# Yet Another Gamecube Documentation

(but one that's worth printing)

December 23, 2006
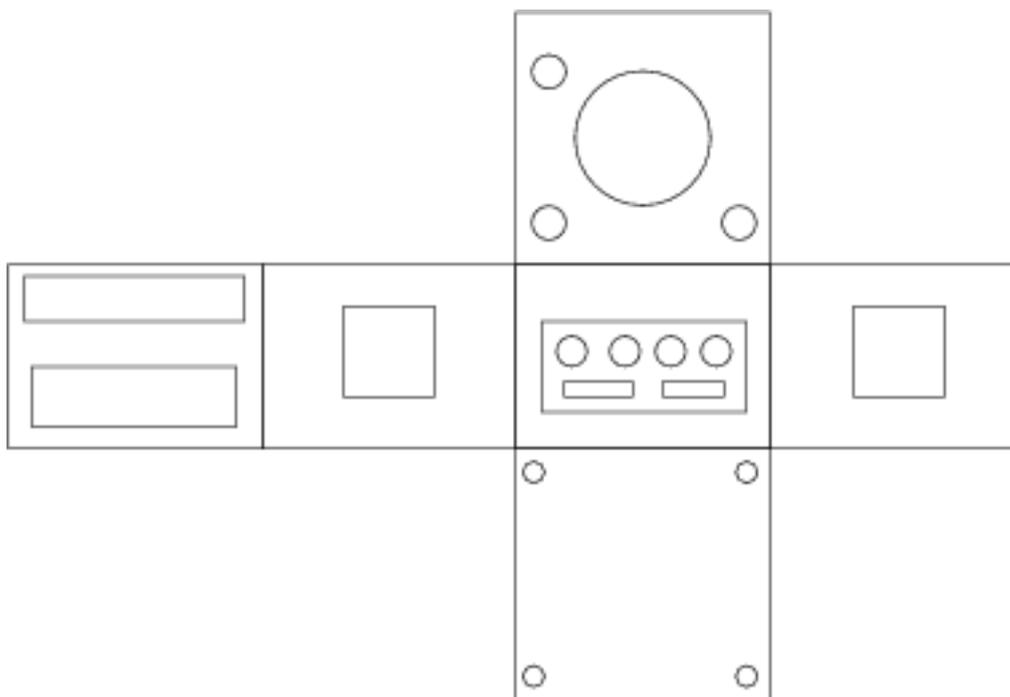
this is the result of myself pasting together various freely available documents aswell as adding some of my own findings. have fun... additions and corrections welcome :)

THIS IS WORK IN PROGRESS! INFORMATION CONTAINED IN THIS DOCUMENT MAY BE MISSING, INCOMPLETE OR EVEN PLAIN WRONG! NO F**N' WARRANTY IMPLIED! IF THE USE OF THE INFORMATION CONTAINED HERE RESULTS IN ULTRA REALISTIC SMOKE EFFECTS, BRAIN DAMAGE OR LOSS OF PHYSICAL AND/OR MENTAL HEALTH PLEASE DON'T COME BACK AND SAY YOU HAVEN'T BEEN WARNED! YOU SHOULDN'T BE USING THIS IN THE FIRST PLACE!**

groepaz/hitmen (groepaz@gmx.net)

Hitmen-Console http://www.hitmen-console.org

Contents

1 Introductional Rant

If you don't know what programming a machine down to the metal is all about, go away! no really, this document is not for you! if you are seeking for advice on using existing solutions, such as SDKs or libraries, you will find little to none information that is of any use for you and you might only become frustrated by figuring out how little you know. If you however aren't afraid of numbers and want to dare jumping into the snake-pit of semi-accurate information based on guesswork done by a bunch of freaks - feel invited. this was made to give you what you need in the most compressed and visually pleasing form possible. *Stuff that matters.*

1.1 Things that are in this document

just about everything explicitly and specifically related to the gamecube hard- and software internals and its programming. everything inside the box is subject to be documented, may it be relevant for actual programming or not. its meant as a reference for everyone who wants to know in all possible detail what makes this thing tick.

one more thing: please notice that this is a technical documentation which is presented for pure educational purposes and higher learning, and not a moral lesson. i have decided against leaving out any information since i believe that information by itself should not be crippled in any way. if you choose to abuse this information for any kind of illegal activities (**PLEASE DON'T!**) so be it, but don't bother me with it.

1.2 Things that are *not* in this document

several things were decided to not being put into this document because they didn't fit into the 'technical documentation' type of concept. They may be documented separatly some time but not now and not here. These things are:

▷ Tips on Emulating the Gamecube on another Host system (this kind of information is only useful for a very limited number of people, and additionally might be highly confusing and/or misleading for those who are writing actual gamecube programs)

▷ Explanation of the PSO (Phantasy Star Online) Exploit that lets you run code on the Gamecube

▷ Explanation of the Codes used with Datels Action Replay

▷ Instructions on using any tools that let you upload and execute code on the Gamecube, or any other development related tools **except** anything related to setting up and using gcc as a cross-compiler targeted to the gamecube.

▷ anything related to gaming, cheat-codes and the like. (this is a tech-doc not a gaming FAQ!)

▷ information on using the datel action replay to patch itself in order to execute code.

▷ detailed and/or complete sourcecode, except when a formal explanation would just over-complicate things. (this is a documentation, not a code library)

▷ building and/or using a custom interface to connect a gamecube memory card to another host and read/write data

▷ anything related to playing/booting/copying pirated games (as you may have noticed, **we do not support piracy!**)

some of these may be arguable, so if you think they should be here - probably along the lines of the appendix - don't hesitate to write the chapter in question and send it to me. i might include it if you write it, but other than that i won't care (there is still enough other stuff to complete).

1.3   Conventions

▷ we count bits starting from 0, the most significant bit of a byte is bit 7. when visualising a byte the most significant bit comes first (left), and the least significant bit comes last (right).

▷ when dealing with 16- or 32 byte values all figures are in big endian byte order. this means that the most significant byte comes first (left), and the least significant byte comes last (right).

Please notice that the above is different to what IBM is using in their PPC documents. They have the (to many people strange and wrong) idea of applying 'big endian' to the order of bits and showing them the other way around as we do in this document.

▷ if known (from patents or other freely available sources) we use the same terminology as Nintendo (or Macronix respectively) does, in particular we try to use the same names and abbreviations for hardware registers, signals and the like as a weak attempt of providing consistency with other existing documentation.

▷ absolute memory addresses are shown as if the gamecube had been initialized by the original IPL and address translation had not been changed. For this matter we dont use physical adresses to avoid confusion for the majority of our readers.

▷ code snippets are in either real or pseudo C language. any logical or arithmetic expressions outside code snippets are loosely similar to C notation according to the following table:

Description	Symbol
logical or bitwise AND	&
logical or bitwise OR	|
logical or bitwise exclusive OR	^
logical or bitwise NOT (inverse)	!
equality or assignment	=
addition	+
substraction	−
multiplication	*
division	/

please notice that -outside code- we do not make a difference between logical and bitwise operations. if in doubt the operation is bitwise, it should however be clearly visible from the context.

1.4   legal Babble

2 Gamecube Hardware Introduction

The GameCube is a powerful piece of hardware. The whole system is based on the IBM PowerPC Gekko processor and the custom ATI Flipper video system. The PowerPC Gekko processor is really just a PowerPC 750 with a few enhancements.

2.1 enhanced PowerPC 750 Specification

- ▷ 486 MHz internal processor clock
- ▷ 200 MHz 64-bit bus width to main memory (1.6 Gigabytes per second maximum)
- ▷ 32KB associative L1 Icache
- ▷ 32KB associative L1 Dcache with 16KB data scratchpad
- ▷ Super-scalar microprocessor with five different execution units:
    - ▷ 2 integer units, 1 FPU, and 1 load\store unit and branch unit
- ▷ DMA unit servicing 16KB data scratchpad.
- ▷ DMA request queue - 15-entry.
- ▷ Write-gather buffer for writing graphics command lists to the video system.
- ▷ Embedded 256KB 2-way set-associative L2 unified cache.
- ▷ 2 32-bit Integer Units (IU)
- ▷ 1 FPU, 32 and 64-bit bus width
- ▷ The FPU supports Floating Point Paired Singles (FP/PS)
- ▷ Branch Unit provides static AND dynamic branch prediction
- ▷ Features Out-Of-Order execution which means that when an instruction delays because of data access, subsequent opcodes can continue to be decoded and executed.

The enhanced PowerPC Gekko processor also contains many features for minimization of processor delays because of data accessing and for maximization of processing throughput:

- ▷ Non-blocking caches
- ▷ Branch prediction through use of the Branch Unit (BU)
- ▷ 8-way set-associative caches
- ▷ 256KB L2 transfer cache
- ▷ Out-of-order execution capabilities

The instruction set of the PowerPC Gekko processor seems to be almost identical to the one of the PowerPC 750 processor.The only visible differences at the moment are that the PowerPC Gekko processor has a few AltiVec\SIMD opcodes added to its final instruction set.

2.2 Consumer Units

2.2.1 Nintendo

2.2.1.1 HW1 HW1 was an initial, buggy version of the GameCube hardware that wasnt sold at retail.

2.2.1.2 HW2 HW2 is the first hardware that was sold in stores to the public.

2.2.1.3 HW2 'second edition' The second edition models are missing the "Serial Port 2" that the first edition had. The plastic cover is still on the bottom of the cube, where the port used to be, but there's just a metal plate underneath it, and no connector.

2.2.1.4 HW2 'third edition' The third edition Gamecubes are missing both the "Serial Port 2", and the Digital A/V connector.

2.2.2 Panasonic Q

There is a Gamecube combined with dvd-player manufactured by Panasonic called 'Panasonic-Q'. It seems to be exactly the same as HW2 for the Gamecube part, except that the dvd drive is different.

2.3 Development Units

Nintendo provides development hardware units to official, licensed GameCube developers. There are namely two different versions: the GDEV and the DDH hardware development kit units. These units are the same as retail GameCube HW2 units with some changes: They have PC communications features (either through SCSI or USB) and they have DVD emulation hardware instead of a proprietary mini-DVD drive. GameCube development units also seem to have slower processor speeds than retail GameCubes, this clock speed ranges from around 150MHz to 400MHz. Development GameCubes also seem to contain more RAM than retail ones, namely around 40MB. SNSystems also provides their own development kit,authorized by Nintendo, called the TDEV. According to specifications directly from SNSystems, the TDEV development hardware contains twice as much memory as retail GameCubes for debugging and a direct PC<->TDEV USB connection for fast uploading of code and\or data. Finally, there is also another proprietary development kit called the NR-Reader. NR-Reader's contain less debugging capabilities than the other development kits and are mostly meant for developers to efficiently get their demos\games to beta testers or media. However, SNSystems reports that their ProDG development kit can be used with a special USB adapter of theirs for directly sending program (debug) code to NR-Reader GameCubes. Also, NR-Reader GameCubes contain different mini-DVD drives than retail GameCubes, but still use a proprietary writing\reading format which is currently unknown. The DVD drives of NR-Readers can only read special DVDs that can only be written correctly with NR-Writer hardware (which is really just a Panasonic\Matshita SW-9501 with modified firmware). Also, the official debug development kits possibly contain J-TAG support, which is a method for debugging hardware. If so, there is a possibility that J-TAG support still remains in retail GameCubes as well, but this is purely hypothesis. If, in fact, retail Game-Cubes contain J-TAG debugging support then it should be possible for (homebrew) code to be uploaded through a J-TAG cable) directly to a GameCube's RAM and executed.

2.4 Hardware Parts List

▷ "MBU" and "MBB" will be used to refer to parts on the top and bottom sides of the GameCube's mother-board respectively.

▷ "DVDB" will be used to refer to parts on the DVD controller board.

▷ "CB" will be used to refer to parts on the controller pad board.

2.4.1 Connectors

The are 10 different connectors on the GameCube's mother-board. The following table contains an ID key and a short functional description.

ID	Description
P1	Motherboard Power Connector - MBB - Top Left
P2	Digital Video Output Connector - MBU - Bottom Left
P3	Controller Pad Board Connector - MBU - Middle Right
P4	Memory Card Slot Connector A - MBU - Top Right
P5	Memory Card Slot Connector B - MBU - Bottom Right
P6	Serial Port Connector 1 - MBB - Top Right
P7	Analog Video Output Connector - MBU - Middle Left
P8	Serial Port Connector 2 - MBB - Top Right
P9	Mini-DVD Drive Port Connector - MBU - Top Right
P10	Hi-Speed Parallel Port Connector - MBB - Bottom Left

2.4.1.1 Memory Card Slots (P4,P5)

pin	Signal
1	EXTIN
2	GND
3	INT
4	3.3V
5	DO
6	5V
7	DI
8	3.3V
9	CS
10	Ground (Shield)
11	CLK
12	EXTOUT

2.4.1.2 High-speed Port (P8)

pin	Signal
1	3.3V
2	GND
3	INT
4	CLK
5	DO
6	DI
7	CS
8	Ground (Shield)

2.4.1.3   SDRAM/Parallel Port (P10)

pin	Signal
1	VCC
2	Ground
3	DQ0
4	DQ7
5	DQ1
6	DQ6
7	DQ2
8	DQ5
9	DQ3
10	DQ4
11	VCC
12	Ground
13	write enable
14	DQM
15	CAS
16	Clock
17	RAS
18	A12
19	CS (Chip Select)
20	A11
21	BA0
22	A9
23	BA1
24	A8
25	A10
26	A7
27	A0
28	A6
29	A1
30	A5
31	A2
32	A4
33	A3
34	INT
35	VCC
36	Ground

2.4.1.4   BBA/Modem Connector (P6)

pin	Signal
1	EXTIN
2	Ground (Shield)
3	INT
4	CLK
5	12V
6	DO
7	3.3V
8	3.3V
9	DI
10	CS
11	Ground
12	Ground

2.4.1.5   DVD Interface Connector (P9)

pin	Signal
1	AISLR (audio bus)
2	5V
3	AISD (audio bus)
4	5V
5	AISCLK (audio bus)
6	5V
7	DIHSTRB
8	5V
9	DIERRB
10	Ground
11	DIBRK
12	DICOVER
13	DIDSTBR
14	DIRSTB
15	DIDIR
16	Ground
17	DID7
18	Ground
19	DID6
20	Ground
21	DID5
22	Ground
23	DID4
24	Ground
25	DID3
26	Ground
27	DID2
28	MONI
29	DID1
30	MONOUT
31	DID0
32	Ground

pin	Signal
1	Ground
2	Ground
3	3.3V
4	3.3V
5	Ground
6	Ground
7	Ground
8	Ground
9	1.8V
10	1.8V
11	1.8V
12	1.8V
13	1.55V
14	1.55V
15	1.55V
16	Ground
17	Ground
18	Ground
19	Thermo detect
20	12V
21	5V
22	5V

2.4.1.6   Power Supply Connector (P1)

2.4.2   Semi-Conductors

ID	Description
U1	Customized NEC Flipper Chip - MBU - Middle
U2	Customized IBM PowerPC Gekko Chip - MBU - Bottom
U3	MoSys (MS3M23B-5 A) 12MB 1-T SRAM - MBU - Top Right
U4	MoSys (MS3M23B-5 A) 12MB 1-T SRAM - MBU - Top Right
U5	NEC (D4891281G5 0125XU621) 16MB ARAM - MBU - Top Left
U6	A\V Encoder (AVE N -DOL RS5C5828) - MBB - Middle Left
U7	Amplifier? (AMP - DOL 128 124) - MBB - Top Left
U8	MX Clock Generator (Part Number?) - MBU - Bottom Left
U9	MX Clock Generator (Part Number?) - MBU - Bottom Right
U10	MX RTC/IPL (8013108-M RTCN-DOL 1R6022A1)

2.4.2.1 IPL (U10)

Pin	Signal
1	Clock
2	
3	
4	
5	CS
6	serial in
7	Ground
8	
9	serial out
10	
11	
12	osc - xtal2
13	osc - xtal1
14	

2.5 Details on the motherboard buses

The GameCube has three main external buses on its mother-board: the North-Bridge, the South-Bridge, and the East-Bridge. The fastest bus is the South-Bridge which connects the two 12MB 1T-SRAM chips to the Flipper. The South-Bridge bus has a bus-width of 64 bits, and data is exchanged through it at rates of about 324MHz. The North-Bridge bus connects the IBM PowerPC Gekko processor to the Flipper and is another 64 bit bus-width bus, however, it is only half as fast as the South-Bridge bus and is clocked at around 162MHz. Finally, the East-Bridge bus connects the 16MB Audio RAM chip to the Flipper chip. This bus only has a bus-width of 8 bits and is by far the slowest one, clocked at only 81MHz.

2.6 Details on the Macronix (MX) Chips

An outer inspection of the two MX chips does not reveal anything of much interest. Both chips are TSOP packages containing 14 pins each. One of the chips has "CLK" inscripted on it, and the other "RTC". It can be easily inferred that the "CLK" chip functions as some sort of a clock controller\generator and the "RTC" chip contains the GameCube's Real-Time Clock unit. Two of the RTC chip's pins are connected to an external crystal which regulates the RTC's timing rate. Another pin is connected to a battery located on the controller board. At least two pins are used for both VCC and GND. That leaves nine unknown pins. The RTC MX chip also contains the GameCube's BIOS. While 14 pins is not nearly enough for parallel Flash ROM, EEPROM, mask ROM, etc., it is quite adequate for a serial connection.

2.7 DVD Protection

The DVD Protection is based on a custom data format on an otherwise pretty standard dvd.

2.7.1 Filesystem

The custom Filesystem (which is described somewhere else in this Document) by itself is not related to the actual protection mechanism. However, since it is not standard, that alone would already make it hard to read (and create) in a regular (pc-) environment.

2.7.2 Barcode

The Barcode is used to authenticate the Disc in the Drive.

2.7.3   Encryption

The entire content of a Gamecube DVD is XORed with a constant cyphertext and it is transparently decrypted by the Disc-Controller when reading from the DVD.

2.7.3.1   Cyphertext algorithm   todo

2.8   IPL/BIOS Encryption

if you XOR an NTSC with a PAL bios (or any other two different ones), you will notice that because they have different sizes, there are some obviously zero encoded areas in one files, giving you plaintext in the other one which proves:

  ▷ encryption is a simple XOR with a constant ciphertext.

  ▷ the key used to generate the cyphertext is the same for different bios's

so we do the math:

given Ci = ciphertext, Cl = cleartext, K = key

encoding data goes like:

Ci1 = Cl1 ^ K
Ci2 = Cl2 ^ K

If Cl1 or Cl2 is nothing but zero, the resultant Ci is just K

decoding it would be:

Cl = Ci ^ K

for the areas where Cl is nothing but zeroes in one bios, we know K

2.8.1   Flipper decryption logic bug

The hardware decryption logic has a really nasty bug which allows us to read almost the full Cleartext (and thus a large part of the cyphertext, by XORing it with encrypted data).

This, combined with the features of the XOR encryption makea the whole encryption useless (at least very insecure) and implementing a new bios is a straight-forward task (provided that "high speed" (30Mhz) programmable logic with enough memory attached to it is available.). The Bios chip, which also includes sram and rtc (but that won't matter here), is attached to the EXI0 bus. The Exi bus (nothing new here, just to refresh it is an SPI-like bus. SPI is nothing complicated, just four interesting lines: CS (used mainly for syncing, since you need a defined start point, and you can easily attach multiple devices (memory card, ...) to the same bus with seperate CS lines), SI (aka MOSI, master out, slave in - the CPU is always master, the IPL-chip is slave. so SI is gamecube -> device), SO (device -> gamecube, tristated when a device is not active), and CLK (generated by the master). a transfer is basically:

- lower CS (it's low active)

for every bit do:
- set SI bit
- clock
- read SO bit
then:
- put CS high again.

(the exact timing (WHEN to sample SO, clock polarity) is different for different SPI modes, and the one
descriped here is not necessarily the one used in the GC. anyway, it doesn't matter here)

so, based on that, we can transfer n-bit messages in BOTH DIRECTIONS. technically this is implemented
with a 32bit shift register, with every clock cycle one bit is shifted out (to SI), and one bit is shifted in (to
SO). so after n clock cycles, you have n new bits in the shift register and shifted n bits out. the used protocol
on the Bus is in most cases very simple but device dependant. In the case of the IPL chip, it's the following:

GC -> IPL
1 bit read/write (0 for read, 1 for write, the latter only valid for RTC/Sram of course)
1 unknown bit
1 bits selection (0 for ROM, 1 for RTC/Sram)
23 bits address
6 bits dummy
after that, the data transfer starts. the 6 dummy cycles are mainly to give the IPL time to read out the first
byte.

So you send 32 bits of data (the "address"), and start receiving the ROM bytes. but hey - we said the
SPI bus always transfers 2 bits per clock cycle (in marketing terms), since it's fullduplex (in technical
terms). we transfer one bit TO the device, and one BACK. we HAVE to. there's no way to NOT send a
bit - but it doesn't matter, since for example the bits send from the IPL to the GC in the first 32bits are
just ignored - they would contain most probably only zeros, ones, or the bus might be tristate. it's simply
not defined, so there's no data to be expected. the same goes for the transfer of the data. the IPL chip sets
the correct data at the SO line, but the gamecube - well, sends dummy bits, too. normally you would send
zeros, ones, or whatever. it's ignored by the IPL chip anyway (unless it's a write, that would turn the whole
thing upside down) now since technically the SPI port is implemented by a shifting register of 32bit length.
after transferring 32bits, we would have to read out the new value, store it into memory, and "start the next
transfer". but what's about CLEARING the register before? yes, they didn't. in the next transfer, the last
32bit are shifted out as dummy bits. well, one might say, it's just the data just shifted in, so it's completely
uninteresting. BUT: the decryption of the loader is done in hardware. it's a part between the SO line of
the IPL and the DI port of the shift register. (the encryption is build into the flipper, so no way to intercept
the content AFTER decryption).so because the (decrypted) data just shifted in (and stored into memory) is
shifted out again - we can get the decrypted data. if you sniff the SI line to the IPL chip, you will get a log
like this:

```
00 00 40 00 (address written to the IPL, in this case:  0x100)
FF FF FF FF (well just dummy data)
xx xx xx xx (the data from the last 4 bytes, decrypted)
...
...
xx xx xx xx (the data from the n-1 transfer, decrypted)
```

so in the end you get every 32bit words except one. For every transfered block you miss 32bits of plaintext
data, but you'll get the rest. This should be enough to decrypt huge parts of the bios, and thus recover a
large part of K.

2.8.2   Cyphertext algorithm

todo

2.8.3   replacing the IPL

using the above gained knowledge it is possible to create a small bootrom replacement (using the, yet incomplete, cyphertext), and get more (most) of the IPL Cleartext.

The Gekko boots from 0x100, that's what you read in almost any ppc instruction manual - the reset vector. well, this isn't the complete truth - it boots from it's exception base + 0x100. And the exception base is normally zero, BUT, as the ppc manual states: there's a bit in a HID (i think) register, which turns the exception base to 0xFFF00000. and this bit is "set usually at boot time". So the processor starts to fetch instructions at 0xFFF000100. If you read a bit further, you'll notice that the CPU always reads 64bits at once for code. The memory at 0xFFF00000 is mapped inside the flipper to an automated exi transfer (with that shift register), with the decryption logic active. so the processor starts executing the decrypted instructions, reading 8 bytes at a time, of which we get 4 bytes in plain - not much, (although enough to make some funny experiments, but that's another topic). Luckily, the IPL itself (the cube menu) isn't executed this way since that wouldn't be possible thanks to the "dumb" decryption logic The first ~0x800 bytes start to read data out of the IPL chip and store it to memory (still using the hardware decryption logic), and jump there. they read 1024 bytes at once. Well - now we know 1020 bytes of each transfer, enough to have a complete block of code we can exchange (we have the ciphertext Cl^K = Ci on SO, and the plaintext (delayed by 32bits on SI), and can XOR them to get Cl^K^Ci = K. now we can encrypt our code with K). so now we can make a small code which just dumps the whole IPL - well, to the EXI bus or whereever you can receive it. Now we have all Cl, and thus we can compute all K, thus we can get the complete Plaintext of all available IPLs aswell as encode a larger custom IPL ourselves.

a small note on why you can not recover the plaintext of the original loader this way:

The decryption logic is, whatever it is, a PRNG. It generates a stream of ciphertext ("K"), which has random properties (non-repeating, at least not in the range of some MB), but is always the same. it is incremented with every EXI-transfer. the address is NOT used in the calculation. thus reading from 0xFFF00100 more than one time will give you each time another result. the first time you get Ci(0)^K(0) (the correct result), the second time you get Ci(0)^K(1) etc., i.e. wrong results. Since we never get the K(n) for odd n, i see no chance of recovering it this way, even if we can read at 0xFFF00000+x (and we can do this if we don't set a specific bit to disable the logic).

3 Gekko CPU Overview

3.1 Registers

▷ General purpose registers (r0-r31)

  ▷ r1 sp stackpointer
  ▷ r2 rtoc global pointer to _SDA2_BASE_
  ▷ r13 global pointer to _SDA_BASE_

▷ Floating point registers (fp0-fp31)

▷ Segment registers (sr0-sr15) - unused in regular (SDK/DolphinOS) applications

▷ Special purpose registers (spr0-spr1023)

  ▷ spr8 (lr) - link register
  ▷ spr920 - HID2
  ▷ spr1010 - instruction breakpoint address
  ▷ spr1013 - data breakpoint address

spr920	4	r/w	HID2

31 24	23 16	15 8	7 0

bit(s)		description
2		PSE - Paired-Single load and store instructions enabled
1		
0		LSQE - Paired-Single mode enabled

3.2 Calling conventions

parameters are passed in r3 (1st) r4 (2nd) and r5 (third) up to r12 (9th), further parameters are passed through the stack.

3.3 PPC Instructions

3.3.1 Integer Instructions

Mnemonic	Opcode	Description
addi		
addis		
add		
addo		
subf		
subfo		
addic		
subfic		
addc		
addco		
subfc		
subfco		
adde		
addeo		
subfe		
subfeo		
addme		
addmeo		
subfme		
subfmeo		
addze		
addzeo		
subfze		
subfzeo		
neg		
nego		
mulli		
mullw		
mullwo		
mulhw		
mulhwu		
divw		
divwo		
divwu		
divwuo		
cmpi		
cmp		
cmpli		
cmpl		

Mnemonic	Opcode	Description
andi		
andis		
ori		
oris		
xori		
xoris		
and		
or		
xor		
nand		
nor		
eqv		
andc		
orc		

Mnemonic	Opcode	Description
extsb		
extsh		
cntlzw		
rlwinm		
rlwnm		
rlwimi		
slw		
srw		
srawi		
sraw		

3.3.2   Floating-Point Instructions

Mnemonic	Opcode	Description
fadd		
fadds (*)		
fsub		
fsubs (*)		
fmul		
fmuls (*)		
fdiv		
fdivs		
fres (*)		
frsqrte		
fsel (*)		
fmadd		
fmadds (*)		
fmsub		
fmsubs (*)		
fnmadd		
fnmadds (*)		
fnmsub		
fnmsubs (*)		
frsp (*)		
fctiw		
fctiwz		
fcmpu		
fcmpo		
mffs		
mcrfs		
mtfsfi		
mtfsf		
mtfsb0		
mtfsb1		
fmr (*)		
fneg		
fabs		
fnabs		

(*) - modified for paired singles

3.3.3   Integer Load and Store Instructions

Mnemonic	Opcode	Description
lbz		
lbzx		
lbzu		
lbzux		
lhz		
lhzx		
lhzu		
lhzux		
lha		
lhax		
lhau		
lhaux		
lwz		
lwzx		
lwzu		
lwzux		
stb		
stbx		
stbu		
stbux		
sth		
sthx		
sthu		
sthux		
stw		
stwx		
stwu		
stwux		
lhbrx		
lwbrx		
sthbrx		
stwbrx		
lmw		
stmw		
lswi		
lswx		
stswi		
stswx		

3.3.4   Floating-Point Load and Store Instructions

Mnemonic	Opcode	Description
lfs		
lfsx		
lfsu		
lfsux		
lfd		
lfdx		
lfdu		
lfdux		
stfs		
stfsx		
stfsu		
stfsux		
stfd		
stfdx		
stfdu		
stfdux		
stfiwx		

3.3.5   Branch Instructions

Mnemonic	Opcode	Description
b		unconditional Jump
ba		
bl		branch and link
bla		
bc		
bca		
bcl		
bcla		
bclr		
bclrl		
bcctr		
bcctrl		

3.3.6   Condition Register Logical Instructions

Mnemonic	Opcode	Description
crand		
cror		
crxor		
crnand		
crnor		
creqv		
crandc		
crorc		
mcrf		

3.3.7 Misc Instructions

Mnemonic	Opcode	Description
twi		
tw		
sc		
rfi		
mtcrf		
mcrxr		
mfcr		
mtmsr		
mfmsr		
mtspr		
mfspr		
lwarx		
stwcx.		
sync		
mftb		
eieio		
isync		
dcbt		
dcbtst		
dcbz		
dcbz_l		
dcbst		
dcbf		
dcbi		
icbi		
eciwx		
ecowx		
mtsr		
mtsrin		
mfsr		
mfsrin		
tlbie		
tlbsync		

3.4 additional Gekko Instructions

The Gekko has some additional (and some modified respectivly) instructions in its Paired-single mode which are useful for fast vector and matrix calculations and which are analog to Intel (and other x86 series) processors "streamed instructions", known as SSE. This extension is unique for the Gekko processor and used to calculate two single-precision numbers ("floats" in C) in one clock cycle. The floating-Point Registers of the Gekko (FPRs) are modified in the following way : one half is used for the first single number, and other for the second. These parts are named as "PS0" and "PS1". PS instructionset is divided into two parts : Load and Store Quantization and Paired-Single Arithmetic instructions. Load and Store Quantization instructions are used for fast integer-float type casting and some specific memory operations, using PS0 and PS1 parts of FPR. If you try to execute any PS instruction without HID2[PSE] and HID2[LSQE] bit set, an illegal instruction exception will be generated.

3.4.1 FPR format in paired-single mode

63	56	55	48	57	40	39	32
1111	1111	1111	1111	1111	1111	1111	1111
31	24	23	16	15	8	7	0
0000	0000	0000	0000	0000	0000	0000	0000

bit(s)		description
32-63	1	PS1
0-31	0	PS0

3.4.2 Arithmetic Instructions

Mnemonic	Opcode	Description
ps_abs	000100 DDDDD 00000 BBBBB 01000 01000 R	absolute value
ps_add	000100 DDDDD AAAAA BBBBB 00000 10101 R	add
ps_cmpo0	000100 DDD00 AAAAA BBBBB 00001 00000 0	compare ordered high
ps_cmpo1	000100 DDD00 AAAAA BBBBB 00011 00000 0	compare ordered low
ps_cmpu0	000100 DDD00 AAAAA BBBBB 00000 00000 0	compare unordered high
ps_cmpu1	000100 DDD00 AAAAA BBBBB 00010 00000 0	compare unordered low
ps_div	000100 DDDDD AAAAA BBBBB 00000 10010 R	divide
ps_merge00	000100 DDDDD AAAAA BBBBB 10000 10000 R	merge high
ps_merge01	000100 DDDDD AAAAA BBBBB 10001 10000 R	merge direct
ps_merge10	000100 DDDDD AAAAA BBBBB 10010 10000 R	merge swapped
ps_merge11	000100 DDDDD AAAAA BBBBB 10011 10000 R	merge low
ps_mr	000100 DDDDD 00000 BBBBB 00010 01000 R	move register
ps_nabs	000100 DDDDD 00000 BBBBB 00100 01000 R	negate absolute value
ps_neg	000100 DDDDD 00000 BBBBB 00001 01000 R	negate
ps_res	000100 DDDDD 00000 BBBBB 00000 11000 R	reciprocal estimate
ps_rsqrte	000100 DDDDD 00000 BBBBB 00000 11010 R	reciprocal square root estimate
ps_sub	000100 DDDDD AAAAA BBBBB 00000 10100 R	substract
ps_madd	000100 DDDDD AAAAA BBBBB CCCCC 11101 R	multiply and add
ps_madds0	000100 DDDDD AAAAA BBBBB CCCCC 01110 R	multiply and add scalar high
ps_madds1	000100 DDDDD AAAAA BBBBB CCCCC 01111 R	multiply and add scalar low
ps_msub	000100 DDDDD AAAAA BBBBB CCCCC 11100 R	multiply and substract
ps_mul	000100 DDDDD AAAAA 00000 CCCCC 11001 R	multiply
ps_muls0	000100 DDDDD AAAAA 00000 CCCCC 01100 R	multiply scalar high
ps_muls1	000100 DDDDD AAAAA 00000 CCCCC 01101 R	multiply scalar low
ps_nmadd	000100 DDDDD AAAAA BBBBB CCCCC 11111 R	negative multiply and add
ps_nmsub	000100 DDDDD AAAAA BBBBB CCCCC 11110 R	negative multiply and substract
ps_sel	000100 DDDDD AAAAA BBBBB CCCCC 10111 R	select
ps_sum0	000100 DDDDD AAAAA BBBBB CCCCC 01010 R	vector sum high
ps_sum1	000100 DDDDD AAAAA BBBBB CCCCC 01011 R	vector sum low

Note : R opcode field (comparsion of result with zero) is unused. (=0)

3.4.2.1 PS_ABS absolute value

```
Clear bit 0 of PS0[B] and copy result to PS0[D]

Clear bit 0 of PS1[B] and copy result to PS1[D]
```

3.4.2.2   **PS_ADD**   add

```
PS0[D] = PS0[A] + PS0[B]
PS1[D] = PS1[A] + PS1[B]
```

3.4.2.3   **PS_CMPO0**   compare ordered high

```
"c" holds result of comparsion
If (PS0[A] is NaN or PS0[B] is NaN) then c = 0001b
Else if (PS0[A] < PS0[B]) then c = 1000b
Else if (PS0[A] > PS0[B]) then c = 0100b
Else c = 0010b
Save result in D field of condition register (CR[D] = c).
```

3.4.2.4   **PS_CMPO1**   compare ordered low

```
"c" holds result of comparsion
If (PS1[A] is NaN or PS1[B] is NaN) then c = 0001b
Else if (PS1[A] < PS1[B]) then c = 1000b
Else if (PS1[A] > PS1[B]) then c = 0100b
Else c = 0010b
Save result in D field of condition register (CR[D] = c).
```

3.4.2.5   **PS_CMPU0**   compare unordered high

```
"c" holds result of comparsion
If (PS0[A] is NaN or PS0[B] is NaN) then c = 0001b
Else if (PS0[A] < PS0[B]) then c = 1000b
Else if (PS0[A] > PS0[B]) then c = 0100b
Else c = 0010b
Save result in D field of condition register (CR[D] = c).
```

3.4.2.6   **PS_CMPU1**   compare unordered low

```
"c" holds result of comparsion
If (PS1[A] is NaN or PS1[B] is NaN) then c = 0001b
Else if (PS1[A] < PS1[B]) then c = 1000b
Else if (PS1[A] > PS1[B]) then c = 0100b
Else c = 0010b
Save result in D field of condition register (CR[D] = c).
These four compare instructions looks same, because I omitted some
unecessary FPSCR stuff.
```

3.4.2.7 **PS_DIV** divide

```
PS0[D] = PS0[A] / PS0[B]
PS1[D] = PS1[A] / PS1[B]
```

3.4.2.8 **PS_MERGE00** merge high

```
PS0[D] = PS0[A]
PS1[D] = PS0[B]
```

3.4.2.9 **PS_MERGE01** merge direct

```
PS0[D] = PS0[A]
PS1[D] = PS1[B]
```

3.4.2.10 **PS_MERGE10** merge swapped

```
PS0[D] = PS1[A]
PS1[D] = PS0[B]
```

3.4.2.11 **PS_MERGE11** merge low

```
PS0[D] = PS1[A]
PS1[D] = PS1[B]
```

3.4.2.12 **PS_MR** move register

```
PS0[D] = PS0[B]
PS1[D] = PS1[B]
```

3.4.2.13 **PS_NABS** negate absolute value

```
Set bit 0 of PS0[B] and copy result to PS0[D]
Set bit 0 of PS1[B] and copy result to PS1[D]
```

3.4.2.14 **PS_NEG** negate

```
Invert bit 0 of PS0[B] and copy result to PS0[D]
Invert bit 0 of PS1[B] and copy result to PS1[D]
```

3.4.2.15 **PS_RES** reciprocal estimate

```
PS0[D] = 1 / PS0[B]
PS1[D] = 1 / PS1[B]
```

3.4.2.16 **PS_RSQRTE** reciprocal square root estimate

```
PS0[D] = 1 / SQRT(PS0[B])
PS1[D] = 1 / SQRT(PS1[B])
```

3.4.2.17 **PS_SUB**   subtract

```
PS0[D] = PS0[A] - PS0[B]
PS1[D] = PS1[A] - PS1[B]
```

3.4.2.18 **PS_MADD**   multiply-add

```
PS0[D] = PS0[A] * PS0[C] + PS0[B]
PS1[D] = PS1[A] * PS1[C] + PS1[B]
```

3.4.2.19 **PS_MADDS0**   multiply-add scalar high

```
PS0[D] = PS0[A] * PS0[C] + PS0[B]
PS1[D] = PS1[A] * PS0[C] + PS1[B]
```

3.4.2.20 **PS_MADDS1**   multiply-add scalar low

```
PS0[D] = PS0[A] * PS1[C] + PS0[B]
PS1[D] = PS1[A] * PS1[C] + PS1[B]
```

3.4.2.21 **PS_MSUB**   multiply-subtract

```
PS0[D] = PS0[A] * PS0[C] - PS0[B]
PS1[D] = PS1[A] * PS1[C] - PS1[B]
```

3.4.2.22 **PS_MUL**   multiply

```
PS0[D] = PS0[A] + PS0[C]
PS1[D] = PS1[A] + PS1[C]
```

3.4.2.23 **PS_MULS0**   multiply scalar high

```
PS0[D] = PS0[A] + PS0[C]
PS1[D] = PS1[A] + PS0[C]
```

3.4.2.24 **PS_MULS1**   multiply scalar low

```
PS0[D] = PS0[A] + PS1[C]
PS1[D] = PS1[A] + PS1[C]
```

3.4.2.25 **PS_NMADD**   negative multiply-add

```
PS0[D] = - (PS0[A] * PS0[C] + PS0[B])
PS1[D] = - (PS1[A] * PS1[C] + PS1[B])
```

3.4.2.26 **PS_NMSUB**   negative multiply-subtract

```
PS0[D] = - (PS0[A] * PS0[C] - PS0[B])
PS1[D] = - (PS1[A] * PS1[C] - PS1[B])
```

3.4.2.27 PS_SEL   select

```
If (PS0[A] >= 0) then PS0[D] = PS0[C] else PS0[D] = PS0[B]

If (PS1[A] >= 0) then PS1[D] = PS1[C] else PS1[D] = PS1[B]
```

3.4.2.28 PS_SUM0   vector sum high

```
PS0[D] = PS0[A] + PS1[B]

PS1[D] = PS1[C]
```

3.4.2.29 PS_SUM1   vector sum low

```
PS0[D] = PS0[C]

PS1[D] = PS0[A] + PS1[B]
```

3.4.3   Load and Store Instructions

Mnemonic	Opcode	Description
psq_lx	000100 DDDDD AAAAA BBBBB WIII 000110 0	Paired Singles Quantized Load indexed
psq_lux	000100 DDDDD AAAAA BBBBB WIII 100110 0	Paired Singles Quantized Load with Update indexed
psq_stx	000100 SSSSS AAAAA BBBBB WIII 000111 0	Paired Singles Quantized Store indexed
psq_stux	000100 SSSSS AAAAA BBBBB WIII 100111 0	Paired Singles Quantized Store with Update indexed

Mnemonic	Opcode	Description
psq_l	111000 DDDDD AAAAA WIII dddddddddddd	Paired Singles Quantized Load
psq_lu	111001 DDDDD AAAAA WIII dddddddddddd	Paired Singles Quantized Load with Update
psq_st	111100 SSSSS AAAAA WIII dddddddddddd	Paired Singles Quantized Store
psq_stu	111101 SSSSS AAAAA WIII dddddddddddd	Paired Singles Quantized Store with Update

3.4.3.1   psq_lx   Paired Singles Quantized Load indexed

3.4.3.2   psq_lux   Paired Singles Quantized Load with Update indexed

3.4.3.3   psq_stx   Paired Singles Quantized Store indexed

3.4.3.4   psq_stux   Paired Singles Quantized Store with Update indexed

3.4.3.5   psq_l   Paired Singles Quantized Load

3.4.3.6   psq_lu   Paired Singles Quantized Load with Update

3.4.3.7   psq_st   Paired Singles Quantized Store

3.4.3.8   psq_stu   Paired Singles Quantized Store with Update

3.4.4 modified floating point instructions

In paired single mode (HID2[PSE] = 1), all the double-precision floating point instructions are still valid, and execute as in non-paired single mode. All single-precision floating-point instructions (fadds, fsubs, fmuls, fdivs, fmadds, fmsubs, fnmadds, fnmsubs, fres, frsp) switch their meaning and operate on the ps0 operand.

