“Gamecube Hacking”

1. Gamecube Hardware - what you can read everywhere
2. Gamecube Hardware - a bit more details
3. Homebrew - how to get your code to the cube
4. The boot process (and how to hack it)
5. Working around the encryption...
6. The ROM emulation hardware
7. Homebrew stuff
8. Linux

Gamecube Hardware

- Codenamed “Dolphin”
- Release: Japan: 2001-09-14, USA: 2002-03-03
- Marketing guys say: “128-bit console”
- Initial price: $199, now as cheap as €99

1 – Gamecube Hardware–

- Built around "Gekko"-CPU (PowerPC) at 486MHz
- External CPU bus: 64bit @ 162MHz, gives 1.3GB/s to the marketing guys
- 32kB instruction cache, 32kB 8-way data cache
- 256kB 2-way second level cache

1 – Gamecube Hardware–

- Custom GPU called “Flipper”, made by ArtX Inc. (now ATi)
- 2.1MB embedded framebuffer memory
- 1MB high-speed texture cache
- GPU supports the usual 3D features

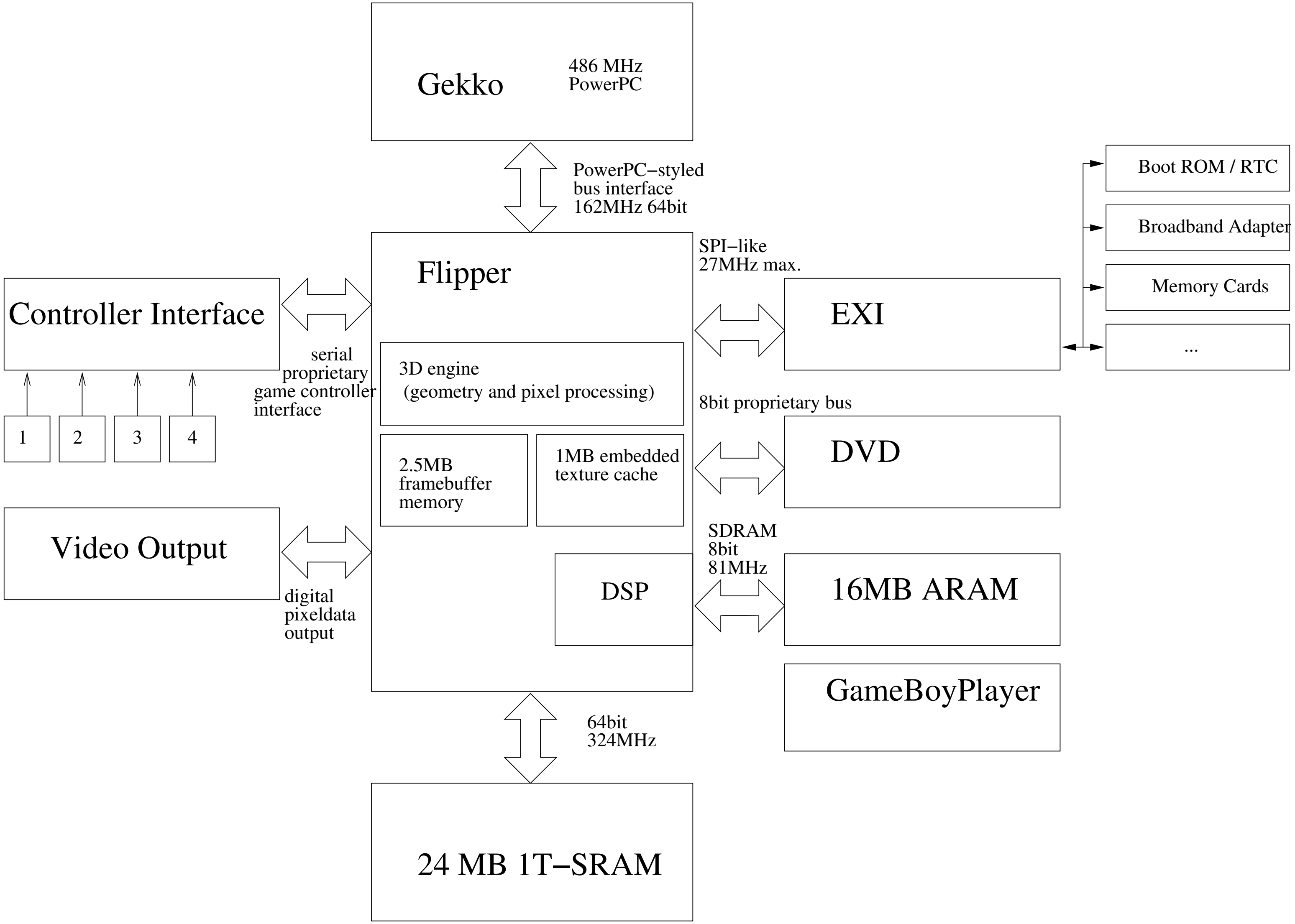
1 – Gamecube Hardware–

- Storage Medium: proprietary 7.5cm (mini-)DVD-based discs
- Of course copy protected ;)
- 1.2GB per disc

1 – Gamecube Hardware–

- External interfaces are proprietary:
- 4 “serial” controllers (N64-compatible)
- 2 memory card slots, 2 “serial” ports (SPI-like) (EXI BUS)

Hardware - More Details



“Gekko”

- Very close relative to the PowerPC 750CXe (“G3”)
- 486MHz clock rate
- PowerPC bus interface
- All memory access through Flipper (but fast!)
- Full features MMU (Linux!)
- No real debugging interface known :(
- Not cache coherent - take care of the cache, cache, cache!
- Special features: DMA-controller to locked cache, write gather pipe, “paired singles”

2 – Hardware - More Details– “Gekko”

writer-gather pipe

- Write any-size words to a fixed location
- CPU will “gather” the writes into whole cachlines
