Panda3DS
Climbing the tree of 3DS emulation

George Poniris  Electrical & Computer Eng. Undergrad @ NTUA  gponiris@pandasemi.co

Paris Oplopoios  Information & Electronic Eng. Undergrad  parisoplop@pandasemi.co

Davit Margarian  Electrical & Computer Eng. @ UC San Diego  mar@pandasemi.co
What is Panda3DS?

Panda3DS is a Nintendo 3DS emulator for Windows, MacOS, Linux and Android. Some goals and aspirations include:

- Providing end-users with a pleasant experience playing their 3DS games on all their devices
- Creating a portable, modern, easy-to-maintain codebase
- Exploring new possibilities in 3DS emulation:
  - Virtualization, ubershaders, and more
- Researching the 3DS software and hardware architecture
- Expanding the red panda cult
- Aiding homebrew devs in writing their own 3DS software
- Fun (...mostly)!
A peek at **Panda3DS**

The SDL frontend  
`panda-sdl`

The Qt frontend  
`panda-qt`

The Android version  
`pandroid`
Agenda

Hardware Architecture
- CPUs (There's too many)
- GPU: The PICA200
- Audio DSP: XpertTeak
- The Rest of the System

3DS Software Stack
- Nintendo's Horizon OS
- The 3DS Userland

Emulating the 3DS
- The “Levels” of Emulation
- Difficulties & Points of Interest
- New & Unexplored Territory
First glance 3DS hardware

Source copetti.org
The Nintendo 3DS is capable of natively running Game Boy Advance, Nintendo DS, and 3DS software. Most of the hardware used to achieve this resides in a small System-on-a-Chip, named “CPU CTR”

Fun facts:
• Many people don’t know the 3DS can run GBA games natively, since only those who were part of Nintendo’s “Ambassador program” could use this feature officially.

Nowadays, there’s an open-source interface for running GBA games natively, called open_agb_firm.
• Diffused by Panasonic in Japan on their 45nm process
Inside the SoC: The ARM11

Running most of the code in 3DS mode

Original 3DS
- ARM11
- MPCore 268M Y2005
- 268 MHz

New 3DS
- ARM11
- MPCore 804M Y2005
- 804 MHz*
Inside ARM11 MPCore

ARmv6K Architecture
32-bit

VFPv2
Vector Floating Point
Faster processing of single/double precision floats

Media Instructions
for Video/Audio

MMU
for running multitasking OSes

16KB I-Cache

Branch Predictor

16KB D-Cache

Out-of-Order Completion
for some instructions

+ Multicore Coherency & More

3 Instruction Sets
ARM
Thumb
Jazelle
Making use of the multiple ARM11 cores

Aiming to make good use of the multicore ARM11, the 3DS OS allocates different tasks to each core.

**Core 0 Appcore**
- Runs userland apps including games & system apps

**Core 1 Syscore**
- Dedicated to OS. Runs many processes often known as “services” used to interface with world.
- Apps can borrow some syscore compute time.

**Core 2 New 3DS Only**
- Reserved for “QTM”, the camera-based head tracking service.

**Core 3 New 3DS Only**
- Available as another Appcore.
Inside the SoC: The **ARM9 & ARM7**

<table>
<thead>
<tr>
<th><strong>ARM946E-S</strong></th>
<th><strong>ARM7TDMI-S</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARMv5TE</strong> Architecture</td>
<td><strong>ARMv4T</strong> Architecture</td>
</tr>
<tr>
<td>32-bit</td>
<td>32-bit</td>
</tr>
<tr>
<td>Same model as DS/DSi ARM9</td>
<td>Same model as GBA/DS/DSi ARM7</td>
</tr>
<tr>
<td>66 MHz in DS compatibility mode</td>
<td>33 MHz in DS compatibility mode</td>
</tr>
<tr>
<td>133 MHz in DSi mode and 3DS mode</td>
<td>16 MHz in GBA compatibility mode</td>
</tr>
<tr>
<td><strong>3DS Mode</strong></td>
<td><strong>DS/DSi Compatibility Mode</strong></td>
</tr>
<tr>
<td>manages storage and cryptography hardware Cartridge/NAND/SD</td>
<td><strong>GBA Compatibility Mode</strong></td>
</tr>
<tr>
<td><strong>DS/DSi Compatibility Mode</strong></td>
<td><strong>Disabled in 3DS Mode</strong></td>
</tr>
</tbody>
</table>
CPU Intercommunication

64b AXI Main Bus
via Snoop Control Unit for inter-ARM11 cache coherency

"PXI" FIFO
64b FIFOs, bidirectional

"IPC" FIFO
64b FIFOs, bidirectional
DMP PICA200 Everyone’s favourite GPU

Me?