Mnemonic	Opcode	Description
fadds		
fsubs		
fmuls		
fdivs		
fmadds		
fmsubs		
fnmadds		
fnmsubs		
fres		
frsp		
fsel		
fmr		

3.4.4.1 fadds

3.4.4.2 fsubs

3.4.4.3 fmuls

3.4.4.4 fdivs

3.4.4.5 fmadds

3.4.4.6 fmsubs

3.4.4.7 fnmadds

3.4.4.8 fnmsubs

3.4.4.9 fres

3.4.4.10 frsp

3.4.4.11 fsel

3.4.4.12 fmr

3.5   Programming Tips and additional information

3.5.1   Machine State Register

to do

3.5.2   Caches

to do

3.5.3   branch unit

to flush branch unit's dynamic prediction logic, you must sequentially execute 3 branches

```
....
b label1
label1:  b label2
label2:  b label3
label3:
....
```

4   Memory Map

4.1   Overview

start	end	size	description
0x00000000	0x017fffff	24MB	Physical address of the RAM
0x80000000	0x817fffff	24MB	Logical address of the RAM, cached
0xC0000000	0xC17fffff	24MB	Logical address of the RAM, not cached
0xc8000000		2MB	Embedded Framebuffer (EFB)
0xCC000000			Hardware registers
0xCC000000			CP - Command Processor
0xCC001000			PE - Pixel Engine
0xCC002000			VI - Video Interface
0xCC003000			PI - Processor Interface (Interrupt Interface)
0xCC004000			MI - Memory Interface
0xCC005000			AI - Audio Interface
0xCC006000			DI - DVD Interface
0xCC006400			SI - Serial Interface
0xCC006800			EXI - External Interface
0xCC006C00			Streaming Interface
0xCC008000			GX FIFO (Graphic display lists)
0xe0000000	0xe0003fff	16k	L2 Cache
0xfff00000		1MB	IPL (mapped here at bootup)

4.2   RAM usage

Variables that are marked as B are changed by bootrom or IPL. Variables marked as A are changed lately in apploader when a game is booting. Variables, which are marked as O are changed after an OSInit call. Remember that the IPL also has a hard-linked Dolphin OS inside, so those variables that are marked as O are also changed in the IPL.

4.2.1   Dolphin-OS globals

In PowerPC architectures the lower 256 bytes of main memory are reserved for internal OS use. This map describes all known OS low memory variables. Dolphin OS accesses low memory as 0x80000000 + offset (cached).

4.2.1.1   Boot Info

4.2.1.1.1   DVD Disc ID

start	size		description	
0x80000000	0x04	B	Gamecode	
0x80000004	0x02	B	Company	
0x80000006	0x01	B	Disk ID	
0x80000007	0x01	B	Version	
0x80000008	0x01	B	Streaming	
			0	audio streaming off
			1	audio streaming on
0x80000009	0x01	B	StreamBufSize	
0x8000000a	0x0f		padding zeros	

start	size		description	
0x8000001c		A	DVD magic word	
			0xc2339f3d	Nintendo Game Disc
0x80000020	4	A	Magic word (how did the console boot?)	

			value	description
			0x0D15EA5E	normal boot
			0xE5207C22	booted from jtag

start	size		description	
0x80000024	4	A	Version (usually set to 1 by apploader)	
0x80000028	4	B	physical Memory Size	
			0x01800000	24MB on retail console
0x8000002C	4	B	Console type	

4.2.1.1.2 system Info

value	Description
0x00000001	Retail1
0x00000002	HW2 production board
0x00000003	The latest production board
0x00000004	Reserved
0x1XXXXXXX	Devkits
0x10000000	MAC emulator
0x10000001	PC Emulator
0x10000002	'Arthur'
0x10000003	'Minnow'
0x10000004	1st Devkit HW
0x10000005	2nd Devkit HW
0x10000006	latest Devkit HW
0x10000007	Reserved
0x2XXXXXXX	TDEV-kits
0x20000005	HW2 TDEV system
0x20000006	The latest TDEV system
0x20000007	Reserved

start	size		description
0x80000030	4	O	ArenaLo (==0x00000000)
0x80000034	4	O	ArenaHi (==0x817fe8c0)
0x80000038	4		FST Location in ram (==0x817fe8c0)
0x8000003C	4		FST Max Length (==0x00000024)

4.2.1.2 Debugger info

start	size		description
0x80000040	4	A	flag for "debugger present" (used by __OSIsDebuggerPresent)
0x80000044	4	A	Debugger Exception mask Bitmap, set to 0 at sdk lib start
0x80000048	4	A	Exception hook destination (physical address)
0x8000004c	4	A	Temp for LR, Return from exception address (to return from hook)
0x80000050	16		padding zeros

4.2.1.3 Debugger Hook
```
.>80000060  38 a0 00 40  li r5,0x40                                    r5=0x40
.>80000064  7c 68 02 a6  mflr r3                                       r3=lr
.>80000068  90 65 00 0c  stw r3,0x0c(r5)
.>8000006c  80 65 00 08  lwz r3,0x08(r5)
.>80000070  64 63 80 00  oris r3,r3,0x8000
.>80000074  7c 68 03 a6  mtlr r3                                       lr=r3
```

```
.>80000078  38 60 00 30  li r3,0x30
.>8000007c  7c 60 01 24  mtmsr r3                                                          msr=0x30
.>80000080  4e 80 00 20  blr                                                               jump (lr)
```

4.2.1.4   Dolphin OS Globals

start	end	size		description
0x80000084	0x800000bf			padding zeros
0x800000c0		4	O	Current OS context (physical address)
0x800000C4		4	O	Previous OS interrupt mask
0x800000C8		4	O	current OS interrupt mask
0x800000CC				TV Mode
				<table><tr><th>value</th><th>description</th></tr><tr><td>0</td><td>ntsc</td></tr><tr><td>1</td><td>pal</td></tr><tr><td>2</td><td>debug</td></tr><tr><td>3</td><td>debug pal</td></tr><tr><td>4</td><td>mpal</td></tr><tr><td>5</td><td>pal 60</td></tr></table>
0x800000d0			B	ARAM size (internal+expansion) in bytes. set by ARAM driver, usually 16mb.
0x800000D4			O	current OS Context (logical address)
0x800000D8			O	default OS thread (logical address)
0x800000Dc			O	active Thread queue, head thread (logical address)
0x800000e0			O	active Thread queue, tail thread (logical address)
0x800000e4			O	Current OS thread
0x800000e8			A	Debug monitor size (in bytes)
0x800000ec			A	Debug monitor location (usually at the top of main memory)
0x800000F0			A	Console Simulated Memory Size, 0x01800000 (usually same as physical memor
0x800000F4			A	DVD BI2 location in main memory (size of BI2 is 0x2000 bytes)
0x800000F8				Bus Clock Speed, 162 MHz (=0x09a7ec80, 162000000)
0x800000FC				CPU Clock Speed, 486 MHz (=0x1cf7c580, 486000000)

4.2.2   Exception Handlers

start	end	size	description
0x80000100			System Reset Interrupt
0x80000200			Machine Check Interrupt
0x80000300			DSI Interrupt
0x80000400			ISI Interrupt
0x80000500			External Interrupt
0x80000600			Alignment Interrupt
0x80000700			Program Interrupt
0x80000800			FP unavailable Interrupt
0x80000900			Decrementer Interrupt
0x80000C00			System Call Interrupt
0x80000d00			Trace Interrupt
0x80000f00			Performance Monitor Interrupt
0x80001300			IABR Interrupt
0x80001400			reserved
0x80001700			Thermal Interrupt
0x80001800	0x80002fff		unused/reserved (*)

(*) note: psoload v2 uses this area to stay resident in memory, it is unused by Dolphin-OS

4.2.3 Dolphin-OS globals

start	end	size		Description
0x80003000				exception handler vectors (from sdk libs & ipl)
0x8000303c		4		padding/unused
0x80003040				external interrupt handler vectors (from sdk libs & ipl)
0x800030a4				padding/unused
0x800030c0		4		?
0x800030c4		4		?
0x800030c8		4		First Module Header Pointer in Module Queue
0x800030cc		4		Last Module Header Pointer in Module Queue
0x800030d0		4		Module String Table Pointer
0x800030d4		4	A	DOL size (total size of text/data sections), in bytes (*1)
0x800030d8		4	B	OS system time (set, when console is powered up)
0x800030dc		4		?
0x800030E0		4		? (6=production pads ?)
0x800030e4		2		?
0x800030e6		1		?
0x800030e7		1		?
0x800030e8		1	O	set by OsInit() (debugger stuff?)
0x800030e9		1	O	set by OsInit() (debugger stuff?)
0x800030ea		2		?
0x800030ec		4		?
0x800030F0		2		?
0x800030F2		1		Boot status
				<table><tr><th>value</th><th>description</th></tr><tr><td>0</td><td>first boot</td></tr><tr><td>1</td><td>already booted</td></tr></table>
0x800030F3		1		?
0x800030F4		4		?
0x800030F8		4		?
0x800030Fc		4		?

(*1) If FST on DVD is placed after DOL, then BB2 FSTLength is added to this value.

4.2.4 User Memory

4.2.4.1 user program area

start	end	size	description
0x80003100			Start of code (usually)
0x80003140			Entry point (early SDK v1.0 applications)
0x81200000			Load Address of the Apploader
0x81300000			Load Address of Bootrom/IPL

note: of course the entrypoint of an application can be anything, those listed here are just some typical examples. Retail game start dol-files are usually located below the apploader.

4.2.4.2 stack area

start	end	size	description
			Bottom of Stack
			Top of Stack

4.2.4.3 heap area

start	end	size		description
?			O	ArenaLo - Bottom of Heap
0x817fe8c0			O	ArenaHi - Top of Heap

note: the address of ArenaHi is not a constant, but should be set to the bottom of the FST which is read from the DVD so its size depends on the application. the value given here is just an example.

4.2.4.4 'high memory'

start	end	size		description
0x817fe8c0		24	O	FST (used by Dolphin-OS)
0x817fffff				Memory Top

5 Hardware Registers

5.1 CP - Command Processor

Registerblock Base	Size of Registerblock	common access size
0xcc000000	0x80	2

0xCC000000	2	R/W	SR - Status Register

15	8	7	0

bit(s)		description
5-15		unused/reserved
4		BP (breakpoint?) interrupt
3		GP is idle for commands (1: idle)
2		GP is idle for reading (1: idle)
1		gx fifo underflow (ptr<lo watermark)
0		gx fifo overflow (ptr>hi watermark)

0xCC000002	2	R/W	CR - Control Register

15	8	7	0
		..bl	.mig

bit(s)		description
6-15		unused/reserved
5	b	bp enable
4	l	gp link enable (enable for linking of cp/pe FIFO)
3		FIFO underflow irq enable (?)
2	m	FIFO overflow irq enable? / cp irq (clear to acknowledge) ?
1	i	cp irq enable (?) (write 1 to clear bp irq?)
0	g	gp FIFO read enable

0xCC000004	2	W	Clear Register

bit(s)		description
2-15		unused/reserved
1		write 1 to clear FIFO underflow
0		write 1 to clear FIFO overflow

0xCC00000E	2	R/W	token register

0xCC000010	2	R/W	bounding box - left

0xCC000012	2	R/W	bounding box - right

0xCC000014	2	R/W	bounding box - top

0xCC000016	2	R/W	bounding box - bottom

`0xCC000020`	2	R/W	cp FIFO base lo

`0xCC000022`	2	R/W	cp FIFO base hi

`0xCC000024`	2	R/W	cp FIFO end lo

`0xCC000026`	2	R/W	cp FIFO end hi

`0xCC000028`	2	R/W	cp FIFO high watermark lo

`0xCC00002a`	2	R/W	cp FIFO high watermark hi

`0xCC00002c`	2	R/W	cp FIFO low watermark lo

`0xCC00002e`	2	R/W	cp FIFO low watermark hi

the low and high watermark control the assertion of the CP interrupt

`0xCC000030`	2	R/W	cp FIFO read/write distance lo

`0xCC000032`	2	R/W	cp FIFO read/write distance hi

`0xCC000034`	2	R/W	cp FIFO write pointer lo

`0xCC000036`	2	R/W	cp FIFO write pointer hi

`0xCC000038`	2	R/W	cp FIFO read pointer lo

`0xCC00003a`	2	R/W	cp FIFO read pointer hi

`0xCC00003c`	2	R/W	cp FIFO bp lo

`0xCC00003e`	2	R/W	cp FIFO bp hi

5.1.1   Token register

You can insert this dirty marker, at the end of command list, by this way :

*(u32 *)GXFIFO = 0x4800XXXX
*(u32 *)GXFIFO = 0x4700XXXX

Where XXXX is the token value. When command processor reaches this stage, it writes XXXX into PE token register (see above), and then raise "PE TOKEN" interrupt. Thus you can monitor the completion of your drawing tasks.

note: its probably a good idea to send a BP 'drawing complete' command (0x45000002) before the in-

sertion of the token.

5.2   PE - Pixel Engine

Registerblock Base	Size of Registerblock	common access size
`0xcc001000`	`0x100`	2

`0xcc001000`	2	R/W	Z configuration

15	8	7	0

bit(s)		description
4		Z update enable
1-3		function
0		z-comperator enable

`0xcc001002`	2	R/W	Alpha configuration

15	8	7	0

bit(s)		description
12-15		blend operator (?)
11		substractive / additive toggle (?)
8-10		source
5-7		destination
4		alpha update enable
3		color update enable
2		dither enable (?)
1		arithmetic blending enable (?)
0		boolean blending enable (?)

`0xcc001004`	2	R/W	destination alpha

15	8	7	0

bit(s)		description
8		enable
0-7		alpha

`0xcc001006`	2	R/W	Alpha Mode

15	8	7	0

bit(s)		description
8-15		mode
0-7		threshold

`0xcc001008`	2	R/W	Alpha Read (?)

15 8	7 0

bit(s)		description
		mode
2		?

`0xcc00100a`	2	R/W	Interrupt Status Register

15 8	7 0

bit(s)		description
3		PE Finish (set to acknowledge)
2		PE Token (set to acknowledge)
1		PE Finish enable (?)
0		PE Token enable (?)

`0xcc00100e`	2	R/W	PE Token ?

15 8	7 0
tttt tttt	tttt tttt

bit(s)		description
0-15	t	PE Token (asserted from last PE Token Interrupt)

5.3   VI - Video Interface

Registerblock Base	Size of Registerblock	common access size
`0xcc002000`	`0x100`	4

`0xCC002000`	2	R/W	VTR - Vertical Timing Register

15 8	7 0
00aa aaaa	aaaa eeee

bit(s)		description
4-13	a	ACV - Active Video (in full Lines) ? other source says halflines
0-3	e	EQU - Equalization pulse in half lines

pal50/pal60/ntsc: 0x11F5, 0x0F06, 0x0F06

The value in ACV is double buffered

| `0xCC002002` | 2 | R/W | DCR - Display Configuration Register |

```
15      8 7       0
0000  00pp  lltt  dire
```

bit(s)		description
	p	FMT - Current Video Format
		<table><tr><td></td><td></td></tr><tr><td>0</td><td>NTSC</td></tr><tr><td>1</td><td>PAL</td></tr><tr><td>2</td><td>MPAL</td></tr><tr><td>3</td><td>Debug</td></tr></table>
	l	LE1 - Enables Display Latch 1
		<table><tr><td></td><td></td></tr><tr><td>0</td><td>Off</td></tr><tr><td>1</td><td>On for 1 field</td></tr><tr><td>2</td><td>On for 2 fields</td></tr><tr><td>3</td><td>Always On</td></tr></table>
	t	LE0 - Enables Display Latch 0
		<table><tr><td></td><td></td></tr><tr><td>0</td><td>Off</td></tr><tr><td>1</td><td>On for 1 field</td></tr><tr><td>2</td><td>On for 2 fields</td></tr><tr><td>3</td><td>Always On</td></tr></table>
	d	DLR - Selects 3D Display Mode
	i	NIN - Interlace Selector
		<table><tr><td></td><td></td></tr><tr><td>0</td><td>Interlaced</td></tr><tr><td>1</td><td>Non-Interlaced, top field drawn at field rate and bottom field is not displayed</td></tr></table>
	r	RST - Reset - Clears all data requests and puts VI into its idle state.
	e	ENB - Enable - Enables video timing generation and data request.

pal50/pal60/ntsc: 0x0101, 0x0001, 0x0001

| `0xCC002004` | 4 | R/W | HTR0 - Horizontal Timing 0 |

```
31      24 23      16 15       8 7        0
0sss  ssss  0eee  eeee  0000  000w  wwww  wwww
```

bit(s)		description
	s	HCS - Horizontal Sync Start to Color Burst Start
	e	HCE - Horizontal Sync Start to Color Burst End
	w	HLW - Halfline Width (W*16 = Width (720))

pal50/pal60/ntsc: 0x4B6A01B0, 0x476901AD, 0x476901AD

0xCC002008	4	R/W	HTR1 - Horizontal Timing 1

31 24	23 16	15 8	7 0
0000 0sss	ssss ssse	eeee eeee	ewww wwww

bit(s)		description
	s	HBS - Half line to horizontal blanking start
	e	HBE - Horizontal Sync Start to horizontal blank end
	w	HSY - Horizontal Sync Width

pal50/pal60/ntsc: 0x02F85640, 0x02EA5140, 0x02EA5140

Setting bit 0 seems to blackout the screen. (Similar to ViBlack?)

0xCC00200C	4	R/W	VTO - Odd Field Vertical Timing Register

31 24	23 16	15 8	7 0
.... ..ss	ssss ssss	.... ..rr	rrrr rrrr

bit(s)		description
16-25	s	PSB - Post blanking in half lines
0-9	r	PRB - Pre-blanking in half lines

pal50/pal60/ntsc: 0x00010023, 0x00030018, 0x00030018

This register sets up the pre-blanking and post-blanking interval of odd fields, PRB and PSB are double-buffered.

0xCC002010	4	R/W	VTE - Even Field Vertical Timing Register

31 24	23 16	15 8	7 0
.... ..ss	ssss ssss	.... ..rr	rrrr rrrr

bit(s)		description
16-25	s	PSB - post-blanking in halflines
0-9	r	PRB - pre-blanking in halflines

pal50/pal60/ntsc: 0x00000024, 0x00020019, 0x00020019

This register sets up the pre-blanking and post-blanking intervals of even fields. PRB and PSB are double-buffered.

0xCC002014	4	R/W	BBEI - Odd Field Burst Blanking Interval Register

31 24	23 16	15 8	7 0

bit(s)		description
21-31		BE3 - Field 3 start to burst blanking end in halflines
16-20		BS3 - Field 3 start to burst blanking start in halflines
5-15		BE1 - Field 1 start to burst blanking end in halflines
0-4		BS1 - Field 1 start to burst blanking start in halflines

pal50/pal60/ntsc: 0x4D2B4D6D, 0x410C410C, 0x410C410C

0xCC002018	4	R/W	BBOI - Even Field Burst Blanking Interval Register

31 24	23 16	15 8	7 0

bit(s)		description
21-31		BE4 - Field 4 start to burst blanking end in halflines
16-20		BS4 - Field 4 start to burst blanking start in halflines
5-15		BE2 - Field 2 start to burst blanking end in halflines
0-4		BS2 - Field 2 start to burst blanking start in halflines

pal50/pal60/ntsc: 0x4D8A4D4C, 0x40ED40ED, 0x40ED40ED

0xCC00201c	4	R/W	TFBL - Top Field Base Register (L) (External Framebuffer Half 1)

31 24	23 16	15 8	7 0
yyy? zzzz	aaaa aaaa	aaaa aaax	xxxx xxxx

bit(s)		description
29-31	y	always zero (maybe some write only control register stuff?, setting bit 31 clears bits 31-28 (?))
28		page offset bit (*1)
24-27	z	XOF - Horizontal Offset of the left-most pixel within the first word of the fetched picture.
9-23	a	FBB - bit 23 - bit 9 of XFB Address (*2)
0-8	x	unused (?)

pal50/pal60/ntsc: 0x00435A4E, 0x00435A4E, 0x00435A4E

This register specifies the display origin of the top field of a picture in 2D mode or for the left picture in 3D mode

(*1) when this bit is set, the framebuffer address is calculated as (address>>5)

(*2) if bit 28 is cleared, highest possible Address: 0x80fffe00 (set register to 0x00fffe00) (aligned to 9bit)

0xCC002020	4	R/W	TFBR - Top Field Base Register (R) (Only valid in 3D Mode)

31 24	23 16	15 8	7 0
0000 0000	ffff ffff	ffff fff0	0000 0000

bit(s)		description
	f	FBB - External Memory Address of frame buffer

pal50/pal60/ntsc: 0x00000000, 0x00000000, 0x00000000

This register specifies the base address of the top field for the right picture in 3D mode.

0xCC002024	4	R/W	BFBL - Bottom Field Base Register (L) (External Framebuffer Half 2)

31 24	23 16	15 8	7 0
yyyy yyyy	aaaa aaaa	aaaa aaax	xxxx xxxx

bit(s)		description
	y	always zero (maybe some write-only control register stuff?)
28		page offset bit (*1)
	a	FBB - bit 23 - bit 9 of XFB Address
	x	unused (?)

pal50/pal60/ntsc: 0x00435A4E, 0x00435A4E, 0x00435A4E

This register specifies the display origin of the bottom field of a picture in 2D mode or for the left picture in 3D mode

(*1) when this bit is set, the framebuffer address is calculated as (address>>5)

0xCC002028	4	R/W	BFBR - Bottom Field Base Register (R) (Only valid in 3D Mode)

31 24	23 16	15 8	7 0
0000 0000	ffff ffff	ffff fff0	0000 0000

bit(s)		description
	f	FBB - External Memory Address of frame buffer

pal50/pal60/ntsc: 0x00000000, 0x00000000, 0x00000000

specifies the base address of the bottom field for the right picture in 3D mode.

0xCC00202C	2	R	DPV - current vertical Position

15 8	7 0
0000 0vvv	vvvv vvvv

bit(s)		description
	v	VCT - current vertical Position of Raster beam

pal50/pal60/ntsc: 0x013C, 0x0005, 0x0000

0xCC00202E	2	R	DPH - current horizontal Position (?)

15	8	7	0
0000	0hhh	hhhh	hhhh

bit(s)		description
	h	HCT - current horizontal Position of Raster beam (?)

pal50/pal60/ntsc: 0x0144, 0x0176, 0x0000

The Horizontal Count is in pixels and runs from 1 to # pixels per line. It is reset to 1 at the beginning of every line.

The Vertical Count is in lines (on a frame basis) and runs from 1 to # lines per frame. It is 1 at the beginning of pre-equalization. This is a frame line count. So for example: for NTSC vcount=264 is the first (full) line in the second field and vcount=525 is the last line in the frame (fields being numbered 1-4). For non-interlaced modes vcount is on a field-by-field basis (for NTSC vcount ranges from 1-263).

This counting scheme applies the Display Position, Display Interrupt, and Display Latch registers.

0xCC002030	4	R/W	DI0 - Display Interrupt 0

31	24	23	16	15	8	7	0
i00e	00vv	vvvv	vvvv	0000	00hh	hhhh	hhhh

bit(s)		description
31	i	INT - Interrupt Status (1=Active) (Write to clear)
28	e	ENB - Interrupt Enable Bit
16-25	v	VCT - Vertical Position
0-9	h	HCT - Horizontal Position

pal50/pal60/ntsc: 0x113901B1, 0x110701AE, 0x110701AE

There are a total of four display interrupt registers (0-3). They are used to generate interrupts to the main processor at different positions within a field. Each register has a separate enable bit. The interrupt is cleared by writing a zero to the status flag (INT).

0xCC002034	4	R/W	DI1 - Display Interrupt 1

pal50/pal60/ntsc: 0x10010001, 0x10010001, 0x10010001

Refer to Display Interrupt 0

0xCC002038	4	R/W	DI2 - Display Interrupt 2

pal50/pal60/ntsc: 0x00010001, 0x00010001, 0x00010001

Refer to Display Interrupt 0

| `0xCC00203C` | 4 | R/W | DI3 - Display Interrupt 3 |

pal50/pal60/ntsc: 0x00010001, 0x00010001, 0x00010001

Refer to Display Interrupt 0

| `0xCC002040` | 4 | R/W | DL0 - Display Latch Register 0 |

bit(s)		Description
31		TRG - Trigger Flag
16-26		VCT - Vertical Count
0-10		HCT - Horizontal Count

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

The Display Latch Register 0 latches the value of the Display Position Register at the rising edge of the gt0 signal. The trigger flag is set if a gun trigger is detected. Writing a zero to the register clears the trigger flag.

| `0xCC002044` | 4 | R/W | DL1 - Display Latch Register 1 |

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

See the description of Display Latch Register 0. This register is latched on the rising edge of the gt1 signal.

| `0xCC002048` | 2 | R/W | HSW - Scaling Width Register |

15	8	7	0

bit(s)		description
0-9		SRCWIDTH - Horizontal Stepping size

pal50/pal60/ntsc: 0x2850, 0x2850, 0x2850

This register is the number of source pixels to be scaled. This is only used when the Horizontal Scaler is enabled. For example, if the image is to be scaled from 320x240 to 640x240, 320 would be written into this register.

| `0xCC00204a` | 2 | R/W | HSR - Horizontal Scaling Register |

15	8	7	0
000e	000v	vvvv	vvvv

bit(s)		description
12	e	HS_EN - Enable Horizontal Scaling
0-8	v	STP - Horizontal stepping size (U1.8 Scaler Value) (0x160 Works for 320)

pal50/pal60/ntsc: 0x0100, 0x0100, 0x0100

This register sets up the step size of the horizontal stepper.

`0xCC00204C`	4	R/W	FCT0 - Filter Coefficient Table 0 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
20-29		T2 - Tap2
10-19		T1 - Tap1
0-9		T0 - Tap0

pal50/pal60/ntsc: 0x1AE771F0, 0x1AE771F0, 0x1AE771F0

sets up part of the low-pass filter. Taps 0 to 9 are in the range (0.0, 2.0)

`0xcc002050`	4	R/W	FCT1 - Filter Coefficient Table 1 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
20-29		T5 - Tap5
10-19		T4 - Tap4
0-9		T3 - Tap3

pal50/pal60/ntsc: 0x0DB4A574, 0x0DB4A574, 0x0DB4A574

`0xcc002054`	4	R/W	FCT2 - Filter Coefficient Table 2 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
20-29		T8 - Tap8
10-19		T7 - Tap7
0-9		T6 - Tap6

pal50/pal60/ntsc: 0x00C1188E, 0x00C1188E, 0x00C1188E

sets up part of the low-pass filter

`0xcc002058`	4	R/W	FCT3 - Filter Coefficient Table 3 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
24-31		T12 - Tap12
16-23		T11 - Tap11
8-15		T10 - Tap10
0-7		T9 - Tap9

pal50/pal60/ntsc: 0xC4C0CBE2, 0xC4C0CBE2, 0xC4C0CBE2

sets up part of the low-pass filter. Taps 9 to tap 24 are in the Rage (-0.125, 0.125)

`0xcc00205c`	4	R/W	FCT4 - Filter Coefficient Table 4 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
24-31		T16 - Tap16
16-23		T15 - Tap15
8-15		T14 - Tap14
0-7		T13 - Tap13

pal50/pal60/ntsc: 0xFCECDECF, 0xFCECDECF, 0xFCECDECF

`0xcc002060`	4	R/W	FCT5 - Filter Coefficient Table 5 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)		description
24-31		T20 - Tap20
16-23		T19 - Tap19
8-15		T18 - Tap18
0-7		T17 - Tap17

pal50/pal60/ntsc: 0x13130F08, 0x13130F08, 0x13130F08

0xcc002064	4	R/W	FCT6 - Filter Coefficient Table 6 (AA stuff)

31 24	23 16	15 8	7 0

bit(s)	description
24-31	T24 - Hardwired to zero
16-23	T23 - Tap23
8-15	T22 - Tap22
0-7	T21 - Tap21

pal50/pal60/ntsc: 0x00080C0F, 0x00080C0F, 0x00080C0F

sets up part of the low-pass filter

0xcc002068	4	R/W	? (AA stuff)

pal50/pal60/ntsc: 0x00FF0000, 0x00FF0000, 0x00FF0000

0xCC00206C	2	R/W	VICLK - VI Clock Select Register

15 8	7 0
0000 0000	0000 000s

bit(s)			description
	s	0	27 MHz video CLK
		1	54 MHz video CLK (used in Progressive Mode)

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

0xCC00206e	2	R/W	VISEL - VI DTV Status Register

15 8	7 0

bit(s)	description
2	VISEL - don't care

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

this register allows software to read the status of two i/o pins

0xCC002070	2	R/W	?

Holds 0x280, but has no effect on change (maybe for Progressive ?)

pal50/pal60/ntsc: 0x0280, 0x0280, 0x0280

`0xCC002072`	2	r/w	HBE - Border HBE

15	8	7	0

bit(s)		description
15		BRDR_EN - Border Enable
0-9		HBE656 - Border Horizontal Blank End

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

This register (in conjunction with the border HBS) sets up a black border around the actual active pixels in debug mode. This was done in order to accommodate certain encoders that only support 720 active pixels. The border HBE and HBS can be programmed for 720 active pixels while the regular HBE and HBS can be programmed to the actual active width. This allows the frame buffer to be of any width without having to manually set up a border in memory. These registers will only take effect if enabled and in debug mode.

`0xcc002074`	2	r/w	HBS - Border HBS

15	8	7	0

bit(s)		description
0-9		HBS656 - Border Horizontal Blank start

pal50/pal60/ntsc: 0x0000, 0x0000, 0x0000

`0xcc002076`	2	?/?	? (unused?)

pal50/pal60/ntsc: 0x00FF, 0x00FF, 0x00FF

`0xcc002078`	4	?/?	? (unused?)

pal50/pal60/ntsc: 0x00FF00FF, 0x00FF00FF, 0x00FF00FF

`0xcc00207c`	4	?/?	? (unused?)

pal50/pal60/ntsc: 0x00FF00FF 0x00FF00FF, 0x00FF00FF

5.3.1 Video Modes

Mode	TV Norm / Region	Framerate	Columns	Lines
NTSC	ntsc (usa, japan)	60Hz	640	480
PAL	pal (europe)	50Hz	640	574
DEBUG				
DEBUG PAL				
MPAL	pal (brazil)	60Hz	640	480
PAL60	pal	60Hz	640	480

note: other modes may be possible using VGA output, although its unlikely.

5.4   PI - Processor Interface

Registerblock Base	Size of Registerblock	common access size
0xcc003000	0x100	4

0xCC003000	4	r	INTSR - interrupt cause

31 24	23 16	15 8	7 0
....	....r	....	....

bit			Description
17-31			unused/reserved
16	r	RSWST	Reset Switch State (1 when pressed)
14-15			unused/reserved
13		HSP	High Speed Port
12		DEBUG	External Debugger
11		CP	Command FIFO
10		PE FINISH	Frame is Ready
9		PE TOKEN	Token Assertion in Command List
8		VI	Video Interface
7		MEM	Memory Interface
6		DSP	DSP
5		AI	Streaming
4		EXI	EXI
3		SI	Serial
2		DI	DVD
1		RSW	Reset Switch
0		ERROR	GP runtime error

0xCC003004	4	r/w	INTMR - interrupt mask

31 24	23 16	15 8	7 0
....	....	....	....

bit			Description
13		HSP	High Speed Port
12		DEBUG	External Debugger
11		CP	Command FIFO
10		PE FINISH	Frame is Ready
9		PE TOKEN	Token Assertion in Command List
8		VI	Video Interface
7		MEM	Memory Interface
6		DSP	DSP
5		AI	Streaming
4		EXI	EXI
3		SI	Serial
2		DI	DVD
1		RSW	Reset Switch
0		ERROR	GP runtime error

0xCC00300c	4	r/w	FIFO Base Start

| 0xCC003010 | 4 | ?/? | FIFO Base End? |

| 0xCC003014 | 4 | ?/? | PI (cpu) FIFO current Write Pointer? |

| 0xCC003018 | 4 | ?/? | ? |

| 0xCC00301c | 4 | ?/? | ? |

| 0xCC003020 | 4 | ?/? | ? |

| 0xCC003024 | 4 | ?/? | Reset? |

Writing anything here seems to cause a complete reset.

| 0xCC00302c | 4 | ?/? | ? |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
28-31		console type (2: hw2)

5.4.1   Operation

5.4.1.1   FIFO/Write Gather Pipe   when CPU writes a byte to 0xcc008000, it is written to mem[writeptr], and writeptr is increased automatically.

