Reference Software 64-bit Migration Application Note

Broadcom
1320 Ridder Park Drive
San Jose, California 95131
broadcom.com

Table of Contents

1 Introduction

The current Set Top Box (STB) chips from Broadcom such as the 7268 and 7271 feature a new processor design based on the 64-bit A53 CPU from ARM. This document gives information on how to migrate applications to these new processors and gives some guidance on the benefits of the different configurations available.

2 64-bit vs 32-bit operation modes

The B53 CPU (Broadcom's implementation of the ARM A53 architecture) can be run in either a 32-bit compatibility or native 64-bit mode.

The 32-bit compatibility mode offers a full hardware emulation of the armv7l architecture so existing software written for 32-bit ARM CPUs should be usable without modification, but doesn't offer any of the advantages of the new processor design.

64-bit native mode may require some software modifications as the size of various data types has changed. Also the Linux kernel uses a different architecture specific driver (arm64 vs arm) so the availability of some features may have changed.

3 BOLT – Boot loader & boot strap options

The 7268 and 7271 have a 32/64-bit boot strap option to control the type of image which can be run. If the strap is set to 32-bit, only 32-bit images may be run. In 64-bit mode it is possible to boot both 32-bit and 64-bit images. It is recommended that you leave the strap set to 64-bit to allow all images to boot.

There is a single BOLT binary for each family of chips (7268, 7271) which can boot both 64-bit and 32-bit images.

4 Linux Kernel & tool chain – stblinux 4.1 & stbgcc 4.8

To support the new CPU architecture Broadcom has release a new port of the 4.1 Linux kernel. This kernel supports both 32-bit and 64-bit architectures. The default kernel image for 7268/7271 is a native 64-bit image, but it is possible at compile time to select a 32-bit image. Broadcom will provide a compiled image for both 32-bit and 64-bit modes. The 32-bit image name has the suffix -32.

For example, the following command will boot a 32 bit kernel:

```
BOLT> boot <network_location>:41-1.2/vmlinuz-initrd-7271a0-32
```

In order to boot a 64 bit kernel use:

```
BOLT> boot <network_location>:41-1.2/vmlinuz-initrd-7271a0
```

The stbgcc tool chain release contains both 32-bit and 64-bit compilers and associated tools. The 32-bit compiler is still arm-linux-gcc whereas the 64-bit compiler executable is aarch64-linux-gcc.

To compile the kernel and root file system from sources:

```
# tar -xvjf rootfs-4.1-1.2.tar.bz2

# tar -xvjf stblinux-4.1-1.2.tar.bz2

# export PATH=$PATH:/opt/toolchains/stbgcc-4.8-1.5/bin

# cd rootfs

# make images-7271a0-32 # (images-7271a0 for 64-bit)
```

N.B. 7268 and 7271 use the same kernel image so always build the kernel for 7271a0.

5 Reference software

The Broadcom reference software can be built for 32-bit or 64-bit mode. At this time, 32-bit Nexus must run on a 32-bit kernel and 64-bit Nexus must run on a 64-bit kernel. No 32-bit to 64-bit translation exists.

Below is a sample environment required to build a 64-bit 7268 image:

```
# export PATH=$PATH:/opt/toolchains/stbgcc-4.8-1.5/bin

# export LINUX=/opt/stblinux-4.1-1.2/

# export B_REFSW_ARCH=aarch64-linux # (arm-linux for 32-bit)

# export NEXUS_PLATFORM=97268 # (or 97271)

# export BCHP_VER=a0
```

Broadcom has ported its reference applications, including nexus examples, NxClient and Atlas to 64-bit mode. Other drivers and applications may require porting to 64-bit mode.

6 64-bit advantages & disadvantages

Pros:

Large virtual address compared to 32-bit platform. For 32-bit platforms, in some configurations, the Nexus kernel mode driver ran out of virtual address space when mapping physical memory.

With a larger available memory map the Linux kernel reserves more space for kernel modules. With the 32-bit ARM kernel 14MB is available for kernel modules and for some projects this limit was an issue. With the arm64 kernel there's 64MB available for modules.

Better support for large memory systems. On a 32-bit system, supporting an address space larger than 4GB (registers and peripherals also have to be memory mapped as well as RAM) required Linux LPAE (large physical address extensions). LPAE creates the larger address space by adding an extra level of lookup table during address translation which impacts memory access performance. On a 64-bit system, all memory can be addressed directly.