Nintendo’s first off-the-shelf GPU in a handheld.

*But most 3DS games don’t actually use GLES

Custom Programmable Shaders

Vertex Shaders

Geometry Shaders

Fragment Lighting
Per-pixel fixed-function effects

Hardware Shadow Mapping & Texture Projection
+ More Procedural Graphics Effects (DMP MAESTRO technology)

OpenGLES 1.1
Compatible Features

OpenGL ES 1.1
// Uniforms
.float fvec projection[4]

// Constants
.float myconst(0.0, 1.0, -1.0, 0.1)
.float myconst2(0.3, 0.0, 0.0, 0.0)
.alias zeros myconst.xxxx // Vector full of zeros
.alias ones  myconst.yyyy // Vector full of ones

// Outputs
.out outpos position
.out outclr color

// Input attributes
.alias inpos v0
.alias inclr v1
.proc main
    // Force the w component of inpos to be 1.0
    mov r0.xyz, inpos
    mov r0.w, ones

    // Output coords = projection matrix * input coords
    // 4 dot-products perform this matrix multiply.
    dp4 outpos.x, projection[0], r0
    dp4 outpos.y, projection[1], r0
    dp4 outpos.z, projection[2], r0
    dp4 outpos.w, projection[3], r0

    // Vertex out color = in color
    mov outclr, inclr

    end
.end
The pixel pipeline

Modern GPUs use programmable shaders to fill in pixels (fragments).

PICA200 uses a configurable pipeline.
Using the **Color Combiners**

The beanstalk texture is mixed with the lighting from a light source using the colour combiner – creating a sheen. On the leaf Kirby is standing on, there’s a darkening gradient from left to right.
Captain Toad: Treasure Tracker, a game known for being clever with all sorts of lighting and shadow effects
Showing off the PICA

The Legend of Zelda: Ocarina of Time 3D
using the PICA’s fog rendering hardware
Also texture compression with ETC1 and ETC1A4!
Showing off the **PICA**

Mario and Luigi: Paper Jam generates the seawater via procedurally-generated textures
Showing off the PICA

Super Mario 3D Land uses all sorts of features, such as stencil testing, logic ops, good usage of lighting, GPU command lists that invoke other command lists, and more
The XpertTeak DSP

More Processors! The 3DS also has a Digital Signal Processor for audio. It’s the same model as the DSi DSP but it’s used far more.

Most games shipped a common DSP firmware which includes:

- Support for up to 24 audio channels
- Multiple Audio Encodings
  - PCM8
  - PCM16
  - ADPCM
- AAC Audio Decoder
  Used in games like Pokémon X/Y
- Effects
  - Reverb
  - Delay
  - Filters
    - One-pole & Biquad
- Mono Input
- Stereo Input
- Stereo Output
Teak architecture

- Signal Processing Instructions
  Plenty for multiply, multiply-add, division, etc.

- 16-bit Bytes
  Instead of the typical 8b

- ARM11 Synchronization
  With data exchange and a semaphore register

- Loop Instructions
  Supporting tight loops

- 256KB Instruction Memory

- 256KB Data Memory

With awful complicated instruction encodings
128 / 256* MB FCRAM
64-bit “Fast Cycle RAM”
Built by Fujitsu

6 / 10 MB of VRAM
Inside the chip

One-Time-Programmable Memory
Stores console-unique keys

Arm CoreLink DMA
programmable
DMA engine

Hardware
Cryptography Engine
AES / SHA / RSA / RNG

WiFi Controller
Xtensa CPU

SD Card Slot
Expandable Game Storage

GBA & DSi Hardware
for game compatibility

eMMC NAND
Stores system data

IR Transceiver
for Amiibo & CirclePadPro

IR Front LED
for New 3DS Face Tracking

NFC Reader
for New 3DS Amiibo Support

Microcontroller
For extra IO & Power Management

2 LCDs
PDC PICA Display Controller
Bottom Display Resistive Touch

Motion Sensors
Gyroscope & Accelerometer

Cameras
2 Back Cameras Front Camera

IR Front LED
for New 3DS Face Tracking

Camera
Front Camera

Microcontroller
For extra IO & Power Management
Software stack!
The 3DS OS: Horizon®