0xcc008000 is the write gather pipe, a way for the CPU to blast sequences of things of various sizes to memory without having to keep track of the write pointer and wrapping manually. the gp then reads what the CPU has written to memory. It is used for Display Lists. it will disconnect the GP from the write-gatherpipe (cc000002 & 0x10 = 0), and change the write ptr to where it wants to write a display list.. then use ordinary GX commands to build it. there's a Call Displaylist GX command.. so it will store render commands for rendering a certain object (for example) in a display list in memory, then send the CallDL with the address to the list instead of sending all the vertices over the FIFO.

5.4.1.2   Interrupts   Each interrupt has one or more "source" devices. It means that some kind of device may generate a couple of different interrupts, represented by a single bit in interrupt registers. To "enable" interrupt, set bit in mask register. To ignore all interrupts write 0 to interrupt mask register. Raising of any interrupt will set corresponding bit in interrupt cause register. Interrupt cause register resets to 0, when read (i.e. you must read it to clear pending interrupts).

Interrupt mask register isn't controlled by hardware logic. Note that masking of interrupt in INTMR doesn't disable it at all. It is only causing masked interrupt to be ignored in the software interrupt handler. You must clear corresponding "source" device registers, to completely disable interrupt.

5.4.1.3   hotreset   this code snippet will reset the machine almost as if powered off/on

```
lis r3,0
lis r9,0xCC00
sth r3, 0x2000(r9)
li r4, 3
stw r4, 0x3024(r9)
stw r3, 0x3024(r9)
nop
loop__:
b loop__
```

5.5 MI - Memory Interface

Protection can be enabled only for pages (page size is 1024 bytes), and you can specify only 4 protected regions of memory. External interrupt will be raised, if CPU try to wrong access in protected region. Because it's allowed to enable protection for 4 regions only, there are a total of 4 possible interrupts which are called MEM_0, MEM_1, MEM_2 and MEM_3.

Registerblock Base	Size of Registerblock	common access size
0xcc004000	0x80	4

0xCC004000	4	r/w	Protected Region No1
0xCC004004	4	r/w	Protected Region No2
0xCC004008	4	r/w	Protected Region No3
0xCC00400c	4	r/w	Protected Region No4

31 24	23 16	15 8	7 0
llll llll	llll llll	hhhh hhhh	hhhh hhhh

bit(s)		Description
16-31	l	Page Address Lo
0-15	h	Page Address Hi

```
note:  the page address can be calculated as (physical_address>>10)
```

0xCC004010	2	r/w	type of the protection, 4*2 bits

15 8	7 0
....	3322 1100

bit(s)		Description	
		unused/reserved	
6	3	Channel 3	
		0	access denied
		1	read only (break on write)
		2	write only (break on read)
		3	read / write (no protection, full access)
4	2	Channel 2 (see Channel 3)	
2	1	Channel 1 (see Channel 3)	
0	0	Channel 0 (see Channel 3)	

`0xCC00401c`	2	?/w	MI interrupt mask

```
15        8 7          0
.... ....  ...m  3210
```

bit(s)		Description
4	m	mask all MI interrupts (1 - enable)
3	3	mask MEM3 interrupt (1 - enable)
2	2	mask MEM2 interrupt (1 - enable)
1	1	mask MEM1 interrupt (1 - enable)
0	0	mask MEM0 interrupt (1 - enable)

`0xCC00401e`	2	r/w	interrupt cause

```
15        8 7          0
.... ....  ...m  3210
```

bit(s)		Description		
4	m	all MI interrupts		
3	3	MEM3 interrupt		
		read	0	irq has not been requested
			1	irq has been requested
		write	0	no effect
			1	clear pending irq assertion
2	2	MEM2 interrupt		
		read	0	irq has not been requested
			1	irq has been requested
		write	0	no effect
			1	clear pending irq assertion
1	1	MEM1 interrupt		
		read	0	irq has not been requested
			1	irq has been requested
		write	0	no effect
			1	clear pending irq assertion
0	0	MEM0 interrupt		
		read	0	irq has not been requested
			1	irq has been requested
		write	0	no effect
			1	clear pending irq assertion

`0xCC004020`	2	?/?	?

```
15        8 7          0
.... ....  ....  ..m.
```

bit(s)		Description
1	1	? (set when MI interrupt has been asserted)
0	0	?

`note:` assume to be zero, after init, and should be cleared by interrupt handler.

| 0xCC004022 | 2 | r/? | ADDRLO - address which failed protection rules |

15 8	7 0
....	....

bit(s)		Description
5-15		bit 5-bit 15 of address
0-4		zero

| 0xCC004024 | 2 | r/? | ADDRHI - address, which failed protection rules |

15 8	7 0
....	....

bit(s)		Description
14-15		zero
0-13		bit 16-bit 29 of address

0xCC004032	2	r/?	TIMERHI
0xCC004034	2	r/?	TIMERLO
0xCC004036	2	r/?	TIMERHI
0xCC004038	2	r/?	TIMERLO
0xCC00403a	2	r/?	TIMERHI
0xCC00403c	2	r/?	TIMERLO
0xCC00403e	2	r/?	TIMERHI
0xCC004040	2	r/?	TIMERLO
0xCC004042	2	r/?	TIMERHI
0xCC004044	2	r/?	TIMERLO
0xCC004046	2	r/?	TIMERHI
0xCC004048	2	r/?	TIMERLO
0xCC00404a	2	r/?	TIMERHI
0xCC00404c	2	r/?	TIMERLO
0xCC00404e	2	r/?	TIMERHI
0xCC004050	2	r/?	TIMERLO
0xCC004052	2	r/?	TIMERHI
0xCC004054	2	r/?	TIMERLO
0xCC004056	2	r/?	TIMERHI
0xCC004058	2	r/?	TIMERLO

note: writing anything to the timer register resets it to zero

| 0xCC00405a | 2 | r/? | ? |

15 8	7 0
.... .xxx	xxxx xxxx

bit(s)		Description
11-15		unused ?

5.6   DSP - Digital Signal Processor Interface

At the heart of the GCN audio hardware is a custom digital signal processor (DSP) which is largely dedicated to pitch modulation and the mixing of voices and effects data. The DSP is augmented by a large quantity of auxiliary RAM (ARAM) which may be used to store audio samples.The GCN audio hardware features a custom digital signal processor (DSP) which has the following characteristics:

 ▷ 81MHz instruction clock.

 ▷ 16-bit data words and addressing.

 ▷ 8KB (4Kword) Data RAM.

 ▷ 4KB (2Kword) Data ROM.

 ▷ 8KB (4Kword) Instruction RAM.

 ▷ 8KB (4Kword) Instruction ROM.

 ▷ 40-bit add-and-multiply-result registers.

 ▷ Hardware ADPCM decoder.

 ▷ Cached memory interface to ARAM.

 ▷ DMA interface to main memory.

 ▷ "Mailbox" registers for communicating with the CPU.

Register block Base	Size of Register block	common access size
0xCC005000	0x200 bytes	16bit words

0xCC005000	2	r/w	DSP Mailbox High (to DSP)
0xCC005002	2	r/w	DSP Mailbox Low (to DSP)

bit31 of DSP Mailbox shows mail delivery status. (it will be cleared when the transfer is done)
to send mail just write data, high word first, with bit31 set.

0xCC005004	2	r	CPU Mailbox High (from DSP)
0xCC005006	2	r	CPU Mailbox Low (from DSP)

bit31 of CPU Mailbox shows mail delivery status.

| `0xCC00500a` | 2 | ?/w | AI DSP CSR - Control Status Register (DSP Status) |

15 8	7 0
....	....

bit(s)		Description			
11		Reset DSP (?)			
10					
9		DSP DMA Int Status			
8		DSPINTMSK - DSP interrupt mask (*1)			
7		DSPINT			
		read	0	no interrupts	
			1	interrupt is active	
		write	0	no effect	
			1	clear interrupt	
6		ARINTMSK - ARAM interupt mask (*2)			
5		ARINT -			
		read	0	no interrupts	
			1	interrupt is active	
		write	0	no effect	
			1	clear interupt	
4		AIDINTMASK - AI interrupt mask (*3)			
3		AIDINT			
		read	0	no interrupts	
			1	interrupt is active	
		write	0	no effect	
			1	clear interrupt	
2		HALT - Halt DSP (?)			
		read	0		
			1		
		write	0	unhalt DSP	
			1	halt DSP (stop task execution)	
1		PIINT - DSP Interrupt Assertion (?)			
		read	0		
			1		
		write	0		
			1	assert PI DSP interrupt	
0		RES - Reset DSP (?)			
		read	0		
			1		
		write	0		
			1	reset DSP	

(*1) disables only PI interrupt, doesnt effect assertion of DSPINT.
(*2) disables only PI interrupt, doesnt effect assertion of ARINT.
(*3) disables only PI interrupt, doesnt effect assertion of AIDINT.

| `0xCC005012` | 2 | ?/? | AR_SIZE |

0xCC005016	2	?/?	AR_MODE

0xCC00501a	2	?/?	AR_REFRESH

0xCC005020	2	?/?	AR_DMA_MMADDR_H

0xCC005022	2	?/?	AR_DMA_MMADDR_L

0xCC005024	2	?/?	AR_DMA_ARADDR_H

0xCC005026	2	?/?	AR_DMA_ARADDR_L

0xCC005028	2	?/?	AR_DMA_CNT_H

bit(s)		description
15		type of transfer (0: write to aram 1: read from aram)
0-14		high bits of transfer length

0xCC00502a	2	?/?	AR_DMA_CNT_L

0xCC005030	2	?/w	DMA Start address (High)

Start of Audio Data

0xCC005032	2	?/w	DMA Start address (Low)

Start of Audio Data

0xCC005036	2	?/w	DMA Control/DMA length (Length of Audio Data)

15	8	7	0
axxx	xxxx	xxxx	xxxx

bit(s)		Description
	a	0=stop sample 1=play sample
	x	length/32 (max len is 0x000fffe0)

`0xCC00503a`	2	r/?	DMA Bytes left

Counts down to zero showing how any bytes are left

5.6.1 internal DSP Registers

Registerblock Base	Size of Registerblock	common access size
`0xffc9`		2

`0xFFC9`	2	r/w	DSCR - DSP dma Control Register

15	8	7	0

bit(s)		description		
3-15		unused/reserved		
2		DSP DMA busy		
		read	0	
			1	Block length counter not yet zero, DMA is still busy
		write	0	no effect ?
			1	no effect ?
1		DSP source/destination (DMA involved DSP memory)		
		0	DSP data memory	
		1	DSP instruction memory	
0		transfer direction		
		0	from main memory to DSP memory	
		1	from DSP memory to main memory	

`0xFFCB`	2	r/w	DSBL - DSp dma Block Length

15	8	7	0

bit(s)		description
2-15		block length - This register is used to specify DSP DMA transfer length from bit 15 to bit 2
0-1		r: 2 bit of its LSBs - The transfer length is a multiple of 4 bytes

`0xFFCD`	2	r/w	DSPA - DSp dma dsP memory Address High

15	8	7	0

bit(s)		description
1-15		DSP memory address - This register is used to specify DSP memory starting/current address from bit 15 to bit 1
0		r: 1 bit of its LSBs - The DSP memory address should be located at 2 word boundary

0xFFCE	2	r/w	DSMAH - DSp dma Main memory Address High

15	8	7	0

bit(s)		description
10-15		r: 6 bits of its MSBs - This register is used to specify DSP DMA main memory starting/current address from bit
0-9		main memory address high word - This register is used to specify DSP DMA main memory starting/current add

0xFFCF	2	r/w	DSMAL - DSp dma Main memory Address Low

15	8	7	0

bit(s)		description
2-15		main memory address - This register is used to specify DSP DMA main memory starting/current address from b
0-1		r: 2 bits of its LSBs - The main memory address of this DMA should be located at 4 byte boundary

0xFFD4	2	r/w	ACSAH - Accelerator aram Starting Address High

15	8	7	0

bit(s)		description
11-15		unused/reserved
0-10		wtarting address high-word - Bit 26 to bit 16 of ARAM starting address

0xFFD5	2	r/w	ACSAL - Accelerator aram Starting Address Low

15	8	7	0

bit(s)		description
0-15		Starting address low-word - Bit 15 to bit 0 of ARAM starting address

0xFFD6	2	w	ACEAH - Accelerator aram Ending Address High

15	8	7	0

bit(s)		description
15-11		unused/reserved
0-10		ending address high-word - Bit 26 to bit 16 of ARAM ending address

`0xFFD7`	2	w	ACEAL - Accelerator aram Ending Address Low

15	8	7	0

bit(s)	description
0-15	ending address low-word - Bit 15 to bit 0 of ARAM ending address

`0xFFD8`	2	r/w	ACCAH - Accelerator aram Current Address High

15	8	7	0

bit(s)	description		
15	direction		
		0	accelerator read ARAM
		1	accelerator write ARAM
11-14	unused/reserved		
0-10	current address high-word - Bit 26 to bit 16 of ARAM current address		

`0xFFD9`	2	r/w	ACCAL - Accelerator aram Current Address Low

15	8	7	0

bit(s)	description
0-15	Bit 15 to Bit 0 of ARAM current address

`0xFFEF`	2	r/w	AMDM - ARAM-Dma request Mask

15	8	7	0

bit(s)	description		
1-15	unused/reserved		
0		0	DMA request ARAM is unmasked
		1	DMA request ARAM is masked

5.6.2 Operation

5.6.2.1 play raw audio sample

▷ load DMA Start Address

▷ setup DMA Control/DMA length

5.6.2.2 transfer from/to ARAM

▷ set main memory address in AR_DMA_MMADDR_H and AR_DMA_MMADDR_L

▷ set aram address in AR_DMA_ARADDR_H and AR_DMA_ARADDR_L

▷ set length and transfer type in AR_DMA_CNT_H and AR_DMA_CNT_L

▷ wait until bit 9 of AI DSP Control register gets cleared

5.6.2.3   reset DSP

▷ set bit0 and bit11 of AI DSP Control Register

5.6.2.4   Boot DSP Task

▷ send mail to DSP: `0x80F3A001`, ram_mmem_addr

▷ send mail to DSP: `0x80F3C002`, ram_addr

▷ send mail to DSP: `0x80F3A002`, ram_length

▷ send mail to DSP: `0x80F3B002`, aram_mmem_addr (==0 ?)

▷ send mail to DSP: `0x80F3D001`, dsp_init_vector

5.7   DI - DVD Interface

Register block Base	Size of Register block	common access size
0xCC006000	0x40	4

`0xCC006000`	4	r/w	DISR - DI Status Register

31	24	23	16	15	8	7	0
....	....	....	....	....	....	....	....

bit(s)	description
7-31	reserved
6	BRKINT - Break Complete Interrupt Status (*1)
	read 0 Interrupt has not been requested
	1 Interrupt has been requested
	write 0 no effect
	1 clear Interrupt
5	BRKINTMASK - Break Complete Interrupt Mask. 0:masked, 1:enabled (*2)
4	TCINT - Transfer Complete Interrupt Status (*3)
	read 0 Interrupt has not been requested
	1 Interrupt has been requested
	write 0 no effect
	1 clear Interrupt
3	TCINTMASK - Transfer Complete Interrupt Mask. 0:masked, 1:enabled (*4)
2	DEINT - Device Error Interrupt Status (*5)
	read 0 Interrupt has not been requested
	1 Interrupt has been requested
	write 0 no effect
	1 clear Interrupt
1	DEINTMASK - Device Error Interrupt Mask. 0:masked, 1:enabled (*6)
0	BRK - DI Break (*7)
	read 0 break not requested or break complete
	1 break requested and pending
	write 0 no effect
	1 request break

(*1) On read this bit indicates the current status of the break complete interrupt. This interrupt is asserted when a Break cycle has completed (break acknowledge received from mass storage access device). When a '1' is written to this register, the interrupt is cleared.

(*2) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of DISR[BRKINT]

(*3) On read this bit indicates the current status of the transfer complete interrupt. The Transfer Complete interrupt is asserted under the following conditions: a DMA mode transfer has completed (DMA finished) or an Immediate mode transfer has completed (transfer to/from DIIMMBUF has completed). When a '1' is written to this register, the interrupt is cleared. The assertion of TCIT is delayed until the DIDSTRBb (low) in order to guarantee the error interrupt occurs before transfer complete interrupt. If DIERRb is asserted during the current transaction, the transaction will be halted and TCINT will not be asserted.

(*4) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of DISR[TCINT]

(*5) On read this bit indicates the current status of the mass storage access device error interrupt. To clear this interrupt, two actions must occur. When a '1' is written to this register, the internal interrupt is cleared. To reset the DIERRb signal, a command must be issued to the external DI device. If error occurs during the command packet, the drive has to delay the error assertion until the completion of the 12 bytes command transfer. In immediate mode, if error occurs during the data packet, the error assertion has to be delayed until the completion of the 4 bytes data transfer. In DMA mode, it has to be delayed until the completion of any 32 bytes data transfer.

(*6) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the

assertion of DISR[DEINT]

(*7) When a '1' is written to this bit, the DI controller interrupts the current command and sends a break signal to the mass storage access device. The break signal interrupts the current command on the mass storage access device. After the break sequence is complete (see TCINT), a new command may be sent to the mass storage access device. This bit is cleared after the break command is complete. Note that DI controller will delay the break signal assertion if it is in the middle of the command transfer. Hence break can only occur during the data transfer or when it is idle.

`0xCC006004`	4	r/w	DICVR - DI Cover Register (status2)

31 24	23 16	15 8	7 0
....	....	....	.... .smc

bit(s)		Description			
2	s	CVRINT - Cover Interrupt Status (*1)			
			read	0	cover interrupt has not been requested
				1	cover interrupt has been requested
			write	0	no effect
				1	clear cover interrupt
1	m	CVRINTMASK - Cover Interrupt Mask. 0: masked, 1: enabled (*2)			
0	c	CVR - State of the DICOVER signal. 0: cover closed, 1: cover opened			

(*1) On read this bit indicates the current status of the Mass Storage Device Cover interrupt. When a '1' is written to this register, the internal interrupt is cleared. The Mass Storage Device Cover Interrupt is asserted when the status of the DICOVER signal changes (e.g., when the cover is opened or closed).

(*2) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of DISR[CVRINT].

`0xCC006008`	4	r/w	DICMDBUF0 - DI Command Buffer 0

31 24	23 16	15 8	7 0
cccc cccc	1111 1111	2222 2222	2222 2222

bit(s)		Description
24-31	c	command
16-23	1	subcommand 1
0-15	2	subcommand 2

`0xCC00600c`	4	r/w	DICMDBUF1 - DI Command Buffer 1 (offset in 32 bit words)

`0xCC006010`	4	r/w	DICMDBUF2 - DI Command Buffer 2 (source length)

`0xCC006014`	4	r/w	DIMAR - DMA Memory Address Register

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		description
26-31		reserved/unused
5-25		DIMAR - Address of source/destination buffer in main Memory
0-4		always zero (Address must be 32 byte aligned)

`0xCC006018`	4	r/w	DILENGTH - DI DMA Transfer Length Register

31	24	23	16	15	8	7	0
....	....	....	....	....	....	....	....

bit(s)		description
26-31		reserved/unused
5-25		DILENGTH - length of DMA data transfer in bytes (*1)
0-4		always zero (transfer length must be 32 byte aligned)

(*1) If a DMA command is interrupted by a break cycle, this register indicates the amount of data that was left to transfer before the DMA command was interrupted. If the length equals zero, it is a special case with command transfer only.

`0xCC00601c`	4	r/w	DICR - DI Control Register

31	24	23	16	15	8	7	0
....	....	....	....	....	....	....	.mbe

bit(s)		Description
2	m	RW - access mode, 0:read, 1:write
1	b	DMA - 0: immediate mode, 1: DMA mode (*1)
0	e	TSTART - transfer start. write 1: start transfer, read 1: transfer pending (*2)

(*1) The only mass storage device packet command which can use immediate mode is the 'Register Access' command. When in immediate mode, the DIMAR and DILENGTH registers are ignored.

(*2) When read this bit represents the current command status. This bit is also cleared after the break completion and after DIERRb is asserted.

`0xCC006020`	4	r/w	DIIMMBUF - DI immediate data buffer (error code ?)

31	24	23	16	15	8	7	0
....	....	....	....	....	....	....	....

bit(s)		description
24-31		REGVAL0 - data of register address+0
16-23		REGVAL1 - data of register address+1
8-15		REGVAL2 - data of register address+2
0-7		REGVAL3 - data of register address+3

`0xCC006024`	4	r	DICFG - DI Configuration Register

31	24	23	16	15	8	7	0
....	....	....	....	....	....	....	....

bit(s)		description
8-31		reserved/unused
0-7		CONFIG - during reset this register latches DIDD bus (only bit 0 used)

5.7.1   Drive Commands

DICMDBUF0	DICMDBUF1	DICMDBUF2	DIMAR	DILENGTH	DIIMMBUF	DICR	Description
0x12000000	0x00000000	0x00000020	ret: Drive-Info	0x00000020	-	DMA read	Inquiry
0xa8000000	Data-Position>>2	Data-Length	ret: Sector-Data	Data-Length	-	DMA read	read Sector
0xa8000040	0x00000000	0x00000020	ret: Disc-ID	0x00000020	-	DMA read	read Disc ID/Init Drive
0xa8000080	?	?	?	?	?	?	?
0xa80000C0	?	?	?	?	?	?	?
0xab000000	Position>>2	-	-	-	-	imm (read)	seek
0xe0000000	-	-	-	-	ret: Error-Code	imm read	request error Status
0xe1??0000	Stream-Position>>2	Stream-Length	-	-	-	imm read	play Audio Stream (?)
0xe2??0000	-	-	-	-	ret: Status (?)	imm read	request Audio Status
0xe3000000	-	-	-	-	-	imm (read)	stop Motor
0xe4000000	-	-	-	-	-	imm (read)	DVD Audio disable
0xe4010000	-	-	-	-	-	imm (read)	DVD Audio enable

5.7.2 Drive Debug Commands

DICMDBUF0	DICMDBUF1	DICMDBUF2	DIMAR	DILENGTH	DIIMMBUF	DICR	Description
0xfe00????	?	?	?	?	?	?	?
0xfe010000	offset	0x00010000	-	-	ret: 32bit value	imm (read)	read memory
0xfe010100	offset	0x00010000	-	-	32bit value	imm (write)	write memory
0xfe018000	offset	0xff000000	-	-	ret: 32bit value	imm (read)	read cache
0xfe018100	offset	0xff000000	-	-	32bit value	imm (write)	write cache
0xfe100000	?	?	-	-	-	imm (read)	?
0xfe110000 (*)	-	-	-	-	-	imm (read)	stop drive
0xfe110100 (*)	-	-	-	-	-	imm (read)	start drive
0xfe114000 (*)	-	-	-	-	-	imm (read)	accept copy
0xfe118000 (*)	-	-	-	-	-	imm (read)	do disc-check
0xfe120000	24bit address	0x66756e63	-	-	-	imm (read)	jsr to address 'func'
0xff004456	0x442d4741	0x4d450300	-	-	-	imm (read)	unlock 2 'DVD-GAME'
0xff014d41	0x54534849	0x5441024f	-	-	-	imm (read)	unlock 1 'MATSHITA'

(*) commands can be ORed to perform several actions at once

5.7.3 Operation

5.7.3.1 Drive Info (Inquiry)

5.7.3.1.1 Structure of the Drive Info Data

start	end	size	Description
0x0000	0x0001	0x02	revision level
0x0002	0x0003	0x02	device code
0x0004	0x0007	0x04	release date
0x0008	0x001F	0x18	padding zeros

5.7.3.2 Read Disc ID / Init Disc

5.7.3.3 Read Sector

5.7.3.4 Seek

5.7.3.5 Request Error

31 24	23 16	15 8	7 0
aaaa aaaa	nnnn nnnn	nnnn nnnn	nnnn nnnn

5.7.3.5.1 Error Codes

bit(s)		description	
	a	0x00	ok
		0x01	lid open
		0x02	no disc/disc changed
		0x03	no disc
		0x04	motor off
		0x05	disc not initialized/disc id not read
	n	0x000000	ok
		0x020400	Motor stopped
		0x020401	Disk ID not read
		0x023A00	Medium not present / Cover opened
		0x030200	No Seek complete
		0x031100	UnRecoverd read error
		0x040800	Transfer protocol error
		0x052000	Invalid command operation code
		0x052001	Audio Buffer not set
		0x052100	Logical block address out of range
		0x052400	Invalid Field in command packet
		0x052401	Invalid audio command
		0x052402	Configuration out of permitted period
		0x056300	End of user area encountered on this track
		0x062800	Medium may have changed
		0x0B5A01	Operator medium removal request

5.7.3.6 Play Audio Stream

5.7.3.7   Request Audio Status

5.7.3.8   Stop Motor

5.7.3.9   DVD Audio Disable

5.7.3.10   DVD Audio Enable

- ▷ Command/Subcommand/Padding <- E4010000
- ▷ Action <- 1
- ▷ ACK (???)
- ▷ Status2 <- Status2
- ▷ INIT (???)
- ▷ Status1 <- 2Ah
- ▷ Status2 <- 0

5.7.3.11   Write Mem debug command   Note: This command is not really a single command but two commands in sequence.

This command writes 'length' bytes to the specified address 'address' in the DVD drive addressable memory. 'length' is specified in bytes and must be in the range 1 to 12. If more data needs to be written, several commands need to be issued. 'address' is a 24 bit value, but a 32 bit value can be safely used. 'length' is a 16 bit value.

- ▷ write the constant `0xfe010100` to DICMDBUF0 (`0xCC006008`)
- ▷ write 'address' at DICMDBUF1 (`0xCC00600C`)
- ▷ write 'length' to the upper 16 bits of DICMDBUF2 (`0xCC006010`) and clear the lower 16 bits.
- ▷ start immediate read transfer: write 1 (TSTART) to DICR (`0xCC00601C`)
- ▷ wait until transfer is complete, poll bit 0 (lowest order bit) of DICR (`0xCC00601C`), it will be cleared on end of transfer
- ▷ write the data (up to 'length' bytes) to `0xCC00600C` (write the first byte at `0xcc00600c`, the second at `0xcc00600d`, and so on)
- ▷ start immediate read transfer: write 1 (TSTART) to DICR (`0xCC00601C`)
- ▷ wait until transfer is complete, poll bit 0 (lowest order bit) of DICR (`0xCC00601C`), it will be cleared on end of transfer

5.7.4   DVD-ROM Subsystem

- ▷ Matsushita MN103S13BGA Optical Disk Controller
- ▷ Matsushita MN102H60GFA MicroComputer

5.7.4.1   Memory Map

start	end	size	description
0x00008000		4kb	internal (cpu) ram
0x00080000		128kb	firmware rom (*)
0x00400000			internal (controller) ram

(*) note: reading the firmware at its real location is prevented by the debug commands (imm buffer will not be changed at all). however you can read its contents from the memory mirrors, ie `0x000a0000`-.

5.8   SI - Serial Interface

Register block Base	Size of Register block	common access size
0xCC006400	0x100	4

0xCC006400	4	r/w	SIC0OUTBUF - SI Channel 0 Output Buffer (Joy-channel 1 Command)
0xCC00640c	4	r/w	SIC1OUTBUF - SI Channel 1 Output Buffer (Joy-channel 2 Command)
0xCC006418	4	r/w	SIC2OUTBUF - SI Channel 2 Output Buffer (Joy-channel 3 Command)
0xCC006424	4	r/w	SIC3OUTBUF - SI Channel 3 Output Buffer (Joy-channel 4 Command)

31 24	23 16	15 8	7 0

bit(s)		description
24-31		unused/reserved
16-23		CMD - (*1)
8-15		OUTPUT0 - (*2)
0-7		OUTPUT1 - (*3)

This register is double buffered, so main processor writes to the SIC0OUTBUF will not interfere with the serial interface output transfer. Internally, a second buffer is used to hold the output data to be transferred across the serial interface. To check if SIC0OUTBUF has been transferred to the second buffer, main processor polls the SISR[WRST0] register. When SICOOUTBUF is transferred, SISR[WRST0] is cleared.

(*1) This byte is the opcode for the command sent to the controller during each command/response packet. This is the first data byte sent from the SI I/F to the game controller in the command/response packet.
(*2) This is the first data byte of the command packet. It is the second data byte sent from the SI I/F to the game controller in the command/response packet.
(*3) This is the second data byte of the command packet. It is the third data byte sent from the SI I/F to the game controller in the command/response packet.

0xCC006404	4	r	Joy-channel 1 Buttons 1
0xCC006410	4	r	SIC1INBUFH - SI Channel 1 Input Buffer High (Joy-channel 2 Buttons 1)
0xCC00641c	4	r	Joy-channel 3 Buttons 1
0xCC006428	4	r	Joy-channel 4 Buttons 1

31 24	23 16	15 8	7 0
...s yxba	..LR udrl	xxxx xxxx	yyyy yyyy

bit(s)		Description	
31		ERRSTAT - Error Status (*1)	
		0	no error on last transfer
		1	error on last transfer
30		ERRLATCH - Error Latch (*2)	
		0	no error latched
		1	error latched (check SISR)
24-29		bit 0-5 of input byte 0 (bit 6 and 7 are assumed to be 0)	
16-23		input byte 1	
8-15		input byte 2	
0-7		input byte 3	

(*1) This bit represents the current error status for the last SI polling transfer on this channel. This register is updated after each polling transfer on this channel.

(*2) This bit is an error status summary of the SISR error bits for this channel. If an error has occurred on a past SI transfer (polling or Com transfer), this bit will be set. To determine the exact error, read the SISR register. This bit is actually an 'or' of the latched error status bits for this channel in the SISR. The bit is cleared by clearing the appropriate error status bits latched in the SISR. The no response error indicates that a controller is not present on thischannel.

0xCC006408	4	r/w	Joy-channel 1 Buttons 2
0xCC006414	4	r/w	Joy-channel 2 Buttons 2
0xCC006420	4	r/w	Joy-channel 3 Buttons 2
0xCC00642c	4	r	SIC3INBUFL - SI Channel 3 Input Buffer Low (Joy-channel 4 Buttons 2)

31 24	23 16	15 8	7 0
xxxx xxxx	yyyy yyyy	llll llll	rrrr rrrr

bit(s)		Description
24-31	x	input byte 4
16-23	y	input byte 5
8-15	l	input byte 6
0-7	r	input byte 7

SIC0INBUFH and SIC0INBUFL are double buffered to prevent inconsistent data reads due to main processor conflicting with incoming serial interface data. To insure data read from SIC0INBUFH and SIC0INFUBL are consistent, a locking mechanism prevents the double buffer from copying new data to these registers. Once SIC0INBUFH is read, both SIC0INBUFH and SIC0INBUFL are 'locked' until SIC0INBUFL is read. While the buffers are 'locked', new data is not copied into the buffers. When SIC0INBUFL is read, the buffers become unlocked again.

0xCC006430	4	r/w	SIPOLL - SI Poll Register (Joy-channel Control (?) (Calibration gun ?))

31 24	23 16	15 8	7 0
....	???? .???	.... ..?.	eeee ....

bit(s)		description		
26-31		unused/reserved		
16-25		X - 7 X lines register (*1)		
8-15		Y - y times register (*2)		
4-7	e	EN - controller port enable (1 bit per port, 1: enabled) (*3)		
0-3		VBCPY - Vblank copy output channel (1 bit per port) (*4)		
			0	copy SICOUTBUF to output buffer after writing
			1	copy SICOUTBUF to output buffer only on vblank

(*1) 7 X lines register: determines the number of horizontal video lines between polling (the polling interval). The polling begins at vsync. 7 is the minimum setting (determined by the time required to complete a single polling of the controller). The maximum setting depends on the current video mode (number of lines per vsync) and the SIPOLL[Y] register. This register takes affect after vsync.

(*2) This register determines the number of times the SI controllers are polled in a single frame. This register takes affect after vsync.

(*3) Enable polling of channel. When the channel is enabled, polling begins at the next vblank. When the channel is disabled, polling is stopped immediately after the current transaction. The status of this bit does not affect communication RAM transfers on this channel.

(*4) Normally main processor writes to the SIC0OUTBUF register are copied immediately to the channel 0 output buffer if a transfer is not currently in progress. When this bit is asserted, main processor writes to channel 0's SIC0OUTBUF will only be copied to the outbuffer on vblank. This is used to control the timing of commands to 3D LCD shutter glasses connected to the VI.

| `0xCC006434` | 4 | r/w | SICOMCSR - SI Communication Control Status Register (command) |

31 24	23 16	15 8	7 0
`r?..  ?ccs`	`.mmm  mmmm`	`.nnn  nnnn`	`eb..  ...?`

bit(s)		description			
31	r	TCINT - Transfer Complete Interrupt Status			
			read	0	transfer complete interrupt not requested
				1	transfer complete interrupt has been requested
			write	0	no effect
				1	clear transfer complete interrupt
30		TCINTMSK - Transfer Complete Interrupt Mask (*1)			
			0	interrupt masked	
			1	interrupt enabled	
29		COMERR - Communication Error			
			0	ok	
			1	error (see SiSr for the cause)	
28		RDSTINT - Read Status Interrupt Status (*2)			
			read	0	Transfer Complete Interrupt not requested
				1	Transfer Complete Interrupt has been requested
			write	0	
				1	
27		RDSTINTMSK - Read Status interrupt Mask (*3)			
			0	masked	
			1	enabled	
25-26	c	Channel Number (?)			
24	s	Channel Enable (?)			
23		unused/reserved			
16-22	m	OUTLNGTH - Communication Channel Output Length (*4)			
15		unused/reserved			
8-14	n	INLNGTH - Communication Channel Input Length (*4)			
7	e	Command Enable (?)			
6	b	callback enable			
			bit	Description	
			0	no callback	
			1	callback enabled	
1-2		CHANNEL - (*5)			
			00	Channel 1	
			01	Channel 2	
			10	Channel 3	
			11	Channel 4	
0		TSTART - Transfer Start (*6)			
			read	0	Command Complete
				1	Command Pending
			write	0	Do not start command
				1	Start command

(*1) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of SICOMCSR[TCINT]

(*2) On read this bit indicates the current status of the Read Status interrupt. The interrupt is set whenever SISR[RDSTn] bits are set. The interrupt is cleared when all of the RdSt bits in the SISR are cleared by

reading from the Si Channel Input Buffers. This interrupt can be used to indicate that a polling transfer has completed and new data is captured in the input registers

(*3) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of SICOMCSR[RDSTINT]

(*4) Minimum transfer is 1 byte. A value of 0 will transfer 128 bytes. These bits should not be modified while SICOM transfer is in progress.

(*5) These bits should not be modified while SICOM transfer is in progress.

(*6) When a '1' is written to this register, the current communication transfer is executed. The transfer begins immediately after the current transaction on this channel has completed. When read this bit represents the current transfer status. Once a communication transfer has been executed, polling will resume at the next vblank if the channel's SIPOLL[ENn] bit is set.

When programming the SICOMCSR after a SICOM transfers has already started (e.g., SICOMCSR[TSTART] is set), the software should read the current value first, then and/or in the proper data and then write the new data back. The software should not modify any of the transfer parameters (OUTLNGTH, INLNGTH, CHANNEL) until the current transfer is complete. This is done to prevent a SICOM transfer already in progress from being disturbed. When writing the data back, the software should not set the TSTART bit again unless the current transfer is complete and another transfer is required.

| 0xCC006438 | 4 | r/w | SISR - SI Status Register (channel select & status2) |

31 24	23 16	15 8	7 0
r??? aaaa	???? bbbb	???? cccc	???? dddd

bit(s)		description			
31	r	WR - Write SICnOUTBUF Register (*1)			
		read	0	buffer copied	
			1	buffer not copied	
		write	0	no effect	
			1	copy all buffers	
30		reserved/unused			
29		RDST0 - Read Status SIC0OINBUF Register (*2)			
		0	New data available, not read by main processor		
		1	No new data available, already read by main processor		
28		WRST0 - Write Status SIC0OUTBUF Register (*3)			
		0	Buffer copied		
		1	Buffer not copied		
27		NOREP0 - No Response Error Channel 0 (*4)			
		read	0	No Response Error not asserted	
			1	No Response Error asserted	
		write	0	No effect	
			1	Clear No Response Error	
26		COLL0 - Collision Error Channel 0 (*5)			
		read	0	Collision Error not asserted	
			1	Collision Error asserted	
		write	0	No effect	
			1	Clear Collision Error	
25		OVRUN0 - Over Run Error Channel 0 (*6)			
		read	0	Over Run Error not asserted	
			1	Over Run Error asserted	
		write	0	No effect	
			1	Clear Over Run Error	
24		UNRUN - Under Run Error Channel 0 (*7)			
		read	0	Under Run not asserted	
			1	Under Run asserted	
		write	0	No effect	
			1	Clear Under Run Error	
22-23		reserved/unused			
16-21	b	Joy-channel 1 bits			
14-15		reserved/unused			
8-13	c	Joy-channel 2 bits			
6-7		reserved/unused			
0-5	d	Joy-channel 3 bits			

(*1) Write SICnOUTBUF Register: This register controls and indicates whether the SICnOUTBUFs have been copied to the double buffered output buffers. This bit is cleared after the buffers have been copied.
(*2) This register indicates whether the SIC0INBUFs have been captured new data and whether the data has already been read by the main processor (read indicated by main processor read of SIC01NBUF[ERRSTAT, ERRLATCH, INPUT0, INPUT1)]
(*3) This register indicates whether the SIC0OUTBUFs have been copied to the double buffered output

buffers. This bit is cleared after the buffers have been copied.

