceder a este contenido de la información de la consola y en el modo que proponeestructura de datos se sugiere utilizar el código del listado 1, sacado del `arm9_main.cpp` de `brightness_test.zip`2[2]. El código muestra otros elementos de la información de la consola que se omite aquí por brevedad del boletín de prácticas.

```
/* Initialization done. Print welcome message. */
char name[11] = {0};
for(int i=0; i<PersonalData->nameLen; i++)
name[i] = (char)(PersonalData->name[i] & 255); // get ascii-bits from utf-16 name
```

Listado 1: Acceso a `PersonalData` en `brightness_test.zip`.

Otras funciones también disponibles en este mismo fichero, `system.h`, que pueden ser de interés son:

- lcdMainOnBottom (void)
- lcdMainOnTop (void)
- lcdSwap (void)
- ledBlink (int bm)
- systemShutDown (void)
- systemSleep (void)

2 Aplicación disponible en <http://library.dev-scene.com/index.php?dir=DS/Developers/>.

Ejercicio: Utilizando el ejemplo de *¡Hola, mundo!* de la práctica anterior o el que se encuentra en `${DEVKITPRO}/examples/templates/arm9` construya una aplicación para mostrar toda la información que contiene la consola y que está accesible desde la definición de `PersonalData` en `system.h`. Utilice las secuencias de códigos de escape ANSI [2] para posicionar la información en pantalla y dar colores diferentes al nombre de la propiedad y a su valor.

2.1.2 Funciones relacionadas con el paso del tiempo

Revisemos lo que nos proporciona el SDK relativo a este tema con los ejemplos que incorpora.

Medir el paso del tiempo (stopwatch)

El ejemplo `time/stopwatch`, véase fig. 1, tiene una salida de modo texto que muestra el número de tics de reloj que han pasado, así como su correspondencia en segundos. El núcleo de este ejemplo es la instrucción:

```
timerStart(0, ClockDivider_1024, 0, NULL);
```

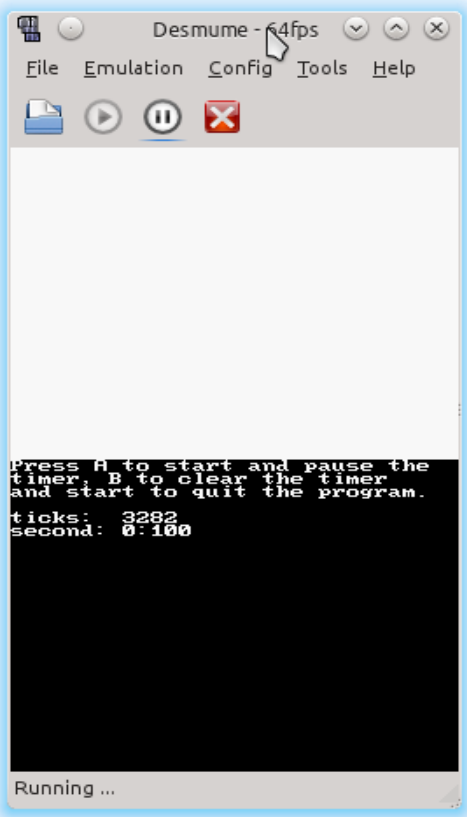


Figura 1: Un instante de la ejecución de stopwatch.

Ejercicio: ¿Cual es la definición (prototpipo) de esta función? Anote su descripción Busque también la definición de los argumentos segundo (`ClockDivider_1024`) y tercero (0)

utilizados.

Función periódica (timercallback)

El ejemplo `time/timercallback` no tiene una salida gráfica ni de texto. Espera a que se pulse el control `A` y, mientras, llama a una función que se encarga de hacer sonar o parar, alternativamente, un sonido previamente generado. El núcleo de este ejemplo es la instrucción

```
timerStart(0, ClockDivider_1024, TIMER_FREQ_1024(5),
           timerCallBack);
```

Ejercicio: ¿Qué significa la función que se le pasa como tercer parámetro ? ¿Cual es la definición (prototpipo) de la función, el cuarto parámetro, que puede ser llamada por el *timer*? ¿Cómo se le pasa/recoge información a/desde esa función ?

Volveremos sobre este ejemplo en el caso de la práctica de audio puesto que tiene una sencilla instrucción para sintetizar sonidos "simples":

```
soundPlayPSG(DutyCycle_50, 10000, 127, 64);
```

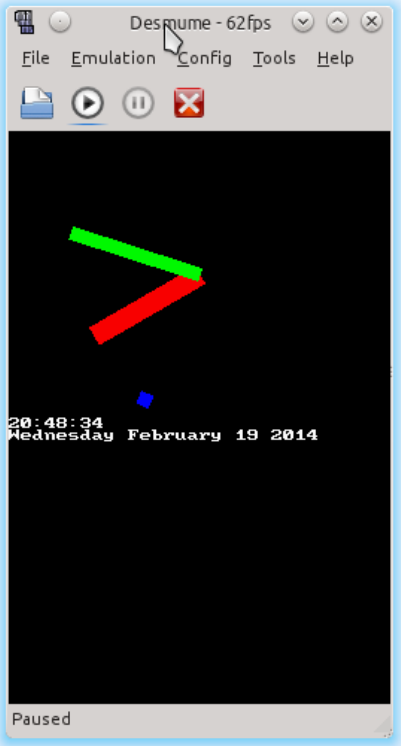


Figura 2: Un instante de la ejecución de "RealTimeClock.nds" sobre DesMuME,

¿Qué hora es? (RealTimeClock)

No tanto por la complejidad de las funciones relativas al tiempo, que son las clásicas, sino por el uso de funciones de dibujo 3D, véase fig. 2, sobre las que volveremos en la práctica de imágenes. El núcleo de este ejemplo es la instrucción

```
timerStart(0, ClockDivider_1024, TIMER_FREQ_1024(5),
           timerCallBack);
```

Ejercicio: ¿Qué dos funciones se utilizan para averiguar la hora y la fecha? Anote sus prototipos y una breve descripción. ¿Cómo inicializa este ejemplo el uso de las dos pantallas de la consola? Anote los nombres y dónde se encuentra la implementación de las funciones encargadas.

3 Gestión de la entrada de usuario

El apartado de la documentación *User Input/ouput* aborda el uso de los controles, un teclado en pantalla y las funciones básicas de configuración de una pantalla en modo texto (consola).

