DIRECTFB 1.4.5 PHASE 1.5

Software Users Guide

Version 1.0

220 Bristol Business Park, Coldharbour Lane, Bristol, BS16 1FJ, UK Phone: +44 1179062700 •

REVISION HISTORY

Revision Number	Date	By	Change Description
1.0	18th March 2011	Rob McConnell	Initial Version

REFERENCES

Reference	Description	Version/ Date
1	*DirectFB-1.4.5_Phase1.5_Feature_List.pdf*	A8
2	*BroadcomReferencePlatformSetup.pdf*	STB_Platform_SWUM101-R
3	BrutusIntallationGuide.pdf	STB_Brutus_SWUM202-R
4	*Nexus Usage Guide*	STB_Nexus-SWUM204-R
5	*Nexus Architecture Guide*	STB_Nexus-SWUM104-R
6	*Nexus Development Guide*	STB_Nexus-SWUM302-R
7	*http://www.directfb.org*	N / A

TABLE OF CONTENTS

LIST OF TABLES

1 INTRODUCTION

1.1 OVERVIEW

DirectFB stands for Direct Frame Buffer. "DirectFB is a thin library that provides hardware graphics acceleration, input device handling and abstraction, integrated windowing system with support for translucent windows and multiple display layers, on top of not only the Linux Frame buffer Device.

It is a complete hardware abstraction layer with software fallbacks for every graphics operation that is not supported by the underlying hardware. DirectFB adds graphical power to embedded systems and sets a new standard for graphics under Linux. (See http://www.directfb.org for more details).

This document describes how to install, build and run DirectFB-1.4.5 on a Broadcom DTV/set-top reference platform.

1.2 AUDIENCE

This document is aimed for individuals who have an engineering background and already know how to build the standard reference software for a STB/DTV platform. This document assumes the user is familiar with the standard reference software tools such as “make”.

1.3 PREREQUISITES

You must have the following before building and running DirectFB on a reference platform:

- A Broadcom reference platform (DTV or Set-Top) and USB keyboard / mouse to connect to it.
- A host PC or build server upon which to install the DirectFB source code and build it. It must have the Broadcom MIPS cross-compilation tool chain installed.
- A DHCP server running on the same network as the reference platform is connected to.
- Knowledge of the “vi” UNIX editor to be able to edit text on the reference platform.
- Bash shell to execute the installation and build steps on the host / build server.

1.4 MAIN CHANGES FROM DIRECTFB-1.4.5 PHASE 1.0

DirectFB-1.4.5 Phase 1.5 adds support for many of the new 40nm chips, such as the 7422, 7425, 7231, 7344 and 7346.

On the graphics side, DirectFB-1.4.5 Phase 1.5 now supports ALUT8, ABGR, YUY2, UYVY and AYUV pixel formats. Both the core DirectFB graphics driver and the Broadcom graphics driver support drawing of trapezoids. The driver now also supports basic graphics operations performed on the ZSP DSP core, for chips that do not have a M2MC (memory-to-memory compositor) core. The *BatchBlit2()* API has also been h/w accelerated to allow dual source blits to occur in a single pass of the M2MC blitter core. We now support improved graphics performance and less CPU loading by directly using the packet buffer interface with Nexus/magnum. This drastically increases the graphics performance when dealing with small sized blits or fills. Finally, the *SetMatrix()* API call has been added to allow translations and transformation to take place on blit and drawing graphics operations.

DirectFB-1.4.5 Phase 1.5 now supports running OpenGL ES 2.0 applications on the VC-4 core using DirectFB surfaces (window or layer surfaces). The OpenGL ES 2.0 applications are all built outside of DirectFB and don't require DirectFB to be configured for 3D graphics operations.

Font caching always used to consume a fixed amount of memory. With DirectFB-1.4.5 Phase 1.5, it is now possible to limit the glyph cache width and number of rows using DirectFB run-time options. This is particularly helpful for applications that create a large number of fonts without destroying them. Without these options, it is possible to run out of physical graphics heap memory.

On the platform side, a new Broadcom DirectFB Platform API has been added to allow multiple applications to communicate Nexus information to the DirectFB platform code. For example, it is now possible for a non-DirectFB application to initialize Nexus, open up its own Nexus Display handle and inform DirectFB of this handle using a combination of "*DFB_Platform_GetDefaultSettings()*" and "*DFB_Platform_Init()*".

The default DirectFB configuration file has been updated to set the smooth upscaling/downscaling options. It now defaults to setting the background color to full transparent black to allow video to be more easily observed behind any graphics.

On the system driver side, many optimizations have been made to improve performance. Below is a list of some of these optimizations:

1. Preallocated surfaces in video memory are now supported, meaning that graphics hardware acceleration is enabled on them.
2. Now use pthread mutexes and conditionals to improve performance over fusion skirmishes.
3. The core DirectFB code and our system driver now support partial updates of the layer code. Instead of blitting the complete back-buffer contents to the front-buffer on a flip() when a portion of the region is updated, the code now effectively swaps the back buffer with the front buffer and then blits only the dirty rectangle from the new front buffer contents to the back buffer to maintain coherency.

The system driver has been re-architected for more stable multi-application support. Only the master DirectFB application can now set the graphics framebuffer and other display settings. The master DirectFB application is also the only process in the system that can create and destroy Nexus surfaces and Nexus memory.

The “ShutdownLayer()” API call was back ported to the system driver to cleanup resources that were claimed during the InitLayer call.

The system driver now also supports additional 24Hz, 25Hz and 30Hz video formats (e.g. 720p/25 and 1080p/24), both statically (with “res=xxx” run-time option) and dynamically (using the screen *SetEncoderConfiguration()* API). It now supports setting the frequency, resolution and scan mode of the video output format allowing the application to support different digital and analog TV standards (e.g. 720p 50/60Hz).

The Broadcom system code now adds the ability to render graphics on the video plane by adding an additional DirectFB layer (“video layer”). This is particularly useful when wanting to use the picture improvement video processing pipeline on graphics (e.g. improve quality of JPEG images).

The IR input driver has been modified to allow run-time selection of the desired IR protocol and keycodes mapping (translation) module. This allows for handsets with different IR protocols to be supported with a single binary, without the need to recompile DirectFB. For windowed applications, it is now possible to determine whether the IR or keypad input event has the repeat flag set.

We now support Bluetooth remote controls that are used in conjunction with the Broadcom Bluetooth software stack.

The Voodoo code has been updated to use “port” rather than “session” when communicating between the remote server and client.

Finally support for building DirectFB with the glibc library has been integrated.

2 DELIVERABLES

This section describes what is contained in the release including software and documents.

Table 1 - Software deliverables

Description	Version/file	Licensing
DirectFB 1.4.5 Phase 1.5 reference software	Phase 1.5 / 20110318	Broadcom SLA

Table 2 - Documentation deliverables

Description	Version/date
DirectFB 1.4.5 Phase 1.5 Software Users Guide (this document)	1.0
DirectFB-1.4.5_Phase1.5_Feature_List.pdf	A8

3 INSTALLATION

3.1.1 Introduction

The DirectFB-1.4.5 Phase 1.5 release can be obtained in two “flavours”. The first is a full release containing both open source software and Broadcom SLA specific code. Typically, the DirectFB graphics, system, input and image provider drivers come under the Broadcom SLA and are built as shared libraries. Please refer to section 3.1.2 for information on installing from a full release.

For customers who have not signed an SLA, a “limited” release can be obtained whereby these modules are supplied in library format only. Please refer to section 3.1.3 for information about how to install from a limited release.

3.1.2 Full Release

The standard reference software (Nexus/Magnum) source code should be available (untared) prior to installation of the DirectFB-1.4.5 Phase 1.5 reference software. If you are unsure of how to do this, please refer to the “Brutus Installation Guide” that comes as part of the reference software release.

On the host (build) machine, navigate to the root of the reference software source tree and type:

```
tar xzvf DirectFB-1.4.5_Phase1.5_20110318.tgz
```

This will overwrite any existing “AppLibs/opensource/directfb” dir (and subdirs).

You will now be ready to build the DirectFB source code from the “AppLibs/opensource/directfb/build/1.4.5” directory.

3.1.3 Limited Release

A limited release does not require any standard reference software to be installed first (e.g. Nexus/magnum). On the host (build) machine choose a directory to navigate to in which the limited DirectFB-1.4.5 Phase 1.5 release will be untared. Then execute the following command:

```
tar xzvf DirectFB-1.4.5_Limited_Phase1.5_20110318.tgz
```

You will now be ready to build the DirectFB source code from the “AppLibs/opensource/directfb/build/1.4.5” directory.

4 BUILDING

4.1.1 Step 1: Host Machine Tools Check

Before commencing the build, please ensure that the version of GNU make is 3.80 or higher on your host build machine. You can test what version of make you are using by issuing the following command:

```
make –version
```

An earlier version of this will result in DirectFB not being built and you will need to upgrade your make package on the host build machine.

> ***NOTE: On the Broadcom build server the latest version can be found in /tools/oss/bin. You would need to set your PATH environment variable as follows:***
>
> ***export PATH=/tools/oss/bin:$PATH***
>
> ***and also the MAKE environment variable as follows:***
>
> ***export MAKE=/tools/oss/bin/make***

4.1.2 Step 2: Environment Variables

Make sure you have the reference software environment variables setup correctly. The important ones are the following:

PLATFORM, BCHP_VER, LINUX, SMP

Example:

```
export PLATFORM=97405
export BCHP_VER=B0
export SMP=y
export LINUX=/opt/brcm/linux-2.6.18-7.7
```

By default the Nexus and magnum drivers will be built in user-space, however you can override this behavior by setting the “MODE” envar to “proxy” and “KERNELMODE=y”. This will ensure the drivers are built in kernel-space with a “proxy” shim layer to translate the API calls to Linux syscalls (ioctl’s) and back again.