- maximum bus utilization for streaming (thus non-cachable) data
- used for 3D geometry data

Paired Singles

- SIMD extension
- not compatible to AltiVec (G4)
- 2x 32bit float operations per cycle
- speed increase over (very fast) FPU
- used for local lighting and other CPU geometry processing

2 – Hardware - More Details– “Gekko”

Debug

- Gekko has full-speed production verification debug ports
- Unfortunately, no information available :(
- Most probably not present in production boards (anymore?)
- JTAG seems to be present on early boards, but not on later ones...

“Flipper”

- Custom graphics processor
- Not related to ATI Tech., Inc. [a]
- Manufactured by NEC in a 0.18 microns process
- Very fast embedded RAM (texture cache: 10.4GB/s!) [b]
- State-of-the-art (well, in 2001) 3D features
- Realtime texture decompression (S3TC), 8 hardware lights, anisotropic filtering
- Very *predicatable* performance
- Very hardwired vertex processing
- More flexible (but still limitated) pixel pipeline (up to 16 stages, 8 textures)

[a] Although there is a sticker “Graphics by ATi” on every cube - ATI bought ArtX after they already completed the chip

[b] But be careful when comparing these peak numbers...

2 – Hardware - More Details– “Flipper”

- Interesting features like (relatively) easy access to Z-buffer, indirect textures (for depth-blur, glass-mapping, ...)

Performance

- Not designed for top-speed peak polygon or pixel rates but to deliver a decent sustained performance in real-world use
- Numbers given by Nintendo (6 to 12 million polygons per second) are quite conservative
- Games like *Star Wars: Rogue Squadron* actually do these 12 million polys/s (and even more...) in *average* (not peak!)
- Keep this in mind when comparing raw numbers to other consoles! Everybody fakes a lot!