Ejercicio: Ejecute los ejemplos siguientes, tomando una captura de pantalla y anotando su funcionalidad. Los ejemplos se encuentran en `${DEVKITPRO}/examples/input/` y son:

- `keyboard/keyboard_async`
- `keyboard/keyboard_stdin`
- `Touch_Pad/touch_area/`
- `Touch_Pad/touch_look/`
- `Touch_Pad/touch_test/`

4 Trabajo autónomo

Ahora que conoce la operativa de acceso a los controles y ha visto varios ejemplos, construya un único ejemplo de código que permita, moviéndose entre menús de texto (utilizando las flechas) y las teclas A (para aceptar una opción) o B (para volver al menú principal) para poder escoger cual de las informaciones que guarda la consola o de las funciones enunciadas en el apartado 2.1.1 se quiere ver o ejecutar.

Como referencia se aconseja estudiar el ejemplo disponible en el fichero `source/main.cpp` del ejemplo `${DEVKITPRO}/examples/Graphics/Backgrounds/all_in_one` y emplear las dos pantallas para mostrar el menú en una y la ejecución de la opción elegida en la otra.

Para completar esta tarea, se comprobará el uso de un entorno de desarrollo, Code::Blocks, como se describe en el Anexo I. Incluya la descripción del sistema utilizado y con qué versiones ha trabajado, junto a la captura de pantalla de la ejecución del ejemplo en que se vea su nombre en el ejecutable y en el código. Guarde junto al código obtenido, el proyecto de Code::Blocks. Si no fuera posible, describa con detalle el paso que ha generado el problema.

Para conocer más detalles de cómo se utilizan y asignan los recursos de la máquina, utilice las aplicaciones web que se describen en el Anexo II para comprobar la asignación de bancos y memoria del sistema a esta aplicación. Guarde las capturas de pantalla que muestran esta comprobación y describa cómo las ha obtenido.

5 Bibliografía

[1] Libnds Documentation, Disponible en la URL <http://libnds.devkitpro.org/index.html>.

[2] Colaboradores de Wikipedia. *Código escape ANSI* [en línea]. Wikipedia, La enciclopedia libre, 2013 [fecha de consulta: 19 de febrero del 2014]. Disponible en <http://es.wikipedia.org/w/index.php?title=C%C3%B3digo_escape_ANSI&oldid=70047232>.

[3] Santofimia Romero, M. J., Fernández Moya, F. J. (2011). Laboratorio de estructura de computadores empleando videoconsolas Nintendo DS. Bubok Publishing. ISBN 978-84-

9981-039-3

[4] NioZero. (2008) Tutorial: Codeblocks + DevKitPro. <http://niozero.blogspot.com.es/2008/08/tutorial-codeblocks-devkitpro.html>

[5] Roi de Janeiro. (2014). How to use DevkitPro with Code::Blocks (+ code-level-debugging). <https://whatroidoes.wordpress.com/2014/01/09/nds-development-tutorial-how-to-use-devkitpro-with-codeblocks-and-debug-with-insight/>,

Anexo I Code::Blocs y DevkitPro

Habitualmente, en esta asignatura, realizaremos nuestros desarrollos a partir de la línea de órdenes para tener una idea clara de qué pasos estamos dando en cada momento. Además, de la misma manera que compilamos con la orden *make* y el fichero *Makefile* es el descriptor de nuestro proyecto, es fácilmente transportable a otra plataforma y seguir utilizando estas mismas herramientas y de la misma forma.

Podemos también realizar una depuración de una aplicación para NDS casi igual que la haríamos para una aplicación típica de escritorio, solo que:

1. Ahora la aplicación ha de ser ejecutada sobre un emulador, puesto que la arquitectura del computador al que va dirigida no es la del computador de desarrollo.
2. El interfaz del depurador utiliza un protocolo preestablecido para comunicarse con esta aplicación "remota", enviando las opciones que el usuario escoja para guiar la depuración por el flujo de ejecución de la aplicación y recibiendo los resultados estas mismas órdenes generen.

Compilar una aplicación devkitARM desde Code::Blocks

Se puede empezar creando un proyecto desde cero con la opción de menú *File* | *New* | *Project*, con lo que aparece la caja de diálogo *New from template*, donde se habrá de escoger el tipo de proyecto (*console application*) y aceptar con el botón "*Go*". A partir de aquí el asistente nos guiará por las opciones de cada plantilla. En nuestro caso hemos seleccionado (véase fig. 3) "Console application" y, en los siguientes pasos, escogeremos lenguaje de programación, directorio donde se ubicarán los ficheros y la configuración para las versiones de desarrollo (*Debug*) y de entrega (*Release*).

Es necesario cambiar el compilador para la plataforma para la que vamos a desarrollar: la NDS basada en ARM. Para ello es necesario ir a *Settings* | *Compiler and debugger ...* y en *Global compiler settings* se puede cambiar el "Selected compiler" seleccionando uno que exista haciendo una copia con el botón que a tal efecto está bajo esta opción, vése fig. 4. La configuración de este "nuevo" compilador se completa con:

- El cambio de la ruta del *Compiler's instalation directory* para el que hemos aprovechado la variable de entorno que se nos sugiere al instalar *devkitPro*.
- El cambio de los *Toolchain executables* especificando los existentes en la instalación de devkitARM, sin explicitar el subidrectorio *bin*, como dice la nota bajo la caja de texto.