To tame this hardware,
We have Nintendo’s beautifully architected operating system

We’ll look at Horizon on ARM11 Syscore
& Horizon on ARM9 for Security & I/O
Getting a firm grasp of FIRMs

The 3DS comes with multiple different “firmware” programs running on the ARM cores with low-level control of the underlying hardware:

- **SAFE_FIRM**
  - Older, bare-bones version of NATIVE_FIRM for recovery
  - Use a button combo to enter Safe Mode with System Updater

- **NATIVE_FIRM**
  - Runs 3DS Games Natively: “3DS Mode”
  - ARM11
    - Runs Userland
    - Majority of OS code

- **AGB_FIRM**
  - Runs GBA Games Natively
  - ARM7
    - Downclocked to GBA speeds
    - Runs game code

- **TWL_FIRM**
  - Runs DS or DSi Games Natively
  - ARM9
    - Downclocked to DS or DSi speeds
    - Runs game code
  - ARM7
    - Downclocked to DS(i) speeds
    - Runs game code

- **NATIVE_FIRM**
  - ARM9
    - Cryptography
    - Cartridge & SD I/O
  - Runs 3DS Games Natively: “3DS Mode”
A microkernel architecture

The kernel is the core of an OS. The 3DS ARM11 kernel is called kernel11.

- **Memory Management**: Map Memory to processes & configure their access permissions
- **Process & Thread Management**: Creation, multithreading primitives, Lifecycling
- **Service & Process Inter-communication**: Via the sync request API and shared memory

Kernel calls are performed with the Arm SVC (Supervisor Call) instruction.

An ARM assembly function for calling the `exitThread SVC`. EXPORT svcExitThread

<table>
<thead>
<tr>
<th>SVC</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>LR</td>
</tr>
</tbody>
</table>
HorizonOS services

**Service Handle**
- Request a service handle from “Service Manager”

**Set Up The Parameter Buffer**
- with the function to call
- and necessary parameters

**Send Request To Service**
- Call SendSyncRequest SVC with service handle & buffer

**Receive Output Buffer**
- Function-Specific Output Data
- Potential Error Codes
Some important services

**FS**
Filesystem IO for Cartridge, SD, Save Data & more

**DSP**
Communication with Audio DSP

**GSP**
GPU & Display Communication

**APT**
Interface with system applets & manage some app controls

**HID**
Interface with many input devices like gamepad, motion, ...

**CFG**
Read system configuration data like console model & region

**AC / HTTP / SSL / SOC**
Networking Tasks

**CAM / MIC / NFC**
Camera, Mic, & NFC Input

---

Function that asks the HID service to enable the gyroscope

```c
Result HIDUSER_DisableAccelerometer(void)
{
    // Get IPC command buffer for service data exchange
    u32* cmdbuf = getThreadCommandBuffer();
    // Setup the command header (0x13 = EnableGyroscope)
    cmdbuf[0] = IPC_MakeHeader(0x13, 0, 0);

    // Send buffer to service and wait for completion
    Result ret = svcSendSyncRequest(hidHandle);
    if(R_FAILED(ret)) // If SendSyncRequest failed,
        return ret;

    // Otherwise, return result code from HID service
    return cmdbuf[1];
}
```

adapted from libctru
Unlike ARM11, the ARM9 in 3DS mode handles all its tasks in one big process called Process9.

**Cryptography**
Data encryption & decryption with crypto hardware

**Device I/O**
Talks to various devices like Cartridge / SD Card / ...

**Overcomplicated C++**
That reverse engineers hate :(

**Really big**
A lil’ breather
Some live demos!
I knew this would fail

Here's a video instead
This failed too?

Here's a video instead
Emulation!
finally...
High & Low-Level Emulation

• **HLE** Reimplementing parts of the emulated system’s software in our own code, to avoid emulating the hardware needed to run said software.

Eg. An LLE Audio DSP is expensive to emulate performance-wise

**LLE & Actual Device**
- Audio DSP Hardware
- DSP Firmware

**HLE**
- C++ model reimplementing DSP firmware features
- DSP Firmware Compatible Interface

For the 3DS, we can HLE the OS:
- Kernel
- Services
- Process9

```c
void FSService::isSdmcDetected(u32 messagePointer) {
    log("FS::IsSdmcDetected\n");
    mem.write32(messagePointer, IPC::responseHeader(0x817, 2, 0));
    mem.write32(messagePointer + 4, Result::Success);
    mem.write8(messagePointer + 8, config.sdCardInserted ? 1 : 0);
}
```

HLE implementation of FS service function that returns if an SD card is inserted

We avoid emulating the complex SD hardware interface.
As an Emudev: What parts to LLE, what to HLE?

