Revolution

Transition Guide

Version 1.00

"Confidential"

Table of Contents

Revision History

Revision No.	Date Revised	Items (Chapter)	Description
Version 1.00	3/22/2006	-	First release by Nintendo of America Inc.

1 Overview

This document highlights the major differences facing developers when migrating applications from Nintendo GameCube to Revolution, with an emphasis on architectural contrasts rather than performance ones.

The Revolution platform heavily leverages Nintendo GameCube to minimize porting challenges. The OS API is nearly identical, with legacy support for most deprecated functions.

This is a living document and is subject to change as new features are enabled in the Revolution Development Environment.

2 Assumptions

This document assumes that:

- The reader is familiar with the Nintendo GameCube development environment (either GDEV or DDH).
- The reader is porting from the HW2 revision of Nintendo GameCube.
- The reader is targeting the NDEV 1.0 version of the Revolution development kit.

As of this writing, there is only one version of the NDEV platform. Anticipated revisions of the NDEV CPU and ASIC do not embody any known differences that will impact developers.

However, device driver support for some peripherals (such as the final Revolution controller) are scheduled for later delivery and may impact existing APIs.

3 Memory Architecture

Nintendo GameCube features a single main memory of 24MB (“Splash”) that is directly addressable by the CPU. Augmenting this memory is an additional 16MB of slower ARAM, intended primarily for audio and accessible by DMA from the CPU.

The arena, heap, and memory allocation functions in the operating system manage the main memory only.

Revolution has two main memories:

- 24MB of 1TSRAM (`MEM1`) (24MB in development systems)
- 64MB of GDDR3 (`MEM2`) (128MB in development systems)

3.1 The Arenas

Both these memories are directly addressable by the CPU and are shared with other devices in the system (such as the graphics and audio cores). These memories are treated as two separate arenas. New arena APIs have been introduced to manage them.

Code 1 - New OS Arena Functions

```
// Arena APIs for 1TSRAM/MEM1 memory.
void* OSGetMEM1ArenaHi ( void );
void* OSGetMEM1ArenaLo ( void );
void  OSSetMEM1ArenaHi ( void* newHi );
void  OSSetMEM1ArenaLo ( void* newLo );

// Arena APIs for GDDR3/MEM2 memory.
void* OSGetMEM2ArenaHi ( void );
void* OSGetMEM2ArenaLo ( void );
void  OSSetMEM2ArenaHi ( void* newHi );
void  OSSetMEM2ArenaLo ( void* newLo );
```

Note that the legacy functions are still supported. The `MEM1` arena functions are simply wrappers for the original arena functions.

Code 2 - Wrapper functions for MEM1 Arena Functions

```
#define OSGetMEM1ArenaHi(x) OSGetArenaHi(x)
#define OSGetMEM1ArenaLo(x) OSGetArenaLo(x)
#define OSSetMEM1ArenaHi(x) OSSetArenaHi(x)
#define OSSetMEM1ArenaLo(x) OSSetArenaLo(x)
```

Applications ported to Revolution will therefore operate from `MEM1` by default.

3.2 Managing Heaps

While the Nintendo GameCube OS supports multiple heaps in a given arena, the heaps must be contiguous. Thus, we cannot use the legacy APIs to create or manage heaps in the `MEM2` (GDDR3) arena.

The Revolution SDK introduces a new library:

```
(REVOLUTION_SDK_ROOT)/build/libraries/mem
```

This is an advanced memory manager that supports both arenas and dynamic memory allocation. Source code is provided. For more details, refer to the MAN pages and the associated demo program.

4 Graphics Library (GX)

The v graphics library is nearly identical to Nintendo GameCube.

4.1 Performance Metrics APIs

The performance-metric APIs are not final. Optimization techniques for GX are pending as well.