The armv8 architecture offers a number of advantages for performance:

- Double the number and double the size of general purpose registers:
    - Cheaper function calls
    - Less stack spillage
    - More registers for functional use
- Hard float ABI by default

For more details see the links below:

ARMv8-A Porting Guide

ARMv8-A Architecture

Cons:

Drivers and applications ported to 64-bit platforms occupy a larger code and data size. As well as data and instructions being larger you may also see issues with different padding inside structures due to alignment constraints.

Kernel mode drivers must be ported to arm64 architecture.

User application could need modification to operate correctly in a 64-bit system if the application needs access to more than 4GB address space, uses pointers, direct memory access, or file access.

7 Reference software image size comparison

Due to the differing sizes of data and instructions in aarch64 there is a noticeable difference in the size of applications and libraries when comparing 32-bit vs 64-bit binaries. Below is a table comparing the sizes of various stripped (all debug information removed) applications and binaries generated for 7268 using the URSR 16.2 release. N.B. URSR defaults to building code optimised with the -Os options.

User mode

Binary / size in Kb	aarch64	arm
bcmdriver.ko	72.9	62.4
libnexus.so	14,617	14,414
nxserver*	224.3	234.9
NxClient play	133.6	127.8

Proxy (kernel) mode

Binary / size in Kb	aarch64	arm
nexus.ko	14,292	11,302
libnexus.so	8,167	8,154
nxserver*	224.3	234.9
NxClient play	133.6	127.8

* The nxserver results are unusual but may be due to the architectural advantages in ARM v8 as most of the code is concerned with argument parsing and executing conditional paths in building the correct software configuration.

8 Reference software memory usage comparison

The results below show the approximate memory usage deltas between 32 and 64-bit modes. The results were obtained by capturing the information from /proc/meminfo before and after loading the relevant kernel module and starting nxserver. The difference between the MemAvailable fields is shown below:

Nexus Mode / size in MB	aarch64	arm
Proxy	15.2	13.2
User mode	7.3	6.0

9 C code porting notes

When writing code for aarch64 a number of data types are of a different size when compared to the arm architecture.

C Data type / bytes	aarch64	arm
char	1	1
short	2	2
int	4	4
long	**8**	**4**
long long	8	8
float	4	4
double	8	8
size_t	**8**	**4**
void*	**8**	**4**

In porting the reference software most of the issues discovered were around the handling of pointers. . Pointers are now 64 bits and can no longer be stored in “int” datatypes.

Some of the things that we would recommend to customers to ease their migration would be:

- Avoid typecasts, as this will allow the compiler to warn or error on any type size issues.
- Fix all compiler warnings. When moving to 64-bit, a compiler “warning” often points to a bug, not a coding preference. Using -Werror may help to keep developers in this mind set.
- NEXUS_CallbackDesc.param is an “int”, but is often used to pass pointers or Nexus handles in 32-bit systems. Application code must be reworked to not do that. Use the void * option to pass a pointer to a structure if you need multiple handles.
- If using pointer arithmetic, stay in the pointer domain. For instance, typecast (void*) to (uint8_t*), then manipulate. Type casting to basic arithmetic types such as int or long should be avoided.
- Watch out for algorithms that requires integer overflow. It may overflow in 32 bit, but not in 64-bit. It’s best to just avoid the overflow.
- Code which relies on bit shifting or other similar operations such as bit fields could have unintended side effects in 64-bit code.

10 Mixed mode operation

Modern x86_64 Linux distributions often offer the ability to run 32-bit applications on top of a 64-bit kernel. It is possible to do this on ARM v8 systems too but requires a significant amount of effort. There are several major system level implications of supporting two ABIs concurrently.

- The system will require two sets of user space libraries e.g. c run time library.
- A custom driver will need to be developed to support calls from the two different ABIs.
- Any inter-process communication between applications must be aware of the ABI differences.

The reference software will also require changes in order to make it compatible.

- The memory layout of structures between 32-bit and 64-bit will differ as the alignment constraints differ between the ABIs so IOCTLs will need to cope with accessing the data from either format.
- Nexus handles are opaque pointers (pointers vary in size between the two ABIs) so a 32-bit Nexus application cannot store a 64-bit pointer for a handle in its data structures.
- In places the public Nexus API contains data types which change size dependent on the ABI. Changing the API would break existing applications.
- Unions in the Nexus API also further complicate the problem as their size would be ABI dependent and use case specific.

The reference software changes to support mixed mode operation are under way and this feature will be available in a future URSR release.