Example:

```
export MODE=proxy
export KERNELMODE=y
```

If you are building for the BCM935230, BCM935251, BCM93556 or BCM93549 DTV platforms, then please ensure that the envar "LINUX_DEFCONFIG" is set to point to the default Linux config file, relative to the Linux source tree root.

Example:

```
export LINUX_DEFCONFIG=arch/mips/configs/bcm93548b0-xxxxxx"
```

Where: xxxxx is the remainder of the config filename.

NOTE: Failure to setup this envar for the above platforms will result in the "linux-fusion" IPC kernel module not being built when in multi-application build mode (i.e. when DIRECTFB_MULTI=y).

NOTE: You can speed up the build process on multi-processor machines by ensuring that either MULTI_BUILD=y or MAKE_OPTIONS=-j? is set where ? specifies how many make jobs can be run in parallel (e.g. `make MAKE_OPTIONS=-j4`).

By default, DirectFB and the drivers will be built in DEBUG mode. This can have performance penalties and it is strongly recommended that the user switch to a non-DEBUG mode after monitoring the drivers and DirectFB for any warnings or errors. To switch to a non-DEBUG mode (a.k.a. RELEASE mode), please ensure that the environment variable "DEBUG" is set to "n".

Example:

```
export DEBUG=n
```

The default is to build DirectFB and the drivers in little-endian mode. If the user wishes to run the platform and DirectFB/drivers in BIG endian mode, then the "ARCH" environment variable needs to be specified as "mips-linux-uclibc".

Example:

```
export ARCH=mips-linux-uclibc
```

Finally, make sure your PATH environment variable is setup correctly to point to your MIPS cross-compilation tool chain.

Example:

```
export  PATH=/opt/toolchains/crosstools_hf_linux-2.6.18.0_gcc-4.2-11ts_uclibc-nptl-0.9.29-20070423_20090508/bin:$PATH
```

4.1.3 Step 3 : Driver Build Check

Please note that this step is only required if building from a "full release". For a "limited" release, this step can be skipped.

Make sure you have already built the Nexus/magnum drivers with the same environment variable settings as in Step 1 above. If not, then you can manually do it this way:

```
cd nexus/build
make
```

NOTE: You can speed up the build process on multi-processor machines by specifying the "-j" option to make (e.g. `make -j4`)

4.1.4 Step 4A: Building DirectFB in Single-Application Mode

DirectFB can be built in different modes of operation known as "single-application" and "multi-application". Single-application mode is typically used in situations where there is only a single application accessing the DirectFB API's. A single application is typically a single process with or without multiple threads. If more than one application or process is required to access DirectFB concurrently, then DirectFB will need to be built in multi-application mode using the "DIRECTFB_MULTI=y" make build option.

To build DirectFB without multi-application mode support you need to follow these steps:

```
cd AppLibs/opensource/directfb/build/1.4.5
make && make tarball
```

NOTE: This will build DirectFB using the zlib, libpng, libjpeg and freetype libraries from the "AppLibs/opensource" directory. If these software components do not exist under this directory or the make option "DIRECTFB_APPLIBS=n" is specified, then these software components are taken from the older "BSEAV/lib" directory.

The build process will first check to see whether your host build tools are at the correct minimum version before proceeding. It will then check to ensure that the Nexus drivers are present. If Nexus cannot be found, then the build process aborts and warns the user to recompile Nexus. DirectFB only supports "proxy" mode drivers (drivers in kernel-space with proxy layer) and user mode drivers (server/master only).

The build process will then check to see whether the DirectFB-1.4.5 source tree already exists in "AppLibs/opensource/directfb/src/DirectFB-1.4.5". If not, then the standard DirectFB-1.4.5.tar.gz tarball from the "AppLibs/opensource/directfb/src/directfb_tarballs" directory will first be untared to create the "AppLibs/opensource/directfb/src/DirectFB-1.4.5" directory. The next step will then be to copy the contents of the "AppLibs/opensource/directfb/src/broadcom_files/public/DirectFB/1.4.5" directory on top of the newly created DirectFB-1.4.5 source tree. This step is necessary, as the standard DirectFB tarballs

don't have all the Broadcom specific changes and drivers. This public directory contains only open-source components.

If the "AppLibs/opensource/directfb/src/broadcom_files/private/DirectFB/1.4.5" directory exists, then the contents of this directory are copied on top of the newly created DirectFB-1.4.5 source tree.

> ***NOTE: The private directory is always provided to customers who have signed an SLA to be able to receive Broadcom reference software. If this directory does not exist, then pre-built shared libraries will still be provided to be placed on the target system. These pre-built libraries will reside in the "broadcom_files/precompiled" directory.***

Before DirectFB source code can be built, the freetype library needs to be built, then jpeg and zlib and finally libpng. The DirectFB source code will then be configured to auto-generate the Makefile(s) and finally the DirectFB source code will be built and installed. The final stage will produce a tarball that can then be copied to the target for extracting and running. This tarball will look similar to the example below:

e.g. DirectFB-1.4.5_debug_build.97405C0.tgz

> ***NOTE: In non-DEBUG (RELEASE) mode, the word "debug" will be replaced with "release".***

4.1.5 Step 4B: Building DirectFB in Multi-Application Mode

Multi-application mode will allow multiple applications/processes to use the DirectFB API concurrently. The build process is the same as for Step 4A above, except that the additional make build option called "DIRECTFB_MULTI=y" needs to be set:

Example:

```
cd AppLibs/opensource/directfb/build/1.4.5
make DIRECTFB_MULTI=y && make DIRECTFB_MULTI=y tarball
```

> ***IMPORTANT NOTE: DirectFB running in multi-application mode requires the Nexus/magnum drivers to be built and run in kernel mode (proxy mode). This restriction will be removed in a future release of DirectFB where multi-process user-space Nexus is also supported.***

The steps the build process takes are slightly different, in that the fusion IPC kernel module (linux-fusion) will be built prior to freetype (and the other libraries) being built. The last stage of the build process is also different in that the SaWMan window manager will be built. SaWMan allow finer control over the

placement and lifecycle of multiple applications on the screen. It replaces the default window manager that comes with DirectFB.

The resulting tarball will be named slightly differently to the one produced in step 4A above. The word "multi" will appear immediately after the "DirectFB-1.4.5"

e.g. DirectFB-1.4.5_multi_debug_build.97405C0.tgz

This tarball can be copied to the target platform in the same way as for step 4A above.

4.1.6 Building DirectFB Tests

DirectFB test applications that reside in the DirectFB-1.4.5/tests directory are not built by default now. To enable these unit tests to be built, please ensure that the make build option "BUILD_TESTS" is set to "y".

```
e.g. make BUILD_TESTS=y
```

4.1.7 Building DirectFB Examples

DirectFB examples are separate test/demo applications that can be built and run on the Broadcom reference platforms. To enable these additional test/demo applications to be built, please ensure that the make build option "BUILD_EXAMPLES" is set to "y".

```
e.g. make BUILD_EXAMPLES=y
```

4.1.8 Building FusionDale

FusionDale is an IPC (inter-process communication) software module that has dependencies on fusion (linux-fusion). It provides a more abstracted way to perform IPC than fusion itself. FusionDale can be used independently from DirectFB and provides distributed event notification, RPC (remote procedure call) and message distribution.

To build FusionDale, the make build option "BUILD_FUSIONDALE" must be set to "y":

```
e.g. make BUILD_FUSIONDALE=y
```

4.1.9 Building ++DFB

++DFB (a.k.a. ppDFB), is a library that C++ application can call into to make DirectFB API calls (effectively a set of C++ bindings). ++DFB is a more advanced version of DFB++, and is incompatible in the way applications can call methods/functions.

To build ++DFB-1.4.2, please ensure that the make build option “BUILD_PPDFB” is set to “y”.

```
e.g. make BUILD_PPDFB=y
```

NOTE: ++DFB requires the standard C++ libraries be present on the target platform.

4.1.10 Building DiVine

DiVine (DirectFB Virtual input extension) is a library that simulates the behavior of a real input device to control DirectFB applications. DiVine has a client/server model to allow input commands to be dispatched to the server over a socket connection

To build DiVine, please ensure that the make build option “BUILD_DIVINE” is set to “y”.

```
e.g. make BUILD_DIVINE=y
```

4.1.11 Building Insignia Test Harness

The Insignia test harness is only available to certain customers who have signed an SLA with YouView. This test harness will check that the Broadcom DirectFB graphics driver matches the software fallback implementation for over 600 test cases. If the Insignia tarball is present, then this package can be built by setting the make build option “BUILD_INSIGNIA” to “y”.

```
e.g. make BUILD_INSIGNIA=y
```

4.1.12 Build Tacho Test Harness

The Tacho test harness is only available to certain customers who have signed an SLA with YouView. This test harness will check the graphics performance of the Broadcom DirectFB graphics driver. If the Tacho tarball is present, then this package can be built by setting the make build option “BUILD_TACHO” to “y”.

```
e.g. make BUILD_TACHO=y
```

4.1.13 Building External Applications

Third party applications and/or external applications/tests can be built with the correct DirectFB compiler flags by using the "directfb-config" utility. The example below shows how a test application called "my_test.c" can be built that reads images from the standard "/usr/local/share/directfb-1.4.5/images" directory.

```
mipsel-linux-uclibc-gcc `./DirectFB-1.4.5/directfb-config --cflags --libs` -
DDATADIR=’”/usr/local/share/directfb-1.4.5/images”’ my_test.c -o my_test
```