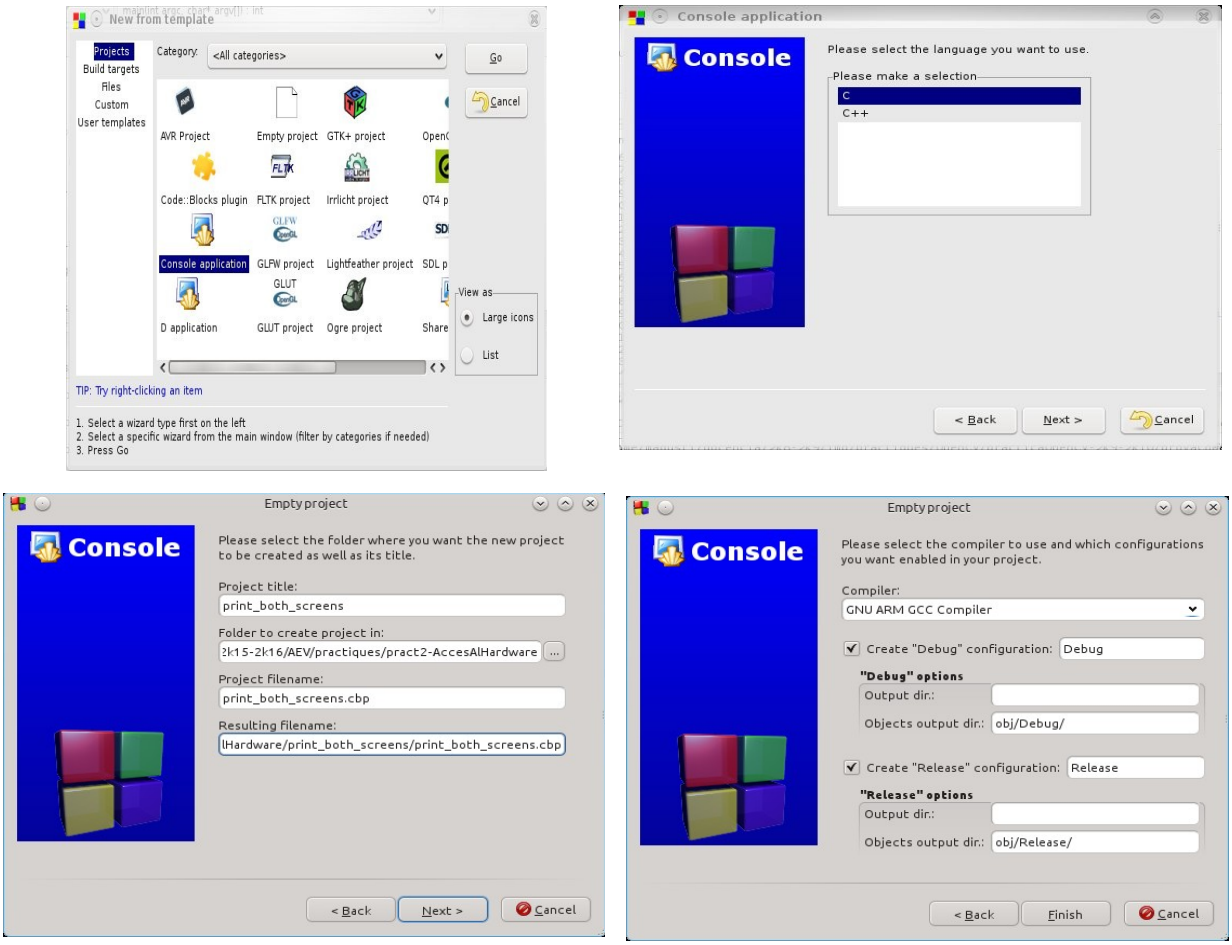


Figura 3: Pasos del asistente de CB para la creación de un nuevo proyecto.

Ahora se podría configura el entorno de desarrollo indicando los parámetros para el compilador, las cabeceras, las bibliotecas de enlace, … pero dado que hay otras etapas en la generación del código binario final y aprovechando que tenemos un fichero *Makefile*, lo vamos a reutilizar para compilar desde *Code::Blocks*.

Para compilar[3] hay que indicarlo mediante el menú *Project* | *Properties*. En la caja de diálogo que aparece, véase fig. 5, hay que seleccionar la opción "*This is a custom makefile*" y comprobar que el nombre del mismo aparece correctamente. Observe como se ha asignado el "*Execution directory*" con el contenido de la variable *PROJECT_DIR*. Así mantendremos todo lo posible el modo de funcionamiento que veníamos utilizando hasta ahora.

Si puede ser, mantendremos la generación de dos versiones de la compilación para que ejecute una acción u otra después: depurar en el caso de *Debug* y ejecutar en el emulador en caso de *Release*. Podría esta última ser ampliada también con *ds2link, ftp* o similar para enviarla a la consola a través de la conexión *Wi-Fi*.