(*4) This register indicates that a previous transfer resulted in no response from the controller. This can also be used to detect whether a controller is connected. If no controller is connected, this bit will be set. Once set this bit remains set until it is cleared by the main processor. To clear this bit write '1' to this register.

(*5) This register indicates data collision between controller and main unit. Once set this bit remains set until it is cleared by the main processor. To clear this bit write '1' to this register.

(*6) This register indicates that the main unit has received more data than expected. Once set this bit remains set until it is cleared by the main processor. To clear this bit write '1' to this register.

(*7) This register indicates that the main unit has received less data than expected. Once set this bit remain set until it is cleared by the main processor. To clear this bit write '1' to this register.

`0xcc00643c`	4	R/W	SIEXILK - SI EXI Clock Lock

31 24	23 16	15 8	7 0

bit(s)		description
31		LOCK - prevents CPU from setting EXI clock to 32MHz
	0	32MHz EXI clock setting permitted
	1	32MHz EXI clock setting not permitted
0-30		unused/reserved (always zero)

`0xCC006480`	0x80	r/w	SI i/o buffer (access by word)

5.8.1 Operation

5.8.1.1 Serial Send Buffer

▷ select channel: unset all bits in 0xcc006438 that are not corresponding to your channel and leave the others untouched.

▷ Put output data into SI buffer (128 bytes maximum), word by word.

▷ Send command: fill in 'c','m','n' and 'b' bits of 0xcc006434 accordingly, and set bits 's' and 'e'. leave other bits untouched.

5.8.1.2 Serial Get Result

▷ simply read the SI buffer

5.9   EXI - External Interface

Upper memory (0xCC000000 and above) can't keep enough data for extra-large arrays, it's limited up to 0xFFFF bytes (suppose to be). EXI was designed to remove this limitation. EXI is used for access to big, unmapped areas of HW memory (such as bootrom or SRAM). This is the main task of EXI. Put another way, EXI can be used for providing access to slow, serial devices, such as memory cards. EXI is a complex of different devices, mapped to a single bus. The EXI bus is divided on 3 channels. Each channel has 3 unique devices. Each device is defined by its ID, and has its own address space.

EXI can be accessed in immediate mode, or via DMA channel. Each EXI device can generate up to 3 interrupts. They are called EXI, TC and EXT :

EXI	Device EXI Interrupt
TC	Transfer Completed (any mode)
EXT	Device Attached / Device Detached

Each EXI channel have its own register set, 5 32bit Registers each.

Register block Base	Size of Register block	common access size
0xCC006800	0x40	4

0xCC006800	4		EXI0CSR - EXI Channel 0 Parameter Register (Status?)
0xCC006814	4		EXI1CSR - EXI Channel 1 Parameter Register
0xCC006828	4		EXI2CSR - EXI Channel 2 Parameter Register

31 24	23 16	15 8	7 0

bit(s)		Description
14-31		unused
13		ROMDIS - (EXI0 only) 1: rom de-scramble logic disabled (*1)
12	d	EXT - Device Connected Bit (R) 1 if a device is connected on the specific channel
11	x	EXTINT - External Insertion Interrupt Status (R) : check to poll EXT interrupt (or to detect device detach) (

		read	0	External Insertion Interrupt has not been requested
			1	External Insertion Interrupt has been requested
		write	0	No effect
			1	Clear External Insertion Interrupt

10	m	EXTINTMASK - EXT Interrupt Mask (1 - enable, 0 - disable) (*5)
7-9	210	CS - devices selected on this channel, each bit selecting one device. (*)
4-6	f	CLK - used frequency (0-5)

		000	1MHz
		001	2MHz
		010	4MHz
		011	8MHz
		100	16MHz
		101	32MHz
		110	reserved
		111	reserved

3	t	TCINT - Transfer Complete Interrupt Status

		read	0	Transfer Complete Interrupt has not been requested
			1	Transfer Complete Interrupt has been requested
		write	0	No effect
			1	Clear Transfer Complete Interrupt

2	m	TCINTMASK - Transfer complete interrupt mask (1 - enable, 0 - disable) (*2)
1	e	EXTINT - Interrupt Status (*6)

		read	0	EXI Interrupt has not been requested
			1	EXI Interrupt has been requested
		write	0	No effect
			1	Clear EXI Interrupt

0	m	EXTINTMASK - EXI interrupt mask (1 - enable, 0 - disable)

(*)Only one of these three bits can be set to signify which device number has been selected on a specific channel.

(*6) This bit indicates the current status of the EXI0 interrupt. The interrupt is cleared by accessing the expansion device and clearing the interrupt on the device itself and cleared locally when a '1' is written to this register. This interrupt input is edge triggered.

(*1) This bit disables access to the IPL Mask ROM attached to CS1. Once this bit is enabled, it can only be disabled again by global reset. The ROM de-scramble logic will become disabled and any reads to the memory mapped ROM area will return all 0.When de-scrambler is enabled all EXI0 data will be de-scrambled, so only the IPL ROM may be accessed through EXI0 until ROMDIS is set to '1'. (this is usually done by the Bootstrap, see Boot process details)

(*2) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of TCINT

(*3) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of EXIINT

(*5) Interrupt masking prevents the interrupt from being sent to the main processor, but does not affect the assertion of EXICPR[EXTINT]

(*4) This interrupt indicates than an external EXI device has been removed from channel 1. To check whether the device has been inserted or removed, check the EXICPR[EXT] bit. When this bit is set, the channel's expansion EXI interface outputs go to high.

0xCC006804	4	r/w	EXI0MAR - EXI Channel 0 DMA Start Address
0xCC006818	4	r/w	EXI1MAR - EXI Channel 1 DMA Start Address
0xCC00682c	4	r/w	EXI2MAR - EXI Channel 2 DMA Start Address

```
31       24 23        16 15        8 7          0
....  ..dd  dddd  dddd  dddd  dddd  ddd.  ....
```

Physical Startaddress for DMA transfer. Must be aligned to 32 byte boundary .

(*) The memory address is the destination address when EXICR[RW] is set to 'read' and is the source address when set to 'write'.

0xCC006808	4	r/w	EXI0LENGTH - EXI Channel 0 DMA Transfer Length
0xCC00681c	4		EXI Channel 1 DMA Transfer Length
0xCC006830	4		EXI Channel 2 DMA Transfer Length

```
31       24 23        16 15        8 7          0
....  ..dd  dddd  dddd  dddd  dddd  ddd.  ....
```

Size of DMA transfer data in bytes. bits 0-4 are always zero (which means the size is 32 byte aligned)

0xCC00680c	4	r/w	EXI0CR - EXI Channel 0 Control Register
0xCC006820	4	r/w	EXI1CR - EXI Channel 1 Control Register
0xCC006834	4	r/w	EXI2CR - EXI Channel 2 Control Register

```
31       24 23        16 15        8 7          0
....  ....  ....  ....  ....  ....  ..ll  ttme
```

bit(s)		Description
6-31	.	unused
4-5	l	TLEN - (data length-1) for immediate mode
		<table><tr><td>00</td><td>1 byte</td></tr><tr><td>01</td><td>2 bytes</td></tr><tr><td>10</td><td>3 bytes</td></tr><tr><td>11</td><td>4 bytes</td></tr></table>
2-3	t	RW - transfer type
		<table><tr><td>00</td><td>read</td></tr><tr><td>01</td><td>write</td></tr><tr><td>10</td><td>read and write, invalid for DMA</td></tr><tr><td>11</td><td>undefined</td></tr></table>
1	m	DMA - transfer mode (0 - immediate, 1 - DMA)
0	e	TSTART - set, to start transfer. will be cleared after transfer completed.

`0xCC006810`	4	r/w	EXI0DATA - EXI Channel 0 Immediate Data
`0xCC006824`	4	r/w	EXI1DATA - EXI Channel 1 Immediate Data
`0xCC006838`	4	r/w	EXI2DATA - EXI Channel 2 Immediate Data

31 24	23 16	15 8	7 0
dddd dddd	dddd dddd	dddd dddd	dddd dddd

Data for read / write immediate operations (up to 4 bytes long).

The EXICPR must be configured to assert one of the devices CS, before the read or write operation can be performed. The actual read/write operation is triggered by the EXI0CR[TSTART] register and EXI0CR[DMA] set to '0'. Data is sent with MSB (bit 31) first.

5.9.1 Operation

5.9.1.1 Initializing the EXI Bus

  ▷ clear all EXIs status registers (set to 0)

  ▷ eventually initialize interrupt callbacks.

If you want to use DMA with EXI, you need your own properly installed EXI interrupt handlers. There is no need in callbacks and interrupts, if you are using EXI in immediate mode (just mask all TCs, to prevent unhandled interrupts).

5.9.1.2 Selecting a Specific EXI Device on an EXI Channel

  ▷ set selected device number and frequency ('210' and f fields in status register)

  ▷ eventually enable any existing callbacks

5.9.1.3 Deselecting EXI Devices on an EXI Channel

  ▷ deselect device (clear corresponding '210' field)

  ▷ eventually disable any existing callbacks

5.9.1.4 Performing an IMM Operation on a EXI Device

5.9.1.4.1 IMM Read

  ▷ setup EXI Control Register (bit4-5:data length-1, bit2: 0 for read, bit1: 0 for immediate, bit1: 1 to start transfer)

  ▷ wait until transfer has been completed (until bit 0 in EXI Control Register has been cleared)

  ▷ get data from EXI IMM Data Register (up to 4 bytes)

5.9.1.4.2 IMM Write

▷ write data to EXI IMM Data Register (up to 4 bytes)

▷ setup EXI Control Register (bit4-5:data length-1, bit2: 1 for write, bit1: 0 for immediate, bit1: 1 to start transfer)

▷ wait until transfer has been completed (until bit 0 in EXI Control Register has been cleared)

5.9.1.5 Performing a DMA Operation on a EXI Device

5.9.1.5.1 DMA Read

▷ setup DMA buffer address

▷ setup DMA length

▷ setup EXI Control Register (bit4-5:data length-1, bit2: 0 for read, bit1: 1 for DMA, bit1: 1 to start transfer)

▷ wait until transfer has been completed (until bit 0 in EXI Control Register has been cleared)

5.9.1.5.2 DMA Write

▷ setup DMA buffer address

▷ setup DMA length

▷ setup EXI Control Register (bit4-5:data length-1, bit2: 1 for write, bit1: 1 for DMA, bit1: 1 to start transfer)

▷ wait until transfer has been completed (until bit 0 in EXI Control Register has been cleared)

5.9.1.6 Wait for EXI transfer completed To detect the end of a transfer on a specific channel either setup a 'transfer completed' callback (only works with DMA transfer) or periodically check bit 0 of the EXI Control Register (until cleared).

5.10 AI - Audio Streaming Interface

Registerblock Base	Size of Registerblock	common access size
0xcc006c00	0x20	4

`0xCC006C00`	r/w	4	AICR - Audio Interface Control Register

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
7-31		reserved/unused
6		DSP Sample Rate
		0	48 kHz sample rate
		1	32 kHz sample rate
5		SCRESET Sample Counter Reset: When a '1' is written to this bit the AISLRCNT register is reset to 0
4		AIINTVLD Audio Interface Interrupt Valid.
		This bit controls whether AIINT is affected by the AIIT register matching AISLRCNT. Once set, AIINT will ho
		0	Match affects AIINT
		1	AIINT hold last value.
3		AIINT Audio Interface Interrupt Status and clear. (*3)
		r	0	Audio Interface Interrupt has not been requested
			1	Audio Interface Interrupt has been requested.
		w	0	No effect
			1	Clear Audio Interface interrupt
2		AIINTMSK Audio interface Interrupt Mask
		0	interrupt masked
		1	Interrupt enabled
1		AFR: Auxiliary Frequency Register (*1)
		0	48 kHz sample rate
		1	32 kHz sample rate
0		PSTAT: Playing Status
		0	Stop or Pause streaming audio (AISLR clock disabled)
		1	Play streaming audio (AISLR clock enabled)

(*3)

On read this bit indicates the current status of the audio interface interrupt. When a '1' is written to this register, the interrupt is cleared. This interrupt indicates that the AIIT register matches the AISLRCNT. This bit asserts regardless of the setting of AICR[AIMSK].

(*1)

Controls the sample rate of the streaming audio data. When set to 32 kHz sample rate, the SRC will convert the streaming audio data to 48 kHz. This bit should only be changed when Streaming Audio is stopped (AICR[PSTAT] set to 0).

(*0)

This bit enables the AISLR clock which controls the playing/stopping of audio streaming. When this bit is 1 AISLRCNT register will increment for every stereo pair of samples output.

`0xCC006C04`	r/w	4	AIVR - Audio Interface Volume Register

31 24	23 16	15 8	7 0
....	....	rrrr rrrr	llll llll

bit(s)		description
16-31		unused/reserved
8-15	r	AVRR - Volume Right Channel (0x00 is muted,0xff is max)
0-7	l	AVRL - Volume Right Channel (0x00 is muted,0xff is max)

`0xCC006C08`	r	4	AISCNT - Audio Interface Sample Counter

Audio interface Sample Counter: This register counts the number of AIS stereo samples that have been output. It is enabled by AICR[PSTAT]. It can be cleared by the AICR[SCRESET] register.

`0xCC006C0C`	r/w	4	AIIT - Audio Interface Interrupt Timing

This register indicates the stereo sample count to issue an audio interface interrupt to the main processor. The interrupt is issued when the value of the AISLRCNT register matches the content of this register.

5.11   GX FIFO (Graphic display lists)

GP have mapped 32-byte FIFO buffer, at 0xCC008000, which is controlled by write gather pipe (WPAR). when FIFO is filled (or overloaded by 32-bytes), WPAR performs burst transaction of primitive data to GP command FIFO. WPAR API also keeps watching for wrapping it on 32-buffer. You can think, that data is always looped and flows like in circle.

Registerblock Base	Size of Registerblock	common access size
0xcc008000	4	any

To access FIFO, you should just write data of any size to 0xCC008000, WPAR will control circularity and gathering automatically. By "data of any size" are assumed command types, vertices, vertex attributes etc stuff. All commands and primitive data are sending through mapped GP FIFO. GP task is only to draw primitives in embedded frame buffer, and then send it to XFB, for VI rendering. All render rules are stored in VI. GP can only change some copy rules, using pixel engine setup.

GP primitives also can be drawn, using Display List. In that case, GP FIFO takes only "CALL_DL" command with pointer to list data, and then GP command FIFO sequentially parsing primitive data from the main memory. Primitives can contains both direct and indexed vertexes as well. In first case, vertex attributes are sent directly using GP FIFO, in the other case the CPU sends only the pointer to vertex attribute data which is located in main memory.

5.11.1   internal BP registers

Registerblock Base	Size of Registerblock	common access size
0x00	0x100	4 (1+3)

Register	Description
`0x00`	GEN_MODE
`0x01`	display copy filter
`0x02`	display copy filter
`0x03`	display copy filter
`0x04`	display copy filter
`0x05`	?
`0x06`	IND_MTXA0
`0x07`	IND_MTXB0
`0x08`	IND_MTXC0
`0x09`	IND_MTXA1
`0x0a`	IND_MTXB1
`0x0b`	IND_MTXC1
`0x0c`	IND_MTXA2
`0x0d`	IND_MTXB2
`0x0e`	IND_MTXC2
`0x0f`	IND_IMASK
`0x10`	IND_CMD0 - tev indirect 0
`0x11`	IND_CMD1 - tev indirect 1
`0x12`	IND_CMD2 - tev indirect 2
`0x13`	IND_CMD3 - tev indirect 3
`0x14`	IND_CMD4 - tev indirect 4
`0x15`	IND_CMD5 - tev indirect 5
`0x16`	IND_CMD6 - tev indirect 6
`0x17`	IND_CMD7 - tev indirect 7
`0x18`	IND_CMD8 - tev indirect 8
`0x19`	IND_CMD9 - tev indirect 9
`0x1a`	IND_CMDA - tev indirect 10
`0x1b`	IND_CMDB - tev indirect 11
`0x1c`	IND_CMDC - tev indirect 12
`0x1d`	IND_CMDD - tev indirect 13
`0x1e`	IND_CMDE - tev indirect 14
`0x1f`	IND_CMDF - tev indirect 15
`0x20`	scissor x0,y0 (0x20156156)
`0x21`	scissor x1,y1 (0x213d5335)
`0x22`	SU_LPSIZE - field mode .. line width - point width
`0x23`	SU Counter (?) (0x23000000)
`0x24`	RAS Counter (?) (0x24000000)
`0x25`	RAS1_SS0 - ind tex coord scale 0
`0x26`	RAS1_SS1 - ind tex coord scale 1
`0x27`	RAS1_IREF
`0x28`	RAS1_TREF0 - tev order 0
`0x29`	RAS1_TREF1 - tev order 1
`0x2a`	RAS1_TREF2 - tev order 2
`0x2b`	RAS1_TREF3 - tev order 3
`0x2c`	RAS1_TREF4 - tev order 4
`0x2d`	RAS1_TREF5 - tev order 5
`0x2e`	RAS1_TREF6 - tev order 6
`0x2f`	RAS1_TREF7 - tev order 7

Register	Description
0x30	SU_SSIZE0 - texture offset 0 (Texture Size X, Y ?)
0x31	SU_TSIZE0 -
0x32	SU_SSIZE1 - texture offset 1
0x33	SU_TSIZE1 -
0x34	SU_SSIZE2 - texture offset 2
0x35	SU_TSIZE2 -
0x36	SU_SSIZE3 - texture offset 3
0x37	SU_TSIZE3 -
0x38	SU_SSIZE4 - texture offset 4
0x39	SU_TSIZE4 -
0x3a	SU_SSIZE5 - texture offset 5
0x3b	SU_TSIZE5 -
0x3c	SU_SSIZE6 - texture offset 6
0x3d	SU_TSIZE6 -
0x3e	SU_SSIZE7 - texture offset 7
0x3f	SU_TSIZE7 -
0x40	PE_ZMODE set z mode
0x41	PE_CMODE0 dithering / blend mode/color_update/alpha_update/set_dither
0x42	PE_CMODE1 destination alpha
0x43	PE_CONTROL comp z location z_comp_loc(0x43000040)pixel_fmt(0x43000041)
0x44	field mask (0x44000003)
0x45	PE_DONE - draw done (end of list marker) ?
0x46	some clock ? (0x46000000|(((162000000/500)/4224)|0x0200))
0x47	PE_TOKEN token B (16 bit)
0x48	PE_TOKEN_INT token A (16 bit)
0x49	EFB source rectangle top left
0x4a	EFB source rectangle width, height-1
0x4b	XFB target address
0x4c	?
0x4d	stride ?
0x4e	DispCopyYScale
0x4f	PE copy clear AR - set clear alpha and red components
0x50	PE copy clear GB - green and blue
0x51	PE copy clear Z - 24-bit Z value
0x52	pe copy execute?
0x53	copy filter
0x54	copy filter
0x55	bounding box (0x550003ff)
0x56	bounding box (0x560003ff)
0x57	?
0x58	? (0x5800000f)
0x59	scissor-box offset (0x5902acab)
0x5a	?
0x5b	?
0x5c	?
0x5d	?
0x5e	?
0x5f	?

Register	Description
0x60	?
0x61	?
0x62	?
0x63	?
0x64	TX_LOADTLUT0
0x65	TX_LOADTLUT1
0x66	?
0x67	metric ? (0x67000000)
0x68	field mode
0x69	some clock ? (0x69000000|(((((162000000/500)>>11)&0x00ffffff))|0x0400)
0x6a	?
0x6b	?
0x6c	?
0x6d	?
0x6e	?
0x6f	?
0x70	?
0x71	?
0x72	?
0x73	?
0x74	?
0x75	?
0x76	?
0x77	?
0x78	?
0x79	?
0x7a	?
0x7b	?
0x7c	?
0x7d	?
0x7e	?
0x7f	?
0x80	TX_SETMODE0_I0 - 0x90 for linear
0x81	TX_SETMODE0_I1
0x82	TX_SETMODE0_I2
0x83	TX_SETMODE0_I3
0x84	TX_SETMODE1_I0
0x85	TX_SETMODE1_I1
0x86	TX_SETMODE1_I2
0x87	TX_SETMODE1_I3
0x88	TX_SETIMAGE0_I0 - texture size ?
0x89	TX_SETIMAGE0_I1
0x8a	TX_SETIMAGE0_I2
0x8b	TX_SETIMAGE0_I3
0x8c	TX_SETIMAGE1_I0
0x8d	TX_SETIMAGE1_I1
0x8e	TX_SETIMAGE1_I2
0x8f	TX_SETIMAGE1_I3

Register	Description
0x90	TX_SETIMAGE2_I0
0x91	TX_SETIMAGE2_I1
0x92	TX_SETIMAGE2_I2
0x93	TX_SETIMAGE2_I3
0x94	TX_SETIMAGE3_I0 - Texture Pointer
0x95	TX_SETIMAGE3_I1
0x96	TX_SETIMAGE3_I2
0x97	TX_SETIMAGE3_I3
0x98	TX_LOADTLUT0
0x99	TX_LOADTLUT1
0x9a	TX_LOADTLUT2
0x9b	TX_LOADTLUT3
0x9c	?
0x9d	?
0x9e	?
0x9f	?
0xa0	TX_SETMODE0_I4
0xa1	TX_SETMODE0_I5
0xa2	TX_SETMODE0_I6
0xa3	TX_SETMODE0_I7
0xa4	TX_SETMODE1_I4
0xa5	TX_SETMODE1_I5
0xa6	TX_SETMODE1_I6
0xa7	TX_SETMODE1_I7
0xa8	TX_SETIMAGE0_I4
0xa9	TX_SETIMAGE0_I5
0xaa	TX_SETIMAGE0_I6
0xab	TX_SETIMAGE0_I7
0xac	TX_SETIMAGE1_I4
0xad	TX_SETIMAGE1_I5
0xae	TX_SETIMAGE1_I6
0xaf	TX_SETIMAGE1_I7
0xb0	TX_SETIMAGE2_I4
0xb1	TX_SETIMAGE2_I5
0xb2	TX_SETIMAGE2_I6
0xb3	TX_SETIMAGE2_I7
0xb4	TX_SETIMAGE3_I4
0xb5	TX_SETIMAGE3_I5
0xb6	TX_SETIMAGE3_I6
0xb7	TX_SETIMAGE3_I7
0xb8	TX_SETTLUT_I4
0xb9	TX_SETTLUT_I5
0xba	TX_SETTLUT_I6
0xbb	TX_SETTLUT_I7
0xbc	?
0xbd	?
0xbe	?
0xbf	?

Register	Description
0xc0	TEV_COLOR_ENV_0 - tev op 0
0xc1	TEV_ALPHA_ENV_0 - tev op 1
0xc2	TEV_COLOR_ENV_1 -
0xc3	TEV_ALPHA_ENV_1
0xc4	TEV_COLOR_ENV_2 -
0xc5	TEV_ALPHA_ENV_2
0xc6	TEV_COLOR_ENV_3 -
0xc7	TEV_ALPHA_ENV_3
0xc8	TEV_COLOR_ENV_4 -
0xc9	TEV_ALPHA_ENV_4
0xca	TEV_COLOR_ENV_5 -
0xcb	TEV_ALPHA_ENV_5
0xcc	TEV_COLOR_ENV_6 -
0xcd	TEV_ALPHA_ENV_6
0xce	TEV_COLOR_ENV_7 -
0xcf	TEV_ALPHA_ENV_7
0xd0	TEV_COLOR_ENV_8 -
0xd1	TEV_ALPHA_ENV_8
0xd2	TEV_COLOR_ENV_9 -
0xd3	TEV_ALPHA_ENV_9
0xd4	TEV_COLOR_ENV_A -
0xd5	TEV_ALPHA_ENV_A
0xd6	TEV_COLOR_ENV_B -
0xd7	TEV_ALPHA_ENV_B
0xd8	TEV_COLOR_ENV_C -
0xd9	TEV_ALPHA_ENV_C
0xda	TEV_COLOR_ENV_D -
0xdb	TEV_ALPHA_ENV_D
0xdc	TEV_COLOR_ENV_E -
0xdd	TEV_ALPHA_ENV_E
0xde	TEV_COLOR_ENV_F -
0xdf	TEV_ALPHA_ENV_F
0xe0	TEV_REGISTERL_0
0xe1	TEV_REGISTERH_0
0xe2	TEV_REGISTERL_1
0xe3	TEV_REGISTERH_1
0xe4	TEV_REGISTERL_2
0xe5	TEV_REGISTERH_2
0xe6	TEV_REGISTERL_3
0xe7	TEV_REGISTERH_3
0xe8	Fog Range (0xe8000156)
0xe9	?
0xea	?
0xeb	?
0xec	? (guessed: tev_range_adj_c)
0xed	? (guessed: tev_range_adj_k)
0xee	TEV_FOG_PARAM_0 (0xee03ce38)
0xef	TEV_FOG_PARAM_1 (0xef471c82)

Register	Description
0xf0	TEV_FOG_PARAM_2 (0xf0000002)
0xf1	TEV_FOG_PARAM_3 (0xf1000000)
0xf2	TEV_FOG_COLOR (0xf2000000)
0xf3	TEV_ALPHAFUNC - alpha compare (0xf33f0000)
0xf4	TEV_Z_ENV_0 - z texture 0
0xf5	TEV_Z_ENV_1 - z texture 1
0xf6	TEV_KSEL_0 - Tev Swap Mode Table 0 (0xf6018064)
0xf7	TEV_KSEL_1 - Tev Swap Mode Table 1 (0xf701806e)
0xf8	TEV_KSEL_2 - Tev Swap Mode Table 2 (0xf8018060)
0xf9	TEV_KSEL_3 - Tev Swap Mode Table 3 (0xf901806c)
0xfa	TEV_KSEL_4 - Tev Swap Mode Table 4 (0xfa018065)
0xfb	TEV_KSEL_5 - Tev Swap Mode Table 5 (0xfb01806d)
0xfc	TEV_KSEL_6 - Tev Swap Mode Table 6 (0xfc01806a)
0xfd	TEV_KSEL_7 - Tev Swap Mode Table 7 (0xfd01806e)
0xfe	SS_MASK - BP Mask Register
0xff	?

0x00	4	w	GEN_MODE

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
19		ZFREEZE		
16		NBMP - Number of Bumpmaps		
14-15		REJECT_EN - Culling Mode		
		0	none	
		1	negative	
		2	positive	
		3	all	
10		NTEV		
9		MS_EN		
4		NCOL - Number of Colors		
0		NTEX - Number of Texture Coords		

0x01	4	w	display copy filter

0x02	4	w	display copy filter

0x03	4	w	display copy filter

0x04	4	w	display copy filter

0x05	4	w	?

`0x06`	4	w	IND_MTXA0
`0x09`	4	w	IND_MTXA1
`0x0c`	4	w	IND_MTXA2

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
22		S
11		MB
0		MA

`0x07`	4	w	IND_MTXB0
`0x0a`	4	w	IND_MTXB1
`0x0d`	4	w	IND_MTXB2

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
22		S
11		MD
0		MC

`0x08`	4	w	IND_MTXC0
`0x0b`	4	w	IND_MTXC1
`0x0e`	4	w	IND_MTXC2

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
22		S
11		MF
0		ME

`0x0f`	4	w	IND_IMASK

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
0		IMASK

`0x10`	4	w	IND_CMD0

0x11	4	w	IND_CMD1
0x12	4	w	IND_CMD2
0x13	4	w	IND_CMD3
0x14	4	w	IND_CMD4
0x15	4	w	IND_CMD5
0x16	4	w	IND_CMD6
0x17	4	w	IND_CMD7
0x18	4	w	IND_CMD8
0x19	4	w	IND_CMD9
0x1a	4	w	IND_CMDA
0x1b	4	w	IND_CMDB
0x1c	4	w	IND_CMDC
0x1d	4	w	IND_CMDD
0x1e	4	w	IND_CMDE
0x1f	4	w	IND_CMDF

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
21-23		PAD0 - padding zeros		
20		FB - addprev		
19		LB - utclod		
16-18		TW - Wrap T		
		0	ITW_OFF	
		1	ITW_256	
		2	ITW_128	
		3	ITW_64	
		4	ITW_32	
		5	ITW_16	
		6	ITW_0	
		7		
13-15		SW - Wrap S		
		0	ITW_OFF	
		1	ITW_256	
		2	ITW_128	
		3	ITW_64	
		4	ITW_32	
		5	ITW_16	
		6	ITW_0	
		7		
9-12		M - Matrix ID		
		0	ITM_OFF	
		1	ITM_0	
		2	ITM_1	
		3	ITM_2	
		5	ITM_S0	
		6	ITM_S1	
		7	ITM_S2	
		9	ITM_T0	
		10	ITM_T1	
		11	ITM_T2	
7-8		BS - Alpha Selection		
		0	ITBA_OFF	
		1	ITBA_S	
		2	ITBA_T	
		3	ITBA_U	
4-6		BIAS		
		0	ITB_NONE	
		1	ITB_S	
		2	ITB_T	
		3	ITB_ST	
		4	ITB_U	
		5	ITB_SU	
		6	ITB_TU	
		7	ITB_STU	
2-3		FMT - Format		
		0	ITF_8	
		1	ITF_5	
		2	ITF_4	
		3	ITF_3	
0-1		BT - Indirect Tex Stage ID (0-3)		

0x20	4	w	SU_SCIS0 - Scissorbox Top Left Corner

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
12		X0 - Scissorbox X0 offset + 342
0		Y0 - Scissorbox Y0 offset + 342

0x21	4	w	SU_SCIS1 - Scissorbox Bottom Right Corner

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
12		X1 - Scissorbox X1 offset + 342
0		Y1 - Scissorbox Y1 offset + 342

0x22	4	w	SU_LPSIZE

31	24	23	16	15	8	7	0

bit(s)		description		
24		RID		
23		PAD0		
22		LINEASPECT		
19		PTOFF		
			0	to 0
			1	to 16th
			2	to 8th
			3	to 4th
			4	to half
			5	to 1
16		LTOFF		
			0	to 0
			1	to 16th
			2	to 8th
			3	to 4th
			4	to half
			5	to 1
8		PSIZE		
0		LSIZE		

0x23	4	w	SU Counter ?

`0x24`	4	w	RAS Counter ?

`0x25`	4	w	RAS1_SS - ind tex coord scale 0

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
12		TS1 - Ind. Tex Stage 1
8		SS1 - Ind. Tex Stage 1
4		TS0 - Ind. Tex Stage 0
0		SS0 - Ind. Tex Stage 0

`0x26`	4	w	RAS1_SS - ind tex coord scale 1

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
12		TS1 - Ind. Tex Stage 3
8		SS1 - Ind. Tex Stage 3
4		TS0 - Ind. Tex Stage 2
0		SS0 - Ind. Tex Stage 2

`0x27`	4	w	RAS1_IREF

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
21		BC3 - Ind. Tex Stage 3 NTexCoord
18		BI3 - Ind. Tex Stage 3 NTexMap
15		BC2 - Ind. Tex Stage 2 NTexCoord
12		BI2 - Ind. Tex Stage 2 NTexMap
9		BC1 - Ind. Tex Stage 1 NTexCoord
6		BI1 - Ind. Tex Stage 1 NTexMap
3		BC0 - Ind. Tex Stage 0 NTexCoord
0		BI0 - Ind. Tex Stage 0 NTexMap

`0x28`	4	w	RAS1_TREF0
`0x29`	4	w	RAS1_TREF1
`0x2a`	4	w	RAS1_TREF2
`0x2b`	4	w	RAS1_TREF3
`0x2c`	4	w	RAS1_TREF4
`0x2d`	4	w	RAS1_TREF5
`0x2e`	4	w	RAS1_TREF6
`0x2f`	4	w	RAS1_TREF7

`31    24`	`23    16`	`15    8`	`7    0`

bit(s)		description		
24		RID		
22		PAD1		
19-21		CC1 - Ind. Tex Stage 1 Channel ID		
		0	Color0	
		1	Color1	
		2	Alpha0	
		3	Alpha1	
		4	Color0A0	
		5	Color1A1	
		6	ColorZero	
		7	Bump	
18		TE1 - Ind. Tex Stage 1 TexMap enable		
15		TC1 - Ind. Tex Stage 1 TexCoord		
12		TI1 - Ind. Tex Stage 1 TexMap		
10		PAD0		
7		CC0 - Ind. Tex Stage 0 Color ID		
6		TE0 - Ind. Tex Stage 0 TexMap enable		
3		TC0 - Ind. Tex Stage 0 TexCoord		
0		TI0 - Ind. Tex Stage 0 TexMap		

0x30	4	w	SU_SSIZE0
0x32	4	w	SU_SSIZE1
0x34	4	w	SU_SSIZE2
0x36	4	w	SU_SSIZE3
0x38	4	w	SU_SSIZE4
0x3a	4	w	SU_SSIZE5
0x3c	4	w	SU_SSIZE6
0x3e	4	w	SU_SSIZE7

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
19		PF - texcoord offset for points enable
18		LF - texcoord offset for lines enable
17		WS - s-cylindrical texcoord wrapping enable
16		BS - s-range bias enable
0		SSIZE - s-scale value -1 (U16)

0x31	4	w	SU_TSIZE0
0x33	4	w	SU_TSIZE1
0x35	4	w	SU_TSIZE2
0x37	4	w	SU_TSIZE3
0x39	4	w	SU_TSIZE4
0x3b	4	w	SU_TSIZE5
0x3d	4	w	SU_TSIZE6
0x3f	4	w	SU_TSIZE7

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
17		WT - t-cylindrical texcoord wrapping enable
16		BT - t-range bias enable
0		TSIZE - t-scale value -1 (U16)

0x40	4	w	PE_ZMODE

31	24	23	16	15	8	7	0

bit(s)		description		
24		RID		
4		MASK - Update enable		
1		FUNC - Z-Buffer Compare Function		
		0	NEVER	
		1	LESS	
		2	EQUAL	
		3	LEQUAL	
		4	GREATER	
		5	NEQUAL	
		6	GEQUAL	
		7	ALWAYS	
0		ENABLE - Z-Buffer enable		

0x41	4	w	PE_CMODE0

31	24	23	16	15	8	7	0

bit(s)		description		
24		RID		
12		LOGICOP		
		0	CLEAR	
		1	AND	
		2	REVAND	
		3	COPY	
		4	INVAND	
		5	NOOP	
		6	XOR	
		7	OR	
		8	NOR	
		9	EQUIV	
		10	INV	
		11	REVOR	
		12	INVCOPY	
		13	INVOR	
		14	NAND	
		15	SET	
11		BLENDOP		
8		SFACTOR		
5		DFACTOR		
4		ALPHA_MASK		
3		COLOR_MASK		
2		DITHER_ENABLE		
1		LOGICOP_ENABLE		
0		BLEND_ENABLE		

`0x42`	4	w	PE_CMODE1

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
8		CONSTANT_ALPHA_ENABLE
0		CONSTANT_ALPHA

`0x43`	4	w	PE_CONTROL

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
7-23		unused ?		
6		Z Comp Loc (1: before tex)		
3-5		Z Format		
			0	linear
			1	near
			2	mid
			3	far
0-2		Pixel Format		
			0	RGB8_ Z24
			1	RGBA6_Z24
			2	RGB565_Z16
			3	Z24
			4	Y8
			5	U8
			6	V8
			7	YUV420

`0x44`	4	w	field mask ?

31 24	23 16	15 8	7 0

bit(s)		description
24		RID

0x45	4	w	PE_DONE - draw done

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
1		1=end of list
0		?

0x46	4	w	? (some clock?)