External Interfaces

- Flipper's registers are memory mapped into the CPU's address space
- Peripherals like DVD-drive[a], the controller ports, the "serial" (EXI) ports are all connected to the flipper
- DMA support for most operations

[a] which has a seperate, intelligent Firmware

RAM

- RAM is often a bottleneck in Games, especially on random-access
- Gamecube has 24MB SRAM-styled RAM with 10ns random access(!) latency
- Not really SRAM, but 1T-SRAM (Real SRAM is too expensive)
- 2.6GB/s raw bandwidth
- Additional 16MB of 81MHz, 8bit SDRAM for “audio” or “auxilliary” use (ARAM)
- Not directly accessible by the CPU, but can be DMA’ed into RAM
- Some games (and Linux) use it, thanks to the MMU, memory-mapped (swapping)

Mass Storage - DVD

- Proprietary, DVD-like media
- Drive made by Matsushita
- Copy Protection using “recorded probabilty”[a]
- Drive’s firmware refuses to read discs without that protection
- Copy protection not yet cracked

[a] More details are documented in Nintendo’s patents, for example US006775227, available at http://www.uspto.gov

Homebrew

- Unbroken copy protection shouldn't prevent anyone from running own code
- Two software hacks appeared:
- First software hack came in the beginning of 2003 ("PSO-Hack")
- Datel's Action Replay (delivered on a "authentic" disc) can be abused, too ("Samson's Bootloader")
- Don't require any soldering, but require a boot each time you load your code
- Hardware hacks are possible, too ("IPL replacement")

“PSO-Hack”

- *Phantasy Star Online* is an internet online RPG
- Contains the possibility to download cheat checks which are executed locally
- Protocol was hacked for Dreamcast
- Hack “ported” to Gamecube
- *PSOload* / *PSUL* emulate the server (using DNS faking)
- Own code can be uploaded
- Required Broadband Adapter (BBA) and the game
- Relatively easy to get and use, but slows down development cycle

“Samson’s Bootloader”

- Datel’s *Action Replay* allows entering encrypted cheat codes for games
- Datel knows how to make “authentic” discs
- Cheats patch memory addresses
- Encryption was reversed
- Own code can be patched into memory
- Small loader code, which loads binary from memory card and/or BBA

“IPL replacement”

- Involves replacing the BIOS
- Hardware modification
- Will be described in more detail

The Boot Process

The Bootrom

- Gamecube doesn't have any parallel bootrom
- Instead, a serial ROM is contained in the RTC chip
- RTC is on the EXI bus
- BIOS is encrypted
- Flipper translates memory-accesses to EXI transfers and decrypts them on-the-fly
- CPU boots from 0xFFF00100 (usual for a PowerPC cpu with EP=1)

What could go wrong?

- NEVER REUSE KEYSTREAMS!
- ... but Nintendo did!
- XORing two different, encrypted ROM images gives XORed plaintexts
- If some image contains zeros, the result gives plaintext
- But it was even worse...

4 – The Boot Process– ROM Access Protocol

ROM Access Protocol

transmit	00000AAA	AAAAAAAA	AAAAAAAA	AAxxxxxx	xxxxxxxx	xx
receive	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	DDDDDDDD	DD

- On every cycle, one bit is transferred in each direction
- Unused bits (if only one direction is used) are ignored (“Dummy bits”)

The stupid encryption bug...

- Sniffing the EXI bus is no problem [a]
- Transfers look like the following: [b]

address sent to ROM chip | encrypted ROM data

interesting dummy data sent back

[a] It's a 27MHz SPI-like bus, i.e. fullduplex serial bus. A homebrew "logic analyzer" was built using a CPLD to parallelize the data and a Cypress FX2 USB2.0 controller to send the data to a PC.

[b] Actual numbers where modified to avoid any copyright issues

4 – The Boot Process– The stupid encryption bug...

```
> 00004000  00000000  00000000
< ffffffff  e8a6c3a4  e48a4ce3

> 00004200  00000000  38840c64
< ffffffff  f89cd6c2  e88c1a34

> 00004400  00000000  3c800123
< ffffffff  e47a9c43  b8a11c23

> 00004600  00000000  7c000456
< ffffffff  4f8ac856  11ae2fc6
```

4 – The Boot Process– The stupid encryption bug...