4.1.14 Additional Make Targets

The DirectFB build system does allow for partial steps or targets to be chosen. These build targets are listed below along with a description:

Table 3 – Make Targets

Make Target	Description
help	List the DirectFB make targets that can be called along with options.
default	This is the default make target and will attempt to install all DirectFB software modules, if they haven't already been installed. If a module hasn't been compiled, then it will be compiled first.
release	This will create a full release of the DirectFB software including both open source and Broadcom SLA specific code. The result is a dated tarball.
limited	This will create a limited release of the DirectFB software including only the open source components. The result is a dated tarball.
all	This will force every DirectFB software module to be reconfigured, rebuilt and installed.
tarball	This will create a tarball of the target output directory that can then be copied over to the target platform for unpacking and running.
install	This option is the same as the default target option and will only install software modules that need installing.
compile	This will attempt to compile all DirectFB software modules that need compiling. If a module hasn't already been configured, then it will be configured first.
config	This will attempt to configure all DirectFB software modules that need configuring. If a module hasn't already been unpacked, then it will be unpacked first.
uninstall	This will cause all installed intermediate files to be uninstalled.
uninstall-target	This will remove all target output directory installed files.
clean	Remove all generated object files, dependencies, binaries and temporary object directories.
distclean	Remove everything including the generated source code. The user will be prompted first to remove any source code. This target should always be called prior to installing a new release of DirectFB.
mrproper	Remove everything including generated source code. No user prompts will be displayed. This target should always be called prior to installing a new release of DirectFB.

Make Target	Description
precompile-clean	This is useful if you want to remove the entire precompiled installation directory.
precompile-distclean	This will remove all files and directories under the precompiled directory.
check-tools	Do a quick check to make sure you have up-to-date tools to build DirectFB.
check-autogen-tools	Do a quick check to make sure you have up-to-date tools to auto generate the autoconf *.in files needed to build DirectFB.
directfb-defines	This will update the graphics and system defines files in the DirectFB-1.4.5 source tree.
xxx-all	Where xxx can be “”, “directfb”, “directfb-examples”, “sawman”, “fusiondale”, “fusion”, “ppdfb”, “divine”, “insignia”, “tacho”, “freetype”, “jpeg”, “png” and “zlib”. This will re-build the chosen target software module including re-configuration and re-compilation if necessary.
xxx-source	Like “xxx-all” above, but the source code for the chosen module “xxx” will be created if not already present.
xxx-config	Like “xxx-source” above, but the configuration step will be called.
xxx-compile	Like “xxx-source” above, but the compilation step will be called.
xxx-install	Like “xxx-source” above, but the installation step will be called.
xxx-uninstall	Like “xxx-source” above, but the uninstallation step will be called.
xxx-clean	Only clean the required software module specified by “xxx”.
xxx-distclean	Perform a complete clean of the chosen software module specified by “xxx” with user prompt.
xxx-mrproper	Perform a complete clean of the chosen software module specified by “xxx”.
yyy-autogen	Where “yyy” can be “directfb”, “directfb-examples”, “sawman”, “fusiondale” and “ppdfb”. This will regenerate the “Makefile.in” and “configure” scripts from the *.am files. This option is only useful for developers who want to change the way the chosen software module is built (e.g. build new test application).

4.1.15 Additional Make Flags

The following table lists the complete set of flags that can be passed to the DirectFB build system to modify the default build behavior. The flags are normally passed on the "make" command line, but can also be set as environment variables.

```
e.g. make DIRECTFB_MULTI=y
```

Table 4 – Make Flags

Make Target	Description
DIRECTFB_VERSION	The default version of DirectFB can be overridden (e.g. DIRECTFB_VERSION=1.0.0)
DIRECTFB_APPLIBS	Use default build normally pulls in the zlib, freetype, png and jpeg libraries from the "AppLibs" directory. By setting this flag to "n", the older versions of these libraries will be used from the "BSEAV/lib" directory instead.
DIRECTFB_MULTI	The default is to build DirectFB in single application mode. Setting this flag to "y" will build DirectFB in multi-application mode.
DIRECTFB_ACCEL	The default is to build DirectFB with hardware graphics acceleration support (blitter/M2MC). Setting this flag to "n" will disable all graphics acceleration and will use the generic DirectFB graphics primitive functions instead. This flag should be set to "n" for chipsets that do not have a blitter/M2MC core.
DIRECTFB_SHARED	The default is to build the zlib, freetype, png and jpeg utility libraries as shared libraries that can then be dynamically linked with DirectFB at run time. Setting this flag to "n" will instead build these libraries as static (.a) and DirectFB will statically link with them at compile time. Setting this flag to "n" generally increases code size, but can be useful if applications use different versions of these utility libraries.
DIRECTFB_PREFIX	The default target prefix is "/usr/local" but this can be overridden using this flag (e.g. DIRECTFB_PREFIX=/usr). This specifies the path to the DirectFB installation on the target platform.
DIRECTFB_IR_PROTOCOL	The default IR protocol can be overridden using this flag. The IR protocol should be the name of the NEXUS IR protocol (e.g. "Generic", "RemoteA", and "CirNec").
DIRECTFB_IR_INPUT	The default is to enable the DirectFB IR input if the platform supports an IR input. However, the user can specify "n" to disable it.

Make Target	Description
DIRECTFB_KEY_INPUT	The default is to enable the DirectFB front-panel keypad input if the platform supports a front panel keypad. However, the user can specify “n” to disable it.
DIRECTFB_SYSTEM	The default is always to build the DirectFB Broadcom system module/lib. However, setting this flag to “n” will prevent this module from being built.
DIRECTFB_SID	The default is to build the DirectFB still image decoder (SID) image provider module/lib, if the platform supports a SID hardware decoder. However, the user can specify “n” to prevent this module from ever being built.
DIRECTFB_SW_DITHERING	Enable software dithering (currently only supported with RGB16 and ARGB4444 formats). When set to “y” advanced software dithering for RGB16 and ARGB4444 formats will be enabled. The downside is that this will increase the size of the data section by at least 64KB.
DIRECTFB_SW_SMOOTH_SCALING	Enable software smooth scaling. When set to “y” software smooth scaling will be enabled and the size of the text section will increase by at least 100K bytes.
DIRECTFB_GFX_PACKET_BUFFER	The default is to enable the packet buffer interface in the Broadcom DirectFB graphics driver. Setting this to “n” will cause the legacy graphics driver to be used that will have lower graphics performance.
DIRECTFB_GFX_TRAPEZOID_SUPPORT	The default is to enable drawing of trapezoids in the graphics driver if the packet buffer interface is enabled too. Setting this option to “n” will result in trapezoids being drawn using only software.
DIRECTFB_GFX_SOFT_MATRIX_SUPPORT	The default is to enable support for the SetMatrix() function in our graphics driver using the PX3D hardware to perform rotation and shearing. If the PX3D hardware is not present or this option is set to “n”, then a much limited feature set for SetMatrix() will be available.
GL_SUPPORT	This flag controls whether the nexus/magnum drivers and DirectFB are built with support for the PX3D 3D graphics core. By default the drivers and DirectFB are not built with support for 3D graphics. To enable support for “DrawLine()”, “TextureTriangles()”, “FillTriangle()” and “FillTriangles()” please set this flag to “y” when building nexus and also when building DirectFB. This flag does not have any effect for the BCM935230 and BCM935125 platforms.
DIRECTFB_GLES_SUPPORT	The default is not to build DirectFB with OpenGL ES 1.0 and EGL support. However, setting both this flag and the “GL_SUPPORT” flag to “y” will build DirectFB with 3D OpenGL ES 1.0 support. This support is only available on devices that have a PX3D graphics core (not available on a BCM935230 device). The nexus/magnum drivers must have also been built with both these flags set to “y” in order to have OpenGL ES 1.0 support.

Make Target	Description
USE_SHIM	The default is not to use the “shim” layer that translates Nexus calls to “shim_NEXUS” calls. Setting this option to “y” will result in the “shim” Nexus code being built.
BUILD_TESTS	The default is not to build the DirectFB unit test applications that are located in the “tests” directory. Setting this flag to “y” will instead build and install the additional DirectFB tests.
BUILD_EXAMPLES	The default is not to build the additional DirectFB examples. Setting this flag to “y” will instead build and install the additional DirectFB examples.
BUILD_FUSIONDALE	The default is not to build the FusionDale library. Setting this flag to “y” will build and install the FusionDale library components and examples.
BUILD_SAWMAN	The default is to build SaWMan only when “DIRECTFB_MULTI=y”. Setting this flag to “n” will disable building SaWMan and will ensure that the default window manager of DirectFB is used. This option is only meaningful when building in multi-application mode.
BUILD_PPDFB	The default is not to build the ++DFB library. Setting this flag to “y” will build and install the library.
BUILD_DIVINE	The default is not to build the DiVine library. Setting this flag to “y” will build and install the library.
BUILD_FUSION	The default is to build the “linux-fusion” kernel module only when DirectFB is built in multi-application mode. Setting this flag to “n” will prevent this module from being built and installed and if used in conjunction with “DIRECTFB_MULTI=y”, the experimental multi-application mode of DirectFB will be built instead.
BUILD_VOODOO	The default is not to build the voodoo library that is part of DirectFB. Setting this flag to ‘y’ will build and install the library.
BUILD_INSIGNIA	The default is not to build the Insignia library. Setting this flag to “y” will build and install the library if the source tarball is present.
BUILD_TACHO	The default is not to build the Tacho library. Setting this flag to “y” will build and install the library if the source tarball is present.