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
9		? (must be 1)
0		((162000000/500)/4224)

0x47	4	w	PE_TOKEN

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
0		Token

0x48	4	w	PE_TOKEN_INT

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
0		

0x49	4	w	EFB Address Top Left

31	24	23	16	15	8	7	0

bit(s)		description
10		Y coordinate
0		X coordinate

| `0x4a` | 4 | w | EFB Address Width, Height-1 |

31 24	23 16	15 8	7 0

bit(s)		description
10		Height-1
0		Width

| `0x4b` | 4 | w | XFB Address |

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
0		physical XFB Address >> 5

| `0x4c` | 4 | w | ? |

| `0x4d` | 4 | w | stride ? |

| `0x4e` | 4 | w | DispCopyYScale |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
24		RID
0		YSCALE - ((u32)(256.0/YSCALEIN))&0x1ff

| `0x4f` | 4 | w | PE copy clear AR - set clear alpha and red components |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
24		RID
8		A
0		R

| `0x50` | 4 | w | PE copy clear GB - green and blue |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
24		RID
8		G
0		B

| 0x51 | 4 | w | PE copy clear Z - 24-bit Z value |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
24		RID
0-23		Z - 24bit Z-Value

| 0x52 | 4 | w | pe copy execute? |

31 24	23 16	15 8	7 0
....	....	....	....

bit(s)		Description
24		RID
14		execute ? (1: to XFB 0: to texture ?!)
12-13		Frame 2 Field Mode
11		clear (1: clear EFB)
10		1: (256-(u32)(256.0/YSCALEIN)) > 0
9		?
7-8		disp copy gamma
4		target (XFB) pixel format
1		clamp
0		clamp

| 0x53 | 4 | w | copy filter |

| 0x54 | 4 | w | copy filter |

| 0x55 | 4 | w | bounding box |

| 0x56 | 4 | w | bounding box |

| 0x57 | 4 | w | ? |

| 0x58 | 4 | w | ? |

| 0x59 | 4 | w | Scissorbox Offset |

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
10		YO - ((Scissorbox Y offset + 342)>>1)
0		XO - ((Scissorbox X offset + 342)>>1)

note: regs `0x5a-0x63` are left out (all unknown)

| 0x64 | 4 | w | TX_LOADTLUT0 |

31 24	23 16	15 8	7 0

bit(s)		description
24-		rid
21-		pad0
0-		tlut base

| 0x65 | 4 | w | TX_LOADTLUT1 |

31 24	23 16	15 8	7 0

bit(s)		description
24-		rid
21-		pad0
10-		count
0		tmem offset

| 0x66 | 4 | w | |

| 0x67 | 4 | w | metric ? |

| 0x68 | 4 | w | field mode |

| 0x69 | 4 | w | ? |

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
10		? (must be 1)
0		((162000000/500)>>11)

note: regs `0x6a-0x7f` are left out (all unknown)

`0x80`	4	w	TX_SETMODE0_I0 - Texture lookup and filtering mode
`0x81`	4	w	TX_SETMODE0_I1 - Texture lookup and filtering mode
`0x82`	4	w	TX_SETMODE0_I2 - Texture lookup and filtering mode
`0x83`	4	w	TX_SETMODE0_I3 - Texture lookup and filtering mode
`0xa0`	4	w	TX_SETMODE0_I4 - Texture lookup and filtering mode
`0xa1`	4	w	TX_SETMODE0_I5 - Texture lookup and filtering mode
`0xa2`	4	w	TX_SETMODE0_I6 - Texture lookup and filtering mode
`0xa3`	4	w	TX_SETMODE0_I7 - Texture lookup and filtering mode

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
21		LODCLAMP / BIASCLAMP		
		0	off	
		1	on	
19		MAXANISO		
		0	1	
		1	2 (requires edge LOD)	
		2	4 (requires edge LOD)	
		3	unused/reserved	
9		LODBIAS (s2.5)		
8		DIAGLOAD		
		0	edge LOD	
		1	diagonal LOD	
5		MIN FILTER		
		0	near	
		1	near mip near	
		2	near mip lin	
		3	unused/reserved	
		4	linear	
		5	lin mip near	
		6	lin mip lin	
		7	unused/reserved	
4		MAG FILTER		
		0	near	
		1	linear	
2		WRAP T		
		0	clamp	
		1	repeat (*)	
		2	mirror (*)	
		3	unused/reserved	
0		WRAP S (same as WRAP T)		

(*) requires the texture size to be a power of two. (wrapping is implemented by a logical AND (SIZE-1))

0x84	4	w	TX_SETMODE1_I0 - LOD Info
0x85	4	w	TX_SETMODE1_I1 - LOD Info
0x86	4	w	TX_SETMODE1_I2 - LOD Info
0x87	4	w	TX_SETMODE1_I3 - LOD Info
0xa4	4	w	TX_SETMODE1_I4 - LOD Info
0xa5	4	w	TX_SETMODE1_I5 - LOD Info
0xa6	4	w	TX_SETMODE1_I6 - LOD Info
0xa7	4	w	TX_SETMODE1_I7 - LOD Info

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
8		MAX LOD (U4.4)
0		MIN LOD (U4.4)

0x88	4	w	SETIMAGE0_I0 - Texture width, height, format
0x89	4	w	SETIMAGE0_I1 - Texture width, height, format
0x8a	4	w	SETIMAGE0_I2 - Texture width, height, format
0x8b	4	w	SETIMAGE0_I3 - Texture width, height, format
0xa8	4	w	SETIMAGE0_I4 - Texture width, height, format
0xa9	4	w	SETIMAGE0_I5 - Texture width, height, format
0xaa	4	w	SETIMAGE0_I6 - Texture width, height, format
0xab	4	w	SETIMAGE0_I7 - Texture width, height, format

31 24	23 16	15 8	7 0

bit(s)		description	
24-		rid	
20-		format	
		0	I4
		1	I8
		2	IA4
		3	IA8
		4	RGB565
		5	RGB5A3
		6	RGBA8
		7	unused/reserved
		8	C4
		9	C8
		10	C14X2
		11	unused/reserved
		12	unused/reserved
		13	unused/reserved
		14	CMP
		15	unused/reserved
10-		height - 1	
0-		width - 1	

`0x8c`	4	w	TX_SETIMAGE1_I0 - even LOD address in TMEM
`0x8d`	4	w	TX_SETIMAGE1_I1 - even LOD address in TMEM
`0x8e`	4	w	TX_SETIMAGE1_I2 - even LOD address in TMEM
`0x8f`	4	w	TX_SETIMAGE1_I3 - even LOD address in TMEM
`0xac`	4	w	TX_SETIMAGE1_I4 - even LOD address in TMEM
`0xad`	4	w	TX_SETIMAGE1_I5 - even LOD address in TMEM
`0xae`	4	w	TX_SETIMAGE1_I6 - even LOD address in TMEM
`0xaf`	4	w	TX_SETIMAGE1_I7 - even LOD address in TMEM

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
21		IMAGE_TYPE		
		0	cached	
		1	preloaded	
18		CACHE_HEIGHT		
		0	unused/reserved	
		1	unused/reserved	
		2	unused/reserved	
		3	32kb	
		4	128kb	
		5	512kb	
		6	unused/reserved	
		7	unused/reserved	
15		CACHE_WIDTH (must be equal to CACHE_HEIGHT)		
0		TMEM_OFFSET (address in TMEM >> 5)		

`0x90`	4	w	TX_SETIMAGE2_I0 - odd LOD address in TMEM
`0x91`	4	w	TX_SETIMAGE2_I1 - odd LOD address in TMEM
`0x92`	4	w	TX_SETIMAGE2_I2 - odd LOD address in TMEM
`0x93`	4	w	TX_SETIMAGE2_I3 - odd LOD address in TMEM
`0xb0`	4	w	TX_SETIMAGE2_I4 - odd LOD address in TMEM
`0xb1`	4	w	TX_SETIMAGE2_I5 - odd LOD address in TMEM
`0xb2`	4	w	TX_SETIMAGE2_I6 - odd LOD address in TMEM
`0xb3`	4	w	TX_SETIMAGE2_I7 - odd LOD address in TMEM

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
18		CACHE_HEIGHT		
		0	none (if odd LOD is unused)	
		1	unused/reserved	
		2	unused/reserved	
		3	32kb	
		4	128kb	
		5	512kb	
		6	unused/reserved	
		7	unused/reserved	
15		CACHE_WIDTH (must be equal to CACHE_HEIGTH)		
0		TMEM_OFFSET - (address in TMEM >> 5)		

`0x94`	4	w	TX_SETIMAGE3_I0 - Address of Texture in main memory
`0x95`	4	w	TX_SETIMAGE3_I1 - Address of Texture in main memory
`0x96`	4	w	TX_SETIMAGE3_I2 - Address of Texture in main memory
`0x97`	4	w	TX_SETIMAGE3_I3 - Address of Texture in main memory
`0xb4`	4	w	TX_SETIMAGE3_I4 - Address of Texture in main memory
`0xb5`	4	w	TX_SETIMAGE3_I5 - Address of Texture in main memory
`0xb6`	4	w	TX_SETIMAGE3_I6 - Address of Texture in main memory
`0xb7`	4	w	TX_SETIMAGE3_I7 - Address of Texture in main memory

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
0		IMAGE_BASE (physical address >> 5)

`0x98`	4	w	TX_SETTLUT_0
`0x99`	4	w	TX_SETTLUT_1
`0x9a`	4	w	TX_SETTLUT_2
`0x9b`	4	w	TX_SETTLUT_3
`0xb8`	4	w	TX_SETTLUT_4
`0xb9`	4	w	TX_SETTLUT_5
`0xba`	4	w	TX_SETTLUT_6
`0xbb`	4	w	TX_SETTLUT_7

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
10		FORMAT		
			0	IA8
			1	RGB565
			2	RGB5A3
			3	reserved/unused
0		TMEM_OFFSET (offset of TLUT from start of TMEM high bank >> 5)		

`0x9c`	4	w	?

`0x9d`	4	w	?

`0x9e`	4	w	?

`0x9f`	4	w	?

`0xbc`	4	w	?

`0xbd`	4	w	?

`0xbe`	4	w	?

`0xbf`	4	w	?

0xc0	4	w	TEV_COLOR_ENV_0
0xc2	4	w	TEV_COLOR_ENV_1
0xc4	4	w	TEV_COLOR_ENV_2
0xc6	4	w	TEV_COLOR_ENV_3
0xc8	4	w	TEV_COLOR_ENV_4
0xca	4	w	TEV_COLOR_ENV_5
0xcc	4	w	TEV_COLOR_ENV_6
0xce	4	w	TEV_COLOR_ENV_7
0xd0	4	w	TEV_COLOR_ENV_8
0xd2	4	w	TEV_COLOR_ENV_9
0xd4	4	w	TEV_COLOR_ENV_A
0xd6	4	w	TEV_COLOR_ENV_B
0xd8	4	w	TEV_COLOR_ENV_C
0xda	4	w	TEV_COLOR_ENV_D
0xdc	4	w	TEV_COLOR_ENV_E
0xde	4	w	TEV_COLOR_ENV_F

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
22		DEST
20		SHIFT
19		CLAMP
18		SUB
16		BIAS
12		SELA
8		SELB
4		SELC
0		SELD

SELA - SELD Format:

0x0	CC_CPREV
0x1	CC_APREV
0x2	CC_C0
0x3	CC_A0
0x4	CC_C1
0x5	CC_A1
0x6	CC_C2
0x7	CC_A2
0x8	CC_TEXC
0x9	CC_TEXA
0xA	CC_RASC
0xB	CC_RASA
0xC	CC_ONE
0xD	CC_HALF
0xE	CC_KONST
0xF	CC_ZERO

`0xc1`	4	w	TEV_ALPHA_ENV_0
`0xc3`	4	w	TEV_ALPHA_ENV_1
`0xc5`	4	w	TEV_ALPHA_ENV_2
`0xc7`	4	w	TEV_ALPHA_ENV_3
`0xc9`	4	w	TEV_ALPHA_ENV_4
`0xcb`	4	w	TEV_ALPHA_ENV_5
`0xcd`	4	w	TEV_ALPHA_ENV_6
`0xcf`	4	w	TEV_ALPHA_ENV_7
`0xd1`	4	w	TEV_ALPHA_ENV_8
`0xd3`	4	w	TEV_ALPHA_ENV_9
`0xd5`	4	w	TEV_ALPHA_ENV_A
`0xd7`	4	w	TEV_ALPHA_ENV_B
`0xd9`	4	w	TEV_ALPHA_ENV_C
`0xdb`	4	w	TEV_ALPHA_ENV_D
`0xdd`	4	w	TEV_ALPHA_ENV_E
`0xdf`	4	w	TEV_ALPHA_ENV_F

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
22		DEST
20		SHIFT
19		CLAMP
18		SUB
16		BIAS
13		SELA
10		SELB
7		SELC
4		SELD
2		TSWAP
0		RSWAP

SELA - SELD Format:

0	CA_APREV
1	CA_A0
2	CA_A1
3	CA_A2
4	CA_TEXA
5	CA_RASA
6	CA_KONST
7	CA_ZERO

`0xe0`	4	w	TEV_REGISTERL_0
`0xe2`	4	w	TEV_REGISTERL_1
`0xe4`	4	w	TEV_REGISTERL_2
`0xe6`	4	w	TEV_REGISTERL_3

31	24	23	16	15	8	7	0

bit(s)		description	
24		RID	
23		TYPE	
		0	Color (?)
		1	Constant (?)
12		A	
0		R	

`0xe1`	4	w	TEV_REGISTERH_0
`0xe3`	4	w	TEV_REGISTERH_1
`0xe5`	4	w	TEV_REGISTERH_2
`0xe7`	4	w	TEV_REGISTERH_3

31	24	23	16	15	8	7	0

bit(s)		description	
24		RID	
23		TYPE	
		0	Color (?)
		1	Constant (?)
12		G	
0		B	

`0x88`	4	w	Fog Range

`0x89`	4	w	

`0x8A`	4	w	

`0x8B`	4	w	

`0xec (guessed)`	4	w	tev_range_adj_c

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
10		CENTER - Screen X Center for range Adjustment
0		ENB - Range-Adjustment enable

		0	TEV_ENB_DISABLE
		1	TEV_ENB_ENABLE

`0xed (guessed)`	4	w	tev_range_adj_k

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
0-11		r2k (u4.8) - specifies the range adjustment function

range adjustment = `sqr((x*x)+(k*k))/k`

`0xee`	4	w	TEV_FOG_PARAM_0 - "a" parameter of the screen to eye space conversion function

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
19		A_SIGN_SHIFT
11		A_EXPN
0		A_MANT (signed 11e8)

`0xef`	4	w	TEV_FOG_PARAM_1 - the "b" parameter of the z screen to eye space conversion function

31	24	23	16	15	8	7	0

bit(s)		description
24		RID
0		B_MAG (unsigned 0.24)

0xf0	4	w	TEV_FOG_PARAM_2 - amount to pre-shift screen z

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
0-4		B_SHF - equivalent to the value of "b" parameter's exponent + 1

The Z-Screen to Eyespace conversion is defined as:

```
Ze = A / (B_MAG - (Zs >> B_SHF))
```

0xf1	4	w	TEV_FOG_PARAM_3 - fog type

31 24	23 16	15 8	7 0

bit(s)		description		
24		RID		
21-23		FSEL		
		0	FSEL_OFF; No fog	
		1	reserved	
		2	FSEL_LIN; linear Fog	
		3	reserved	
		4	FSEL_EXP; Exponential Fog	
		5	FSEL_EX2; Exponential Squared Fog	
		6	FSEL_BXP; Backward Exp Fog	
		7	FSEL_BX2 Backward Exp Squared Fog	
20		PROJ		
		0	PERSP; Perspective projection	
		1	ORTHO; Orthographic projection	
19		C_SIGN (*)		
11		C_EXPN (*)		
0-10		C_MANT (*)		

(*) Specifies the amount to subtract from eye-space Z after range adjustment.

0xf2	4	w	TEV_FOG_COLOR - Value of Fog Color

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
16		R
8		G
0		B

0xf3	4	w	TEV_ALPHAFUNC

31	24	23	16	15	8	7	0

bit(s)		description	
24		RID	
22		LOGIC	
		0	AND
		1	OR
		2	XOR
		3	XNOR
19		OP1	
16		OP0	
8		A1	
0		A0	

0xf4	4	w	TEV_Z_ENV_0

31	24	23	16	15	8	7	0

bit(s)		description
24		RID -
0-23		ZOFF/BIAS -

0xf5	4	w	TEV_Z_ENV_1

31	24	23	16	15	8	7	0

bit(s)		description	
24		RID	
2-3		OP	
		0	disable
		1	add
		2	replace
		3	unused/reserved
0-1		TYPE/FORMAT	
		0	u8
		1	u16
		2	u24
		3	unused/reserved

0xf6	4	w	TEV_KSEL_0
0xf7	4	w	TEV_KSEL_1
0xf8	4	w	TEV_KSEL_2
0xf9	4	w	TEV_KSEL_3
0xfa	4	w	TEV_KSEL_4
0xfb	4	w	TEV_KSEL_5
0xfc	4	w	TEV_KSEL_6
0xfd	4	w	TEV_KSEL_7

31 24	23 16	15 8	7 0

bit(s)		description
24		RID
19		KASEL1
14		KCSEL1
9		KASEL0
4		KCSEL0
2		XGA
0		XRB

KCSEL - tev const color sel

0	1
1	7_8
2	3_4
3	5_8
4	1_2
5	3_8
6	1_4
7	1_8
8	
9	
10	
11	
12	K0
13	K1
14	K2
15	K3
16	K0_R
17	K1_R
18	K2_R
19	K3_R
20	K0_G
21	K1_G
22	K2_G
23	K3_G
24	K0_B
25	K1_B
26	K2_B
27	K3_B
28	K0_A
29	K1_A
30	K2_A
31	K3_A

KASEL - tev const alpha sel

0	1
1	7_8
2	3_4
3	5_8
4	1_2
5	3_8
6	1_4
7	1_8
8	
9	
10	
11	
12	
13	
14	
15	
16	K0_R
17	K1_R
18	K2_R
19	K3_R
20	K0_G
21	K1_G
22	K2_G
23	K3_G
24	K0_B
25	K1_B
26	K2_B
27	K3_B
28	K0_A
29	K1_A
30	K2_A
31	K3_A

`0xfe`	4	w	SS_MASK - BP Mask Register

31 24	23 16	15 8	7 0
**** ****			

bit(s)		description
24	*	RID
0-23		MASK (*)

(*) This Register can be used to limit to which bits of BP registers is actually written to. the mask is only valid for the next BP command, and will reset itself.

`0xff`	4	w	?

5.11.2 internal CP Registers

Registerblock Base	Size of Registerblock	common access size
0x20	0xa0	4

Register		description
0x20		?
0x30		MATINDEX_A - Texture Matrix Index 0-3
0x40		MATINDEX_B - Texture Matrix Index 4-7
0x50		VCD_LO - Vertex Descriptor (VCD) low, format 0
0x51		VCD_LO - Vertex Descriptor (VCD) low, format 1
0x52		VCD_LO - Vertex Descriptor (VCD) low, format 2
0x53		VCD_LO - Vertex Descriptor (VCD) low, format 3
0x54		VCD_LO - Vertex Descriptor (VCD) low, format 4
0x55		VCD_LO - Vertex Descriptor (VCD) low, format 5
0x56		VCD_LO - Vertex Descriptor (VCD) low, format 6
0x57		VCD_LO - Vertex Descriptor (VCD) low, format 7
0x60		VCD_HI - Vertex Descriptor (VCD) high, format 0
0x61		VCD_HI - Vertex Descriptor (VCD) high, format 1
0x62		VCD_HI - Vertex Descriptor (VCD) high, format 2
0x63		VCD_HI - Vertex Descriptor (VCD) high, format 3
0x64		VCD_HI - Vertex Descriptor (VCD) high, format 4
0x65		VCD_HI - Vertex Descriptor (VCD) high, format 5
0x66		VCD_HI - Vertex Descriptor (VCD) high, format 6
0x67		VCD_HI - Vertex Descriptor (VCD) high, format 7
0x70		VAT_A - Vertex Attribute Table (VAT) group 0, format 0
0x71		VAT_A - Vertex Attribute Table (VAT) group 0, format 1
0x72		VAT_A - Vertex Attribute Table (VAT) group 0, format 2
0x73		VAT_A - Vertex Attribute Table (VAT) group 0, format 3
0x74		VAT_A - Vertex Attribute Table (VAT) group 0, format 4
0x75		VAT_A - Vertex Attribute Table (VAT) group 0, format 5
0x76		VAT_A - Vertex Attribute Table (VAT) group 0, format 6
0x77		VAT_A - Vertex Attribute Table (VAT) group 0, format 7
0x80		VAT_B - Vertex Attribute Table (VAT) group 1, format 0
0x81		VAT_B - Vertex Attribute Table (VAT) group 1, format 1
0x82		VAT_B - Vertex Attribute Table (VAT) group 1, format 2
0x83		VAT_B - Vertex Attribute Table (VAT) group 1, format 3
0x84		VAT_B - Vertex Attribute Table (VAT) group 1, format 4
0x85		VAT_B - Vertex Attribute Table (VAT) group 1, format 5
0x86		VAT_B - Vertex Attribute Table (VAT) group 1, format 6
0x87		VAT_B - Vertex Attribute Table (VAT) group 1, format 7
0x90		VAT_C - Vertex Attribute Table (VAT) group 2, format 0
0x91		VAT_C - Vertex Attribute Table (VAT) group 2, format 1
0x92		VAT_C - Vertex Attribute Table (VAT) group 2, format 2
0x93		VAT_C - Vertex Attribute Table (VAT) group 2, format 3
0x94		VAT_C - Vertex Attribute Table (VAT) group 2, format 4
0x95		VAT_C - Vertex Attribute Table (VAT) group 2, format 5
0x96		VAT_C - Vertex Attribute Table (VAT) group 2, format 6
0x97		VAT_C - Vertex Attribute Table (VAT) group 2, format 7

Register	description
0xA0	ARRAY_BASE - vertices ptr
0xa1	ARRAY_BASE - normals ptr
0xa2	ARRAY_BASE - color 0 ptr
0xa3	ARRAY_BASE - color 1 ptr
0xa4	ARRAY_BASE - texture 0 coordinate ptr
0xa5	ARRAY_BASE - texture 1 coordinate ptr
0xa6	ARRAY_BASE - texture 2 coordinate ptr
0xa7	ARRAY_BASE - texture 3 coordinate ptr
0xa8	ARRAY_BASE - texture 4 coordinate ptr
0xa9	ARRAY_BASE - texture 5 coordinate ptr
0xaa	ARRAY_BASE - texture 6 coordinate ptr
0xab	ARRAY_BASE - texture 7 coordinate ptr
0xac	ARRAY_BASE - IndexRegA - general purpose array 0 ptr
0xad	ARRAY_BASE - IndexRegB - general purpose array 1 ptr
0xae	ARRAY_BASE - IndexRegC - general purpose array 2 ptr
0xaf	ARRAY_BASE - IndexRegD - general purpose array 3 ptr
0xB0	ARRAY_STRIDE - size of vertices
0xb1	ARRAY_STRIDE - size of normals
0xb2	ARRAY_STRIDE - size of colors 0
0xb3	ARRAY_STRIDE - size of colors 1
0xb4	ARRAY_STRIDE - size of texture 0 coordinates
0xb5	ARRAY_STRIDE - size of texture 1 coordinates
0xb6	ARRAY_STRIDE - size of texture 2 coordinates
0xb7	ARRAY_STRIDE - size of texture 3 coordinates
0xb8	ARRAY_STRIDE - size of texture 4 coordinates
0xb9	ARRAY_STRIDE - size of texture 5 coordinates
0xba	ARRAY_STRIDE - size of texture 6 coordinates
0xbb	ARRAY_STRIDE - size of texture 7 coordinates
0xbc	ARRAY_STRIDE - IndexRegA - general purpose array 0 stride
0xbd	ARRAY_STRIDE - IndexRegB - general purpose array 1 stride
0xbe	ARRAY_STRIDE - IndexRegC - general purpose array 2 stride
0xbf	ARRAY_STRIDE - IndexRegD - general purpose array 3 stride

0x20	4	w	?

0x30	4	w	MATIDX_REG_A

31 24	23 16	15 8	7 0

bit(s)	description
24	TEX3IDX - Index for Texture 3 matrix
18	TEX2IDX - Index for Texture 2 matrix
12	TEX1IDX - Index for Texture 1 matrix
6	TEX0IDX - Index for Texture 0 matrix
0	POSIDX - Index for Position/Normal matrix

0x40	4	w	MATIDX_REG_B

31 24	23 16	15 8	7 0

bit(s)		description
18		TEX7IDX - Index for Texture 7 matrix
12		TEX6IDX - Index for Texture 6 matrix
6		TEX5IDX - Index for Texture 5 matrix
0		TEX4IDX - Index for Texture 4 matrix

0x50	4	R/W	VCD_LO - Vertex Descriptor low Format 0
0x51	4	R/W	VCD_LO - Vertex Descriptor low Format 1
0x52	4	R/W	VCD_LO - Vertex Descriptor low Format 2
0x53	4	R/W	VCD_LO - Vertex Descriptor low Format 3
0x54	4	R/W	VCD_LO - Vertex Descriptor low Format 4
0x55	4	R/W	VCD_LO - Vertex Descriptor low Format 5
0x56	4	R/W	VCD_LO - Vertex Descriptor low Format 6
0x57	4	R/W	VCD_LO - Vertex Descriptor low Format 7

31 24	23 16	15 8	7 0

bit(s)		description
17-31		unused
15-16		COL1 - Color1 (Specular)
13-14		COL0 - Color0 (Diffused)
11-12		NRM - Normal or Normal/Binormal/Tangent
9-10		POS - Position
8		T7MIDX
7		T6MIDX
6		T5MIDX
5		T4MIDX
4		T3MIDX
3		T2MIDX
2		T1MIDX
1		T0MIDX - Texture Coordinate 0 Matrix Index
0		PMIDX - Position/Normal Matrix Index (*1)

```
(*1) position and normal matrices are stored in 2 seperate areas of internal XF memory,
but there is a one to one correspondence between normal and position index.If index 'A'
is used for the position, then index 'A' needs to be used for the normal as well.
```

`0x60`	4	R/W	VCD_HI - Vertex Descriptor high Format 0
`0x61`	4	R/W	VCD_HI - Vertex Descriptor high Format 1
`0x62`	4	R/W	VCD_HI - Vertex Descriptor high Format 2
`0x63`	4	R/W	VCD_HI - Vertex Descriptor high Format 3
`0x64`	4	R/W	VCD_HI - Vertex Descriptor high Format 4
`0x65`	4	R/W	VCD_HI - Vertex Descriptor high Format 5
`0x66`	4	R/W	VCD_HI - Vertex Descriptor high Format 6
`0x67`	4	R/W	VCD_HI - Vertex Descriptor high Format 7

31 24	23 16	15 8	7 0
			`..tt`

bit(s)		description
16-		unused
14-15		`TEX7` - texture coordinate 7
12-13		`TEX6` - texture coordinate 6
10-11		`TEX5` - texture coordinate 5
8-9		`TEX4` - texture coordinate 4
6-7		`TEX3` - texture coordinate 3
4-5		`TEX2` - texture coordinate 2
2-3		`TEX1` - texture coordinate 1
0-1	t	`TEX0` - texture coordinate 0

vertex descriptor data

value	Vertex/Color	Pos/Tex Matrix Index
0	no data present	no data present
1	direct	direct
2	i8 - indirect/8 bit index	n/a
3	i16 - indirect/16 bit index	n/a

0x70	4	w	CP_VAT_REG_A – Format 0
0x71	4	w	CP_VAT_REG_A – Format 1
0x72	4	w	CP_VAT_REG_A – Format 2
0x73	4	w	CP_VAT_REG_A – Format 3
0x74	4	w	CP_VAT_REG_A – Format 4
0x75	4	w	CP_VAT_REG_A – Format 5
0x76	4	w	CP_VAT_REG_A – Format 6
0x77	4	w	CP_VAT_REG_A – Format 7

31 24	23 16	15 8	7 0

bit(s)	description		
31	NORMALINDEX3 (*1)		
		0	single index per normal
		1	triple-index per nine-normal
30	BYTEDEQUANT (should always be 1)		
		0	shift does not apply to u8/s8 components
		1	shift applies to u8/s8 components
25	TEX0SHFT		
22	TEX0FMT		
21	TEX0CNT		
18	COL1FMT (Specular)		
17	COL1CNT (Specular)		
14	COL0FMT (Diffused)		
13	COL0CNT (Diffused)		
10	NRMFMT		
9	NRMCNT		
4	POSSHFT		
1	POSFMT		
0	POSCNT		

(*1) when nine-normals are selected in indirect mode, input will be treated as three
staggered indices (one per triple biased by components size), into normal table (note:
first index internally biased by 0, second by 1, third by 2)

0x80	4	w	CP_VAT_REG_B - Format 0
0x81	4	w	CP_VAT_REG_B - Format 1
0x82	4	w	CP_VAT_REG_B - Format 2
0x83	4	w	CP_VAT_REG_B - Format 3
0x84	4	w	CP_VAT_REG_B - Format 4
0x85	4	w	CP_VAT_REG_B - Format 5
0x86	4	w	CP_VAT_REG_B - Format 6
0x87	4	w	CP_VAT_REG_B - Format 7

31 24	23 16	15 8	7 0

bit(s)		description
31		VCACHE_ENHANCE (must always be 1)
28		TEX4FMT
27		TEX4CNT
22		TEX3SHFT
19		TEX3FMT
18		TEX3CNT
13		TEX2SHFT
10		TEX2FMT
9		TEX2CNT
4		TEX1SHFT
1		TEX1FMT
0		TEX1CNT

0x90	4	w	CP_VAT_REG_C - Format 0
0x91	4	w	CP_VAT_REG_C - Format 1
0x92	4	w	CP_VAT_REG_C - Format 2
0x93	4	w	CP_VAT_REG_C - Format 3
0x94	4	w	CP_VAT_REG_C - Format 4
0x95	4	w	CP_VAT_REG_C - Format 5
0x96	4	w	CP_VAT_REG_C - Format 6
0x97	4	w	CP_VAT_REG_C - Format 7

31 24	23 16	15 8	7 0

bit(s)		description
27		TEX7SHFT
24		TEX7FMT
23		TEX7CNT
18		TEX6SHFT
15		TEX6FMT
14		TEX6CNT
9		TEX5SHFT
6		TEX5FMT
5		TEX5CNT
0		TEX4SHFT

Vertex Attribute Data Formats

`CompCount`

value	coords	normals	tex coords	colors
0	two (x,y)	three	one (s)	three (r,g,b)
1	three (x,y,z)	nine	two (s,t)	four (r,g,b,a)

`CompSize`

value	coords	normals	colors
0	u8	n/a	16 bit rgb565
1	s8	s8	24 bit rgb888
2	u16	n/a	32 bit rgb888x
3	s16	s16	16 bit rgba4444
4	f32	f32	24 bit rgba6666
5	n/a	n/a	32 bit rgba8888
6	unused	unused	unused
7	unused	unused	unused

`Shift`

coords	normals	colors
location of decimal point	n/a (byte: 6, short: 14)	n/a

`This shift applies to all s16/u16 components, and all s8/s8 components when ByteDequant is asserted.`

0xA0	4	w	ARRAY_BASE
0xA1	4	w	ARRAY_BASE
0xA2	4	w	ARRAY_BASE
0xA3	4	w	ARRAY_BASE
0xA4	4	w	ARRAY_BASE
0xA5	4	w	ARRAY_BASE
0xA6	4	w	ARRAY_BASE
0xA7	4	w	ARRAY_BASE
0xA8	4	w	ARRAY_BASE
0xA9	4	w	ARRAY_BASE
0xAA	4	w	ARRAY_BASE
0xAB	4	w	ARRAY_BASE
0xAC	4	w	ARRAY_BASE
0xAD	4	w	ARRAY_BASE
0xAE	4	w	ARRAY_BASE
0xAF	4	w	ARRAY_BASE

31 24	23 16	15 8	7 0

bit(s)		description
26-		unused
0-25		array base addres in main memory

0xB0	4	w	ARRAY_STRIDE
0xB1	4	w	ARRAY_STRIDE
0xB2	4	w	ARRAY_STRIDE
0xB3	4	w	ARRAY_STRIDE
0xB4	4	w	ARRAY_STRIDE
0xB5	4	w	ARRAY_STRIDE
0xB6	4	w	ARRAY_STRIDE
0xB7	4	w	ARRAY_STRIDE
0xB8	4	w	ARRAY_STRIDE
0xB9	4	w	ARRAY_STRIDE
0xBa	4	w	ARRAY_STRIDE
0xBb	4	w	ARRAY_STRIDE
0xBc	4	w	ARRAY_STRIDE
0xBd	4	w	ARRAY_STRIDE
0xBe	4	w	ARRAY_STRIDE
0xBf	4	w	ARRAY_STRIDE

31 24	23 16	15 8	7 0

bit(s)		description
8-		unused
0-7		array stride

5.11.3   internal XF Memory

Every register in the transform unit is mapped to a unique 32b address. All addresses are available to the xform register load command (command 0x30).