What did we see?

- The CPU fetches instructions from 0x100 upwards

  (The instruction cache will be enabled very early, so the bootup code can be fetched in a linear order)

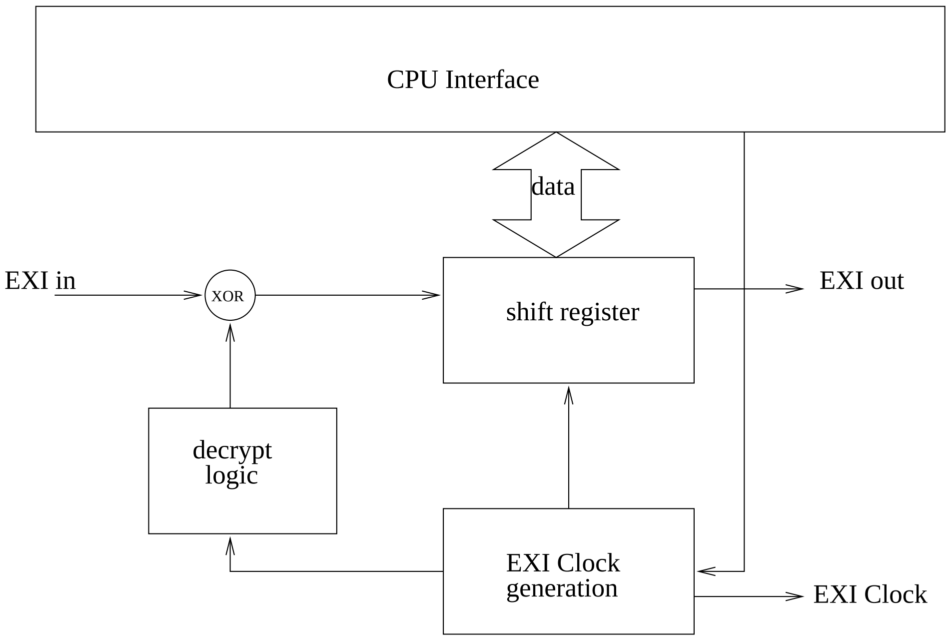
- The ROM transfers the (encrypted) data to the Flipper

- But the Flipper sends back decrypted data as dummy bits!

4 – The Boot Process– The stupid encryption bug...

What the hell...?

- Flipper's EXI interface is implemented with a shift register
- Data from EXI bus shifts in, data to EXI bus shifts out
- Decryption is added before the shift register
- Shift register isn't cleared after derypted data is in! (lol)



[a]

[a] This is only a model! There is no proof that the hardware works that way!

4 – The Boot Process– The stupid encryption bug...

- Clearly a bug in the design!
- Maybe they didn't notice it? (Unlikely... Hardware gets tested a LOT)
- Maybe added in last hardware revision, and they could afford a new mask revision?
- Other people suggested they were just stupid... But intelligent enough to build an otherwise full-featured chip? I don't believe that...

Working around the encryption...

How does this help?