Make Target	Description
MODE	The default is to build NEXUS for user-space. Setting this option to “proxy” will result in the NEXUS drivers being compiled for kernel-space.
SMP	Setting this option to “y” will build “linux-fusion” in SMP mode. Setting this flag to “n” will build “linux-fusion” in UP mode.
ARCH	The default is to build DirectFB and the associated libraries in little endian mode. Setting this flag to “mips-uclibc” will result in DirectFB and its libraries being built in big endian mode instead. It should be noted that Nexus and magnum must be built in big endian mode prior to rebuilding DirectFB.
DEBUG	The default is to build DirectFB in debugging mode. However, setting this flag to “n” will build DirectFB in release mode and no debugging information will be available. Setting this option to “n” will also improve graphics performance.
VERBOSE	The default is to build DirectFB with minimal information. Setting this flag to “y” will increase the amount of information available during the building stages.
TRACE	The default is to build DirectFB without any tracing information. Setting this flag to “y” will allow tracing information to be enabled.
DIRECTFB_EXAMPLES_VERSION	The default is to build the DirectFB examples 1.2.1 software package. If a different version is available, then seeing this flag will result in that version being build instead (e.g. DIRECTFB_EXAMPLES_VERSION=1.2.3). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
FUSION_VERSION	The default is to build linux-fusion version 8.1.1. If an alternative tarball version is available to be built, it should first be placed in the “directfb_tarballs” directory and this flag should be set accordingly (e.g. FUSION_VERSION=8.1.2).
SAWMAN_VERSION	The default is to build SaWMan version 1.4.5, but if a different version of SaWMan is available, then setting this flag will result in that version being built instead (e.g. SAWMAN_VERSION=1.4.11). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
FUSIONDALE_VERSION	The default is to build FusionDale version 0.8.1. Setting this flag will allow an alternative version of FusionDale to be built (e.g. FUSIONDALE_VERSION=0.8.2). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.

Make Target	Description
PPDFB_VERSION	The default is to build ++DFB version 1.4.2. However, this behavior can be overridden by specifying an alternative version (e.g. PPDFB_VERSION=1.4.0). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
DIVINE_VERSION	The default is to build DiVine version 0.4.0. However, this behavior can be overridden by specifying an alternative version (e.g. DIVINE_VERSION=0.4.1). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
INSIGNIA_VERSION	The default is to build Insignia version 0.1.2. However, this behavior can be overridden by specifying an alternative version (e.g. INSIGNIA_VERSION=0.1.3). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
TACHO_VERSION	The default is to build Insignia version 0.1.2. However, this behavior can be overridden by specifying an alternative version (e.g. TACHO_VERSION=0.1.3). The alternative software tarball should be placed in the “directfb_tarballs” directory before building the software.
DFB_FREETYPE_VERSION	The default is to build Freetype version 2.3.7 from “AppLibs/opensource/freetype” or version 2.1.5 from “BSEAV/lib/freetype-2.1.5”. This behavior can be overridden by setting this flag appropriately.
DFB_JPEG_VERSION	The default is to build JPEG version “6b” from “AppLibs/opensource/jpeg” or “BSEAV/lib/jpeg-6b”. This behavior can be overridden by setting this flag appropriately.
DFB_PNG_VERSION	The default is to build PNG version 1.2.29 from “AppLibs/opensource/libpng” or version 1.2.8 from “BSEAV/lib/libpng”. This behavior can be overridden by setting this flag appropriately.
DFB_ZLIB_VERSION	The default is to build zlib version 1.2.3 from “AppLibs/opensource/zlib” or version 1.1.3 from “BSEAV/lib/zlib”. This behavior can be overridden by setting this flag appropriately.
APPLIBS_INSTALL_PREFIX	The default is to install all files relative to “/usr/local” on the target platform. This option can be overridden to place the target files in a different directory structure.
APPLIBS_TARGET_TOP	This specifies the final output directory on the host build machine in which the DirectFB binaries and libraries are to be installed prior to being packed ready for transfer to the target platform. The default is “AppLibs/target”, but this can be overridden

5 RUNNING DIRECTFB ON THE TARGET PLATFORM

5.1.1 Single Application Mode

Once you have generated the tarball in Step4A or Step4B, you need to copy it to the target platform. You can use tftp to do this. For example, you need to ensure your host machine is running a tftp server and then you can copy the tarball to the tftp root dir (e.g. /tftpboot). For more information about setting up a tftp server on you host machine, please refer to the "BroadcomReferencePlatformSetup.pdf" guide that is part of the standard reference software release.

On the target platform, please ensure that you boot it with the same kernel version and setup (e.g. SMP) that you built DirectFB and the Nexus/magnum drivers with. Once you have booted to the login prompt, and logged in (as root), you need to enter in these commands to create a tmpfs filesystem at /dev/shm:

```
mkdir /dev/shm
mount –t tmpfs none /dev/shm –o size=20m
```

Next you need to copy the DirectFB tarball to /dev/shm using tftp and unpack it to /usr/local. Please follow these steps:

```
cd /dev/shm
tftp –g –r DirectFB-1.4.5_xxxxxx.tgz <name of host machine or IP address>
cd /
tar xzvf /dev/shm/DirectFB-1.4.5_xxxxxx.tgz
```

This will create a new "/usr/local" tree which is where the DirectFB libraries and unit tests will reside.

> ***NOTE: If you have permission problems writing to the root file system, then it may be mounted read-only (RO). To change the permissions to read-write (RW), you can enter in the following command:***
>
> ***`mount –o remount,rw /`***

If your target platform has a HDD connected, you can choose to install the DirectFB libraries on the HDD instead of the root files system. This is necessary if your root files system is initrd or you do not have sufficient capacity on the flash device. To do this, you need to ensure your HDD partition is mounted first and that you created a symbolic link from /usr on the root files-system to /usr on the HDD partition.

Example:

```
mount /dev/sda1 /mnt/hd
cd /
ln –s /mnt/hd/usr
cd /mnt/hd
tar xzvf /dev/shm/ DirectFB-1.4.5_xxxxxx.tgz
```

Once you have untared DirectFB to /usr/local, you are now ready to run the installation script.

```
cd /usr/local/bin/directfb/1.4
./rundfb.sh install
```

You are now ready to run any DirectFB application from this directory. For example, to run the Denis's (DOK's) benchmarking unit test, please enter the following command:

```
./rundfb.sh df_dok
```

NOTE: On the DTV platforms such as the BCM93556 and BCM935230, the graphics layer size may need to be different to that of the display resolution. This restriction depends entirely on the bvn config setup (RTS settings). In some cases, the maximum size of the graphics layer is 1366x1080 on the BCM93556 and 1366x768 on the BCM935230. You will need to use the "mode=HxW" DirectFB run-time option to be able to run applications succesfully. For example, on the BCM935230 you would need to run "df_dok" as follows:

./rundfb.sh df_dok --dfb:mode=1366x768

On non-DTV products, you can also specify the output resolution of the connected display (the default is 720p on most chipsets). For example, to run with a 1080i output resolution, you can enter the following command:

```
./rundfb.sh df_dok --dfb:res=1080i
```

The default graphics heap memory size is 16MB. For some resolutions and applications, this will need to be increased. For example, to run the "Penguins" test app (df_andi) at 1080i resolution, it will be necessary to increase the graphics heap size from 16MB to 33MB. To do this you need to set an environment variable prior to running any DirectFB application.

```
export gfx_heap_size=34603008
./rundfb.sh df_andi --dfb:res=1080i
```

The value after the equals sign (=) on the first line is the requested heap size in bytes. This cannot be larger than the total size of ram minus the size of the Linux kernel top of ram and Nexus/magnum drivers ram footprint. This graphics heap is also known as VIDEO MEMORY in DirectFB terminology and is not SYSTEM MEMORY (a.k.a Linux memory).

On some platforms, the memory architecture is unknown until after the drivers have been loaded and initialized. For example, on the BCM97405 platform the memory architecture can be UMA or non-UMA (unified or non-unified). Because of this, there can be a lag between issuing the "rundfb.sh" command and the application appearing on the display. This is because the drivers have to be loaded, then unloaded and loaded again with the correct memory configuration.

To help reduce this lag time, you can set an environment variable to tell DirectFB exactly what architecture the memory is. For example, if the memory is unified, then set please enter the following command prior to running any DirectFB application:

```
export mem_non_uma=n
```

On the other hand, if the memory architecture is non-unified, then you can set the environment variable as follows:

```
export mem_non_uma=y
```

If you are unsure what the memory architecture is set to, then please refer to the CFE boot information and look for the line named "Memory Config:". For example, if this line is set to "64-bit UMA", then this indicates that the memory architecture is unified and you will need to set "mem_non_uma=n".

5.1.2 Multi-Application Mode

With DirectFB running in multi-application mode, more than one application can access the DirectFB API's at the same time and from different processes. The same steps can be taken as for single-application mode when it comes to running the very first application. However, for any subsequent application, the "rundfb.sh" script needs to accept the "join" option. An example of running both df_andi (Penguins) and df_window in multi-application mode using different processes is given below:

```
cd /usr/local/bin/directfb/1.4
export gfx_heap_size=36000000
./rundfb.sh df_window &
./rundfb.sh join df_andi --dfb:force-windowed
```

You should now be able to see the DirectFB windows on top of the Penguins. If you have a USB keyboard and mouse connected to the platform, you should be able to move the windows around the screen. Pressing Q or ESC on the USB keyboard will quit the application(s).