The first block is formed by the matrix memory. Its address range is 0 to 1 k, but only 256 entries are used. This memory is organized in a 64 entry by four 32b words. Each word has a unique address and is a single precision floating point number. For block writes, the addresses auto increment. The memory is implemented in less than 4-32b rams, then it is possible that the memory writes to this block will require a minimum write size larger than 1 word.

start	end	size	description
0x0000		32	Matrix Ram word 0
0x0001	0x00ff		Matrix Ram word (n)
0x0100	0x03ff	0x300	not used

```
0 - position matrix (4*3)
0xF0 - (texture?)  transform matrix (4*3)
```

The second block of memory is the normal matrix memory. It is organized as 32 rows of 3 words. Each word has a unique address and is a single precision floating point number. Also, each word written is 32b, but only the 20 most significant bits are kept. For simplicity, the minimum granularity of writes will be 3 words:

start	end	size	description
0x0400	0x0402	20	Normal Ram words 0,1,2
0x0403	0x045f		Normal Ram word (n)
0x0460	0x05ff		not used

```
0x400 - normal transform matrix (3*3)
```

The third block of memory holds the dual texture transform matrices. The format is identical to the first block of matrix memory. There are also 64 rows of 4 words for these matrices. These matrices can only be used for the dual transform of regular textures:

start	end	size	description
0x0500		32	Matrix Ram word 0
0x0501	0x05ff		Matrix Ram word (n)

```
0x5F4 - dual texture transform matrix (4*3)
```

The fourth block of memory is the light memory. This holds all the lighting information (light vectors, light parameters, etc.). Both global state and ambient state are stored in this memory. Each word written is 32b, but only the 20 most significant bits are kept. Each row is 3 words wide. Minimum word write size is 3 words.

start	end	size	description
0x0600			reserved
0x0601			reserved
0x0602			reserved
0x0603		32 bit	Light0 - RGBA
0x0604		20 bit	Light0A0 - cos atten. A-0
0x0605		20 bit	Light0A1 - cos atten. A-1
0x0606		20 bit	Light0A2 - cos atten. A-2
0x0607		20 bit	Light0K0 - dist atten. A-0
0x0608		20 bit	Light0K1 - dist atten. A-1
0x0609		20 bit	Light0K2 - dist atten. A-2
0x060a		20 bit	Light0Lpx - x light pos, or inf ldir x
0x060b		20 bit	Light0Lpy - y light pos, or inf ldir y
0x060c		20 bit	Light0Lpz - z light pos, or inf ldir z
0x060d		20 bit	Light0Dx/Hx - light dir x, or 1/2 angle x
0x060e		20 bit	Light0Dy/Hy - light dir y, or 1/2 angle y
0x060f		20 bit	Light0Dz/Hz - light dir z, or 1/2 angle z
0x0610	0x067f		Light(n)data - see Light0 data
0x0680	0x07ff		not used

5.11.4   internal XF Registers

Registerblock Base	Size of Registerblock	common access size
0x1000	0x54	4

Register	description
0x1000	Error (=0x3f)
0x1001	Diagnostics
0x1002	State0 - Internal State Register 0
0x1003	State1 - Internal State Register 1
0x1004	Xf_clock - Enables Power Saving Mode
0x1005	ClipDisable - clip mode (=0)
0x1006	Perf0 - Performance monitor selects (=0)
0x1007	Perf1 - Xform target performance register
0x1008	InVertexSpec - INVTXSPEC - (=0x01)
0x1009	NumColors - NUMCOLORS - (=0x00)
0x100a	Ambient0 - chan Ambient color 0 (=0x00)
0x100b	Ambient1- chan Ambient color 1 (=0x00)
0x100c	Material0 - chan Material ID 0 (=0xffffffff)
0x100d	Material1 - chan Material ID 1 (=0xffffffff)
0x100e	COLOR0CNTRL (=0x0401)
0x100f	COLOR1CNTRL (=0x0401)
0x1010	ALPHA0CNTRL (=0x0401)
0x1011	ALPHA1CNTRL (=0x0401)
0x1012	DualTexTrans - (=0x01)
0x1013	?
0x1014	?
0x1015	?
0x1016	?
0x1017	?
0x1018	MatrixIndex0 - MATINDEX A
0x1019	MatrixIndex1 - MATINDEX B
0x101a	ScaleX - Viewport Scale X
0x101b	ScaleY - Viewport Scale Y
0x101c	Scale Z - Viewport Scale Z
0x101d	OffsetX - Viewport Offset X
0x101e	OffsetY - Viewport Offset Y
0x101f	OffsetZ - Viewport Offset Z
0x1020	ProjectionA - A parameter in projection equations
0x1021	ProjectionB - B parameter in projection equations
0x1022	ProjectionC - C parameter in projection equations
0x1023	ProjectionD - D parameter in projection equations
0x1024	ProjectionE - E parameter in projection equations
0x1025	ProjectionF - F parameter in projection equations
0x1026	ProjectOrtho

Register	description
0x103f	NUMTEX - Number of active Textures
0x1040	TEX0
0x1041	TEX1
0x1042	TEX2
0x1043	TEX3
0x1044	TEX4
0x1045	TEX5
0x1046	TEX6
0x1047	TEX7

Register	description
0x1050	DUALTEX0
0x1051	DUALTEX1
0x1052	DUALTEX2
0x1053	DUALTEX3
0x1054	DUALTEX4
0x1055	DUALTEX5
0x1056	DUALTEX6
0x1057	DUALTEX7

0x1000	4	w	Error

0x1001	4	w	Diagnostics

0x1002	4	w	State 0 - Internal State Register 0

0x1003	4	w	State 1 - Internal State Register 1

0x1004	4	w	Xf_clock

31 24	23 16	15 8	7 0

bit(s)		description	
0		0	no power saving when idle
		1	enable Power saving when idle

0x1005	4	w	ClipDisable

31 24	23 16	15 8	7 0

bit(s)		description
2		when set, disable cpoly clipping acceleration (default==0)
1		when set, disable trivial rejection (default==0)
0		when set, disable clipping detection (default==0)

0x1006	4	w	Perf0 - Performance monitor selects

0x1007	4	w	Perf1 - Xform target performance Register

31 24	23 16	15 8	7 0

bit(s)		description
0-6		Xform internal target performance (Cycles per Vertex)

0x1008	4	w	INVTXSPEC

31 24	23 16	15 8	7 0

bit(s)		description		
4-7		`HOST_TEXTURES` – number of host supplied texture coordinates		
		0	no host supplied textures	
		1	1 host supplied texture pair (S0, T0)	
		2-8	2-8 host supplied texturepairs	
		9-15	reserved/unused	
2-3		`HOST_NORMAL` – host supplied normal		
		0	no host supplied normal	
		1	host supplied normal	
		2	host supplied normal and binormals	
0-1		`HOST_COLORS` – host supplied color0 usage		
		0	no host supplied color information	
		1	host supplied color 0	
		2	host supplied color 0 and color 1	

0x1009	4	w	NUMCOLORS

31 24	23 16	15 8	7 0

value		description
0		`No colors`
1		`One color` – Xform supplies 1 color (host supplied or computed)
2		`Two colors` – Xform supplies 2 colors (host supplied or computed)

`Selects the number of output colors`

0x100a	4	w	XF_AMBIENT0 – Ambient color 0 specifications

31 24	23 16	15 8	7 0

bit(s)		description
24		`RED`
16		`GREEN`
8		`BLUE`
0		`ALPHA`

0x100b	4	w	XF_AMBIENT1 – Ambient color 1 specifications

31 24	23 16	15 8	7 0

bit(s)		description
24		RED
16		GREEN
8		BLUE
0		ALPHA

0x100c	4	w	XF_MATERIAL0 – global color0 material specification

31 24	23 16	15 8	7 0

bit(s)		description
24		RED
16		GREEN
8		BLUE
0		ALPHA

0x100d	4	w	XF_MATERIAL1 – global color1 material specification

31 24	23 16	15 8	7 0

bit(s)		description
24		RED
16		GREEN
8		BLUE
0		ALPHA

0x100e	4	w	COLOR0CNTRL

31 24	23 16	15 8	7 0

bit(s)		description	
14		LIGHT7 – Light 7 is source	
		0	Do not use Light
		1	Use light
13		LIGHT6 – Light6 is source	
		0	Do not use Light
		1	Use light
12		LIGHT5 – Light5 is source	
		0	Do not use Light
		1	Use light
11		LIGHT4 – Light4 is source	
		0	Do not use Light
		1	Use light
10		ATTENSELECT – Attenuation Select function	
		0	Select specular (N.H) attenuation
		1	Select diffuse spotlight (L.Ldir) attenuation
9		ATTENENABLE – Attenuation Enable function	
		0	Select 1.0
		1	Select Attenuation fraction
7-8		DIFFUSEATTEN – Diffuse Attenuation function	
		00	Select 1.0
		01	Select N.L, signed
		10	Select N.L clamped to [0,1.0]
		11	
6		AMBIENT_SRC – Ambient source	
		0	Use register Ambient0 register
		1	Use CP supplied vertex color 0
5		LIGHT3 – Light3 is source	
		0	Do not use light
		1	Use light
4		LIGHT2 – Light2 is source	
		0	Do not use light
		1	Use light
3		LIGHT1 – Light1 is source	
		0	Do not use light
		1	Use light
2		LIGHT0 – Light0 is source	
		0	Do not use light
		1	Use light
1		LIGHTFUNC – Color0 Light Function	
		0	Use 1.0
		1	Use Illum0
0		MATERIAL_SRC – Color0 Material source	
		0	Use register (Material 0)
		1	Use CP supplied Vertex color 0

0x100f	4	w	COLOR1CNTRL

31 24	23 16	15 8	7 0

bit(s)		description	
14		LIGHT7 – Light 7 is source	
	0	Do not use Light	
	1	Use light	
13		LIGHT6 – Light6 is source	
	0	Do not use Light	
	1	Use light	
12		LIGHT5 – Light5 is source	
	0	Do not use Light	
	1	Use light	
11		LIGHT4 – Light4 is source	
	0	Do not use Light	
	1	Use light	
10		ATTENSELECT – Attenuation Select function	
	0	Select specular (N.H) attenuation	
	1	Select diffuse spotlight (L.Ldir) attenuation	
9		ATTENENABLE – Attenuation Enable function	
	0	Select 1.0	
	1	Select Attenuation fraction	
7-8		DIFFUSEATTEN – Diffuse Attenuation function	
	00	Select 1.0	
	01	Select N.L, signed	
	10	Select N.L clamped to [0,1.0]	
	11		
6		AMBIENT_SRC – Ambient source	
	0	Use register Ambient1 register	
	1	Use CP supplied vertex color 1	
5		LIGHT3 – Light3 is source	
	0	Do not use light	
	1	Use light	
4		LIGHT2 – Light2 is source	
	0	Do not use light	
	1	Use light	
3		LIGHT1 – Light1 is source	
	0	Do not use light	
	1	Use light	
2		LIGHT0 – Light0 is source	
	0	Do not use light	
	1	Use light	
1		LIGHTFUNC – Color1 Light Function	
	0	Use 1.0	
	1	Use Illum1	
0		MATERIAL_SRC – Color1 Material source	
	0	Use register (Material 1)	
	1	Use CP supplied Vertex color 1	

| 0x1010 | 4 | w | ALPHA0CNTRL |

31 24	23 16	15 8	7 0

bit(s)	description
14	LIGHT7 – Light 7 alpha is source
	0 Do not use Light
	1 Use light
13	LIGHT6 – Light6 alpha is source
	0 Do not use Light
	1 Use light
12	LIGHT5 – Light5 alpha is source
	0 Do not use Light
	1 Use light
11	LIGHT4 – Light4 alpha is source
	0 Do not use Light
	1 Use light
10	ATTENSELECT – Attenuation Select function
	0 Select specular (N.H) attenuation
	1 Select diffuse spotlight (L.Ldir) attenuation
9	ATTENENABLE – Attenuation Enable function
	0 Select 1.0
	1 Select Attenuation fraction
7-8	DIFFUSEATTEN – Diffuse Attenuation function
	00 Select 1.0
	01 Select N.L, signed
	10 Select N.L clamped to [0,1.0]
	11
6	AMBIENT_SRC – Ambient source
	0 Use register Ambient0 alpha register
	1 Use CP supplied vertex color 0 alpha
5	LIGHT3 – Light3 alpha is source
	0 Do not use light
	1 Use light
4	LIGHT2 – Light2 alpha is source
	0 Do not use light
	1 Use light
3	LIGHT1 – Light1 alpha is source
	0 Do not use light
	1 Use light
2	LIGHT0 – Light0 alpha is source
	0 Do not use light
	1 Use light
1	LIGHTFUNC – Color0 alpha Light Function
	0 Use 1.0
	1 Use Illum0
0	MATERIAL_SRC – Color0 alpha Material source
	0 Use register (Material 0 alpha)
	1 Use CP supplied Vertex color 0 alpha

0x1011	4	w	ALPHA1CNTRL

31 24	23 16	15 8	7 0

bit(s)		description	
14		LIGHT7 – Light 7 alpha is source	
	0	Do not use Light	
	1	Use light	
13		LIGHT6 – Light6 alpha is source	
	0	Do not use Light	
	1	Use light	
12		LIGHT5 – Light5 alpha is source	
	0	Do not use Light	
	1	Use light	
11		LIGHT4 – Light4 alpha is source	
	0	Do not use Light	
	1	Use light	
10		ATTENSELECT – Attenuation Select function	
	0	Select specular (N.H) attenuation	
	1	Select diffuse spotlight (L.Ldir) attenuation	
9		ATTENENABLE – Attenuation Enable function	
	0	Select 1.0	
	1	Select Attenuation fraction	
7-8		DIFFUSEATTEN – Diffuse Attenuation function	
	00	Select 1.0	
	01	Select N.L, signed	
	10	Select N.L clamped to [0,2.0]	
	11		
6		AMBIENT_SRC – Ambient source	
	0	Use register Ambient1 alpha register	
	1	Use CP supplied vertex color 1 alpha	
5		LIGHT3 – Light3 alpha is source	
	0	Do not use light	
	1	Use light	
4		LIGHT2 – Light2 alpha is source	
	0	Do not use light	
	1	Use light	
3		LIGHT1 – Light1 alpha is source	
	0	Do not use light	
	1	Use light	
2		LIGHT0 – Light0 alpha is source	
	0	Do not use light	
	1	Use light	
1		LIGHTFUNC – Color0 alpha Light Function	
	0	Use 1.0	
	1	Use Illum0	
0		MATERIAL_SRC – Color0 alpha Material source	
	0	Use register (Material 0 alpha)	
	1	Use CP supplied Vertex color 0 alpha	

`0x1012`	4	w	DualTexTrans

31 24	23 16	15 8	7 0

bit(s)		description	
0		0	disable dual texture transform feature
		1	enable dual transform for all texture coordinates

`0x1013`	4	w	?

`0x1014`	4	w	?

`0x1015`	4	w	?

`0x1016`	4	w	?

`0x1017`	4	w	?

`0x1018`	4	w	MatrixIndex0

31 24	23 16	15 8	7 0

bit(s)		description
24-29		Tex3 matrix index
23-18		Tex2 matrix index
12-17		Tex1 matrix index
6-11		Tex0 matrix index
0-5		Geometry matrix index

`0x1019`	4	w	MatrixIndex1

31 24	23 16	15 8	7 0

bit(s)		description
18-23		Tex7 matrix index
12-17		Tex6 matrix index
6-11		Tex5 matrix index
0-5		Tex4 matrix index

`0x101A`	4	w	Viewport
`0x101B`	4	w	Viewport
`0x101C`	4	w	Viewport
`0x101D`	4	w	Viewport
`0x101E`	4	w	Viewport
`0x101F`	4	w	Viewport

Viewport Matrix

		description
`0x101A`	f32	wd / 2
`0x101B`	f32	-ht / 2
`0x101C`	f32	ZMAX * (farZ - nearZ)
`0x101D`	f32	xOrig + wd / 2 + 342
`0x101E`	f32	yOrig + ht / 2 + 342
`0x101F`	f32	ZMAX * farZ

ZMAX is 16777215.0 (maximum 24-bit Z buffer value, or 'infinite')

`0x1020`	4	w	Projection Matrix
`0x1021`	4	w	Projection Matrix
`0x1022`	4	w	Projection Matrix
`0x1023`	4	w	Projection Matrix
`0x1024`	4	w	Projection Matrix
`0x1025`	4	w	Projection Matrix

Projection Matrix

		orthogonal	perspective
`0x1020`	f32	2.0 / (r - l)	(1.0f / tanf(fovy * 0.5F)) / aspect
`0x1021`	f32	-(r+l) / (r-l)	0
`0x1022`	f32	2.0 / (t-b)	(1.0f / tanf(fovy * 0.5F))
`0x1023`	f32	-(t+b)/(t-b)	0
`0x1024`	f32	-1.0/(f-n)	-n * 1.0f / (f-n)
`0x1025`	f32	-(f)/(f-n)	-(f*n) * 1.0f / (f-n)

`0x1026`	4	w	ProjectOrtho

31 24	23 16	15 8	7 0

bit(s)		description

If set selects orthographic otherwise non-orthographic (Zh or 1.0 select)

note: regs `0x1027-0x103e` skipped (all unknown)

`0x103f`	4	w	NUMTEX - Number of active Textures

0x1040	4	w	TEX0
0x1041	4	w	TEX1
0x1042	4	w	TEX2
0x1043	4	w	TEX3
0x1044	4	w	TEX4
0x1045	4	w	TEX5
0x1046	4	w	TEX6
0x1047	4	w	TEX7

31 24	23 16	15 8	7 0

bit(s)		description		
15-17		EMBOSS_LIGHT - Bump mapping source light (*1)		
12-14		EMBOSS_SOURCE - bump mapping source texture (*2)		
7-11		SOURCE_ROW - regular texture source row (*3)		
		0	GEOM_INROW -	
		1	NORMAL_INROW -	
		2	COLORS_INROW -	
		3	BINORMAL_T_INROW -	
		4	BINORMAL_B_INROW -	
		5	TEX0_INROW -	
		6	TEX1_INROW -	
		7	TEX2_INROW -	
		8	TEX3_INROW -	
		9	TEX4_INROW -	
		a	TEX5_INROW -	
		b	TEX6_INROW -	
		c	TEX7_INROW -	
		d		
		e		
		f		
4-6		TEXGEN_TYPE		
		0	REGULAR - Regular transformation (transform incoming data)	
		1	EMBOSS_MAP - texgen bump mapping	
		2	COLOR_STRGBC0 - Color texgen: (s,t)=(r,g:b) (g and b are concatenated), color 0	
		3	COLOR_STRGBC1 - Color texgen: (s,t)=(r,g:b) (g and b are concatenated), color 1	
3		reserved/unused		
2		INPUT_FORM - format of source input data for regular textures		
		0	AB11 - (A, B, 1.0, 1.0) (used for regular texture source)	
		1	ABC1 - (A, B, C, 1.0) (used for geometry or normal source)	
1		PROJECTION		
		0	ST - (s,t): texmul is 2x4	
		1	STQ - (s,t,q): texmul is 3x4	
0		reseved/unused		

(*1) n: use light #n for bump map direction source (10 to 17)

(*2) n: use regular transformed tex(n) for bump mapping source

(*3) Specifies location of incoming textures in vertex (row specific) (i.e.: geometry is row0, normal is row1, etc . . .) for regular transformations

note: regs 0x1048-104f skipped (all unknown)

0x1050	4	w	DUALTEX0
0x1051	4	w	DUALTEX1
0x1052	4	w	DUALTEX2
0x1053	4	w	DUALTEX3

31 24	23 16	15 8	7 0

bit(s)		description
8		`NORMAL_ENABLE` - specifies if texture coordinate should be normalized before send transform.
6-7		unused
0-5		`DUALMTX` - base row of the dual transform matrix for regular texture coordinate0 (63 max, similar to 0x1018/0x1

5.11.5   GP packet description

The first thing in a GP Packet is the command type (8 bit).Next follows actual primitive data. It may vary
on each opcode type.

5.11.5.1   Command Type

7 0
oooo ovvv

bit(s)		description
	o	Opcode
	v	Vertex Attribute Table Index (VAT)

5.11.5.1.1   opcodes

opcode	Description
0x00	NOP - No Operation
0x08	Load CP REG
0x10	Load XF REG
0x20	Load INDX A
0x28	Load INDX B
0x30	Load INDX C
0x38	Load INDX D
0x40	CALL DL - Call Displaylist
0x48	Invalidate Vertex Cache
0x61	Load BP REG (SU_ByPassCmd)
0x80	QUADS - Draw Quads (*)
0x90	TRIANGLES - Draw Triangles (*)
0x98	TRIANGLESTRIP - Draw Triangle Strip (*)
0xA0	TRIANGLEFAN - Draw Triangle Fan (*)
0xA8	LINES - Draw Lines (*)
0xB0	LINESTRIP - Draw Line Strip (*)
0xB8	POINTS - Draw Points (*)

(*) all draw opcodes must be Or-ed with used VAT index (0...7)

8 bits	16 bits	n
opcode	number of vertices	vertex data

5.11.5.2   Drawing Commands

Vertex data may be in one of many formats. The VCD tells wether data for a component exists (and if yes, if it is direct or indexed) and the VAT tells the actual format of the respective component. Each individual component may or may not exist, but the order is fixed as follows:

1. PNMTXIDX - Position/Normal Matrix Index

2. TEX0MTXIDX - Texture 0 Matrix Index

3. TEX1MTXIDX - Texture 1 Matrix Index

4. TEX2MTXIDX - Texture 2 Matrix Index

5. TEX3MTXIDX - Texture 3 Matrix Index

6. TEX4MTXIDX - Texture 4 Matrix Index

7. TEX5MTXIDX - Texture 5 Matrix Index

8. TEX6MTXIDX - Texture 6 Matrix Index

9. TEX7MTXIDX - Texture 7 Matrix Index

10. POS - Position Vector

11. NRM - Normal or NBT - Binormal vector (T, B)

12. CLR0 - Color0 (Diffused)

13. CLR1 - Color1 (Specular)

14. TEX0 - Texture 0 data

15. TEX1 - Texture 1 data

16. TEX2 - Texture 2 data

17. TEX3 - Texture 3 data

18. TEX4 - Texture 4 data

19. TEX5 - Texture 5 data

20. TEX6 - Texture 6 data

21. TEX7 - Texture 7 data

Notice that the Position/Normal and Texture Matrix Indices are different from the other data in that they are 8 bit and must always be sent as direct data.

5.11.5.2.1   Quads   draws a series of non planar quads, using v0,v1,v2,v3 then v4,v5,v6,v7 and so on. (the quad is actually drawn using 2 triangles so the 4 vertices do not have to be coplanar). The minimum number of vertices is 4.

5.11.5.2.2   Triangles   draws a series of triangles, from v0,v1,v2 then v3,v4,v5 and so on. The number of vertices should be a multiple of 3

5.11.5.2.3   Trianglestrip   draws a series of triangles, from v0,v1,v2 then v1,v3,v2, then v2,v3,v4 amd so on. The number of vertices must be at least 3.

5.11.5.2.4   TriangleFan   draws a series of triangles, from v0,v1,v2 then v0,v2,v3 and so on. The number of vertices must be at least 3.

5.11.5.2.5   Lines   draws a series of unconnected lines, from v0 to v1, then from v2 to v3 and so on. The number of vertices should be a multiple of 2

5.11.5.2.6   Linestrip   draws a series of connected lines, from v0 to v1, then from v1 to v2 and so on. If n vertices are drawn, n-1 lines are drawn

5.11.5.2.7   Points   draws a Point at each of the n vertices

5.11.5.3   NOP - No Operation   Use it to pad primitive data to 32-byte boundaries and to terminate a display list.

5.11.5.4   CALL DL - Call Display List   used to call one display list from another.

8 bits	
7	0
0100	0000

opcode == 0x40

32 bits							
31	24	23	16	15	8	7	0
0000	000.	....	....	....	....	....	....

list address

32 bits							
31	24	23	16	15	8	7	0
0000	000.	....	....	....	....	....	....

list size in bytes (32 bit words?)
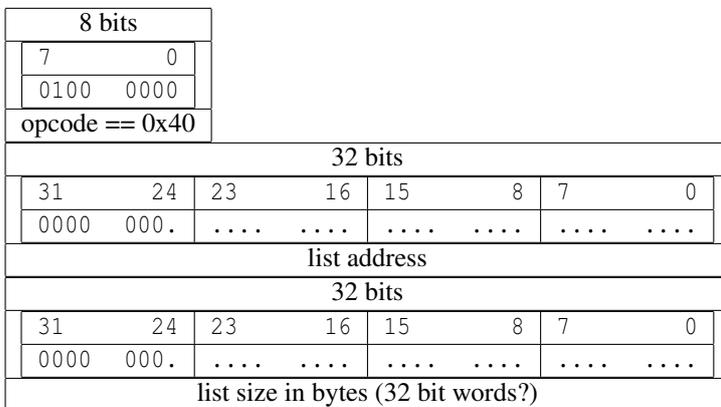
5.11.5.5   Invalidate Vertex Cache
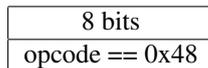
8 bits
opcode == 0x48

5.11.5.6   BP command (Bypass Raster State Registers)

8 bits	8 bits	24 bits
opcode == 0x61	reg. addr.	reg. value
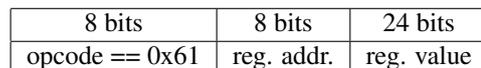
5.11.5.7   CP command (Command Processor Registers)

8 bits	8 bits	32 bits
opcode == 0x08	reg. addr.	reg. value
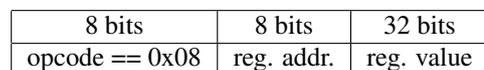
5.11.5.8   XF command (Transform Unit Registers)

8 bits	16 bits	16 bits	32 bits * length
opcode == 0x10	length - 1	1st addr.	reg. value(s)

note : "length" is limited to 16.

5.11.5.9   Indexed XF command

8 bits	16 bits	4 bits	12 bits
opcode	index value	length-1	1st address

note : "length" is limited to 16.

There are 4 different XF index units, which are typically used as follows: A: pos. mtx's B: nrm. mtx's C: tex. mtx's D: light obj's.

6 Exception and Interrupt Processing

6.1 Hardware Exception Sources

Handler Start	Exception
`0x80000100`	System Reset Interrupt
`0x80000200`	Machine Check Interrupt
`0x80000300`	DSI Interrupt
`0x80000400`	ISI Interrupt
`0x80000500`	External Interrupt
`0x80000600`	Alignment Interrupt
`0x80000700`	Program Interrupt
`0x80000800`	FP unavailable Interrupt
`0x80000900`	Decrementer Interrupt
`0x80000C00`	System Call Interrupt
`0x80000d00`	Trace Interrupt
`0x80000f00`	Performance Monitor Interrupt
`0x80001300`	IABR Interrupt
`0x80001700`	Thermal Interrupt

6.1.1 System Reset Interrupt

Triggered at a system reset

6.1.2 Machine Check Interrupt

6.1.3 DSI Interrupt

Triggered if an attempt to store or read data from/to an illegal address was made

6.1.4 ISI Interrupt

Triggered if an attempt to fetch an instruction from an illegal address was made.

6.1.5 External Interrupt

14 sources, triggered by external chips (you may look at it as the traditional IRQ).

6.1.5.1 Setup

▷ set corresponding bit(s) in PI Interrupt Mask Register (`0xcc003004`)

▷ enable external Interrupts in the Machine State Register

6.1.5.2 Handling

▷ check PI Interrupt Cause Register (`0xcc003000`) for flags

▷ acknowledge by setting corresponding bits in PI Interrupt Cause Register (`0xcc003000`)

▷ handle different interrupt sources

bit		Description
13	HSP	High Speed Port
12	DEBUG	External Debugger
11	CP	Command FIFO
10	PE FINISH	Frame is Ready
9	PE TOKEN	Token Assertion in Command List
8	VI	Video Interface
7	MEM	Memory Interface
6	DSP	DSP
5	AI	Audio Streaming
4	EXI	EXI
3	SI	Serial
2	DI	DVD
1	RSW	Reset Switch
0	ERROR	GP runtime error

6.1.6 Alignment Interrupt

6.1.7 Program Interrupt

Triggered if the instruction that was about to execute is invalid.

6.1.8 FP unavailable Interrupt

6.1.9 Decrementer Interrupt

Triggered by an underflow of the decrementer register.

6.1.10 System Call Interrupt

Triggered when the PowerPC instruction 'sc' is executed.

6.1.11 Trace Interrupt

6.1.12 Performance Monitor Interrupt

6.1.13 IABR Interrupt

6.1.14 Thermal Interrupt

6.2 External Interrupt Sources

6.2.1 HSP - High Speed Port

3 Sources

6.2.1.1 TX Mailbox Interrupt

6.2.1.2 RX Mailbox Interrupt

6.2.1.3 ID Interrupt

6.2.2 Debug

1 Source

6.2.3 CP - Command Processor

2 Sources (read/write pointer watermark over- and underflow) check bit 0 and bit 1 of `0xcc000000`

6.2.3.1 FIFO underflow

6.2.3.1.1 setup

  ▷ setup CP FIFO low watermark

  ▷ set bit 1 of CP interrupt status register (`0xcc000002`) to enable

6.2.3.1.2 handling

  ▷ clear bit 3 of CP interrupt status register (`0xcc000002`) to acknowledge

6.2.3.2 FIFO overflow

6.2.3.2.1 setup

  ▷ setup CP FIFO high watermark

  ▷ set bit 0 of CP interrupt status register (`0xcc000002`) to enable

6.2.3.2.2 handling

  ▷ clear bit 2 of CP interrupt status register (`0xcc000002`) to acknowledge

6.2.4 PE - Pixel Engine Finished

1 Source (Frame finished)

6.2.4.1 setup

  ▷ set bit 1 in PE Interrupt status register (`0xcc001000a`)

6.2.4.2 handling

▷ set bit 3 in PE Interrupt status register (`0xcc001000a`) to acknowledge the interrupt

6.2.5 PE - Pixel Engine Token

1 Source (Token in GP Command List)

6.2.5.1 setup

▷ set bit 0 in PE Interrupt status register (`0xcc001000a`)

6.2.5.2 handling

▷ check PE Token Register (`0xCC00000E`) for the token that triggered the interrupt.

▷ set bit 2 in PE Interrupt status register (`0xcc001000a`) to acknowledge the interrupt

6.2.6 VI - Video Interface

4 Sources, check (`0xcc002030`) (`0xcc002034`) (`0xcc002038`) (`0xcc00203c`) for flags

6.2.6.1 Setup

▷ setup desired position of the interrupt

▷ set enable bit in Display Interrupt Register (`0xcc002030`) (`0xcc002034`) (`0xcc002038`) (`0xcc00203c`)

6.2.6.2 Handling

▷ clear Status bit in Display Interrupt Register (`0xcc002030`) (`0xcc002034`) (`0xcc002038`) (`0xcc00203c`) to acknowledge the interrupt.

6.2.7 Memory Interface

4 sources (4 regions of memory can be protected independently)

6.2.7.1 protection fault

6.2.7.1.1 Setup

▷ clear (`0xCC004020`)

▷ setup the regions in (`0xCC004000`)(`0xCC004004`)(`0xCC004008`) (`0xCC00400c`)

▷ setup type of protection in (`0xCC004010`)

▷ enable protecting of regions in irq mask (`0xCC00401c`)

6.2.7.1.2 Handling

▷ clear (`0xCC004020`)

▷ read status bits from (`0xCC00401e`) to check what region triggered the protection fault

▷ set status bits in (`0xCC00401e`) to acknowledge the interrupt

▷ read the address that triggered the protection fault from (`0xCC004022`) and (`0xCC004024`)

6.2.8 DSP Interface

3 Sources, check (`0xcc00500a`) for flags

6.2.8.1 Audio DMA finished   asserted when audio DMA transfer has been completed.

6.2.8.1.1 Setup

▷ set bit 4 of DSP Control Register (`0xcc00500a`)

6.2.8.1.2 Handling

▷ set bit 3 of DSP Control Register (`0xcc00500a`) to acknowledge the interrupt.

6.2.8.2 ARAM transfer complete   asserted when a transfer from/to auxiliary ram has been completed.

6.2.8.3 DSP

6.2.9 Audio Streaming Interface

1 Source, check (`0xcc006c00`) for flag. asserted based on the disk streaming sample counter.

6.2.10 EXI

3 Sources each EXI Channel (TCINT,EXTINT,EXIIRQ), making 9 total, check (`0xcc006800`) (`0xcc006814`) (`0xcc006828`) for flags.

6.2.10.1 DMA Transfer finished (TCINT channel 0, channel 1)

6.2.10.2 Ethernet Adapter (EXIIRQ channel 2)

6.2.10.2.1 setup

6.2.10.2.2 handling

▷ check command register 3 (irq status) for the exact source (MX chip, killing irq,cmd error, challenge/response request, challange/response status)

▷ if from MX chip, check register 8 and 9 for the exact cause

6.2.10.3 Memory Card removed (EXTINT channel 0, channel 1)

6.2.10.3.1 setup

6.2.10.3.2 handling

▷ To check whether the device has been inserted or removed, check the EXICPR[EXT] bit.

6.2.11 Serial Interface

6.2.12 DVD Interface

4 Sources (Break Complete, DMA finished, Device Error, Cover state changed), check (`0xcc006000`)(`0xcc006004`) for flags.

6.2.12.1 Break Complete

6.2.12.2 DMA finished

6.2.12.3 Device Error

6.2.12.4 Cover State changed

6.2.13 Reset Button

1 Source (Reset Button pressed)

6.2.13.1 setup

▷ no further setup required

6.2.13.2 handling

▷ nothing special required, its however recommended to clear the PI Interrupt Mask to avoid multiple interrupts.

6.2.14 Error

1 Source (GP runtime error)

7   Video Processing

7.1   Used VI terms

   ▷ XFB - external Framebuffer

The external framebuffer resides in main memory and can be directly displayed by the video processor.

7.2   init VI

7.2.1   Videomodes

7.3   render to XFB

7.4   vertical retrace

7.5   set XFB Address

8   3D Graphics Processing

Main
Memory

XFB
external
Framebuffer

Main CPU

start

XF
Transform
Unit

SU
Setup
Unit

RAS0
Edge–/Z–
Rasterizer

CP
Command
Processor

Cache

PE
Pixel
Engine

EFB
embedded
Framebuffer

RGB Framebuffer
Z–Buffer

Top Z–
Compare

PI
Processor
Interface

end

bottom Z–
compare

Memory
Interface

TEV
Texture
Environment

RAS2
Color
Rasterizer

Texture
Unit

TMEM
Texture
memory

RAS1
Texture Coordinate
Rasterizer

VI
Video
Interface

3D Graphics Pipeline

Display

8.1   basic operations

8.1.1   load BP Register

▷ write byte 0x61 to GXFIFO

▷ write 4 bytes of data to GXFIFO

8.1.2   load CP Register

▷ write byte 0x08 to GXFIFO

▷ write 1 byte address to GXFIFO

▷ write 4 bytes of data to GXFIFO

8.1.3   load XF Register

▷ write byte 0x10 to GXFIFO

▷ write 0x0000 (16 bit) to GXFIFO

▷ write addr (16 bit) to GXFIFO

▷ write 4 bytes of data to GXFIFO

8.1.4   load XF Register Indexed

▷ write byte 0x10 to GXFIFO

▷ write n-1 (16 bit) to GXFIFO

▷ write addr (16 bit) to GXFIFO

▷ write n bytes of data to GXFIFO

8.2   example processing loop

```
gx_init();
while(running)
{
gx_begin();
drawframe();
gx_end();
waitvsync();
}
gx_close();
```

8.2.1   init GX

8.2.1.1   setup the fifos

▷ fifo start/end must be 32 byte aligned

▷ minimum fifo size is 64kb

▷ typical size for the hi watermark is fifo size - 16kb

8.2.1.2 enable gx command processing   while (mfwpar () & 1);

PI[3] = 0x100000;

PI[4] = 0x110000;

PI[5] = 0x100000;

mtwpar(0xC008000); // GXFIFO physical address

mtspr(920, mfspr(920) | 0x40000000);

8.2.1.3 send setup frame   after setting up and enabling gx command processing it is recommended to send a first initial 'frame' to setup the various internal registers to useful values.

8.2.1.3.1 Videomodes   remember that depending on the videomode some things must be setup differently (such as the scissor, viewport, ...)

8.2.2 begin frame

8.2.3 draw frame

8.2.4 end frame

   ▷ load draw-done to BP register (0x45000002)

   ▷ copy EFB to XFB

   ▷ terminate the list by writing 32 zeroes

8.2.4.1 copy EFB to XFB   #define XY(x, y) (((y) << 10) | (x))

GX_LOAD_BP_REG(0x4000001f); // set z mode

GX_LOAD_BP_REG(0x410004bc); // set color mode 0

GX_LOAD_BP_REG(0x49000000 | XY(0, 0)); // set source top left

GX_LOAD_BP_REG(0x4a000000 | XY(639, 479)); // set source bottom right

GX_LOAD_BP_REG(0x4d000028); // stride? (0x1280>>5) ... 640*2 ; 320*YuYv

GX_LOAD_BP_REG(0x4b000000 | (0xC00000 >> 5)); // xfb target address

GX_LOAD_BP_REG(PE_COPY_CLEAR_AR | 0x0000);

GX_LOAD_BP_REG(PE_COPY_CLEAR_GB | 0x0000);

GX_LOAD_BP_REG(PE_COPY_CLEAR_Z | 0xFFFFFF);

GX_LOAD_BP_REG(0x52004803); // do it (efb copy execution command?)

8.2.4.2 copy EFB to Texture   simelar to copying EFB to XFB, setup BP registers 0x4a,0x4a,0x4d,0x4b and then `(0x52000003|(format<<4))`

8.2.5 close GX

9 Joy-Bus Devices

9.1 ID and Device List

The device ID can be read by sending the SI Command 0x00, and then reading 3 bytes from the respective device. The response-data looks like this:

first comes a 16bit device id:

ID	Device
0x0500	? N64 Controller
0x0001	? N64 Microphone
0x0002	? N64 Keyboard
0x0200	? N64 Mouse
0x0004	GBA
0x0800	GBA (n/a)
0x0900	GC Standard Controller
0xe960	? GC Wavebird Receiver
0xe9a0	? GC Wavebird
0xa800	? GC Wavebird
0xebb0	? GC Wavebird
0x0820	GC Keyboard
0x0800	? GC Steering Wheel
0x0900	DKongas (same as std Controller)
?	Resident Evil4 Chainsaw

bit(s)		description
15		wireless (1: wireless Controller)
14		wireless receive (0: not wireless 1: wireless)
13		Rumble Motor (0: available 1: not available)
12		Type of Controller (always 0 ?)
11		Type of Controller (0: N64 1: Dolphin)
10		wireless type (0:IF 1:RF)
9		wireless state (0: variable 1: fixed)
8		0: non standard Controller 1: Dolphin Standard Controller
7		
6		
5		wireless origin (0:invalid 1:valid)
4		wireless fix id (0:not fixed 1:fixed)
3		wireless type - 0:normal 1: non-controller (?)
2		wireless type - 0:normal 1: lite controller
1		wireless type -
0		wireless type -

the device id is followed by 8 status bits:

bit(s)		description
7		?
6		?
5		?
4		?
3		? rumble motor running
2		?
1		?
0		?

9.2   standard Controller

9.2.1   Init

▷ enable all controllers in 0xcc006430

▷ set Joy-channel 1-4 Command Register to 0x00400300

▷ clear SI i/o buffer

▷ wait until bit 31 of 0xCC006434 is 0, then set it to 1

Command Word

0x00400000 - enable polling

0x00000300 - ?

0x00000100 - ?

0x00000001 - enable rumble motor

Commands:

0x00 - get id+status

0x40 - ?

0x41 - get origins

0x42 - calibrate ?

9.2.2   Read Controller Status

▷ simply read all Joy-channel registers and extract the info you want

first input word

bit(s)		Description
31		ERRSTAT - (assumed 0)
30		ERRLATCH - (assumed 0)
29		unused (?)
28	s	Start Button
27	y	Y Button
26	x	X Button
25	b	B Button
24	a	A Button
23		unused (?)
22	L	L Trigger
21	R	R Trigger
20		Z Trigger
19	u	D-Pad Up
18	d	D-Pad Down
17	r	D-Pad Right
16	l	D-Pad Left
8-15	x	Analog Stick X (8bit unsigned, ~32...128...~224)
0-7	y	Analog Stick Y (8bit unsigned, ~32...128...~224)

second input word

bit(s)		Description
24-31	x	Analog C Stick X (8bit unsigned, ~32...128...~224)
16-23	y	Analog C Stick Y (8bit unsigned, ~32...128...~224)
8-15	l	L Trigger Analog (8bit unsigned, ~32...~224)
0-7	r	R Trigger Analog (8bit unsigned, ~32...~224)

9.2.3 rumble Motor On

(volatile unsigned long)0xCC006400 = 0x00400001;

(volatile unsigned long)0xCC006438 = 0x80000000;

9.2.4 rumble Motor Off

(volatile unsigned long)0xCC006400 = 0x00400000;

(volatile unsigned long)0xCC006438 = 0x80000000;

9.3 Keyboard

9.3.1 Types

9.3.1.1 ASCII Official Nintendo/SEGA keyboard for the GameCube. It has 80 keys plus an Fn key, some of the keys have Japanese labelings. It has an LShift and an RShift key, but only a single Ctrl and Alt key. The Fn key is internal to the keyboard. It makes the keyboard send different scancodes if it is pressed, and an Fn keypress alone cannot be detected.