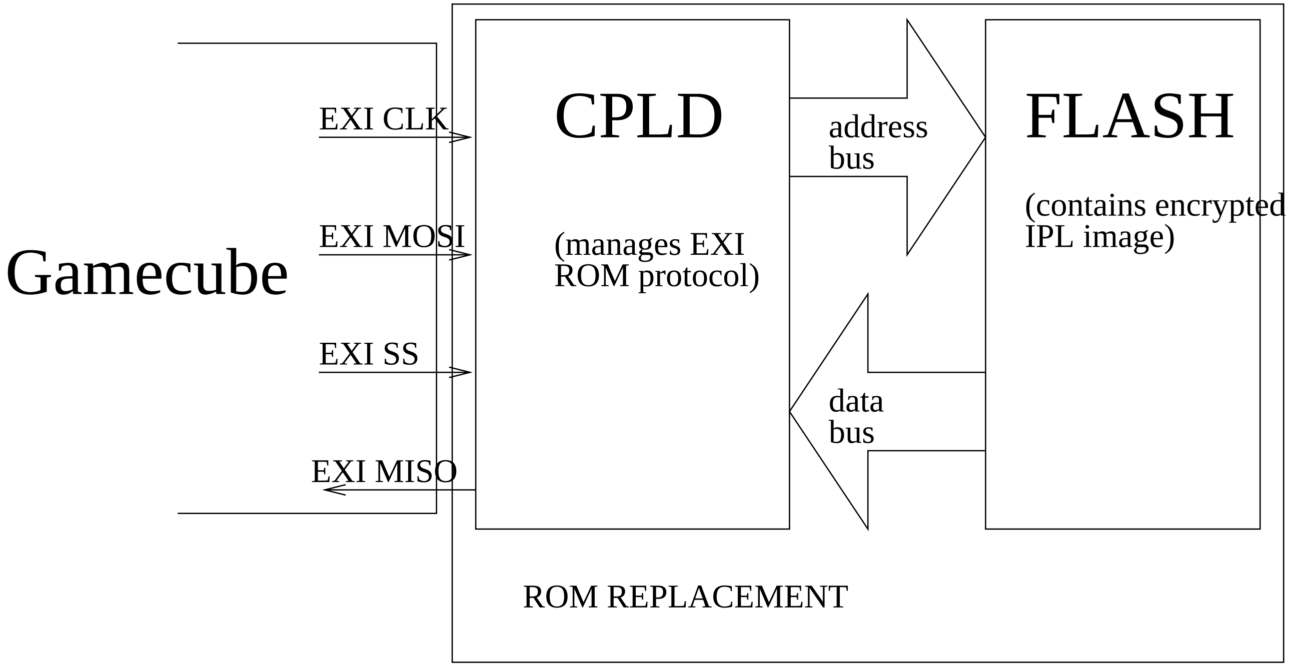
- The last 4 bytes are missing in the decrypted output
- Gives only 50% of the fetched data
- Fortunately, only the first 0x700 bytes are executed directly (called *BS*)
- The rest is transferred using large DMA blocks (1024 bytes) (called *BS2*)
- 1020 bytes of them come back decrypted!
- Now custom code can be encrypted (simple XOR) and injected (using modified hardware which emulates ROM protocol and replaces/overrides original ROM)
- This code can dump the memory
- Dumped memory can be XORed with the encrypted data to yield keystream

The first 0x700 bytes

- 50% of the first 0x700 are still missing as they are transferred in 8 byte blocks
- JTAG isn't available (at least not for me)
- Decryption is one-way - no way to go backward or re-decrypt data other than resetting the Flipper
- Every second instruction is known in plain
- Second instruction can be patched to "jump"
- Jump where? Into memory.
- Dumpcode must be placed there first, using *BS2*-injection
- Then modify ROM to have jump in the first fetched word.
- Dumpcode fetches the rest, recovering nearly complete Keystream
- First instruction still missing, but can be guessed.

The ROM emulation hardware

Now the full ROM can be replaced with a custom bootloader. A hardware was built, using a CPLD and Flash memory, which emulates the original ROM.



The IPL replacement

- Presented here at the 21c3
- Completely open (software, schematics, VHDL, tools, ...)
- Can’t boot pirate games (because the DVD-firmware won’t be modified this way)
- Can boot homebrew codes in seconds!
- Option to be invisible after boot
- Additional features like an UART-port (maybe...)

Homebrew examples

Extending the GC hardware

Rob Reilink

Extending the GC hardware

Why? GC can be an embedded computer

- Home automation
- Cinema set
- Car Infotainment system
- ...