You can also specify the size of the graphics surface/layer independent of the output resolution of the display. For example, if you would like to have a 640x480 graphics layer with a 1280x720p output resolution, you can use the "mode" DirectFB option. An example of this is given below:

```
./rundfb.sh join df_andi --dfb:mode=640x480
```

The graphics will be stretched horizontally and vertically to fill the display window.

NOTE: The first DirectFB application that runs is known as the "master" and subsequent DirectFB applications are known as "slaves". The "master" application is normally a module that should not under normal circumstances be terminated. Internally within DirectFB, it will manage the Nexus display settings and will be responsible for creating and destroying all Nexus surfaces and memory. If this application is terminated before any of the client applications are closed, then the system will be in an unrecoverable state.

5.1.3 Running Texture Mapped Graphics Applications

DirectFB can now be built with hardware support for the “TextureTriangles()”, “FillTriangle()” and “FillTriangles()” graphics functions. This support is only available for Broadcom devices that have the PX3D 3D graphics core (e.g. BCM93556, BCM97413, BCM97420). Both Nexus and DirectFB must be built with the “GL_SUPPORT=y” environment flag set.

To test “TextureTriangles()”, the user can run the “df_texture” unit test.

```
e.g. ./rundfb.sh df_texture
```

5.1.4 Running OpenGL ES 1.0 Graphics Applications

DirectFB can now be built with support for OpenGL ES 1.0 and EGL. Both Nexus and DirectFB must be built with the following two environment flags set:

“GL_SUPPORT=y”

“DIRECTFB_GLES_SUPPORT=y”

To test the OpenGL ES 1.0 and EGL support within DirectFB, you may run any of the following test applications:

1. dfbtest_egl_only
2. dfbtest_egl_pixmap (use mouse to XYZ rotation)
3. dfbtest_gl (use mouse to change XYZ rotation)

```
e.g. ./rundfb.sh dfbtest_gl
```

5.1.5 Run-time Environment Variables

The table below lists the environment variables that affect the run-time behavior of DirectFB. These environment variables can be set using the bash "export" command as follows:

```
e.g. export mem_non_uma=n
```

Table 5 – Run-time environment variables

Make Target	Description
mem_non_uma	Setting to "n" will force the memory architecture to UMA, whereas setting to "y" will force the memory architecture to non-UMA. When this environment variable is not set (or is not "y" or "n"), then when DirectFB starts it will interrogate the memory architecture. This can introduce an unwanted delay at startup. By setting this envar appropriately, this delay can be eliminated all together.
gfx_heap_size	This overrides the default graphics heap memory size (in bytes) of 16MB. For example, to increase the graphics heap size to 32MB, ensure that this envar is set to "33554432" (size in bytes).
dfb_slave	This determines whether the DirectFB application should "join" Nexus (set to "y") or initialize Nexus (set to "n"). This option can be set to "y" if another non-DirectFB application is the primary application in the system and has already initialized Nexus with "NEXUS_Platform_Init()". After initializing or joining Nexus, DirectFB can decide whether to open the display, graphics and picture decoder Nexus handles itself or use the handles provided to it in the DFB_Platform_Init() call.
sw_picture_decode	This envar only affects platforms that have a still image decoder (SID). Normally, JPEG, GIF and PNG images are rendered using the SID (if available). However, setting this envar to any value will result in the generic software DirectFB picture decoding functions being used instead.
dfb_no_platform_init	This "forces" the DirectFB application to behave like a pure "slave" application joining Nexus and using the Nexus handles already provided by another "master" DirectFB application.
panel_type	This envar is only available on DTV platforms (e.g. BCM935230, BCM935125 and BCM93556). This indicates the panel type and the following options are available: 1. "JEIDA" 2. "CMO" 3. "AUO" 4. "B552" (Note: use this option for LVDS to DVI board) 5. "OPENLDI" (Note: only available on BCM935230 / BCM935125 platforms)
bvn_usage	This envar is only used on DTV platforms such as the BCM93556 and the BCM935230. It is necessary to set this envar to the correct "configXX" value. For example, on the BCM935230, this is typically set to "config200".

Make Target	Description
pixel_swap	This envar is only available on DTV platforms (e.g. BCM935230, BCM935251 and BCM93556) and when “panel_type” is set to “B552”. If this envar is set to “yes”, then the pixel components are swapped.
output_type	This envar is only available on DTV platforms (e.g. BCM935230, BCM935251 and BCM93556). Setting this envar to “component”, will result in the primary display resolution taking the value from the DirectFB “res” option, rather than defaulting to the native resolution of the panel. This option is only useful if a panel is not connected and the component output is used instead.
hdsd_mode	Set the HD/SD display mode. If this envar is set to “0”, then the composite/CVBS output is connected to the primary display 0 output. In this configuration, only SD display output resolutions are supported on both primary and secondary display outputs (e.g. “res=576i”). If this envar is not set or is a value other than “0”, then the composite/CVBS output is connected to the secondary display 1 output and the primary display output can be configured to be either HD or SD.
enable_rfm	This envar is only available on platforms that have an RFM output (e.g. BCM97325). The RFM output is connected to the same display output as the composite/CVBS output and is dictated by the “hdsd_mode” envar. Setting this envar to “y” will enable the RFM output. Setting this envar to any other value will disable the RFM output. The same restrictions for display output size apply as for the “hdsd_mode” envar above.
DFBARGS	This is the standard DirectFB arguments envar that can be used to specify the DirectFB run-time options (e.g. export DFBARGS=”res=1080i”).

5.1.6 Running DirectFB Examples

DirectFB examples are additional example applications and tests that are built when the “BUILD_EXAMPLES=y” option is set. To test any of these additional example applications, please follow the example steps below:

```
cd /usr/local/bin/directfb-examples/1.2
./rundfb.sh df_matrix
```

5.1.7 Running FusionDale

As previously mentioned, FusionDale provides an abstracted IPC mechanism for applications to communicate with each other when in different processes. There are a few unit test applications that get built when the make option “BUILD_FUSIONDALE=y” is set. These are:

1. **simple**: very simple application to test the basic FusionDale interface functions correctly.
2. **fdmaster**: simple application that initializes and creates a FusionDale instance and then pauses.
3. **data_test**: send a message and receive a notification for it.
4. **t2_sender**: sends a simple event using the messenger interface.
5. **t2_receiver**: listens for an event using the messenger interface.

The “t2_sender” and “t2_receiver” are the best applications to test IPC between different processes.

Example:

```
cd /usr/local/bin/fusiondale/0.8
./runfus.sh t2_receiver &
./runfus.sh join t2_sender
```

5.1.8 Running SaWMan (multi-application mode)

SaWMan is the **S**hared **A**pplication and **W**indow **Man**ager that overrides the "default" window manager of DirectFB. It can act as an application lifecycle manager, deciding what application/processes can be spawned or terminated and which application(s) receive input events. Many multi-application environments use SaWMan to help fulfill their requirements for displaying and managing multiple applications simultaneously.

If DirectFB has been built in multi-application mode, then the SaWMan becomes the default window manager. There are two specific applications that can be used to test SaWMan functionality. They are "testman" and "testrun". "testman" is the main application manager and is used to register what applications can be spawned or terminated. It also has full control over the layout of multiple applications on the display. "testrun" on the other-hand, is used to signal what pre-registered application can be run. "testrun" can be called from different processes multiple times, thus helping to simulate a real-world multi-process / multi-application environment. To test SaWMan please follow the steps below:

```
cd /usr/local/bin/sawman/1.4
./runsaw.sh testman &
./runsaw.sh join testrun Penguins
./runsaw.sh join testrun Penguins2
./runsaw.sh join testrun Penguins3
./runsaw.sh join testrun Penguins4
```

On the screen you should see 4 windows each with their own df_andi (Penguins) moving around. You can move the mouse over any of the windows and press **<Q>** to quit the application. Each of the applications is running in a separate process.

5.1.9 Running ++DFB

++DFB requires that the C++ standard libraries are installed on the target platform. These libraries are usually located in the "/lib" directory and are called "libstdc++.so". If the libraries are not present, then you will need to manually copy them from your host/build cross-compilation toolchains directory as follows:

On host/build machine:

```
cp /opt/toolchains/crosstools_hf-linux-2.6.18.0_gcc-4.2-11ts_uclibc-nptl-
0.9.29_20070423-20090508/mipsel-linux-uclibc/lib/libstdc++.so.6.0.9 /tftpboot
```

On target platform:

```
cd /usr/local/lib
tftp -g -r libstdc++.so.6.0.9  <name of host/build machine>
ln -s libstdc++.so.6.0.9  libstdc++.so.6
ln -s libstdc++.so.6.0.9  libstdc++.so
```