**LLE**
- Tedious to implement so much hardware
- That much hardware reduces performance and is error-prone.
- Beneficially, it can run any 3DS software incl. baremetal firms like 3DS Linux / GodMode9

**HLE**
- Tedious to implement so many services
- Performant but still error-prone.
- Many elements left to reverse engineer.

**Hybrid**
- We can HLE kernel11 & process9
- We can LLE most OS services.

**Balance**
- Minimizing work
- Improving Performance
- Maintaining Accuracy
As an Emudev: The CPU

**Interpreters**
Loop and process CPU instructions in normal code. Slow, portable, good for a start but not for fullspeed emulation.

**Just-in-Time (JIT) Recompilation**
Convert ARM code to host CPU code. This is the most common solution. Citra & Panda3DS both use the Dynarmic library to perform Arm32 to x86 / Arm64.

**Virtualization (Potentially)**
On Arm32/Arm64 devices, virtualization could be used to execute 3DS code natively. An ongoing Panda3DS PR is aiming to add this.

**Ahead-of-Time (AOT) Recompilation (Potentially)**
Recompile ARM code from the code section of 3DS executables to host CPU assembly ahead of time.
As an Emudev: The GPU

**Software Rendering**

Draws emulated triangle on the CPU in software.

Very slow but portable and simpler™

**How can we speed it up?**

- **Multithreading** drawing with several concurrent threads
- **Recompilers** Much like the CPU JIT, we can parse the PICA200 configuration for a draw call and generate optimized rasterization code at runtime.

**Hardware Rendering**

Draws on the GPU via a graphics API like OpenGL / Vulkan / DirectX / Metal. Much faster - suitable for gameplay.

**Challenges**

- Choosing the ideal API
- Efficient & Correct Surface Management
textures, color buffers, depth buffer
- Many, many other problems to solve
Emulating PICA shaders

### Interpreter
- Simple
- Too slow

### JIT on CPU
- Decent Performance
- But could be better

### Recompiling Shaders for GPU
- Good Performance
- Only for HW Rendering
- Might not be possible for select PICA shaders

```assembly
.proc main
    mov r0.xyz, inpos
    mov r0.w, ones

    ; outpos = projectionMatrix * inpos
dp4 outpos.x, projection[0], r0
    dp4 outpos.y, projection[1], r0
    dp4 outpos.z, projection[2], r0
    dp4 outpos.w, projection[3], r0

    ; outclr = inclr
    mov outclr, inclr

; We're finished
.end
```
Emulating the PICA pixel pipeline

**Specialized Shaders**
- Compile a specialized shader for each PICA pixel pipeline configuration.
- Low GPU Usage
- However lots of time is spent compiling shaders, causing stutters.
- Most common approach, currently WIP in Panda3DS

**Ubershaders**
- Include an entire “emulator” for the pixel pipeline inside a GPU fragment shader.
- Higher GPU usage but no compilation stutter.
- Works well on modern PC GPUs but struggles on mobile GPUs.
- Implemented in Panda3DS

**Hybrid emulation**
- Compile specialized shaders in the background. The ubershader is used for each draw call until the relevant shader is ready.
- Good performance with minimum stutter.
- Works well on all GPUs.
- Higher code complexity.

What Panda3DS wishes to achieve.
As an Emudev Audio DSP

**LLE**
How do we optimize it?
- Recompiling firmware
- AOT Compilation

**HLE**
Improving current DSP reverse engineering efforts
By making test ROMS & RE tooling.

**Teakra**
An emulator / assembler / disassembler for the Teak DSP used in Citra & MelonDS

**Techniques for optimized audio mixing**
SIMD / Multithreading / ...
Exploring new territory in 3DS emulation

Panda3DS comes with Lua scripting, including a text editor, so developers can make all sorts of scripts & mods, testing them fully within the app!
Exploring new territory in 3DS emulation

A Panda3DS dev branch running CTRAging, a factory test program some other emulators may struggle with

Panda3DS running on Wii, via HTTP streaming
Exploring new territory in 3DS emulation

Revolutionary UI (Has panda icons)

Play on Discord with all of your friends!
Star us on GitHub

Thx