3 Code::Blocks and Makefiles": <http://wiki.codeblocks.org/index.php/Code::Blocks_and_Makefiles>

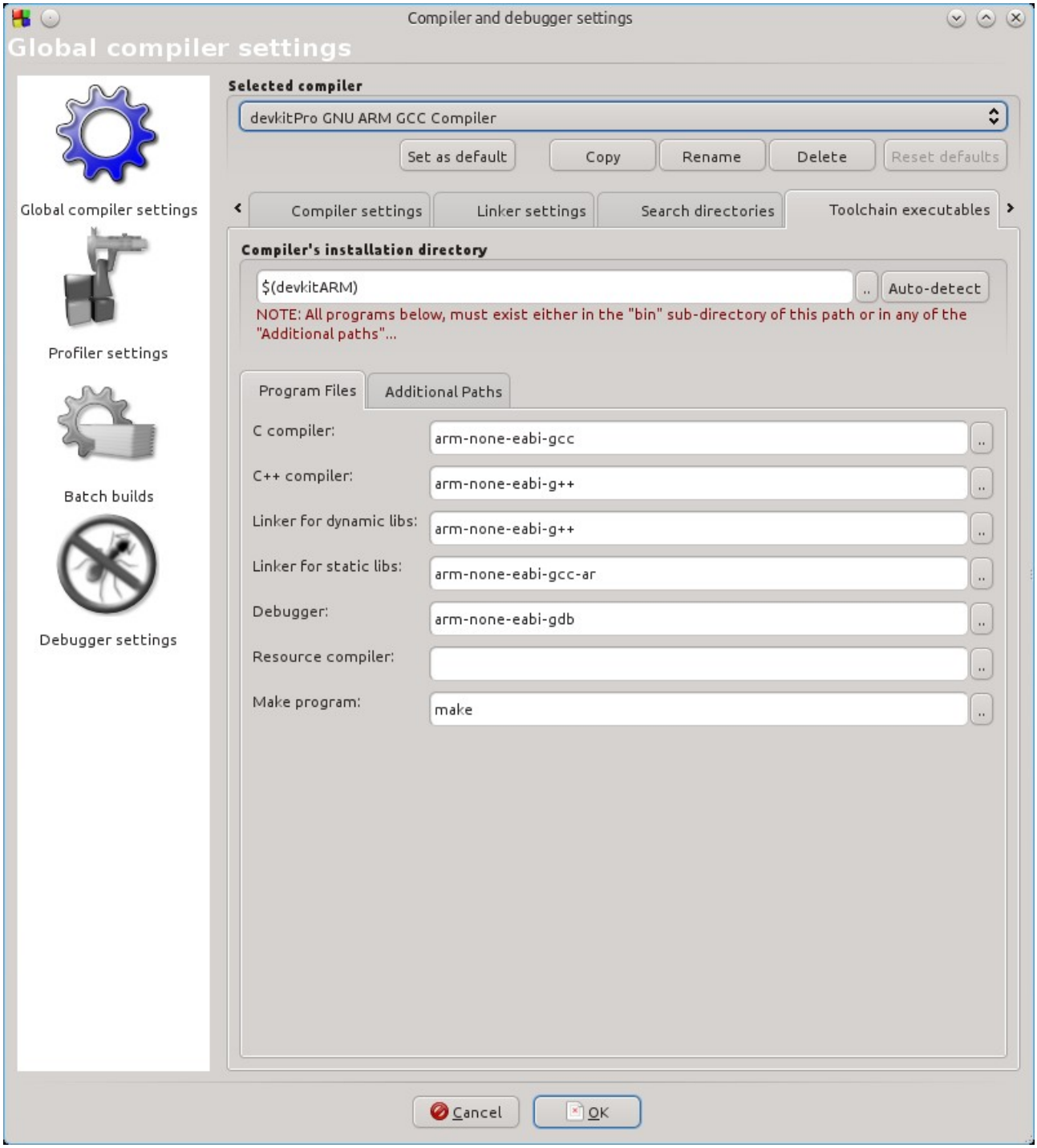


Figura 4: Configurando el "nuevo" compilador para NDS,

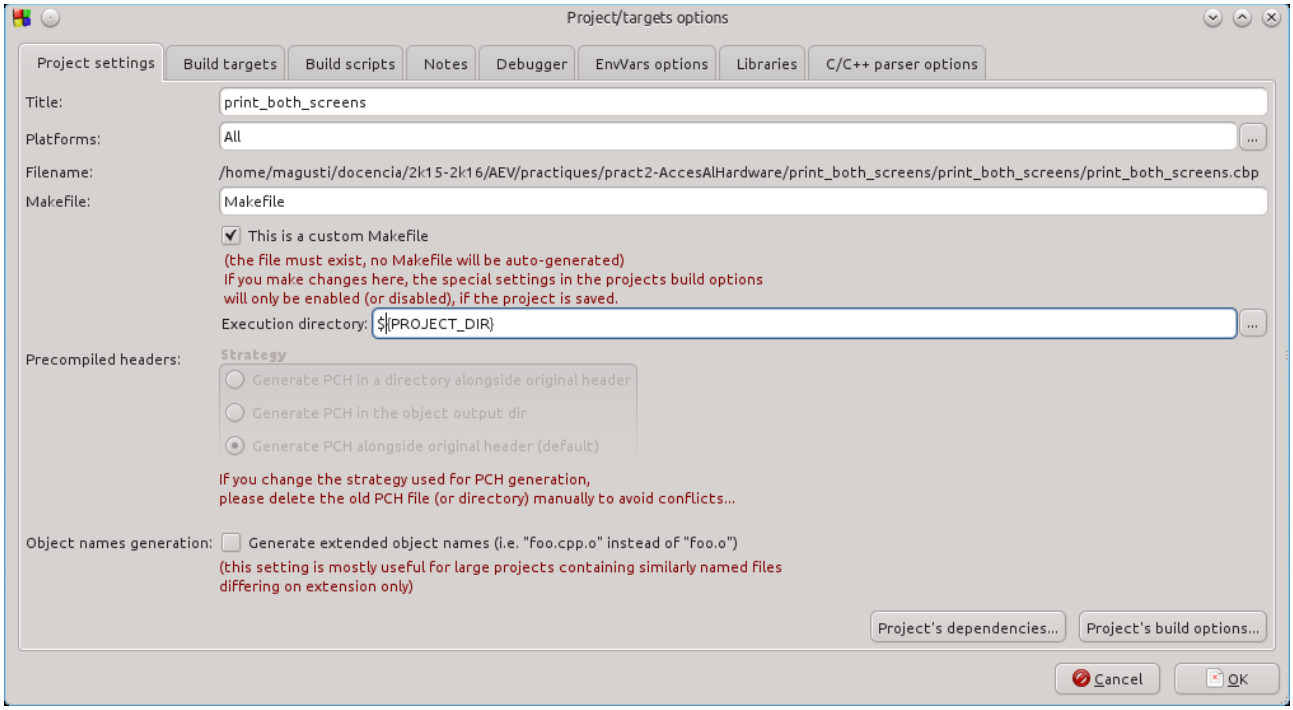


Figura 5: Aprovechando el Makefile para compilar desde Code:Blocks.

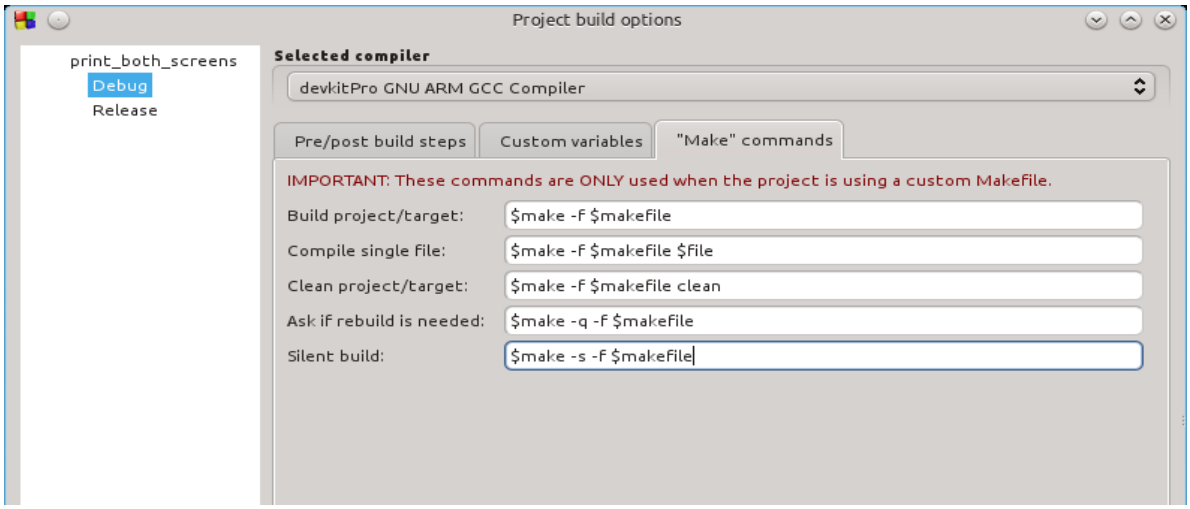
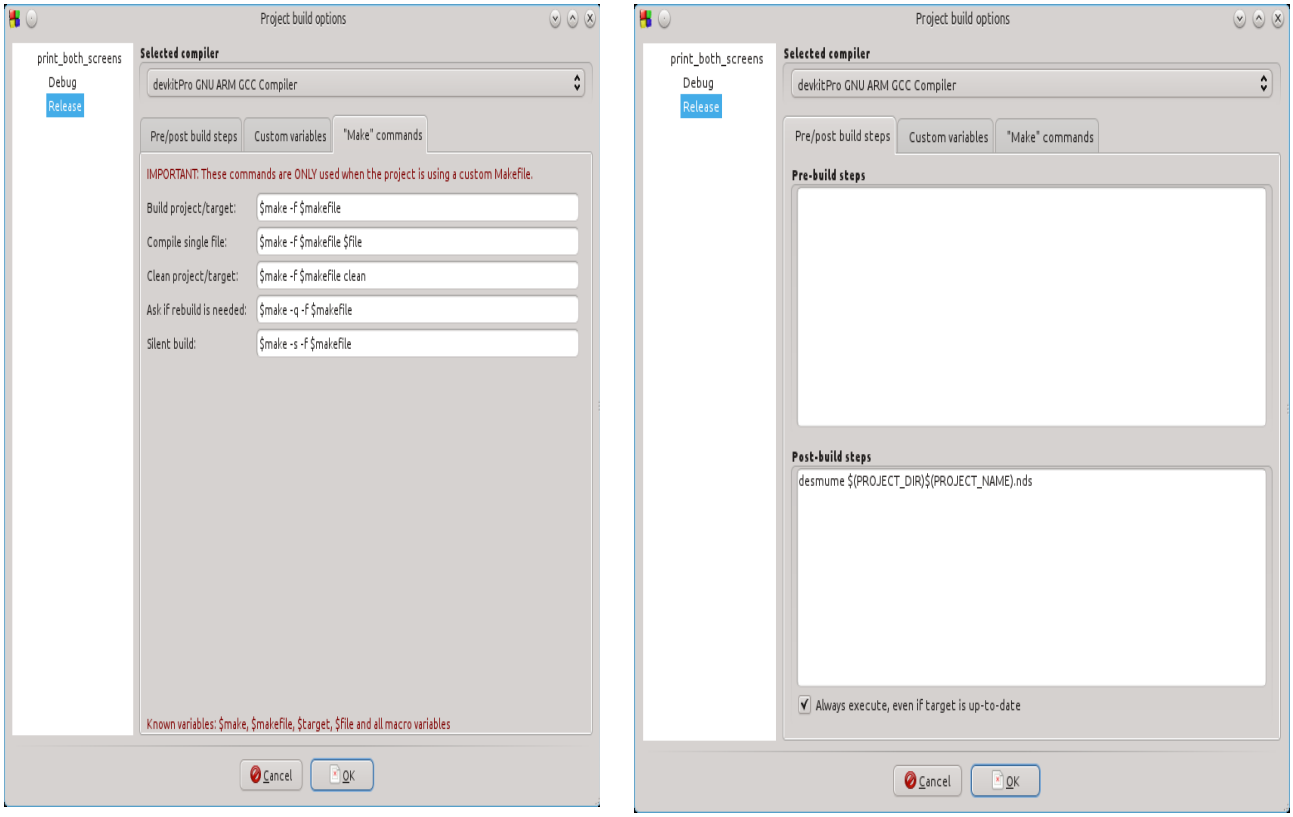