4.2 Graphics FIFO

Some minor API changes exist for the graphics FIFO. The following APIs have been deleted:

Code 3 - Deleted GX FIFO APIs

```
GXGetFifoStatus()
GXSaveCPUFifo()
GXSaveGPFifo()
```

The following APIs have revised arguments and return types:

Code 4 - Revised GX FIFO APIs

```
GXGetCPUFifo()
GXGetGPFifo()
GXGetFifoPtrs()
```

Please refer to the GX MAN pages for more information.

4.3 Graphics FIFO Restriction

It should be noted that the FIFO is restricted to MEM1; it cannot be placed in MEM2. All other data, however, can be placed in either memory at the developer's discretion.

5 Video Interface (VI)

The VI library is nearly identical to that of Nintendo GameCube. New APIs have been added to accomodate the new trap filter and gamma correction features. Refer to the VI man pages for more information.

Note that the external frame buffer (XFB) can be placed in either MEM1 or MEM2.

6 Audio Library (AX)

6.1 Frame Time and New Effects Bus

Audio frame times have been changed from 5 msec to 3 msec. This modification frees memory in the DSP core each frame, allowing the addition of an extra effects bus (AUXC).

This impacts the DLS conversion tool (DLS1WT) because its calculations for LFOs and envelopes are based on frame size. The tool has been revised. If you are using DLS samples in your application, they will need to be re-converted.

6.2 No ARAM

ARAM has been removed from the memory hierarchy and replaced by GDDR3 (MEM2). The associated APIs (AR/ARQ and AM) have been obsoleted and are no longer available.

The DSP can access samples from either MEM1 or MEM2. Applications must be modified to:

- Allocate a block of memory.
- Read samples into said block of memory from disk.
- Notify the Sound Pipeline library (or other abstraction) of the samples' addresses.

Note that the application is still responsible for specifying a zero buffer.

For examples of how to make these modifications, please refer to the AX and SP demos.

6.3 Voice Count

As of this writing, the voice count for the audio subsystem is the same as Nintendo GameCube because the DSP and AX libraries have not been optimized for the new memory hierarchy. Revised libraries are forthcoming.

7 Audio Library (AI)

The Revolution console does not support hardware audio streaming from disc. The related APIs have been obsoleted.

Note that a library to support software-based audio streaming from disk is forthcoming.

8 GC Memory Cards

Nintendo GameCube Memory cards are supported through the legacy CARD library. This allows applications to import data from Nintendo GameCube titles.

Writes to Nintendo GameCube memory cards are permitted, but strongly discouraged for production applications. The CARD library is supplanted by the NAND API for saving and restoring game data.

For more information, please refer to the Revolution API Manual and the NAND demo programs.

9 System Configuration (SC)

Nintendo GameCube system preferences are accessed through a series of OS APIs. These are maintained for access to the legacy Nintendo GameCube system preferences in the SRAM. The SRAM is maintained for Nintendo GameCube backwards compatbility mode.

Code 5 - Legacy Nintendo GameCube System Preference APIs

```
OSGetEuRgb60Mode()
OSGetLanguage()
OSGetPhysicalMemSize()
OSGetPhysicalGDDR3Size()
OSGetProgressiveMode()
OSGetSoundMode()
OSSetEuRgb60Mode()
OSSetProgressiveMode()
OSSetSoundMode()
OSSetLanguage()
```

While applications can access the Nintendo GameCube system preferences, they must eventually migrate to the Revolution-specific system configuration library (SC).

The Revolution configuration parameters are a superset of the Nintendo GameCube parameters. Common parameters will be synchronized between the legacy SRAM and NAND-based system configuration file during boot.

Please refer to the Revolution API Manual for details on the SC library. A demo program is also available.

10 Network Support and Host I/O

For the intial release of the NDEV system and SDK, network is not supported. These are expected in a subsequent release.

However, NDEV does support Host I/O similar to that of Nintendo GameCube. Refer to the SDK documentation for more details.