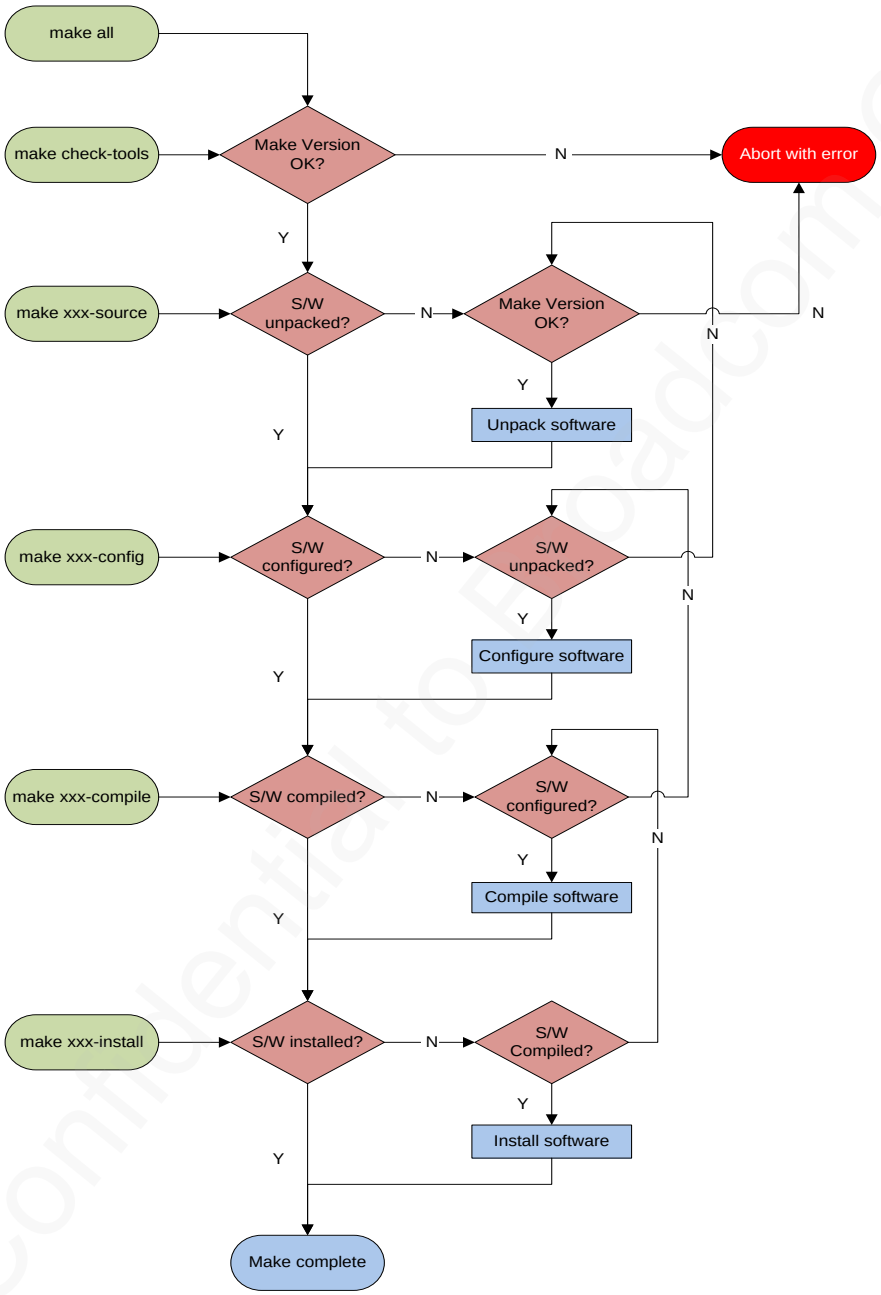
At this point you should be able to run any of the ++DFB test applications. For example, you can run the "dfbshow" test application as follows:

```
cd /usr/local/bin/++dfb/1.4
./runppd.sh dfbshow /usr/local/share/directfb-1.4.5/images/biglogo.png
```

6 ADDITIONAL INFORMATION

6.1 BUILD SYSTEM INFORMATION

The build system for DirectFB and its associated software components all reside in a top-level Makefile and make include file (directfb_common.inc) in the "AppLibs/opensource/directfb/build/1.4.5" directory. The make process is broken down into different steps, each of which depends on a previous step. The following diagram shows how the user can enter any step directly, but the make system knows whether the previous step(s) have already been completed (dependencies).



It is worth noting that the build system does not track modified source code files between the stages in green. For example, if the user built and installed DirectFB by typing “make” and then modified a DirectFB-1.4.5 source code file, the build system does NOT know that a source code file was modified if the user were to type “make” again. Instead, the user can type “make directfb-compile” and this will rebuild only the source files needed within DirectFB. The user can then type “make” and the build system is intelligent enough to know that the directfb installation phase needs to be completed next.

This approach saves time during the build process when making lots of source code modifications. If the makefile had to call each software modules “compile” stage, then it would also force an installation which would all consume valuable time. The recommended approach is to make source code modifications, type “make xxx-compile” (where xxx is the software module like “directfb”) and then type “make” for the build system to complete any further necessary steps (e.g. installation). The same can be said if the user wants to reconfigure DirectFB or a software module. The user should type “make xxx-config” first to re-configure the software module, and then type “make”. Usually this will involve the source code being re-compiled and re-installed.

6.2 RUNNING MULTIPLE APPLICATIONS IN SEPARATE PROCESSES

DirectFB can be built in what is known as “multi-application” mode. This mode allows multiple DirectFB and non-DirectFB applications to run in separate processes simultaneously. Currently, DirectFB multi-process support is only available when the Nexus drivers are built for “proxy” mode (kernel mode). This limitation will be removed in the next release to allow user-space multi-process Nexus drivers to run.

The first DirectFB application to run is normally the “master” application. This application needs to always remain running as it is responsible for receiving remote procedure calls (RPC) from client DirectFB applications. This “master” DirectFB application is responsible for creating and destroying Nexus surfaces, allocating and freeing Nexus memory and handling Nexus display settings (e.g. setting the framebuffer). When a client DirectFB application tries to create a surface or set the graphics framebuffer, it will issue a RPC to this “master” DirectFB application to service the request.

If the master application is terminated either intentionally or unintentionally, then the system will be in an unstable state as client applications won’t be able to have their requests serviced. It is recommended that the master DirectFB application is also the “application manager” in the complete multi-process system.

6.2.1 Running non-DirectFB and DirectFB Applications

There are some situations where the system has non-DirectFB applications and DirectFB applications. In this scenario, the non-DirectFB application might already be initializing Nexus and opening Nexus modules that the DirectFB applications rely on (e.g. Nexus display, graphics2d, graphics3d).

To allow for this usage scenario, there is a new “dfb_platform.h” file that contains a light-weight API to allow non-DirectFB and DirectFB applications to use. The non-DirectFB application can continue to initialize Nexus and open up the Nexus modules it needs to. Then, it can inform the Broadcom DirectFB platform layer code what handles it has opened through the “*DFB_Platform_Init()”* call. This call accepts a pointer to a structure that tells the code whether to initialize or join Nexus and what (if any) handles are available. It places the handles in System V shared memory to allow other processes to access them.

The “master” and “slave” DirectFB applications should also call this function to ensure that the handles are known to them (read from the System V shared memory). To help initialize this structure, the applications can call “*DFB_Platform_GetDefaultSettings()*” to obtain default platform settings, modify them and then pass the settings to “*DFB_Platform_Init()*”.

To allow this scenario to work, the platform init code (DirectFB-1.4.5/libinit/platform_init.c) needs to be modified to prevent the constructor and destructor from being called. The master application can then link with the modified platform code (libinit.so) and should call the “*DFB_Platform_Init()*” with appropriate arguments. The DirectFB applications should also link with the platform init code (libinit.so) and should call this function right at the beginning of the main function. If the master application opens up any display handles, then it has the responsibility for connecting any outputs to the display such as HDMI and component. The Broadcom DirectFB platform code will then not attempt to initialize the display or connect any outputs to the display.

Each application in the system should call *“DFB_Platform_Uninit()”* before it terminates. This ensures that the system is in a consistent state.

6.3 CONFIGURING THE KERNEL TO ENABLE USB INPUT DEVICES

Some of the pre-built Linux kernels do not have USB HID (human input device) support enabled by default and the USB connected keyboard/mouse will not function. To rectify this problem, you need to recompile the linux kernel with USB HID support enabled. You will need to have the reference software Linux kernel and root file system source code untared on your host/build machine. Please refer to the "BroadcomReferencePlatformSetup.pdf" guide for more information about unpacking these sources.

6.3.1 Broadcom 2.6.12 and 2.6.18 Kernels

Once you have both root file system and kernel unpacked, you will need to manually configure the kernel to enable additional options for the USB keyboard and mouse. First you will need to choose and copy the required kernel configuration file. These can be found in the following subdirectory from the Linux root:

"arch/mips/configs"

There are many different configurations for each platform, with options for SMP/non-SMP, BIG (be) / little endian, type of flash (e.g. NAND) and type of root filesystem (e.g. initrd). For example, to configure the kernel for an SMP-enabled BCM97405B0 platform, you will need to copy the "bcm97405b0-smp_defconfig" to .config.

Example:

```
cd $LINUX
cp arch/mips/configs/bcm97405b0-smp_defconfig .config
```

Next, you will need to configure the kernel with additional options. To do these enter the following commands from the Linux kernel root directory:

```
make oldconfig
make menuconfig
```

You will need to select the "**Device Drivers**" option and then "**Input Device Support**". Please press <**Y>** to choose "**Generic input layer (needed for keyboard, mouse, …)**". Next navigate to the "**Event interface (NEW)**" option and press <**Y>** to select it.

Exit this menu and then within the "**Device Drivers**" menu choose "**Character devices**". Press <**Y>** to select "**Virtual terminal**" and then exit this menu.

Still within the "**Device Drivers**" menu, navigate to "**USB support**" and press <**ENTER>**. Then within the "**USB support**" menu, navigate to the option "**USB Human Interface Device (full HID) support**" and press <**Y>**. Finally exit all menus and choose "**YES**" to save your new kernel configuration.

Once you have created your new kernel configuration, you will need to overwrite the original one. Please enter the following commands from the Linux source root directory (i.e. $LINUX):

```
chmod u+w arch/mips/config/bcm97405b0-smp_defconfig
cp .config arch/mips/config/bcm97405b0-smp_defconfig
```

Where: "bcm97405b0-smp_defconfig" needs to be substituted with the name of your preferred configuration file.