Figura 6: Reescribiendo las órdenes para el compilador asignado al proyecto de NDS en el perfil Debug.

Ahora para aprovechar el *Makefile* original hay que instruir a Code::Blocks sobre como invocar a este Makefile en las diferentes variantes que permite utilizar el entorno (compilar el proyecto, un fichero, recompilar, eliminar todos los archivos binarios creados, …) se deberá revisar la opción de menú "*Project | Build options* ...". Por ejemplo, la fig. 6 muestra como se ha modificado la línea de órdenes para que la versión de depuración se genere con los símbolos de ídem.

Ejecutar una aplicación con DesMuMe desde Code::Blocks

Asegúrese de revisar que también los otros perfiles existentes (como el de *Release* en este caso) también han actualizado el compilador a utilizar y las órdenes, fig. 7a. Antes de salir de esta caja de diálogo haremos una última modificación. En la pestaña *Pre/Post build steps* podemos configurar los *Post-build steps* para que ejecuten automáticamente el emulador con el resultado de la compilación, indicando la orden que teclearíamos en el terminal adaptada a la sintáxis de *Code::Blocks* para que se configure a cada proyecto, como se muestra en la fig. 7b. Con lo cual compilará y ejecutará el simulador al acabar, correctamente, la compilación



a) *b)*

Figura 7:Reescribiendo las órdenes para el compilador asignado al proyecto de NDS en el perfil Release.

Lo que se puede reflejado en el panel de salida, fig. 8, que muestra la ejecución de estas órdenes. Si el panel de salida no muestra las órdenes que el make ha llevado a cabo, compruebe que está activada desde el menú *Settings* | *Compiler and debugger settings* la opción de *Compiler logging* a *Full command line*, fig. 9.

Si el panel de salida, al hacer " Build" muestra un error como el de la fig. 10, es por qué se han configurado mal las opciones de creación de los perfiles Debug y/o Release. La fig. 11 muestra un caso típico en que se ha ido a modificar la orden a ejecutar para hacer el *Build* y se ha omitido el cambio del compilador seleccionado. ¡Por eso dice que es un compilador inválido!. Los valores correctos los hemos visto en las figuras 6 y 7.

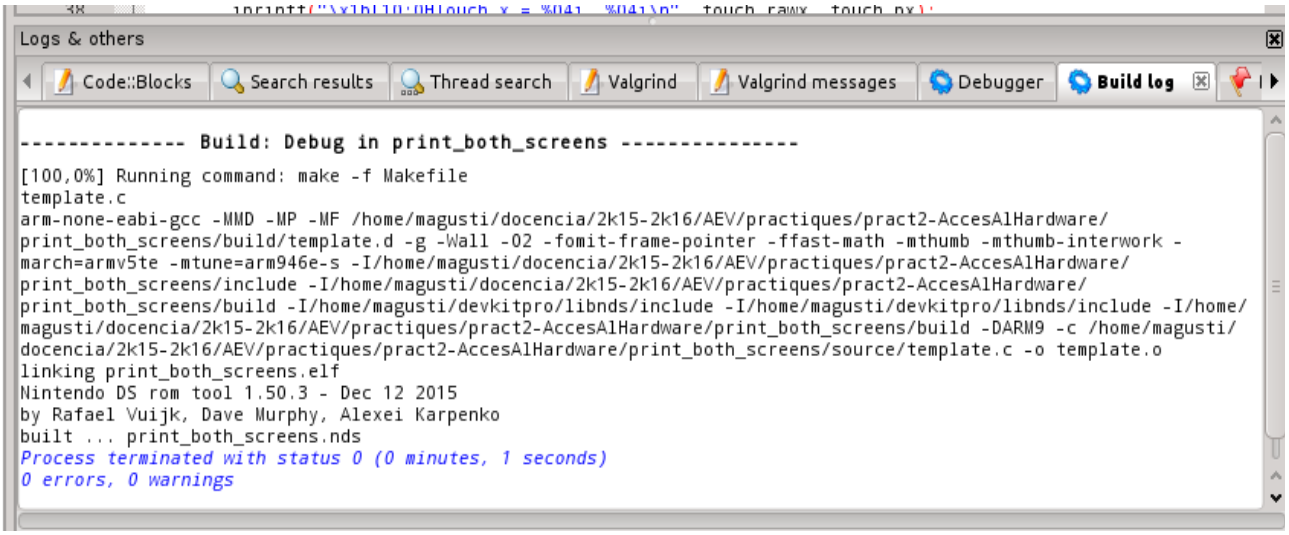


Figura 8: Panel de salida o "Logs & others" de Code::Blocks.