But: Essential hardware lacks:

- Data storage (harddisk and/or flash)
- Keyboard/mouse
- Generic I/O
- RAM is only 40MB (Main RAM+ARAM together)

Required extra hardware for embedded systems

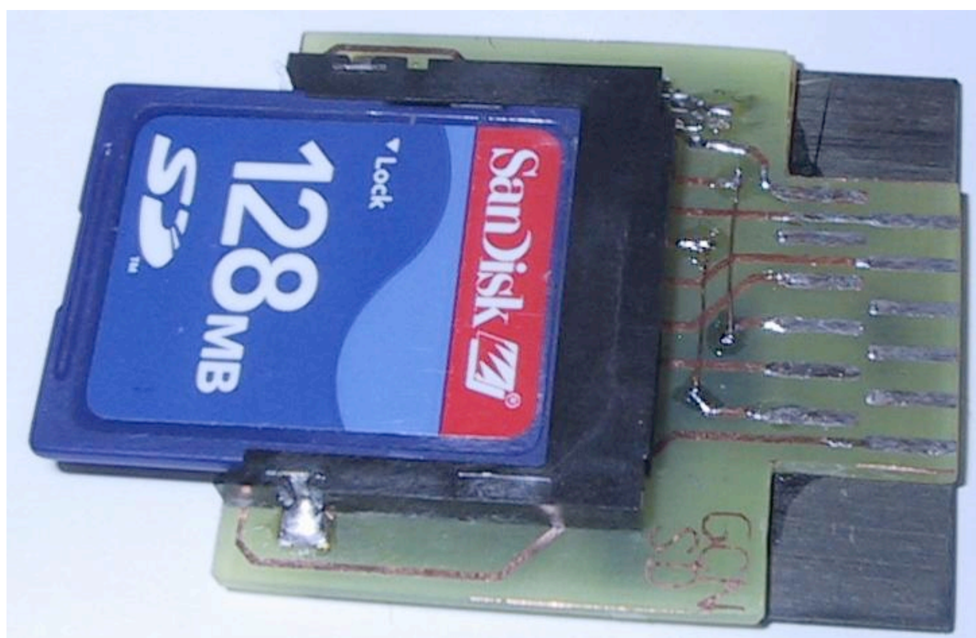
- Harddisk interface
- Flash storage interface (Compactflash, SD, MMC, ...)
- RAM extension
- USB interface (for all other devices)
- Keyboard/mouse interface
- Generic I/O interface

Memory (EXI) interfaces (2)

- Standard SPI interface (Clk, /CS, DataIn, DataOut)
- 3.3V and 5V power available
- Connector is etched on PCB, so no special connector required

Memory (EXI) interfaces (2)

- SPI: can easyly be interfaced to standard hardware: shift register IC's for digital I/O or SD and MMC cards for data storage
- Can be interfaced to USB 1.1 host controller with litte logic (USB 2.0 not possible: only PCI controllers available)



Serial interfaces (for the controllers) (4)