To rebuild just the kernel you will need to enter your root file system source directory and issue the following command:

```
make -f build.mk vmlinuz-7405b0-smp TFTPDIR=/tftpboot
```

Where:

"7405b0-smp" needs to be substituted for your chip and kernel configuration.

"TFTPDIR" points to place on host/build machine where the newly generated kernel will be located.

At this point, you can then re-flash the kernel on to your target platform and USB keyboard and mouse should function (please refer to the "BroadcomReferencePlatformSetup.pdf" guide for more information on how flash a kernel). To test the keyboard and mouse, you can run the "df_input" test application and you should immediately see what key presses or mouse movements/clicks are captured and displayed.

6.3.2 Broadcom 2.6.31 Kernels

The Linux kernel build process for 2.6.31 kernels has been overhauled and simplified compared to previous Linux kernel versions. To configure the kernel with additional options for USB, you will need to enter the following command from the root file system source directory:

```
make defaults-XXXXYY
```

Where:

XXXX is the chip number (e.g. 7420)

YY is the chip revision (e.g. b0)

Next type:

```
make menuconfig-linux
```

You will need to navigate to the "**Device Drivers**" option and press **<ENTER>**. Then you will need to navigate to the "**Input device support**" option and press **<ENTER>**. Within this sub-menu you will need to press **<Y>** for "**Generic input layer (needed for keyboard, mouse, …)**". Next navigate down to the "**Event interface (NEW)**" option and press **<Y>** to enable it. Next navigate down to the "**Hardware I/O ports**" line and press **<ENTER>**. Ensure all options in this sub-menu are **NOT** selected. If any are, then please navigate to the option and press **<N>** to deselect it. Exit this menu and the "**Input device support**" menu.

Within the "**Device Drivers**" menu, navigate down to the "**Character devices**" option and press **<ENTER>**. Next press **<Y>** at the top of this sub-menu to enable the "**Virtual terminal**". Exit this menu.

Within the "**Device Drivers**" menu, please ensure that the "**HID Devices**" and "**USB Support**" options have both been selected (they should have an asterisk by them). Finally exit all menus selecting "**YES**" to save your new kernel configuration.

Finally type:

```
make kernel
```

To rebuild the kernel with USB HID and input event interface support.

Once the new kernel has been created (in "images" directory), you can re-flash the target platform with it (please refer to the "BroadcomReferencePlatformSetup.pdf" guide for more information on how to do this).

To test the keyboard and mouse, you can run the "df_input" test application and you should immediately see what key presses or mouse movements/clicks are captured and displayed.

6.4 MAKING A LIMITED DIRECTFB RELEASE

If the user has a DirectFB-1.4.5 Phase 1.5 full release from Broadcom, then a limited release can be made that doesn't have any Broadcom specific SLA source code. This limited release can be useful to provide to customers who have not signed a Broadcom SLA, but who would still like to test and/or develop DirectFB applications on suitable Broadcom reference platforms.

A limited release does not contain the "DirectFB/src/broadcom_files/private" directory, but does contain an alternative directory "DirectFB/src/broadcom_files/precompiled". This "precompiled" directory contains all necessary precompiled libraries and kernel modules as well as scripts to be able to run the DirectFB applications. A subdirectory under "precompiled" is made for each platform, Linux version, architecture (big or little endian) and debug/release mode. For example, for a BCM97405C0 platform running with a Linux 2.6.18-7.1 SMP kernel in DEBUG mode with the default little endian tool chain, the subdirectory would be named:

97405c0_mipsel-uclibc_2.6.18-7.1-smp_debug

Both user-mode and kernel-mode (proxy) Nexus drivers would be placed in this directory, but they would be all compiled with debugging enabled.

To make a full featured "limited" release, four passes are required in order to have both user-space and kernel-space nexus drivers in debugging and release modes:

1. Build linux-fusion and nexus with debugging enabled and nexus drivers in kernel mode (proxy). Then do a "make limited".
2. Build nexus with debugging enabled but with the drivers in user-space. Then do a "make limited".
3. Build linux-fusion and nexus with debugging disabled and nexus drivers in kernel mode (proxy). Then do a "make limited".
4. Finally build nexus with debugging disabled and nexus drivers in user-space. End with a "make limited".

To make a full "limited" release, please follow the four steps that follow on the next page. It is recommended that the user initiate a "make distclean" first and answer "y" to clean out any previous "precompiled" directory (and subdirectories).

```
e.g.
cd /AppLibs/opensource/directfb/build/1.4.5
make DIRECTFB_MULTI=y distclean
```

After the four steps have been completed, the user can then use the resulting tarball "DirectFB-1.4.5_Limited_Phase1.5_YYYYMMDD.tgz", as a finished "limited" release.

6.4.1 Step 1 - Making a debug limited release with kernel-space Nexus drivers

```
cd /AppLibs/opensource/directfb/build/1.4.5
export MODE=proxy
export KERNELMODE=y
export DEBUG=y
make -C /nexus/build clean
make -C /nexus/build
make DIRECTFB_MULTI=y uninstall-target limited
```

6.4.2 Step 2 - Making a debug limited release with user-space Nexus drivers

```
cd /AppLibs/opensource/directfb/build/1.4.5
unset MODE
unset KERNELMODE
export DEBUG=y
make -C /nexus/build clean
make -C /nexus/build
make DIRECTFB_MULTI=y uninstall-target limited
```

6.4.3 Step 3 - Making a non-debug limited release with kernel-space Nexus drivers

```
cd /AppLibs/opensource/directfb/build/1.4.5
export MODE=proxy
export KERNELMODE=y
export DEBUG=n
make -C /nexus/build clean
make -C /nexus/build
make DIRECTFB_MULTI=y uninstall-target limited
```

6.4.4 Step 4 - Making a non-debug limited release with user-space Nexus drivers

```
cd /AppLibs/opensource/directfb/build/1.4.5
unset MODE
unset KERNELMODE
export DEBUG=n
make -C /nexus/build clean
make -C /nexus/build
make DIRECTFB_MULTI=y uninstall-target limited
```

7 CHANGES

This lists the changes to the deliverables since the previous DirectFB-1.4.5 Phase 1.0 release.

7.1 BUILD SYSTEM

The build system also now allows for graphics packet buffer support to be built (default) or not. Packet buffer support provides increased graphics performance with lower CPU loading. With packet buffer support enabled, Trapezoid drawing can be accelerated.

The IR protocol and keycodes mapping default settings has been updated. The user can now specify a more human readable name for the protocol and keycodes file. Further information can be found later in section 7.3.

7.2 GRAPHICS DRIVER

The graphics driver has been overhauled to provide improved performance. This has been achieved by using the magnum “packet-buffer” APIs. These APIs allow the DirectFB graphics driver to access the software GRC command ring buffer directly to insert the desired M2MC commands. By accessing the command buffer directly, the software overhead of performing a blit or fill operation has been significantly reduced. Packet buffer operation in the graphics driver is now enabled by default as long as the “magnum/portinginterface/grc/$(BCHP_CHIP)/bm2mc_packet.h” packet buffer API header file is present.

The driver now supports the SetMatrix() API. This API allows affine matrix operations to be applied during blitting and drawing operations. If the platform has a 3D PX3D core, then shearing and arbitrary rotation is enabled. If not, then only translations and mirroring are enabled.

Support for the “FillTrapezoid()” API has now been incorporated into the core DirectFB code and the graphics driver. The “FillTrapezoid()” call is h/w accelerated by either the PX3D or M2MC cores. If the PX3D core is present, then the “DrawLine() / DrawLines()” API are also h/w accelerated.

In previous versions of the graphics driver, overlapping blits were not h/w accelerated. This restriction has been removed for both Blit() and StretchBlit() cases.

Dual-source batch blitting has now been accelerated in our graphics driver. Dual-source batch blitting allows both the source and destination feeders to be used together to blend two source surfaces in one pass of the M2MC.

Support for YUY2, AYUV and UYVY pixel formats are now available in the graphics driver. Prior to this, it was not possible to h/w accelerate YUV images and a black screen would be displayed.

7.3 IR AND FRONT PANEL DRIVER

The IR input driver has been modularized to allow run-time selection of the IR protocol and IR keycodes mapping file. The default IR protocol and keycodes mapping file are still defined in the build system, but the user can now override this default at compile-time and/or run-time. This means that it is now possible to change the protocol used at run-time without having to recompile DirectFB.

The run-time selection is available with the following DirectFB config options:

bcmnexus-ir-protocol

bcmnexus-ir-keycodes

By default, the STB platforms will use the "RemoteA" keycodes and IR protocol whereas the newer DTV platforms will use "Generic" instead. "Generic" is used with the Broadcom small silver handset on DTV platforms like the BCM935230 and BCM935125. "RemoteA" is the One-For-All handset or black slim handset and is used on all STBs and older DTV platforms like the BCM93556.

If the user would rather use the silver handset to control STBs, then the "CirNec" IR protocol and keycodes file can be specified at run-time.

Example1: Choose silver handset (NEC protocol) on a STB:

```
./rundfb.sh df_input --dfb:bcmnexus-ir-protocol=CirNec,bcmnexus-ir-
keycodes=CirNec
```

Example2: Choose silver handset (NEC protocol) on a BCM935230 DTV platform:

```
./rundfb.sh df_input -dfb:bcmnexus-ir-protocol=BrcmGeneric,bcmnexus-ir-
keycodes=generic
```

The choice of the IR protocol name can be found in the "DirectFB-1.4.5/inputdrivers/bcmnexus/core/bcmnexus_ir_inputmode.h" header file. This is an auto-generated header file that extracts the name of the input modes from Nexus. The choice of IR keycode mapping file can be found in what keycodes modules are built and present in the /usr/local/lib/directfb-1.4-5/inputdrivers/bcmnexus" target directory. For example, if "libdirectfb_bcmnexus_ir_keycodes_cirnec.so" is present, then "cirnec" can be specified as the keycodes mapping file at run-time.