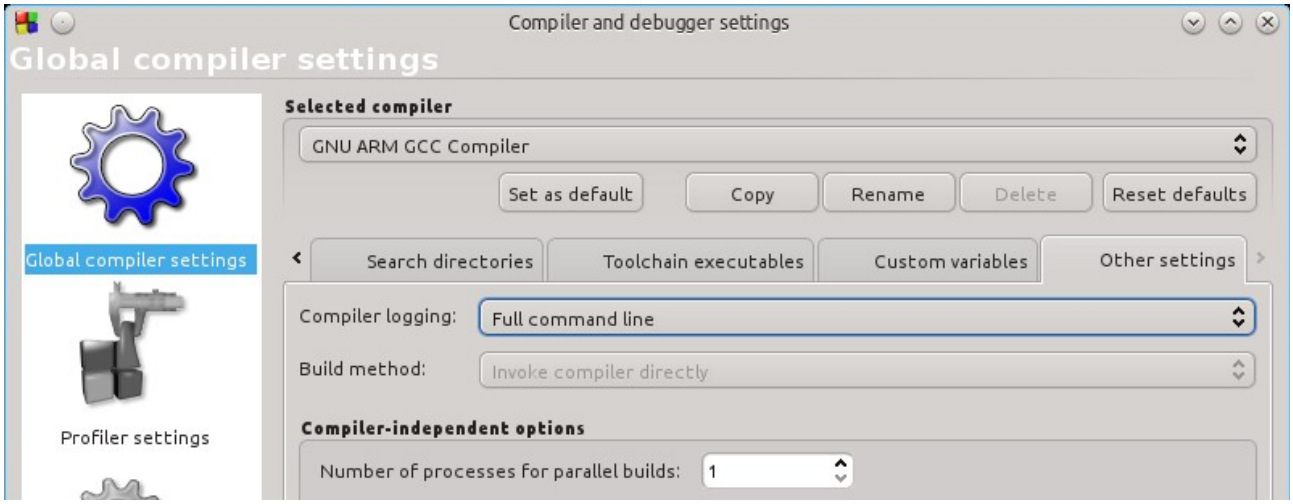


Figura 9: Habilitar la visualización de la orden de compilación.

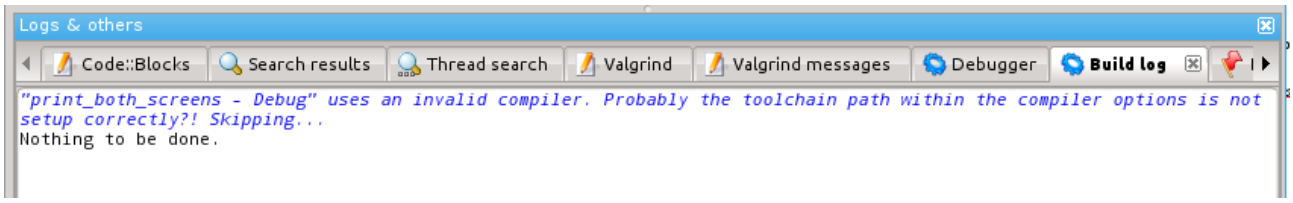


Figura 10: Error del proceso de creación del ejecutable.

Pruebe a compilar el ejemplo habiendo activado el perfil de *Release* y obtendrá un resultado similar al de la fig. 12, en el cual hemos desplazado la ventana del emulador sobre la descripción del proyecto para que se pueda observar el resto de salidas obtenidas.

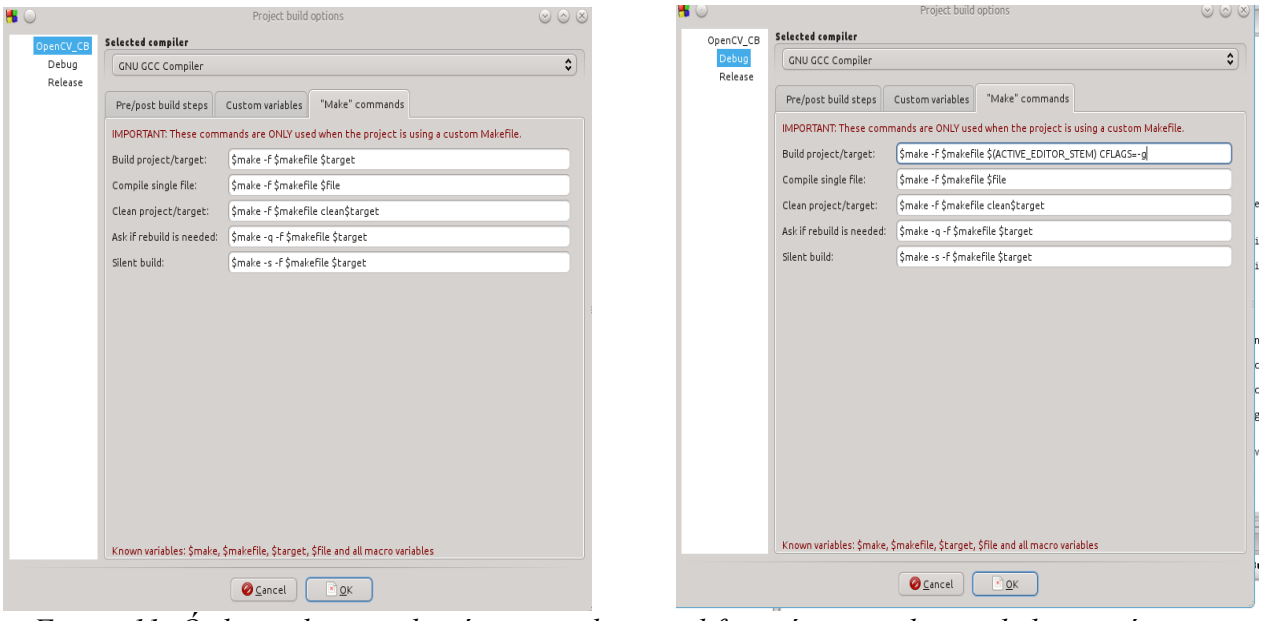


Figura 11: Órdenes de compilación con make y modificación para el caso de la versión para depuración.