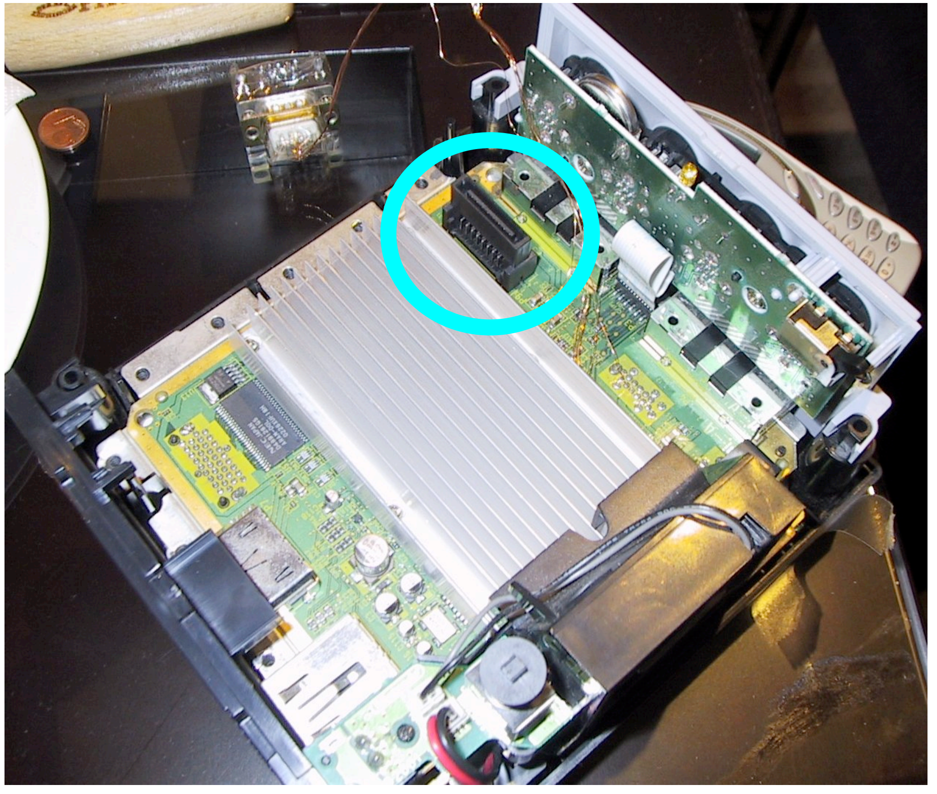
- 1 wire interface, open drain bus with custom protocol
- Speed is low enough to be bit-banged by standard microcontroller which can then be interfaced to PS/2 (mouse/kbd) or custom hardware (UART, digital I/O)

High-speed interface (1)

- SDRAM interface 8 bit to extend the ARAM; can be interfaced to standard SDRAM chip(s), maximum RAM size unknown;

DVD interface (1)

- 8 bit, bidirectional, DMA-able, interrupt
- Can be used to interface IDE HDD with CPLD
- DVD cannot be used anymore, so only for GC's with modchip
- Connector seems to be custom made or is hard to find



Summary hardware extensions

- Harddisk interface → DVD interface
- Flash storage interface → SD, EXI interface
- RAM extension → High speed interface
- USB interface → EXI interface
- Keyboard/mouse interface → SI (controller) interface
- Generic I/O interface → SI (controller) interface

For every required hardware extension a suitable interface is available!

GameCube Linux

Michael Steil

Who?

- The GameCube Linux Project
- http://www.gc-linux.org/
- started by the Xbox Linux people
- most information had been reverse engineered before

Why?

- The GC is silent, small and cheap
- Linux = maximum flexibility
  - **“PC”/thin client**: KDE, VNC on TV
  - **media player**: play DivX from network
  - **server**: PowerPC!
  - **development**: e.g. emulators on Linux

Drivers

- Interrupt controller
- Framebuffer
  - YUV colour space problem
  - 3D acceleration in the works
- Gamepad
- Keyboard: original keyboard, 2 different adapters

Drivers

- Broadband Adapter
- ARAM block device (can be used as swap!)
- Audio (ARAM!)
- Memory card
- SD card
- RAM/RTC

What is working?

- as a computer:
  - X-Window!
  - KDE? memory problems
  - Remote Desktop (X, VNC, RDC) works great
- as a media player:
  - enough power to play fullscreen DivX

What is working?

- as a server:
  - any full Linux distribution can be used without much hassle
  - all software works, if memory is enough
  - PostgreSQL regression test works flawlessly

What is working?

- games and emulators:
  - powerful enough to run many games and emulators
  - software should be based on SDL
  - initrd with bundled emulator possible

Current drawbacks

- rootfs and media are typically taken from network (NFS), network is slow (10 MBit)
- few ready-to-use soltions
- booting often cumbersome