11 Controller Libraries (PAD, WPADemu, and WPAD)

The Revolution platform supports legacy Nintendo GameCube controllers via the PAD library.

As of this writing, the Revolution controller is emulated using a serial-I/O based hardware prototype and the WPADEmu library.

Note that the WPADEmu and PAD libraries cannot be used simultaneously due to conflicts at the serial interface.

Developers are encouraged to link one or the other depending on their needs. For applications in which legacy PAD calls exist, the PADEmpty library is provided to facilitate compilation.

The build system accomodates this through the WPADEMU_LIB flag. Simply assert this flag on the command line:

```
% make WPADEMU_LIB=TRUE
```

or within your makefile. Note that the WPADEMU_LIB flag must be asserted before commondefs is included.

Code 6 - Makefile example for WPADEMU_LIB

```
#
# Must specify WPADEMU_LIB for this demo
# Must be specified before commondefs is included!
#

WPADEMU_LIB=TRUE

include $(REVOLUTION_SDK_ROOT)/build/buildtools/commondefs
.
.
.
```

If the WPADEMU_LIB flag is asserted for a program, then WPADEmu.a and PADEmpty.a are linked. Otherwise, PAD.a is linked.

As of this writing, the majority of the demos in the Revolution SDK are largely unchanged from Nintendo GameCube. As such, they rely on the Nintendo GameCube controller and therefore use the PAD library.

11.1 Plans for WPAD

As of this writing, the final wireless version of the Revolution controller is not ready for distribution. While it is functionally similar to the serial-I/O based device supported by WPADEmu, the wireless nature of the final controller may entail substantial API changes. Details are not available at this time.

Furthermore, the WPAD library will not be able to read Nintendo GameCube controllers through the same interface. This may impact the intrepid developers who have been using WPADEmu to facilitate debugging with Nintendo GameCube controllers.

Code 7 - Example: How some developers are using WPAD to read RVL and GC Controllers

```
#include <revolution/wpad.h>
#include <revolution/pad.h>

int main( void )
{
    WPADStatus   core;    // RVL controller(Game RIMOKON)
    WPADFSStatus fs;      // RVL controller(NUNCHAKU controller)
    PADStatus    gc;      // GC controller

    WPADInit();

    while( 1 )
    {
        s32 err=0x00u;
        u32 dev=0x00u;

        err = WPADProbe(WPAD_CHAN0, &dev);
        if (err == WPAD_ERR_NONE) {
            switch (dev) {
            case WPAD_DEV_CORE:      // RVL controller(Game RIMOKON)
                WPADSetDataFormat(WPAD_CHAN0, WPAD_FMT_CORE);
                WPADRead(WPAD_CHAN0, &core);
                break;
            case WPAD_DEV_FREESTYLE: // RVL NUNCHAKU
                WPADSetDataFormat(WPAD_CHAN0, WPAD_FMT_FREESTYLE);
                WPADRead(WPAD_CHAN0, &fs);
                break;
            case WPAD_DEV_DOLPHIN:   // GC controller
                WPADSetDataFormat(WPAD_CHAN0, WPAD_FMT_DOLPHIN);
                WPADRead(WPAD_CHAN0, &gc);
                break;
            default:
                break;
            }
        }
    }
    return 0;
}
```

Upon release of the final wireless Revolution controller, the associated `WPAD` library can be used simultaneously with `PAD`. Developers must modify their code to use these libraries for the appropriate controller.

We apologize for the inconvenience.

Table 1 - What devices need what libraries?

Device	SDK 1.0	SDK 2.0 (tentative)
Nintendo GameCube controller	`WPADEmu` or `PAD`	`PAD` only
SI-based RVL prototype	`WPADEmu`	n/a
Final RVL Controller (wireless)	n/a	`WPAD` only

IMPORTANT! The `WPADEmu` library will be deleted upon release of the wireless controller hardware and `WPAD` library. Developers will have to modify their code accordingly.