9.3.1.2 Datel   A British IBM PS/2 keyboard that ships with an adapter.

9.3.1.3 Tototek Adapter   Converts the IBM PS/2 protocol to the GameCube SI protocol and also converts the PS/2 scancodes into GameCube scancodes. The keys that have Japanese labelings on the ASCII keyboard get mapped to keys like PrintScreen and Pause.

  ▷ 0-9 and Enter send the scancodes of their counterparts on the alphanumeric part of the keyboard, regardless of the status of NumLock, so the Numpad cannot be used as cursor keys.

  ▷ The key right of LShift on non-US keyboards sends no scancode.

  ▷ The Pause/SysReq key only sends the keycode of 0x37 once, even if it's pressed down continuously.

  ▷ Numpad-* sends 0x37, the same as Pause/SysReq, but this one continues sending it.

  ▷ All combinations of Pause/SysReq with other keys are possible, except for these: LStrg or LStrg together with Pause/SysReq doesn't send anything. This would have been SysReq.

  ▷ The adapter easily gets confused by two keys for which it produces the same GameCube scancodes: If you hold down LStrg and press RStrg, the LStrg scancode will disappear even though LStrg is srill pressed down. The same is true for Ctrl and Alt.

9.3.2 Scancodes

	.0	.1	.2	.3	.4	.5	.6	.7
0.							Home	End
1.	A	B	C	D	E	F	G	H
2.	Q	R	S	T	U	V	W	X
3.	7 NP 7	8 NP 8	9 NP 9	0 NP 0	-_ () NP -	=+	PrntScrn NP /	Pause/SysReq NP *
4.	F1	F2	F3	F4	F5	F6	F7	F8
5.	Backspace	Tab		CapsLock	LShift (*54)		LCtrl (*56)	LAlt (*57)
6.		Enter NP Enter						
	.8	**.9**	**.A**	**.B**	**.C**	**.D**	**.E**	**.F**
0.	PgUp	PgDn	ScrollLock					
1.	I	J	K	L	M	N	O	P
2.	Y	Z	1 NP 1	2 NP 2	3 NP 3	4 NP 4	5 NP 5	6 NP 6
3.	[{	;: NP + (*39)	”	]}	,<	.> NP .	/?	|
4.	F9	F10	F11	F!2	ESC	Ins	Del	'~
5.	LWin	space	RWin	Menu	Left	Down	Up	Right
6.			NumLock (*6a)					

(*39) Tototek adapter: makes only sence for japenese ASCII labeling
(*54) Tototek adapter sends this code for LShift and RShift
(*56) Tototek adapter sends this code for LCtrl and RCtrl
(*57) Tototek adapter sends this code for LAlt and RAlt
(*6a) Tototek adapter: undefined for GameCube

9.3.3 Init

▷ enable controller in 0xcc006430

▷ set Joy-channel Command Register to 0x00540000

▷ clear SI i/o buffer

▷ wait until bit 31 of 0xCC006434 is 0, then set it to 1

Command Word

0x00540000 - enable polling

9.3.4 Read Keyboard

first input word

bit(s)		Description
31		ERRSTAT - (assumed 0)
30		ERRLATCH - (assumed 0)
24-29		?
16-23		?
8-15		?
0-7		?

second input word

bit(s)		Description
24-31		key1
16-23		key2
8-15		key3
0-7		?

9.4 GBA

to do

9.5 Wavebird

to do

9.6 steering wheel

to do

9.7 DKongas

These work exactly like the standard controllers from the programmers point of view, and they even have the same ID.

9.7.1 Read Controller Status

▷ simply read all Joy-channel registers and extract the info you want

first input word

bit(s)		Description
31		ERRSTAT - (assumed 0)
30		ERRLATCH - (assumed 0)
29		unused (?)
28	s	Start Button
27	y	left Konga, top/left (Y Button)
26	x	right Konga, top/left (X Button)
25	b	left Konga, bottom/right (B Button)
24	a	right Konga, bottom/right (A Button)
23		unused (?)
22	L	unused (L Trigger)
21	R	unused ? (R Trigger)
20		unused (Z Trigger)
19	u	unused (D-Pad Up)
18	d	unused (D-Pad Down)
17	r	unused (D-Pad Right)
16	l	unused (D-Pad Left)
8-15	x	unused (Analog Stick X)
0-7	y	unused (Analog Stick Y)

second input word

bit(s)		Description
24-31	x	unused (Analog C Stick X)
16-23	y	unused (Analog C Stick Y)
8-15	l	unused (L Trigger Analog)
0-7	r	Microphone (R Trigger Analog) (8bit unsigned, ~16...?)

9.8 Resident Evil4 Chainsaw

to do

10 EXI Devices

10.1 EXI Channel and Device List

The following table shows the GameCube devices which use the EXI bus and their channel and device numbers, the EXI frequency commonly used with them and their virtual offset in EXI mapping.

channel	device	freq	offset	Description
0	0	4		Memory Card (Slot A)
0	1	3	0x00000000	Mask ROM
0	1	3	0x20000000	Real-Time Clock (RTC)
0	1	3	0x20000100	SRAM
0	1		0x20010000	UART
1	0	4		Memory Card (Slot B)
2	0			AD16 (trace step)
0	2			Serial Port 1
0	2	5		Ethernet Adapter (SP1)

Note: The Real-Time Clock (RTC), SRAM, and Mask ROM are actually one device mapped to different address offsets. The SRAM should only be accessed by the IPL and contains non-volatile system data. The Mask ROM contains the IPL itself (encrypted) and the system font data.

10.2 Retrieving the ID of an EXI Device

To retrieve the ID of an EXI Device, an EXI IMM write operation must be used to send the ID command (0x0000) and an EXI IMM read operation should follow it to read the actual 4 byte ID.

ID	Device
0x00000004	Memory Card 59
0x00000008	Memory Card 123
0x00000010	Memory Card 251
0x00000020	Memory Card 507
0x00000040	Memory Card 1019
0x00000080	Memory Card 2043
0x01010000	USB Adapter
0x01020000	NPDP GDEV
0x05070000	IS Viewer
0x04120000	AD16
0x03010000	Marlin (?)
0x02020000	Modem
0x04020200	Ethernet Adapter

10.3 Mask ROM

Mask ROM also referred as bootrom or IPL. Total size of bootrom is 2 MB.

10.3.1   Memory Map (Europe/PAL)

Start	End	Size	Description
0x00000000	0x000000ff	0x00000100	Copyright message (*1)
0x00000100	0x001aeee8	0x001aede8	BIOS data (*2)
0x001AFF00	0x001FA0E0	0x0004D000	'Yay0' - ROM Fonts #1 (SJIS)
			0x61 bytes of 0xFF, 0x62 , followed by zeros until 0x1FCF00
0x001FCF00	0x001FF474	0x00003000	'Yay0' - ROM Fonts #2 (ANSI)
0x001FF474	0x001FFEF0		filled with 0x00
0x001FFF00		0x001FFFFF	filled with 0xff

(*1) "(C) 1999-2001 Nintendo. All rights reserved.(C) 1999 ArtX Inc. All rights reserved.PAL Revision 1.0 " and zeros up to 0x100.
(*2) encrypted by an XOR cyphertext which is generated by a yet unknown algorithm (probably not a single LFSR)

10.3.2   Memory Map (USA/NTSC)

Start	End	Size	Description
0x00000000	0x0015ee40	0x0015ee40	BIOS data (*1)
0x001AFF00		0x0004D000	ROM Fonts #1 (SJIS)
0x001FCF00		0x00003000	ROM Fonts #2 (ANSI)
0x001FFF00		0x001FFFFF	filled with 0x00

(*1) encrypted by an XOR cyphertext which is generated by a yet unknown algorithm (probably not a single LFSR)

note: all unused space is filled with 0x00, no pieces with 0xFF.

10.3.3   Memory Map (Japenese/NTSC)

10.3.4   Memory Map (Japenese/NTSC - Panasonic Q)

10.3.5   Font Encoding

The 'Yay0' data is compressed similar to the the Zelda 64 'Yaz0' compression. Besides the Raw Font data it also contains some information about the Font.

10.3.6   Font Layout

10.3.6.1   SJIS Font (ROM Font #1)

10.3.6.2   ANSI Font (ROM Font #2)   The ANSI Font is a 512x512 Pixel Texture in I4 Format. It consists of 21 colums and 11 rows of characters which are in a 24x24 pixel grid.

	!	’	#	$	%	&	t’	(	)	*	+	,	-	.	/	0	1	2	3	4
5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
_	’	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
t	u	v	w	x	y	z														

10.3.7   Operation

10.3.7.1   read

▷ IMM write 32 bit: (offset<<6)

▷ DMA read X bytes

max offset is 2*1024*1024, max block len is ?

10.4   RTC (Real-Time Clock)

Real-Time Clock (RTC) is 32-bit value, counting time intervals in seconds. To get the real time (seconds since January 1st, 2000 12am) add the counter bias saved in SRAM.

10.4.1   Operation

10.4.1.1   read

▷ IMM write 32 : `0x20000000` | (0<<6) (RTC offset)

▷ IMM read 32-bit RTC value

Since it is uncertain if the hardware prevents fragmented reads of the time interval (eg byte 1 from tick n and byte 2-4 from tick n+1) it is recommended to repeatedly retrieve the value until there is no difference between two consecutive reads.

10.4.1.2   write

▷ IMM write 32 : `0xA0000000` (== `0x20000000` | `0x80000000` (write flag) | (0<<6) (RTC offset))

▷ IMM write 32-bit RTC value

10.5   SRAM

SRAM is battery backed memory, used for saving some non volatile settings. Size of SRAM is 64 bytes.

10.5.1   Memory Map

offset	end	size	Description
0x00		2	Checksum 1
0x02		2	Checksum 2
0x04		4	ead 0
0x08		4	ead 1
0x0c		4	counter Bias (add to RTC value to get actual time)
0x10		1	display offset H (signed value, -32...32)
0x11		1	ntd
0x12		1	language

value	Description
0	english
1	german
2	french
3	spanish
4	italian
5	dutch

offset	end	size	Description
0x13		1	flags

bit(s)	Description
7	? (=0)
6	? (=0)
5	? (=1)
4	? (=0)
3	? (=1)
2	0: mono 1: stereo
1	? (=0)
0	? (=0)

offset	end	size	Description
0x14		2*12	Flash ID
0x2c		4	wireless Keyboard ID
0x30		4*2	wireless PAD ID
0x38		1	last DVD Errorcode
0x39		1	padding/unused/reserved
0x3a		2*2	Flash ID checksum
0x3e		2	padding/unused/reserved

10.5.2   Operation

10.5.2.1   read

▷ IMM write 32 : 0x20000100 (== 0x20000000 | (8 << 6) (SRAM offset))

▷ DMA read 64 bytes.

10.5.2.2   write

▷ IMM write 32 : 0xa0000100 (== 0x20000000 | 0x80000000 (write flag) | (8 << 6) (SRAM offset))

▷ use sequential IMM writes

10.5.3   Checksums

the SRAM data is protected against corruption by a simple additive checksum which is calculated like this:

```
void checksums (unsigned short *buf, unsigned short *c1, unsigned short *c2)
{
int i;
*c1 = 0;  *c2 = 0;
for (i = 0;i<4;++i)
{
*c1 += buf[0x06 + i];
*c2 += (buf[0x06 + i] ^ 0xFFFF);
}
}
```

10.6   AD16

AD16 is on channel 2, as device 0. Probably its used for debugging purposes. AD16 is the 32-bit register, keeping bootrom "trace-step".

10.6.1   Operation

10.6.1.1   init

  ▷ IMM write : `0x0000`

  ▷ IMM read 32 bit ID and check it for error (it should be `0x04120000`)

10.6.1.2   write

  ▷ IMM write : `0xa0`

  ▷ IMM write 32-bit value (trace-step)

10.6.1.3   read

  ▷ IMM write : `0xa2`

  ▷ IMM read 32-bit AD16 register value

10.6.2   Trace-Step Values

10.6.2.1   BS

0x01000000	AD16 Inited, cache lines 320, 340,360, 380, 3a0 prefetched
0x02000000	cache line 0x3e0 prefetched
0x03000000	rest of cachelines prefetched
0x04000000	ramtest passed
0x05000000	ramtest error
0x06000000	ramtest error

	0x00000800	System Init
	0x00000900	DVD Init
10.6.2.2 BS2	0x00000a00	Card Init
	0x00000b00	video init
	0x00000c00	final before menu

10.7 Memory Cards

Product	Blocks	Mega bits	Mega bytes	Vendor	
Memory Card 59	64	4	0.5	Nintendo	
Action Replay memory card	64	4	0.5	Datel	
Memory Card '4 Mega'	64	4	0.5	3rd Party	WINBOND 512K*8 CM
Memory Card 123	128	8	1	Nintendo (*)	
Memory Card '8 Mega'	128	8	1	3rd Party	
Memory Card 251	256	16	2	Nintendo	
Memory Card 507	512	32	4	Nintendo (*)	
Memory Card 1019	1024	64	8	Nintendo	
Memory Card '64 Mega'	1024	64	8	3rd Party	
USB Memory Adaptor 64M / 1019 blocks	1024	64	8	EMS	
Memory Card 2043	2048	128	16	Nintendo (*)	

(*) never seen those, appearently supported but never manufactured.

as you can see the products are named in reference to their *useable* capacity in blocks or total size in mega bits. The theoretical maximum size for a memory card is 128 mega bits (16 mega bytes, 2048 blocks).

10.7.1 Commands

Command	len	indata	len	outdata	len	Description
`0x8500`	2	-	–	ID	2	get ID
`0x8300`	2	-	–	Status	1	get Card Status
`0x89`	1	-	–	-	–	clear Card Status
`0x52`	1	offset, 0x00000000	8	Block Data	<=0x200	read Block
`0xf40000`	3	-	–	-	–	erase Card
`0xf1`	1	Sector	2	-	–	erase Sector
`0xf2`	1	Block offset	4	Block Data	<=0x80	write Block

10.7.2 Operation

10.7.2.1 unlocking original memory cards (those manufactured by nintendo) need to be 'unlocked' before they can be accessed. this is appearently done by a small dsp program.

to do

10.7.2.2 get ID

▷ IMM write 0x85,00

▷ IMM read 2 bytes ID

10.7.2.2.1 Card IDs

10.7.2.3 get Status

- ▷ IMM write 0x83,00

- ▷ IMM read 1 byte Status

10.7.2.3.1 Status Bits

7		0
x...		...r

bit(s)		description
7	x	1: erase in Progress (?)
0	r	1: Card ready (?)

10.7.2.4 clear Status

- ▷ IMM write 0x89

10.7.2.5 read Block

- ▷ select

- ▷ IMM write: 0x52

- ▷ IMM write: (offset >> 17), (offset >> 9), (offset >> 7) & 3, offset & 0x7F

- ▷ IMM write: 0x00, 0x00, 0x00, 0x00

- ▷ read 8 bit values

- ▷ deselect

max offset is 16*1024*1024, max block len is 512 bytes.

10.7.2.6 erase Card

- ▷ select

- ▷ IMM write 0xf4,0x00,0x00

- ▷ deselect

10.7.2.7 erase Sector

- ▷ select

- ▷ IMM write 0xf1

- ▷ IMM write (sector >> 17) & 0x7F, (sector >> 9) & 0xFF

- ▷ deselect

- ▷ wait until bit 7 of card status is cleared

10.7.2.8 write Block

▷ wait while bit 0 of card status is cleared (= card ready)

▷ select

▷ IMM write 0xf2

▷ IMM write (offset >> 17) & 0x3F, (offset >> 9) & 0xFF, (offset >> 7) & 3, offset & 0x7F

▷ write 0x80 bytes

▷ deselect

10.8 Ethernet Adapter

The Macronix chip found in the ethernet adapter (mx98730ec) seems to be a reenginered version of the mx98726 or mx98728 (or the ec type respectivly).

10.8.1 registers

Registerblock Base	Size of Registerblock	common access size
0x00	0x1000	1

These are the actual Chip Registers which match with the descriptions in MXs documents.

0x00	1	r/w	NCRA - Network Control Register A (0x08 ?)

```
7        0
???? ?ep?
```

bit(s)		description
7		INTCLK - must be 0 for normal operation.
6		INTMODE - Interrupt Mode: Set for the active high interrupt, reset for the active low interrupt case.
5		LB1 - Loopback mode
4		LB0 <table><tr><th>LB1</th><th>LB0</th><th>description</th></tr><tr><td>0</td><td>0</td><td>Mode0 - Normal mode</td></tr><tr><td>0</td><td>1</td><td>Mode1 - internal FIFO Loopback</td></tr><tr><td>1</td><td>0</td><td>Mode2 - internal NWAY Loopback</td></tr><tr><td>1</td><td>1</td><td>Mode3 - internal PMD Loopback</td></tr></table>
3		SR - Start Receive : Enable the MAC to receive packets. Default is disabled.
2	e	ST1 - Start Transmit Command/Status
1	p	ST0
0		RESET - Software reset.

bit 1 and bit 2 will get cleared after a packet has been sent

`0x01` | 1 | r/w | NCRB - Network Control Register (0x11, 0x12 ?)

`7   0`

bit(s)		description
6-7		RXINTC - Recieve Interrupt Counter
5		HBD - Heart Beat Check Disable
4		AB - "pass all broadcast frames"
3		PB - "pass bad frames"
2		PM - "pass all multicast"
1		CA - capture effect mode
0		PR - promiscuous mode

`0x02` | 1 | r/w | GMAC Test Register A : TRA

bit(s)		Description
7		SB - Start/Stop Back-off counter
6		FC - Forced Collision
5		RWD - Receive Watchdog Disable
4		RWR - Recieve Watchdog Release
1-3		TMODE - Test Moder Select bits
0		TEST - Test mode enable

`0x03` | 1 | r/w | GMAC Test Register B : TRB

bit(s)		Description
7		BFSTATUS
6		BFS1
5		BKCNTLB
4		BFS0
3		COLCNTCB
2		RDNCNTSB
1		RDNCNTCB
0		FKD - Flaky Oscillator Disable

`0x04` | 1 | r/w | LTPS - Last Transmitted Packet Status (transmit error code ?)

bit(s)		Description
7		TERR - Transmit Error
6		OWC - Out of Window Collision
5		UF - TX FIFO Underflow
4		CRSLOST - Carrier Sense Lost
3		CC3 - Collision Count Bit 3
2		CC2
1		CC1
0		CC0

| 0x05 | 1 | r/w | LRPS - Last Recieved Packet Status |

bit(s)		Description
7		RERR - Recieve Error
6		RF - Runt Frame
5		MF - Multicast Frame address
4		RW - Recieve Watchdog
3		FO - FIFO Overrun
2		FAE - Frame Alignment Error
1		CRC - CRC error
0		BF - RX Packet Buffer Full Error

| 0x06 | 1 | r/w | MPCL - Missed Packet Counter |

bit(s)		Description
0-7		MISSCNT - Miss Packet Counter LSB

| 0x07 | 1 | r/w | MPCL - Missed Packet Counter |

bit(s)		Description
0-7		MISSCNT - Miss Packet Counter MSB

| 0x08 | 1 | r/w | IMR - Interrupt Mask Register (IRQ Mask) |

bit(s)		Description
7		RBFIM - RX Buffer Full Interrupt Mask
6		BUSEIM - Bus Error Interrupt Mask
5		FIFOEIM - FIFO Error Interrupt Mask
4		TEIM - Transmit Error Interrupt Mask
3		REIM - Receive Error Interrupt Mask
2		TIM - Transmit Interrupt Mask
1		RIM - Recieved Interrupt Mask
0		FRAGIM - Fragment Counter Interrupt Mask

| 0x09 | 1 | r/w | IR - Interrupt Register (irq status) |

7		0
???t	esr?	

bit(s)		description
7		RBFI - RX Buffer Full Interrupt
6		BUSEI - Bus Error Interrupt
5		FIFOEI - FIFO Error Interrupt
4	t	TEI - Transmit Error Interrupt (1: transmit error)
3	e	REI - Recieve Error Interrupt (1: receive error ?)
2	s	TI - Transmit OK Interrupt (1: packet sent ?)
1	r	RI - Receive OK Interrupt (1: packet received ?)
0		FRAGI - Fragment Counter Interrupt

| 0x0a | 2 | r/w | BP - Boundary Page Pointer Register (0x0100 ?) |

| `0x0c` | 2 | r/w | TLBP - TX Low Boundary Page Pointer Register |

| `0x0e` | 2 | r/w | TWP - Transmit Buffer Write Page Pointer Register |

| `0x10` | 2 | ?/? | unused/reserved |

| `0x12` | 2 | r/w | TRP - Transmit Buffer Read Page Pointer Register |

| `0x14` | 2 | r/w | RXINTT - Receive Interrupt Timer |

| `0x16` | 2 | r/w | RWP - Receive Buffer Write Page Pointer Register |

| `0x18` | 2 | r/w | RRP - Receive Buffer Read Page Pointer Register |

| `0x1a` | 2 | r/w | RHBP - RX High Boundary Page Pointer Register (0x0f00 ?) |

| `0x1c` | 1 | r/w | EEPROM Interface Register: |

bit(s)		Description
6-7		unused/reserved
5		EELD -
4		EESEL -
3		EEDO - Serial Data Output from external EEPROM clock device
2		EEDI - Serial Data Input to external EEPROM clock device
1		EECK - Serial Clock output to external EEPROM clock device (< 1 MHz)
0		EECS - Chip Select output to external EEPROM clock device

| `0x1d` | 1 | r/w | BICT - Bus Integrity Check Timer |

| `0x1e` | 2 | r/? | IORDP - IO Data Port Page Pointer Register |

| `0x20` | 6 | r/w | PAR0-PAR5 Network Address Filtering Registers - Physical (MAC) Address |

| `0x26` | 8 | r/w | MAR0-PAR7 Network Address Filtering Registers - Hash Table Register |

| `0x2e` | 1 | r/w | ANALOG - Transceiver Control Register |

bit(s)		Description
6-7		unused/reserved (must be 0)
5		RST100 - Reset for NORMAL mode
4		RSQ - Reduced SQuelch Enable
3		PWD100 - Reset for NORMAL mode
2		PWD10B - Set for NORMAL mode
1		DS130 - Must be 1 for NORMAL mode with auto-compensation
0		DS120 - Must be 1 for NORMAL mode with auto-compensation

| `0x2f` | 1 | r/w | DINTVAL - DMA Interval Timer |

`0x30`	1	r/w	NWAYC - NWAY Configuration Register

bit(s)		Description
7		LTE - Link Test Enable
6		NTTEST - reserved
3-5		ANS - Autonegotiation Status
2		ANE - Autonegotiation Enable
1		PS100/10 - Port Select 100/10 Mbit
0		FD - Full Duplex Mode

`0x31`	1	r/-	NWAYS - NWAY Status Register

bit(s)		description
7		10TXH - NWAY 10 TX Half duplex Mode
6		10TXF - NWAY 10 TX Full duplex Mode
5		100TXH - NWAY 100 Half duplex Mode
4		100TXF - NWAY 100 TX Full duplex Mode
3		ANCLPT - Auto-negotiation Completion
2		LPNWAY - Link Partner NWAY Status
1		LS100 - Physical Link Status of 100Mbps TP
0		LS10 - Physical Link Status of 10 Mbps TP

`0x32`	1	r/w	GCA - GMAC Configuration A Register (0x08 ?)

bit(s)		description
7		unused/reserved (must be 0)
6		TXFIFOCNTEN
5		AUTOPUB - Auto Page Update option
4		unused/reserved
3		ARXERRB - Accept RX packet with error
2		SLOWSRAM
1		PBW - Packet Buffer Data Width
0		BPSCRM - Bypass Scrambler

`0x33`	1	r/w	GCB - GMAC Configuration B Register

bit(s)		description
4-7		unused/reserved
2-3		RTHD - Recieve FIFO Threshold
0-1		TTHD - Transmit FIFO Threshold

`0x34`	4	-/w	TWD - IO Mapped Data Port

`0x38`	2	-/-	unused/reserved

| 0x3a | 1 | r/w | Host Interface Protocol Register (0x08 ?) |

```
7    0
```

bit(s)	description
2	DREQB - DREQB-pin status bit
1	STIORD/RRDYB - (no data available ?)
0	WRDYB - Write Packet Memory Ready Bar Status Indication

| 0x3b | 1 | r/- | LPC - Link Partner Link Code Register |

| 0x3c | 1 | r/w | TX/RX DMA Status Register |

| 0x3d | 1 | r/w | MISC1 - MISC Control Register 1 |

| 0x3e | 2 | r/w | TXFIFOCNT - TX FIFO Byte Counter |

| 0x40 | 4 | r/- | RRD - RX Burst Read Data Port |

| 0x44 | 2 | r/- | ID1 - 'MX' |

| 0x46 | 2 | r/- | ID2 - '0001' |

| 0x48 | 4 | w | WRTXFIFOD - Write TX FIFO Data Port Register (output queue) |

| 0x4c | 4 | r/- | IORD - IO Read Data Port Register |

| 0x50 | 1 | r/w | MISC2 - MISC Control Register 2 (0x80 ?) |

| 0x51 | 1 | ? | (?) |

note: register `0x51` is not documented in mx98728 datasheet

| 0x52 | 2 | r/- | HRPKTCNT - Host Recieve Packet Counter |

| 0x54 | 3 | r/w | FRAGCNT - Host DMA Fragment Counter |

note: `0x56` is the last reg of a mx98726 and mx98728

| 0x5b | ? | ?/? | (?) |

| 0x5c | ? | ?/? | (?) |

0x36 = disconnect network

| 0x5e | ? | ?/? | (0x01 ?) |

| 0x60 | ? | ?/? | (0x00 ?) |

| 0x100 | 0xf00 | ?/w | input queue (?) |

- select for reading
- IMM read block

10.8.2 command-registers

Registerblock Base	Size of Registerblock	common access size
0x00	0x10	1

These Registers are related to the EXI Interface on the Ethernet-Adapter, not to the actual Ethernet Chip.

| 0x00 | ? | ?/? | EXI id |

- select register for reading
- IMM read 4 bytes (0x04,0x02,0x02,0x00)

| 0x01 | ? | ?/? | ? |

| 0x02 | ? | ?/? | irq mask |

| 0x03 | ? | ?/? | irq status |

7			0
r??p			h???

bit(s)		description
7	r	irq from MX chip
6	?	'killing' irq (should not get this one)
5	?	command error
4	p	challange/response request
3	h	challange/response status

| 0x04 | ? | ?/? | ? (revid_eth) |

- select register for writing
- IMM write 0xd1,0x07 (16bit)

| 0x05 | ? | ?/? | ? (0x4e ?) |

| 0x06 | ? | ?/? | ? |

| 0x07 | ? | ?/? | ? |

| 0x08 | ? | ?/? | Challenge |

- select for reading
- IMM read 4 bytes

| 0x09 | ? | ?/? | Response |

- select for writing
- IMM write 4 bytes

| 0x0a | ? | ?/? | ? |

| 0x0b | ? | ?/? | Status |

1: challenge/response ok
2: challenge/response failed

| 0x0c | ? | ?/? | ? |

| 0x0d | ? | ?/? | ? |

| 0x0e | ? | ?/? | ? |

| 0x0f | ? | ?/? | ? (0x00 ?) |

10.8.3   Operation

10.8.3.1   selecting a register for reading

▷ IMM write 0x80000000 | (register<<8)

10.8.3.2   selecting a register for writing

▷ IMM write 0xc0000000 | (register<<8)

10.8.3.3   selecting command-register for reading

▷ IMM write (register<<8) (16 bits)

10.8.3.4   selecting command-register for writing

▷ IMM write 0x4000 | (register<<8) (16 bits)

10.8.3.5   init

   ▷ write reg - 1 byte to 0x60, 0x00

   ▷ read reg - 1 byte from 0x0f (?)

   ▷ delay

   ▷ write reg - 1 byte to 0x00, 0x01

   ▷ write reg - 1 byte to 0x00, 0x00 (not necessary ?)

   ▷ read reg - 1 byte from 0x01, 0x00

   ▷ read reg - 1 byte from 0x5b

   ▷ write reg - 1 byte to 0x5b, write back previously read value AND ~(1<<7)

   ▷ write reg - 1 byte to 0x5e, 0x01 (or same as 0x5b ?!)

   ▷ read reg - 1 byte from 0x5c

   ▷ write reg - 1 byte to 0x5c, write back previously read value OR 4

   ▷ write reg - 1 byte to 0x01, 0x11

   ▷ write reg - 1 byte to 0x50, 0x80 (?)

   ▷ write reg - 1 byte to 0x08, 0xff

   ▷ write reg - 1 byte to 0x09, 0xff

   ▷ write reg - 1 byte to 0x02, 0x00 (?)

   ▷ write reg - 1 byte to 0x00, 0x08

   ▷ delay

   ▷ write reg - 2 bytes to 0x16, 0x0100 (recv buffer write ptr)

   ▷ write reg - 2 bytes to 0x18, 0x0100 (recv buffer read ptr)

10.8.3.6   challenge/response calculation

```
u32 ETHChallResp(u32 val,u32 revid_0,u32 revid_eth_0,u32 revid_eth_1)
{
    u32 c0,c1,c2,c3;

    c0=(
          ( ((val&0xff000000)>>24) +
            ((val&0x00ff0000)>>16) * 0xc1 + 0x18 + revid_0 )
        ^ ( ((val&0x000000ff) * ((val&0x0000ff00)>>8)) + 0x90 )
        ) & 0xff;
```

```
  c1=(
        ( ((val&0x00ff0000)>>16) + ((val&0x0000ff00)>>8) + 0x90 )
       ^ ( c0 + ((val&0xff000000)>>24) - 0xc1 )
      ) & 0xff;


  c2=(
        ( ((val&0x0000ff00)>>8) + 0xc8 )
       ^ ( c0 + ((revid_eth_0 + revid_0 * 0x23) ^ 0x19) )
      ) & 0xff;


  c3=(
        ( ((val&0xff000000)>>24) + 0xc1 )
       ^ ( (val&0x000000ff) + ((revid_eth_1 + 0xc8) ^ 0x90) )
      ) & 0xff;


  return ((c0 << 24) | (c1 << 16) | (c2 << 8) | (c3));
}
```

10.8.3.7   send packet (outside interrupt)

▷ write reg - X bytes to 0x48, <senddata> (X must be >=0x3c!)

▷ read reg - 1 byte from 0x00, status ?

▷ write reg - 1 byte to 0x00, 0x00

▷ write reg - 1 byte to 0x00, previously read status OR 0x04 (set bit3, ACK error?)

▷ read reg - 1 byte from 0x00,status ;do while (status AND 0x06)!=0 (wait until packet sent)

10.8.3.8   poll received packets (outside interrupt)

▷ read reg - 2 bytes from 0x16, write_ptr

▷ read reg - 2 bytes from 0x18, read_ptr

▷ if write_ptr==read_ptr, no more packets are available

▷ write reg - 1 byte to 0x3a, 0x02

▷ read reg - 1 byte from 0x3a, if (value&2)!=0, no more data is available

▷ read reg- 4 bytes from read_ptr, packet descriptor

▷ read reg - X bytes from 0x100+read_ptr, <data> (X must be >=0x3c! ; wrap around to 0x100 if read_ptr=0xf00)

▷ write reg - 2 bytes to 0x18, first byte of packet descriptor, 0x00 (advance read pointer)

▷ write reg - 1 byte to 0x09, 0x02

▷ write cmd - 1 byte to 0x02, 0xf8 (ACK?)

10.8.3.9 received packet format first 4 bytes of a received packet contains a descriptor about the packet that has been received

```
31        24 23        16 15        8 7        0
....  .... | yyyy  .... | xxxx  xxxx | ....  ....
```

bit(s)		Description
20-23	y	length of received packet lo bits (shift 4 down)
8-15	x	length of received packet hi bits (shift 4 up and OR with lo bits)

length is inclusive the 4 byte descriptor!

10.9   UART

(note: the existance of an UART is highly speculative and was never proved for a fact)

to do

10.10   SD Cards

SD cards support an SPI mode, which is essentially the same hardware protocol that official memory cards use. Notice the use of the word hardware, not software. SD cards uses the MMC command set for communication and data transfer, while Nintendo memory cards use a custom Macronix command set. This means that you should not waste your time trying to get standard GC programs to recognize the SD card as a normal memory card, because it will not work without specific code to access the SD card. Even if you make the raw sector data on the SD card the same as the Nintendo memory card, the low-level commands to access the sectors differ between the two. However, since the hardware bus and protocol are the same, an SD adapter can essentially be made with straight-through connections from the EXI bus to the SD card signals.

to do

10.11   Viper 'Modchip'

this is a 3rd-Party 'modchip' used to override the IPL with a custom program.

todo

10.12   Ripper III GC 'Modchip'

todo

10.13   Qoob 'Modchip'

todo

10.14   NinjaMOD 'Modchip'

todo

10.15   Mario Party Microphone

todo

11 HSP Devices

HSP devices seem to be accessable through the ARAM interface with offsets beyond 16MB.

11.1 GB Player

to do

12 Memory Card Structure

12.1 Overview

one "block" on memcard equals 0x2000 bytes, the first 5 blocks are used for the filesystem (0xa000 bytes).

Offset	Size	Description
0x0000	0x2000	Header
0x2000	0x2000	Directory
0x4000	0x2000	Directory backup (*)
0x6000	0x2000	Block Allocation Map
0x8000	0x2000	Block Allocation Map backup (*)
0xa000		file(s) data

(*) If a change is to be made that will alter the Master File Table, such as moving or deleting a file, copying a file from another memory card, or creating a new save game file, the GameCube will first backup the Master File Table to this location. Presumably, if the operation fails for certain reasons, the GameCube will restore the Backup File Table to the Master File Table.

12.2 Header

Offset	Size	Description
0x0000		?
0x000c	8	time of format (OSTime value)
0x0014		unique card id (?)
0x0020	2	padding zeroes
0x0022	2	size of memcard in Mbits
0x0024	2	encoding (ASCII or japanese)
0x0026		unused (0xff)
0x01fa	2	update Counter (?, probably unused)
0x01fc	2	Checksum 1 (?)
0x01fe	2	Checksum 2 (?)
0x0200	0x1e00	unused (0xff)

12.3 Directory

Offset	Size	Description
0x0000		Directory Entries (max 127)
0x0ffa	2	update Counter
0x0ffc	2	Checksum 1
0x0ffe	2	Checksum 2

12.3.1 Directory Entries

offset	length	description		
0x00	0x04	Gamecode		
0x04	0x02	Makercode		
0x06	0x01	reserved/unused (always 0xff, has no effect)		
0x07	0x01	banner gfx format and icon animation (Image Key)		
		bit(s)	**description**	
		2	Icon Animation 0: forward 1: ping-pong	
		1	0: No Banner 1: Banner present	
		0	Banner Color 0: RGB5A3 1: CI8	
0x08	0x20	filename		
0x28	0x04	Time of file's last modification in seconds since 12am, January 1st, 2000		
0x2c	0x04	image data offset		
0x30	0x02	icon gfx format (2bits per icon)		
		bits	**Description**	
		00	no icon	
		01	CI8 with a shared color palette after the last frame	
		10	RGB5A3	
		11	CI8 with a unique color palette after itself	
0x32	0x02	animation speed (2bits per icon) (*1)		
		bits	**Description**	
		00	no icon	
		01	Icon lasts for 4 frames	
		10	Icon lasts for 8 frames	
		11	Icon lasts for 12 frames	
0x34	0x01	file-permissions		
		bit	**permission**	**Description**
		4	no move	File cannot be moved by the IPL
		3	no copy	File cannot be copied by the IPL
		2	public	Can be read by any game
0x35	0x01	copy counter (*2)		
0x36	0x02	block no of first block of file (0 == offset 0)		
0x38	0x02	file-length (number of blocks in file)		
0x3a	0x02	reserved/unused (always 0xffff, has no effect)		
0x3c	0x04	Address of the two comments within the file data (*3)		

(*1) Clearly, the animation rate is unimportant when there is only one frame of icon data; nevertheless, a value for that one frame must still be set, or that one frame will not be shown. It is illegal to specify that a frame does not exist if it does; a value of 00 indicates that no frame exists, and should not be mistaken for meaning that this frame should not be shown. If you specify blank frames to slow the frame rate, these also cannot be 00.

(*2) This byte contains an 8-bit integer that indicates how many times the file has been copied from one memory card to another.