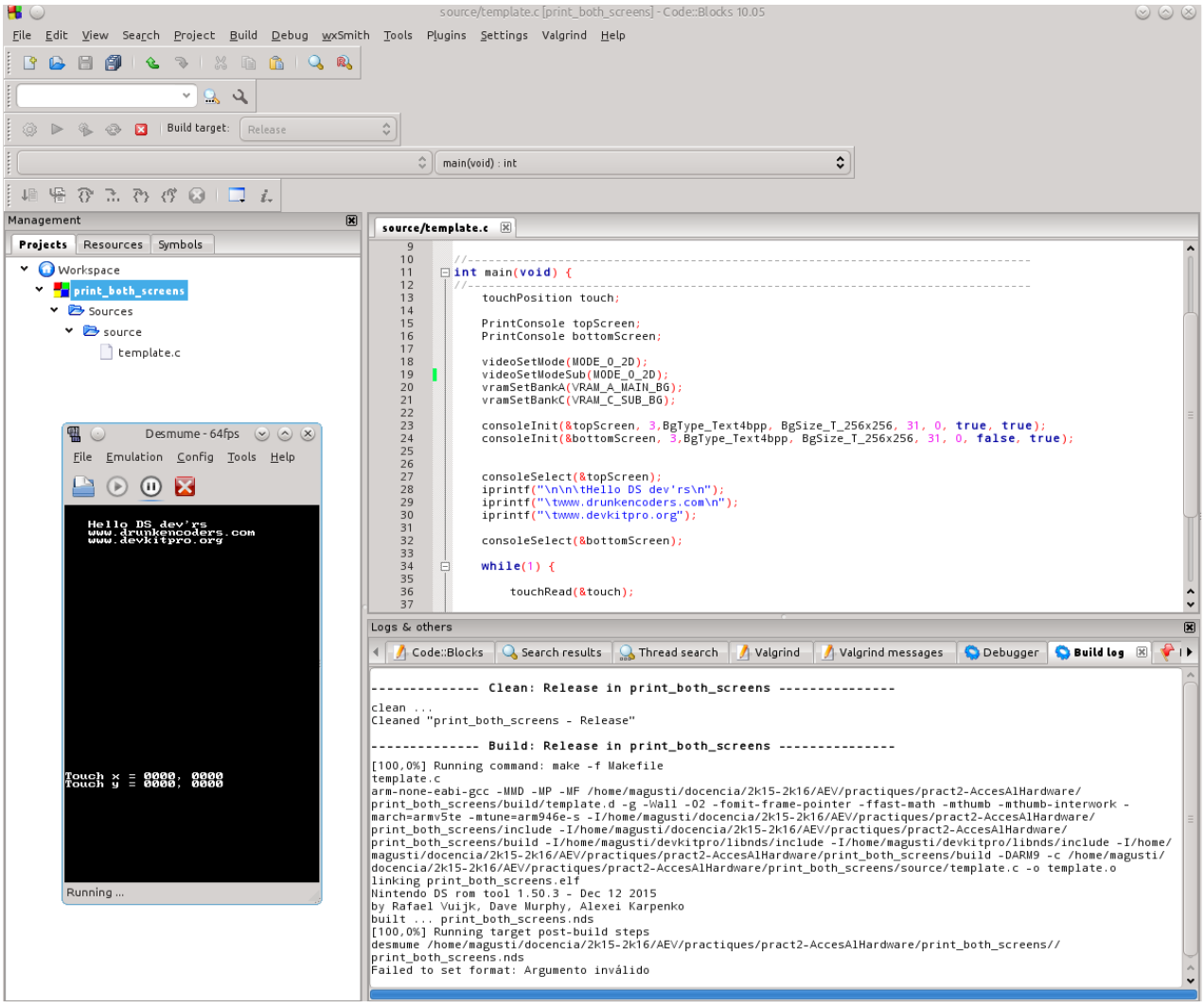


Figura 12: Versión Release en funcionamiento.

Depurar una aplicación con DesMuMe desde Code::Blocks

Si se conoce el repertorio de órdenes disponibles para el depurador de ARM (una variante del *GDB*[4]) podemos ya ampliar nuestra colección de herramientas para desarrollar incluyendo la depuración. Así, en la línea de órdenes, en dos terminales diferentes, podemos emplear [3] la pareja de órdenes:

```
$ desmume-glade --arm9gdb=4444 print_both_screens.nds
$ ${DEVKITARM}/bin/arm-none-eabi-gdb -q -ex "target remote :4444" print_both_screens.elf
```

Lo que nos permitiría ver la ejecución de *printf_booth_screens.nds* que está en el apartado *Graphics* de los ejemplos de NDS de las distribución de devkitPro. El puerto indicado puede ser cualquiera de los que estén libres en un sistema. Podemos averiguar los puertos asignados en una máquina bajo Linux y Windows, al menos, con la orden

```
$ netstat -an
```

Así como también comprobar ese número de puerto en concreto que queremos utilizar con otras como

```
$ nmap localhost -p 4444 | grep -i tcp
$ nc -zv  localhost  4444
```

También se puede recurrir a un interfaz gráfico. Lo cual es además aconsejable para no tener que memorizar la lista de órdenes que acepta el depurador y su sintaxis. La instalación del *devkitPro* para *MS/Windows* instala *Insight*, un interfaz gráfico para *GDB* sobre *Tcl/Tk*, que no se instala en la versión de *GNU/Linux*. Aquí podemos utilizar herramientas como *kdbg* o un entorno de desarrollo integrado como *Eclipse* o *Code::Blocks*.

Veamos cómo adaptar Code::Blocks, que es más ligero e igual de portable que *Eclipse* para compilar con *make* y depurar en remoto (con el depurador de ARM de *devkitPro*) contra un emulador de *NDS* como *DesMuMe.*

4 Se puede consultar en local y en la red en la url <http://sourceware.org/gdb/current/onlinedocs/gdb/>.

Anexo II Bancos de memoria

Escoger la asignación de bancos de memoria para los diferentes elementos de una aplicación para la NDS es un tema complejo. A continuación, se presentan dos herramientas que pueden ayudar a tomar la mejor decisión:

- *NDS Homebrew VRAM Banks Selector* <https://mtheall.com/banks.html#>.
  Nos va a permitir ver de forma grafica, como asignamos los modos de funcionamiento a cada pantalla y el numero de fondos (*backgrounds*) y características de estos. Observe que se va generando una lista de funciones que escribir en el código para obtener el resultado que se va escogiendo con los desplegables o bien indicaciones de error.