If the user wishes to support a different IR protocol or handset, then a new keycodes mapping file will have to be created and added to the DirectFB build system.

If the user would like to override the default IR protocol and keycodes file at build time, then the following environment variables can be set to override the defaults:

- DIRECTFB_IR_PROTOCOL=xxxxx
- DIRECTFB_IR_KEYCODES=yyyyy

NOTE: The default IR protocol is "BrcmGeneric" on the BCM935230 and BCM935125 platforms and "RemoteA"on all other platforms. The default IR keycodes setting is "generic" on the BCM935230 and BCM935125 platforms and "remotea"on all other platforms.

The front panel and IR receiver drivers have now been updated to mimic the behavior of a keyboard by default. This means that if the user presses a key, a single DIET_KEYPRESSED event is generated and when the key is released a single DIET_KEYRELEASED event is generated. If the key is held down for longer than the "skip" count, single DIET_KEYPRESSED events are generated with the DIET_REPEAT flag set.

If the user would like to revert to using the original mechanism whereby both DIET_KEYPRESSED and DIET_KEYRELEASED events are generated together, then the following runtime DirectFB options can be specified in the directfbrc file (or set in the **DFBARGS** envar):

bcmnexus-ir-timeout=0

bcmnexus-key-timeout=0

The list of all IR and KEYPAD options can be found at run-time by using the help option:

```
e.g. ./rundfb.sh <app> --dfb-help
```

For example, the IR repeat filter time can be specified with the following run-time option:

bcmnexus-ir-repeat-time=xxx

7.4 System Driver

The system driver has been re-architected to better support multi-process applications. The "master" DirectFB application handles all Nexus surface creation and destruction in addition to Nexus memory allocations/freeing. On top of this all Nexus display calls are routed to the "master" DirectFB application to ensure that only one process receives the callbacks.

The screen part of the system driver has now been updated to support the DirectFB Encoder API. The functions "*SetEncoderConfiguration()*" can now be used to specify the output resolution, frequency and scanmode. Prior to this API being implemented, the user could only specify the video output resolution and not whether 50Hz/60Hz output should be made available (using the "*SetOutputConfiguration()*" API call).

Support for 1080p/720p 24Hz / 25Hz /30Hz has now been added to the system driver. The user can now specify either one of these resolutions at startup using the DirectFB configuration run-time option "res=xxx". Alternatively, one of these video output resolutions can be chosen using either of the above API calls. To see this in action, the user can run the "df_andi" test application and press the "O" button on the target platform's USB connected keyboard. Each press will cycle around a different Output resolution.

The layer handling code inside the system driver has been overhauled to support multi-process applications better. It now issues linux fusion RPC calls from DirectFB clients to the "master" DirectFB client when the graphics framebuffer needs to be set or when display settings need to be adjusted.

7.5 ImageProvider Driver

The ImageProvider() driver for Nexus has been overhauled to use internal DirectFB APIs, rather than using Nexus surface and graphics calls to blit from the decoded output to the final destination surface. This means that the DirectFB graphics driver is used to perform the blit (or stretch blit) from the decoded surface to the final destination surface. This change was necessary to support the new packet-buffer graphics APIs and to allow multi-application DirectFB to work reliably. The changes to support this, required a new DirectFB pixel format to be added called DSPF_ALUT8. This format is what the still image decoder (SID) hardware outputs for palletized images.

7.6 PUBLIC API CHANGES

Table 6 – Public Function API changes

Function	Change
FillTrapezoids()	This is a new API that has been added to DirectFB to allow Trapezoids to be drawn using either hardware acceleration or software fallbacks.

Table 7 – Public Definition API changes

Definition	Change
DSPF_ABGR	This new pixel format is necessary to support VC-4's texture pixel output format.
DSPF_ALUT8	This new format is necessary to support the still image decoder's output.
DFBSurfaceDescription	The "preallocated" structure inside this structure has been increased to hold 3 buffers along with a handle. This change is necessary to support VC-4 OpenGLES 2.0 applications.

8 TEST NOTES

8.1 TESTING THE IR INPUT

DirectFB can accept input not only from a USB keyboard or mouse, but also from an IR handset. The current DirectFB IR driver supports the standard “One-For-All” programmable remote control that comes with the set-top reference platform and older DTV platforms. The remote control needs to be set to “Cable” (CBL) or “STB” for DirectFB to recognize the key presses.

For newer DTV platforms like the BCM935230 and BCM935125, the DirectFB IR driver can support the NEC protocol as used in the “silver” Broadcom “generic” remote control. Set-top reference platforms can also support this protocol if they specify the “CirNec” IR protocol and keycodes mapping at run-time.

The best application to test the IR remote control is “df_input”. You can run this test application by entering in the following command:

```
cd /usr/local/bin/directfb/1.4
./rundfb.sh df_input
```

Then you can press any key on the handset and you should see the name of the key and key code displayed on the display. If the key is held down on the remote control, the repeat event should be set.

8.2 TESTING THE FRONT PANEL INPUT

The front panel input can also be tested in the same manner as for the IR input. However, there is a known issue with the Nexus/magnum drivers that requires the LED controller to be initialized first. Instead of placing the burden on DirectFB to initialize the LED controller (which could interfere with an application that already opens the LED controller), it was decided instead to wait for the drivers to rectify this initialization problem. As a result, it is necessary to run a test application first to initialize the front panel LED controller prior to running any DirectFB test application that requires front panel input control. The nexus "frontpanel" example application is recommended to be run first prior to running the require DirectFB application.

Example:

```
./nexus frontpanel
./rundfb.sh df_input
```

NOTE1: This assumes you have already compiled the Nexus example applications and have copied the executables and "nexus" script to "/usr/local/bin/directfb/1.4". If you are unsure of how to compile the Nexus example applications, then please refer to the document "Nexus_Usage.pdf" that should be part of the reference software release.

NOTE2: The BCM935230 platform does not have support for the keypad driver at this time.

8.3 TESTING DIFFERENT BLITTING AND DRAWING MODES

There is a specific test called “df_brcmTest” that runs through many different blitting/blending and drawing operations with the hardware acceleration output displayed in a window on the left-hand side the screen and the software fallback mechanism displayed in its own window on the right-hand side the screen (side-by-side for comparison). The user simply needs to press the **<OK>** or **<SELECT>** key on the IR handset to progress through the different tests.

The test also accepts setting the “blittingflags” and “drawingflags” environment variables to test additional blitting and drawing scenarios (such as Destination color keying). For example, to test source color keying on all blitting/blending test cases, you need to set the “blittingflags” environment variable as follows (prior to running the test).

```
export blittingflags=0x08
```

To test destination color keying for all blitting test cases, you need to set the environment variable as follows:

```
export blittingflags=0x10
```

NOTE: These values are determined by looking at the “DirectFB-1.4.5/include/directfb.h” header file and reviewing the “DFBSurfaceBlittingFlags” typedef.

The test defaults to an ARGB pixel format for the graphics layer/plane, but any valid pixel format can be set by modifying the df_brcmTest.c file in the “DirectFB-1.4.5/tools” directory and setting the following define to the required format:

```
#define PRIMARY_PIXELFORMAT    DSPF_XXXX
```

Where: XXXX is one of the pixel formats as defined in “DirectFB-1.4.5/include/directfb.h”.

8.4 PERFORMANCE TESTS

Both “df_andi” (Penguins) and “df_dok” are good benchmarking tests to check the overall graphics performance of the target platform.

“df_andi” measures real-world blitting performance by seeing how many penguin blits can be sustained at a given number of frames per second. The graphics performance is proportional to the number of penguins and fps (frames per second) displayed at the top left-hand corner of the screen. The number of penguins can be increased by pressing the **<S>** key on the target platform’s USB keyboard. Also, the number of penguins can be decreased by pressing the **<D>** key on the keyboard. Pressing the **<SPACE>** bar will cause the penguins to form a logo and pressing **<R>** will cause them to revert to moving around the screen. Pressing **<P>** will power down or power up the screen. Pressing **<O>** will cause the output resolution to change. Pressing **<Q>** will quit the application.

“df_dok” is a true benchmarking test that measures CPU load and graphics blitting / drawing performance. It shows the CPU load in square brackets along with a print out of the number of graphics operations per second of each test (e.g. Mpixels/s, Kchars/s).

NOTE: There will be a significant difference in CPU load when running DirectFB in release mode vs debug mode. To obtain the best results, always build DirectFB with DEBUG=n (release mode).

NOTE: Building the Nexus and magnum drivers in release mode (DEBUG=n) will also provide a significant increase in some hardware accelerated operations.

8.5 SUPPORTED PLATFORMS

The table below lists the DTV and Set-Top platforms that this release of DirectFB supports.

Table 8 – Supported Platforms

Platform	Comment
BCM93548	DTV ATSC market
BCM93556	DTV DVB-T market
BCM935230	DTV ATSC / DVB market
BCM935125	DTV ATSC / DVB market
BCM97019	
BCM97125	
BCM97208	
BCM97231	
BCM97325	
BCM97335	
BCM97336	
BCM97340	
BCM97342	
BCM97344	
BCM97346	
BCM97400	
BCM97401	
BCM97403	
BCM97405	
BCM97408	
BCM97413	
BCM97420	
BCM97422	
BCM97425	
BCM97468	
BCM97540	
BCM97550	.