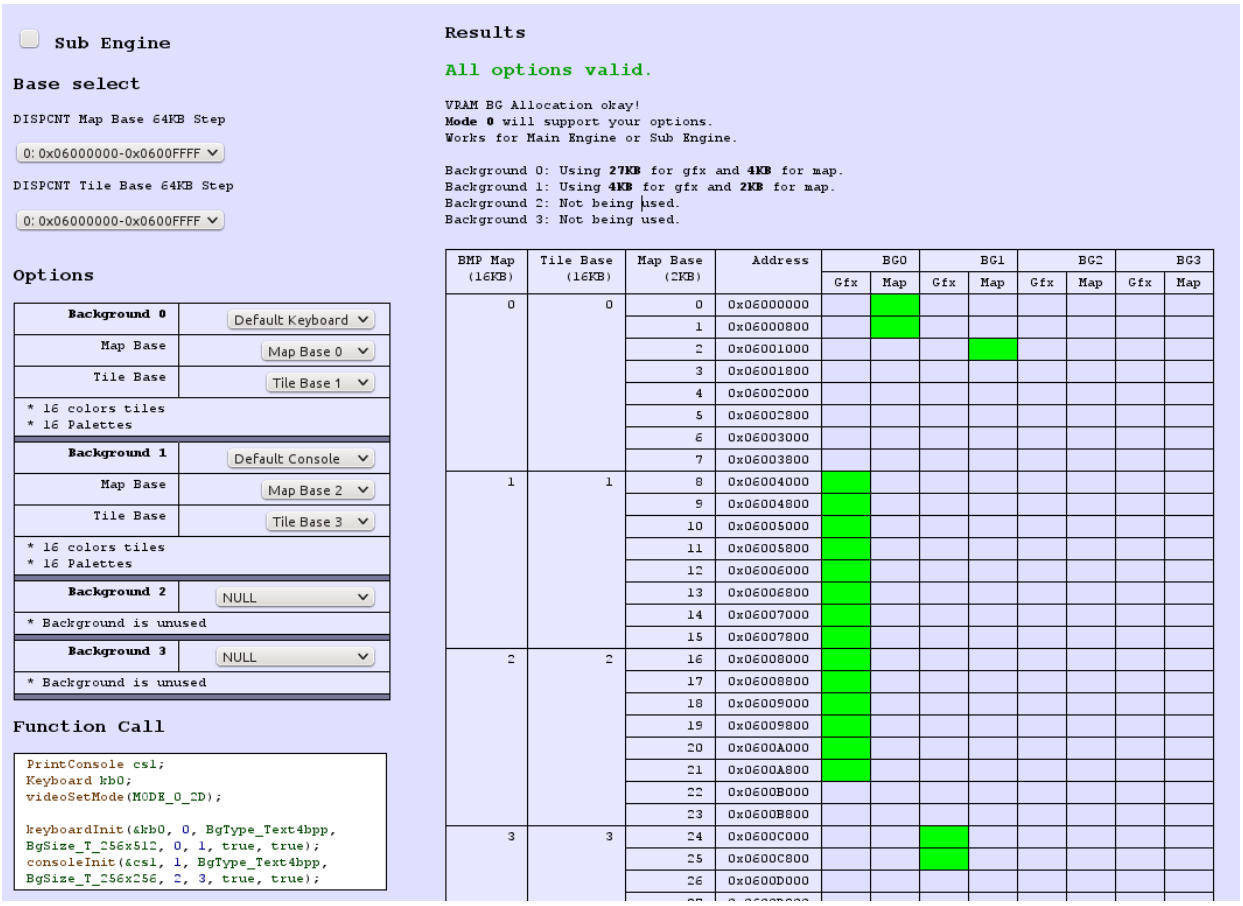


Figura 13: Configuración de vídeo del motor principal.

La fig. 13 muesta un posible caso de configuración del motor gráfico principal para albergar un teclado en el BG0 y un modo teselado con 16 *tiles* y 16 paletas.

- *NDS Homebrew VRAM BG Allocation Conflict Viewer* <https://mtheall.com/vram.html#>.
  Esta nos ayudara a definir los modos de inicialización de las pantallas, con un determinado numero de fondos (*backgrounds*), mapas y teselas (*tiles* o *gfx*).
  La fig, 14 muestra un posible ejemplo de cómo el motor principal tendría mapeado en memoria de vídeo sus componentes, la memoria disponible y si hubiera algún conflicto de selección repetida de un banco de memoria aparecería en rojo la casilla que lo identificaría.

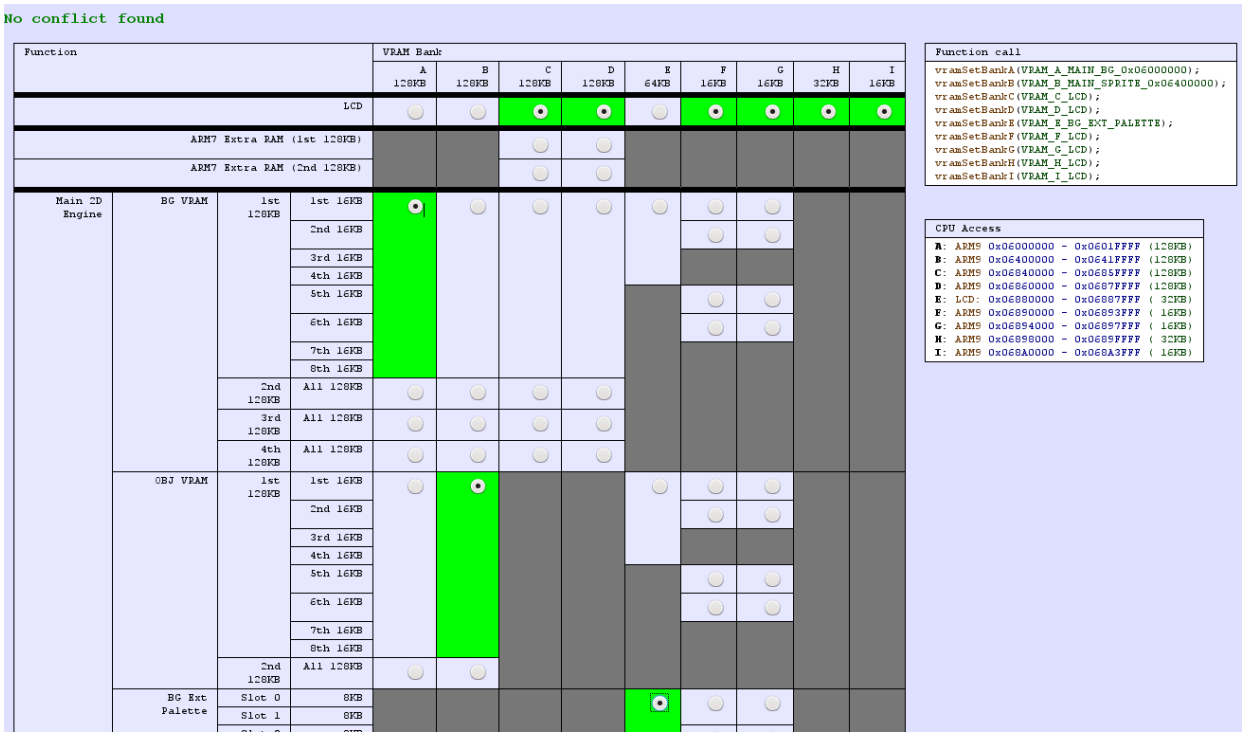


Figura 14: Una posible asignación de los bancos de memoria a los componentes del motor principal.