(*3) Each file has two 32 character strings which the IPL displays at the bottom of the memory card screen, next to the banner. The two strings (64 bytes) must fit within one block (8192 bytes), they are not allowed to cross sector boundaries.

12.3.1.1   Image Data   Image data consists of a banner image and an icon. The banner image is not required, dependant on the value of the Image Key. If the banner image is not present, the icon image is displayed where the banner image would be displayed (centered horizontally). The icon image is required, and immediately follows the banner if present. Otherwise, it is located at the start of the image data.

12.3.1.1.1   Banner Image   The banner size is 96*32 pixels, making 3072 pixels in total (= 0x0c00 bytes in 8bit, 0x1800 bytes in 16bit mode). If the Banner is in CI8 mode, the palette follows immediately after the banners pixel data.

12.3.1.1.2   Icon Image   Immediately following the banner (if present) is the Icon Image. This can have a variable number of frames (up to eight), each 32*32 pixels, making 1024 pixels per frame in total. (= 0x0400 bytes in 8bit, 0x0800 byte in 16bit mode). If the Icon is in CI8 mode, its palette either follows immediately after its pixel data or after the pixel data of all 8 icons, depending on the icon gfx format field.

12.3.1.1.3   Palettes   Palettes in the image data are in RGB5A3 pixel format, and are 0x100 entries large. (= 0x200 bytes)

12.4   Block Allocation Map

Offset	Size	Description
0x0000	2	Checksum 1
0x0002	2	Checksum 2
0x0004	2	update Counter
0x0006	2	free Blocks
0x0008	2	last allocated Block
0x000a	0x1ff8	Map of allocated Blocks

This is an array of 0x0ffc 16 bit values, each holding info about one allocated block on the memory card. (thus the maximum memcard size is limited to 2048 blocks (16 Megabytes, 128Mbit))

each 16 bit value at position X in the array has the following meaning:

value	Description
0x0000	block is not allocated (ie, free)
0xffff	last allocated block of a file
any other	allocated block, usually equals (x+1) (==next block of file)

scan through a file like this:

```
thisblock=firstblock;

do

{

    // process block

    // next block

    thisblock=((unsigned short*)0x6000)[thisblock];

}
```

```
while (thisblock!=0xffff);
```

note:

although this scheme could do it, i have never stumbled about a file yet that is NOT linear on the memcard anyway. from this point of view using this allocation map seems to be a bit stupid...more testing needed :)

12.5   Checksums

The Checksums for the Directory and Block Allocation Map are simple 16bit additive checksums (ie nothing fancy or particular safe) which can be easily calculated like this:

```
void checksums(unsigned short *buf, int num, unsigned short *c1, unsigned short *c2)

{
    int i;
    *c1 = 0;*c2 = 0;
    for (i = 0; i < num; ++i)
    {
        *c1 += buf[i];
        *c2 += (buf[i] ^ 0xffff);
    }
    if (*c1 == 0xffff)
    {
        *c1 = 0;
    }
    if (*c2 == 0xffff)
    {
        *c2 = 0;
    }
}
```

13 DVD Structure

total capacity of disc data is 1,459,978,240 bytes (1.5 GB approx.). that's exactly 712880 DVD raw sectors (each 2048 bytes).

start	end	size	Description
0x00000000		0x0440	Disk header ("boot.bin")
0x00000440		0x2000	Disk header Information ("bi2.bin")
0x00002440		(0x2000 ?)	Apploader ("appldr.bin")
			FST ('fst.bin')

13.1 Disk header

start	end	size	Description		
0x0000	0x0003	0x0004	Game Code		
			1	Console ID	
			2	Gamecode	
			1	Country Code	
0x0004	0x0005	0x0002	Maker Code		
0x0006		0x0001	Disk ID		
0x0007		0x0001	Version		
0x0008		0x0001	Audio Streaming		
0x0009		0x0001	Stream Buffer Size		
0x000a	0x001b	0x0012	unused (zeros)		
0x001c	0x001f	0x0004	DVD Magic Word (0xc2339f3d)		
0x0020	0x03ff	0x03e0	Game Name		
0x0400	0x0403	0x0004	offset of debug monitor (dh.bin) ?		
0x0404	0x0407	0x0004	addr (?) to load debug monitor ?		
0x0408	0x041f	0x0018	unused (zeros)		
0x0420	0x0423	0x0004	offset of main executable DOL (bootfile)		
0x0424	0x0427	0x0004	offset of the FST ("fst.bin")		
0x0428	0x042B	0x0004	size of FST		
0x042C	0x042F	0x0004	maximum size of FST (usually its same as FST size) (*)		
0x0430	0x0433	0x0004	user position (?)		
0x0434	0x0437	0x0004	user length (?)		
0x0438	0x043b	0x0004	(?)		
0x043c	0x043f	0x0004	unused (zeros)		

(*) multiple DVDs must use it, to properly reside all FSTs.

13.2 Disk header Information

this is loaded to the Address in 0x800000f4 when a disc is initialized by the IPL

offset	end	size	Description
0x0000		4	Debug-monitor Size
0x0004		4	Simulated Memory Size
0x0008		4	Argument offset
0x000c		4	Debug flag
0x0010		4	Track Location
0x0014		4	Track size
0x0018		4	Countrycode
0x001c		4	?

13.3 Apploader

offset	end	size	Description
0x0000	0x0009		Date (version) of the apploader in ASCII
0x000A	0x000F		padding (0)
0x0010	0x0013	4	Apploader entrypoint
0x0014	0x0017	4	size of the apploader (32 bit) (usually 0x2000)
0x0018	0x001b	4	trailer size
0x0020			Apploader code (loaded to 0x81200000 in RAM)

13.4 Format of the FST

start	end	size	Description
0x00	0x0c	0x0c	Root Directory Entry
0x0c	...	0x0c	more File- or Directory Entries
...	...	...	String table

13.4.1 Format of a File Entry

start	end	size	Description
0x00		1	flags; 0: file 1: directory
0x01		3	filename, offset into string table
0x04		4	file_offset or parent_offset (dir)
0x08		4	file_length or num_entries (root) or next_offset (dir)

14 general File Formats

14.1 BNR (Banner file format)

this is the format of the file 'opening.bnr' (file size: always 6.496 bytes) found in the root directory of every Gamecube disc. This file is the little image that is displayed in the cube menu when inserting a disc into the gamecube, when in menu mode.

start	end	size	Description
0x0000	0x0003	0x0004	Magic Word "BNR1" (US/JP) or 'BNR2' (EU)
0x0004	0x001f		padding zeroes
0x0020	0x181f	0x1800	Graphical Data (Pixel-format is RGB5A1)
0x1820	0x183f	0x0020	Gamename (*)
0x1840	0x185f	0x0020	Company/Developer (*)
0x1860	0x189f	0x0040	Full Game Title (*)
0x18a0	0x18df	0x0040	Company/Developer Full name, or description (*)
0x18e0	0x195f	0x0080	Game Description (*)

(*) All Text is all stored in either SHIFT-JIS or ASCII, depending on the region of the Game.

note: In the filesystem of european Games with multi-lingual text there may be several .bnr files (opening.bnr, openingUS.bnr, openingEU.bnr, openingJP.bnr). The opening.bnr is a BNR2 file, it is just like a regular BNR file, except that the metadata at the end repeats several times in different languages. 0x1820 through 0x1960 are the first, and it continues in blocks of 0x0140.

14.2 DOL (Gamecube Executable)

This is a custom GameCube program file format, which is directly booted by GameCubes' BIOS (to be exact, by the apploader on retail discs. a different apploader could well load a binary in whatever different format.).

start	end	size	description
0x0000	0x001B		Text[0..6] sections File Positions
0x001C	0x0047		Data[0..10] sections File Positions
0x0048	0x0063		Text[0..6] sections Mem Address
0x0064	0x008F		Data[0..10] sections Mem Address
0x0090	0x00AB		Text[0..6] sections Sizes
0x00AC	0x00D7		Data[0..10] sections Sizes
0x00D8		0x04	BSS Mem address
0x00DC		0x04	BSS Size
0x00E0		0x04	Entry Point
0x00e4		0x1c	unused
0x0100			Start of sections data (body)

14.3 ELF (Executable and linkable Format)

The ELF format is a standard, known format for debugging target specific code, etc. GCC targeted for the PPC 750 processor or even for the specialized Gekko processor has a final output of ELF format files. The exact GameCube ELF file format details are currently unknown, but they should be similar to the standard specification.

14.4   GCB (QOOB Flash Files)

start	end	size	description		
0x00	0x03	4	ID, indicates whats in the block		
			0x28432920	'(C) ' - qoob bios file	
			0x454c4600	'ELF\0' - ELF File	
			0x42494e00	(not yet) 'BIN\0'	
			0x444f4c00	(not yet) 'DOL\0'	
0x04	0xf7		description, will be shown in boot menu (by the qoob bios)		
0xf8	0xfb	4	reserved		
0xfc	0xff	4	size of block		
0x100	...		data		

14.5   GCM (Gamecube Disc Image)

These files are always 1.4GB's exactly and each contains a complete binary image of a proprietary format GameCube DVD. This file format is used for the NR-Writer DVD writing software which writes special DVDs that can only be read by NR-Reader GameCubes. The GCM file format can probably be closely compared to the ISO file format for CDs in its purpose.

14.6   GCI (Gamecube Game Save)

Used by the EMS Memory Adapter.

64 byte header (equal to FST entry on memcard), followed by the file data (as on memory card)

14.7   GCP (Gamecube Memorycard Image)

Used by the EMS Memory Adapter.

this is a raw image of all blocks of a memory card.

14.8   TGC

a proprietary image format found on demo discs and eg the zelda n64 emu

note: there seem to be tgc files on european discs that follow a different layout (no header).

14.8.1   Header

start	end	size	description
0x0000	0x0003	0x0004	TGC-Magic (0xae0f38a2)
0x0004	0x0007	0x0004	? (=0x00000000)
0x0008	0x000b	0x0004	TGC-Header Size (=0x00008000)
0x000c	0x000f	0x0004	? (=0x00100000)
0x0010	0x0013	0x0004	Offset to FST inside embedded GCM
0x0014	0x0017	0x0004	FST Size
0x0018	0x001b	0x0004	max FST Size
0x001c	0x001f	0x0004	Offset to Boot-DOL inside embedded GCM
0x0020	0x0023	0x0004	Boot-DOL Size
0x0024	0x0027	0x0004	?
0x0028	0x002b	0x0004	?
0x002c	0x002f	0x0004	Offset to Banner inside embedded GCM ?
0x0030	0x0033	0x0004	Banner Size ?
0x0034	0x0037	0x0004	?

14.8.2   embedded GCM

usually starts at offset 0x00008000 (after the TGC Header) and follows exactly the same layout as a GCM file, with the following exceptions:

▷ Boot-DOL offset, FST offset contain bogus data and must be substituted by the data found in the TGC header

▷ offsets within the embedded GCM file must be calculated relative to the start of the embedded GCM (obviously)

14.9   VGC (Viper Flash Files)

start	end	size	description		
0x00	0x03	4	Viper Magic ('VIPR')		
0x04		1	Configuration Flags		
			bit(s)	**description**	
			7	GC_FLASH_ACCESS - allows access to the flashrom	
			6	?	
			5	?	
			4	?	
			3	COMMAND_MODE - enables modchip command mode (requires extended mode)	
			2	EXTENDED_MODE - allows reading of original IPL	
			1	COBRA_ENCRYPTION - enables additional encryption mode	
			0	DISABLE_CHIP - disables the modchip	
0x05		1	Lid Sensor Status at Boot Time (*2)		
			0	LID_OPEN	
			1	LID_CLOSED	
			2	LID_PASSTHROUGH	
0x06	0x0f	10	padding (zeros)		
0x10	0x1f	16	BIOS Name in Ascii		
0x20	...		Encrypted (*1) BIOS, loaded to 0x81300000		

(*1) encrypted with the IPL XOR-Stream

(*2) original docs state that the default value is `0xff`, however actually using this value seems to cause problems.

15 Game File Formats

This Section contains information about files used in, or produced by the official SDK, and thus is primarily useful for those who are hacking retail games.

15.1 AFC (audio stream)

15.2 AST (audio stream)

like afc but with tags?

15.3 ARC (RARC Archive)

This file is an archive file and contains several other files.

15.3.1 Header

The file starts with an Rarc-Header:

start	end	size	description
		4	type - 'RARC'
		4	size, size of the file
		4	unknown
		4	dataStartOffset, where does the actual data start? You have to add 0x20 to this value.
		16	unknown
		4	numNodes
		8	unknown
		4	fileEntriesOffset
		4	unknown
		4	stringTableOffset, where is the string table stored? You have to add 0x20 to this value.
		8	unknown

15.3.2 Nodes

Next are RarcHeader.numNodes Node structures:

start	end	size	description
		4	type
		4	filenameOffset, directory name, offset into string table
		2	unknown
		2	numFileEntries, how many files belong to this node?
		4	firstFileEntryOffset

Each RARC file contains at least one Node, the 'ROOT' node. For each subdirectory in the archive, there's another Node (so each Node represents a directory). Each Node contains files and directories, represented by FileEntry structures:

15.3.3 File Entries

start	end	size	description
		2	id, file id. If this is 0xFFFF, then this entry is a subdirectory link
		2	unknown
		2	unknown
		2	filenameOffset, file/subdir name, offset into string table
		4	dataOffset, offset to file data (for subdirs: index of Node representing the subdir)
		4	dataSize, size of data
		4	zero, seems to be always '0'

To read the archive, you read the root node and its file entries. For each subdir in the root node's fileentries, you read the corresponding node and its file entries. For each file in the fileentries, you dump its data.

15.4 ARC (audio stuff)

if a .arc file doesnt start with 'RARC' it may contain audio data

15.5 ASN

audio related, contains strings

15.6 AW ("audio wave"?)

15.7 BAS ("audio script" ?)

seems to have to do with audio (check mkdd file names...)

15.8 BCA

looks very similar to a .col file, only with some tags in it

15.9 BCK (animation of a .bmd skeleton)

15.10 BDL

same as .bmd

15.11 BFN (font)

images of characters + mapping from character code to corresponding image part

15.12 BIN (binary file)

scene.bin in sms contains scene layout

15.13   BLO (screen layout for dialog screens)

15.14   BMD (3d model with texture and skeleton)

15.15   BMG

messages, subtitles, ... (text)

15.16   BMP (window bitmap (!))

15.17   BMT

seems to contain a MAT3 block of a .bmd file

15.18   BCK ("Pack" file)

15.19   BRK

animation stuff? rotation keys?

15.20   BTI

Note: some .bti files are Yaz0-compressed (if the first 4 bytes are 'Yaz0'), if this is the case you have to uncompress them first.

A .bti file stores a single image, but can store several mipmaps. The file starts with a texture-header (which is used in the TEX! section of bmd/bdl and jpa files to store textures aswell):

15.20.1 Texture Header

start	end	size	description		
`0x00`		1	format		
			0	I4 (4 bit intensity, 8x8 tiles)	
			1	I8 (8 bit intensity, 8x4 tiles)	
			2	IA4 (4 bit intensity with 4 bit alpha, 8x4 tiles)	
			3	IA8 (8 bit intensity with 8 bit alpha, 4x4 tiles)	
			4	RGB565 (4x4 tiles)	
			5	RGB5A3 (*) (4x4 tiles)	
			6	RGBA8 (4x4 tiles in two cache lines - first is AR and second is GB)	
			8	CI4 (4 bit color index, 8x8 tiles)	
			9	CI8 (8 bit color index, 8x4 tiles)	
			10	CI14X2 (14 bit color index, 4x4 tiles)	
			14	CMP (S3TC compressed, 2x2 blocks of 4x4 tiles)	
`0x01`		1	unknown		
`0x02`	`0x03`	2	width		
`0x04`	`0x05`	2	height		
`0x06`	`0x07`	2	unknown		
`0x08`		1	unknown		
`0x09`		1	Palette Format		
			0	IA8	
			1	RGB565	
			2	RGB5A3 (*)	
`0x0a`	`0x0b`	2	Palette Entries - number of entries in the Palette		
`0x0c`	`0x0f`	4	Palette Offset - offset to Palette Data		
`0x10`	`0x13`	4	unknown		
`0x14`	`0x15`	2	unknown		
`0x16`	`0x17`	2	unknown		
`0x18`		1	mipmap count		
`0x19`		1	unknown		
`0x1a`	`0x1b`	2	unknown		
`0x1c`	`0x1f`	4	Data Offset - offset to image Data		

(*) RGB5A3 is RGB5 if color value is negative and RGB4A3 otherwise.

Offsets are relative to the Texture Header (this is important in bmd/bdl files).

15.21 BTP

99% sure that this contains texture animation (NOT texture coordinate animation)

15.22 BTK

(curves??? kinematics??) translation keys? probably some material animation as well (texture coord anim?)

15.23 COL (collision triangles)

15.24 DZB

collision data ?

15.25   H4M

a proprietary Movie Format found on some Gamecube Game DVDs. Probably related to the HVQ (Hirarchical Vector Quantization) format developed by Hudson.

15.26   JPA (particle data)

(TEX1 section contains .bti images)

15.27   JPC

collection of .jpa files ("Particle paCk"?)

15.28   MTH ('Mute thp?')

video format, has the same video frame format as thp, but headers are a bit different

15.29   PAD

recorded controller data?

15.30   PRM ('Parameters?')

15.31   REL (relocatable module)

some kind of .dll or similar, contains text and data sections and relocation info

15.32   SB

?, contains a stringtable

15.33   SZS (packed RARC Archive)

This is a Yaz0-compressed RARC archive

15.34   THP (video format)

.thp is a video format on the gamecube. The video frames are independent "quasi-jpegs", and if audio frames are present, they are in an adpcm format (described below).

15.34.1 Header data

The file starts with a thp header:

start	end	size	description
0x00	0x03	4	Magic Bytes, 0x54485000 ('THP\0')
0x04	0x07	4	Version
			0x00010000 — v1.0
			0x00011000 — v1.1
0x08	0x0b	4	maxBufferSize - maximal buffer size needed for one complete frame (header + video + audio)
0x0c	0x0f	4	maxAudioSamples - != 0 if sound is stored in file, maximal number of samples in one frame.
0x10	0x13	4	FPS (float value)
			0x41efc28f — ~29; NTSC
0x14	0x17	4	numFrames - number of frames in the thp file
0x18	0x1b	4	firstFrameSize - size of first frame (header + video + audio)
0x1c	0x1f	4	dataSize - size of all frames (not counting the thp header structures)
0x20	0x23	4	componentDataOffset - ThpComponents stored here (see below)
0x24	0x27	4	offsetsDataOffset - if != 0, offset to table with offsets of all frames?
0x28	0x2b	4	firstFrameOffset - offset to first frame's data
0x2c	0x2f	4	lastFrameOffset - offset to last frame's data

15.34.2 Components structure

At ThpHeader.componentDataOffset, a ThpComponents structure is stored:

start	end	size	description
0x00	0x03	4	numComponents - usually 1 or 2 (video or video + audio)
0x04	0x13	16	componentTypes - each byte specifies the type of one component as follows:
			0x00 — video
			0x01 — audio
			0xff — no component

The first ThpComponents.numComponents entries of ThpComponents.componentTypes are valid. For each component, an information structure is stored after the ThpComponents struct.

15.34.3 VideoInfo Structure

Component type 0 is video, a ThpVideoInfo struct looks like this:

start	end	size	description
0x00	0x03	4	width
0x04	0x07	4	height
0x08	0x0b	4	unknown (only v1.1 files)

15.34.4 AudioInfo Structure

Component type 1 is audio (not always included), a ThpAudioInfo struct looks like this:

start	end	size	description
0x00	0x03	4	numChannels
0x04	0x07	4	frequency
0x08	0x0b	4	numSamples
0x0c	0x0f	4	numData (only for v1.1 files) - amount of audio blocks stored after each video block

15.34.5   Frame data

A frame is made up of a frame header followed by a video frame followed by ThpAudioInfo.numData audio frames (only if the video contains sound).

The frame header consists of 3 (or 4, if the video contains sound) 32bit values:

start	end	size	description
0x00	0x03	4	nextTotalSize - total size of NEXT frame (frame header, video and audio)
0x04	0x07	4	prevTotalSize - total size of PREVIOUS frame
0x08	0x0b	4	imageSize - size of image frame of THIS frame
0x0c	0x0f	4	audioSize - size of one audio frame of THIS frame (only if the file contains audio)

Directly after the frame header FrameHeader.imageSize bytes video information follow. Directly after the video information, ThpAudioInfo.numData audio frames follow, each Frameheader.audioSize bytes large (only if the file contains audio).

15.34.6   Video Frames

A video frame is more or less a jpeg image. A jpeg file is structured by several markers. A marker is a two-byte code, the first of the two bytes is `0xff`. The jpeg standard states that if you want to store the value `0xff`, you have to store it as `0xff 0x00` (else it would be confused with a marker). This is NOT the case in .thp files, the value `0xff` is stored simply as `0xff` in the image data. So if you want to use jpeglib to read the frame, you have to convert the thp "quasi-jpeg" to a real jpeg by converting `0xff` values to `0xff 0x00` in the image data. You have to be careful that you don't convert the terminating End-Of-Image marker, though.

- search for Start-Of-Image marker (`0xff 0xda`)

- search for End-Of-Image marker (`0xff 0xd9`) (start search at end of buffer and search backwards!)

- convert each `0xff` between image data start and image data end to `0xff 0x00`

- the resulting buffer can be passed to jpeglib to let it decode the image for you

15.34.7   Audio Frames

An audio frame starts with a ThpAudioFrameHeader (total size is 80 bytes)

start	end	size	description
0x00	0x03	4	channelSize - size of one channel in bytes (*1)
0x04	0x07	4	numSamples - number of samples/channel
0x08	0x27	32	table for first channel (*2)
0x28	0x47	32	table for second channel (stored for one channel videos as well) (*2)
0x48	0x49	2	signed value, channel1Prev1
0x4a	0x4b	2	signed value, channel1Prev2
0x4c	0x4d	2	signed value, channel2Prev1
0x4e	0x4f	2	signed value, channel2Prev2

(*1) `audio frame size = sizeof(ThpAudioFrameHeader) + ThpAudioInfo.numChannels * ThpAudioFrameHeader.`

(*2) tables stored as 16bit signed 5.11 fixed point numbers

Directly after the ThpAudioFrameHeader ThpAudioFrameHeader.channelSize bytes follow for the first channel, and if the video is stereo (ThpAudioInfo.numChannels = 2), that many bytes follow for the second channel.

The audio data is made up of small packets of 8 byte, each packet contains 14 samples. Some kind of adpcm coding is used. A sample is calculated like this:

```
newSample = previousSample*factor1 + sampleBeforePreviousSample*factor2 + (sampleData
* 2^exponent);
```

For each packet, the first byte stores factor1, factor2 and exponent:

```
u8 index = (firstByte >> 4) & 0x7; //highest bit of byte is ignored
```

```
u8 exponent = firstByte & 0xf;
```

```
float factor1 = ThpAudioFrameHeader.table[2*index]/pow(2.f, 11);
```

```
float factor2 = ThpAudioFrameHeader.table[2*index + 1]/pow(2.f, 11);
```

The following 7 bytes store 14 sampleData (each 4 bit, interpreted as a signed two's complement number).

15.35   TPL (Texture Palette)

Another custom GameCube file format that holds texture and texture palette data. Many textures can be stored in one TPL file format, and it is commonly used by the SDK to hold texture data for GameCube games.

note: appearently there are different formats of .TPL files, the following applies only to those with the magic `0x00,0x20,0xaf,0x30`.

TPL Header

start	end	size	description
0x0000	0x0003	4	Magic (`0x00, 0x20, 0xAF, 0x30`)
0x0004	0x0007	4	ntextures - Number of Textures in File
0x0008	0x000b	4	size of Header (always `0x0c` in files with this structure)

TPL Texture

After the header goes 'ntextures' times the TPLTexture structure:

start	end	size	description
		4	Texture Header Offset
		4	Texture Palette Offset (0 if no palette)

TPL Texture Header

For every texture at position 'Texture Header Offset' there is the TPL Texture Header:

start	end	size	description
		2	height
		2	width
		4	format

0	I4 (4 bit intensity, 8x8 tiles)
1	I8 (8 bit intensity, 8x4 tiles)
2	IA4 (4 bit intensity with 4 bit alpha, 8x4 tiles)
3	IA8 (8 bit intensity with 8 bit alpha, 4x4 tiles)
4	RGB565 (4x4 tiles)
5	RGB5A3 (*) (4x4 tiles)
6	RGBA8 (4x4 tiles in two cache lines - first is AR and second is GB)
8	CI4 (4 bit color index, 8x8 tiles)
9	CI8 (8 bit color index, 8x4 tiles)
10	CI14X2 (14 bit color index, 4x4 tiles)
14	CMP (S3TC compressed, 2x2 blocks of 4x4 tiles)

start	end	size	description
		4	offset to Texture Data
		4	wrap s
		4	wrap t
		4	min filter
		4	mag filter
		4	lod bias (float value)
		1	edge lod
		1	min lod
		1	max lod
		1	unpacked

TPL Palette Header

For every palette (not every texture has one) there is the TPL Palette Header:

start	end	size	description
		2	nitems
		1	unpacked
		1	pad
		4	format

0	IA8
1	RGB565
2	RGB5A3 (*)

start	end	size	description
		4	offset to Palette Data

(*) RGB5A3 is RGB5 if color value is negative and RGB4A3 otherwise.

15.36   YMP (height map)

16 Compression Formats

16.1 Yay0

This format is used to store the fonts in the BIOS/IPL. It is compressed similar to the the Zelda 64 'Yaz0' compression.

16.1.1 compression

start	end	size	description
0x0000		4	'Yay0' signature
0x0004		4	size of decoded data in bytes
0x0008		4	offset to link table
0x000c		4	offset to non-linked chunks and count modifiers table
0x0010			packed data (32 bit words)

The packed data is a bitstream (padded to a multiple of 32bits), with each bit having the following meaning:

0	linked chunk, copy block from the link table (offset 0x0008)
1	non linked chunk, copy next byte from non-linked chunks and count modifiers table (offset at 0x000c)

todo

16.1.2 de-compression Code

```
void Decode(void *s, void *d)
{
u32 i, j, k;
u32 p, q;
u32 cnt;
i = r21 = *(u32 *)(s + 4); // size of decoded data
j = r29 = *(u32 *)(s + 8); // link table
k = r23 = *(u32 *)(s + 12); // byte chunks and count modifiers
q = r31 = 0; // current offset in dest buffer
cnt = r28 = 0; // mask bit counter
p = r24 = 16; // current offset in mask table
do
{
// if all bits are done, get next mask
if(cnt == 0)
{
// read word from mask data block
r22 = *(u32 *)(s + p);
p += 4;
cnt = 32; // bit counter
}
// if next bit is set, chunk is non-linked
if(r22 & 0x80000000)
{
// get next byte
*(u8 *)(d + q) = *(u8 *)(s + k);
k++; q++;
```

```
}
// do copy, otherwise
else
{
// read 16-bit from link table
r26 = *(u16 *)(s + j);
j += 2;
// 'offset'
r25 = q - (r26 & 0xfff);
// 'count'
r30 = r26 >> 12;
if(r30 == 0)
{
// get 'count' modifier
r5 = *(u8 *)(s + k);
k++;
r30 = r5 + 18;
}
else r30 += 2;
// do block copy
r5 = d + r25;
for(i=0; i<r30; i++)
{
*(u8 *)(d + q) = *(u8 *)(r5 - 1);
q++; r5++;
}
}
// next bit in mask
r22 <<= 1;
cnt--;
} while(q < i);
}
```

16.1.3   Font Data

start	end	size	description
0x0000		2	Font Type
0x0002		2	first Character in Font
0x0004		2	last Character in Font
0x0006		2	Character to use for substituting invalid Characters
0x0008		2	ascent Units
0x000a		2	descent Units
0x000c		2	width of widest Character
0x000e		2	leading Space
0x0010		2	Cell width
0x0012		2	Cell Height
0x0014		4	Texture Size
0x0018		2	Texture Format
0x001a		2	Texture Columns
0x001c		2	Texture Rows
0x001e		2	Texture Width
0x0020		2	Texture Height
0x0022		2	offset to Character-width Table
0x0024		4	offset to Tile-Data
0x0028		4	Tile-Data Size

16.2   Yaz0

Yaz0 compression is reportedly used in quite a few Nintendo datafiles. I have seen it in SuperMario Sunshine's .szs files for example, and I heard that it is used in Windwaker and Majoras Mask as well.

The first 16 bytes of a Yaz0-compressed data block are the data header. The first 4 bytes of the header are 'Y', 'a', 'z', '0', so you can easily see in your hex editor that there's a Yaz0 block waiting for you :-) The second 4 bytes are a single uint32 (big-endian of course) that tells you the size of the decompressed data, so you know how large your working buffer has to be. The next 8 bytes are always zero.

Next comes the actual compressed data. Yaz0 is some kind of RLE compression. You decode it as follows: First you read a "code" byte that tells you for the next 8 "read operations" what you have to do. Each bit of the "code" byte represents one "read operation" (from left to right, that is, 0x80 first, 0x01 last). If the bit is 1, copy one byte from the input buffer to the output buffer. Easy. If the bit is 0, things are a little bit more complicated, RLE compressed data is ahead. You have to read the next two bytes to decide how long your run is and what you should write to your output buffer.

15	8	7	0
a	b		

The upper nibble of the first byte (a) contains the information you need to determine how many bytes you're going to write to your output buffer for this "read operation". if a == 0, then you have to read a third byte from your input buffer, and add 0x12 to it. Otherwise, you simply add 2 to a. This is the number of bytes to write ("count") in this "read operation". byte2 and the lower nibble of byte1 (b) tell you from where to copy data to your output buffer: you move (dist = (b << 8)|byte2 + 1) bytes back in your outputBuffer and copy "count" bytes from there to the end of the buffer. Note that count could be greater than dist which means that the copy source and copy destination might overlap.

16.2.1   de-compression Code

```
//src points to the yaz0 source data (to the "real" source data, not at the header!)
//dst points to a buffer uncompressedSize bytes large (you get uncompressedSize from
```

```
//the second 4 bytes in the Yaz0 header).

void decode(u8* src, u8* dst, int uncompressedSize)
{
int srcPlace = 0, dstPlace = 0; //current read/write positions
u32 validBitCount = 0; //number of valid bits left in "code" byte
u8 currCodeByte;
while(dstPlace < uncompressedSize)
{
//read new "code" byte if the current one is used up
if(validBitCount == 0)
{
currCodeByte = src[srcPlace];
++srcPlace;
validBitCount = 8;
}
if((currCodeByte & 0x80) != 0)
{
//straight copy
dst[dstPlace] = src[srcPlace];
dstPlace++;
srcPlace++;
}
else
{
//RLE part
u8 byte1 = src[srcPlace];
u8 byte2 = src[srcPlace + 1];
srcPlace += 2;
u32 dist = ((byte1 & 0xF) << 8) | byte2;
u32 copySource = dstPlace - (dist + 1);
u32 numBytes = byte1 >> 4;
if(numBytes == 0)
{
numBytes = src[srcPlace] + 0x12;
srcPlace++;
}
else
numBytes += 2;
//copy run
for(int i = 0; i < numBytes; ++i)
{
dst[dstPlace] = dst[copySource];
copySource++;
dstPlace++;
}
}
//use next bit from "code" byte
currCodeByte <<= 1;
validBitCount-=1;
}
}
```

17 Graphic Formats

17.1 YCbYCr

This is the Format used for image data in the external framebuffer (XFB). It exploits the fact that the resolution of color on a PAL/NTSC screen is lower than the resolution of luminance (brightness), and thus stores only separate luminance info for each pixel and combines the color information of two pixels each, saving 2 bytes versus traditional RGB-per-pixel framebuffers. This means that in XFB you cant modify the color of a single pixel without affecting its neighbour. (you can however, seperatly modify its luminance/brightness). It also means that you can not accurately convert one single pixel into XFB framebuffer format, you will always have to convert two pixels at once.

To convert two pixels to YCbYCr, first average their RGB values

```
R = (R1+R2)/2
G = (G1+G2)/2
B = (B1+B2)/2
```

now calculate the luminance portion of each pixel

```
Y1 = (77/256)R1 + (150/256)G1 + (29/256)B1
Y2 = (77/256)R2 + (150/256)G2 + (29/256)B2
```

then calculate the combined color portion

```
Cb = -(44/256)R -  (87/256)G + (131/256)B + 128
Cr = (131/256)R - (110/256)G -  (21/256)B + 128
```

now a 32 bit value to be written to XFB (to a 32 bit aligned address of course) can be made up like this

31 24	23 16	15 8	7 0
1111 1111	bbbb bbbb	2222 2222	rrrr rrrr

bit(s)		description
24-31	1	Y1 - luminance Portion of first Pixel
16-23	b	Cb - combined color
8-15	2	Y2 - luminance Portion of first Pixel
0-7	r	Cr - combined color

converting a single pixel back to RGB looks like this

```
R = Y + 1.371(Cr - 128)
G = Y - 0.698(Cr - 128) - 0.336(Cb - 128)
B = Y + 1.732(Cb - 128)
```

17.2   I4 (4bit indexed)

17.3   IA4 (4bit indexed with alpha)

17.4   I8 (8bit indexed)

17.5   IA8 (8bit indexed with alpha)

17.6   CI4 (compressed 4bit indexed)

17.7   CIA4 (compressed 4bit indexed with alpha)

17.8   CI8 (compressed 8bit indexed)

Used for Icons and Banners on Memory Card. This Format uses a palette in RGB5A1 Format, the Pixel data is stored in 8x4 pixel tiles.

17.9   CIA8 (compressed 8bit indexed with alpha)

17.10   RGB4A3

Used for Icons and Banners on Memory Card. This Format uses no palette and is stored in 4x4 pixel tiles.

17.10.1   RGB4A3 Pixel Format

```
15          8 | 7          0
.ttt   rrrr | gggg   bbbb
```

bit(s)		description
15		unused (?)
12-14	t	transparency
8-11	r	red channel
4-7	g	green channel
0-3	b	blue channel

17.11   RGB5A1

Used for Icons and Banners on Memory Card. This Format uses no palette and is stored in 4x4 pixel tiles.

17.11.1   RGB5A1 Pixel Format

```
15          8 | 7          0
trrr   rrgg | gggb   bbbb
```

bit(s)		description
15	t	transparency
10-14	r	red channel
5-9	g	green channel
0-4	b	blue channel

17.12   RGB565

17.12.1   RGB565 Pixel Format

```
15           8 | 7           0
rrrr   rggg | gggb   bbbb
```

bit(s)		description
11-15	r	red channel
5-10	g	green channel
0-4	b	blue channel

17.13   RGBA8

17.13.1   RGBA8 Pixel Format

```
31        24 | 23        16 | 15         8 | 7          0
rrrr   rrrr | gggg   gggg | bbbb   bbbb | aaaa   aaaa
```

bit(s)		description
24-31	r	red channel
16-23	g	green channel
8-15	b	blue channel
0-7	a	alpha channel

17.14   S3TC

WARNING: this section is screwed! any advice/corrections/help/etcblabla welcomed! (thanx to **Aaron Kaluszka** for pointing this out)

S3TC is a compression method for textures, developed by S3 and licenced by Nintendo for the Gamecube (and also by Microsoft for DirectX 6.0). It basically gives you one more MIP level for free, with relatively small quality loss and a simple implementation in hardware. You basically store 2 colour values and then you have a few bits per pixel to interpolate between them. It works in blocks of 4x4 pixel.

There are 5 variants:

DXT1 allows one bit of alpha
DXT2/3 allows 4 bits of alpha
DXT4/5 stores 2 alpha values and has 3 bits to interpolate between them

The difference between DXT2/3 and 4/5 is, if colour values are pre-multiplied with alpha. The blending equation is normally (c*(1-a))+(t*a)), so with pre-multiplied alpha the texture contains (t*a) in each colour channel and the blending becomes (c*(1-a) + t).

Each image is made up of tiles placed linearly from left to right then top to bottom.

Each tile is made up of 4 blocks

0 1
2 3

Each block is made up of 8 words. ăThese 8 words represent 16 pixels using S3TC compression.

RRRRRGGG - GGGBBBBB - rrrrrggg - gggbbbbb - 00112233 - 44556677 - 8899UUVV- WWXXYYZZ

R = Color 0 Red
G = Color 0 Green
B = Color 0 Blue
r = Color 3 Red
g = Color 3 Green
b = Color 3 Blue
0 - 9, U - Z = Pixel color (2-bits each)
Colors 1 and 2 are interpolated from colors 0 and 3

The tiles are 32 bytes each. Depending on the image format the width and height of the tiles will differ. A 16bit format (ie RGB5 or RGB4A3) will have a 4x4 pixel tile since 4 * 4 * 2 bytes = 32. An 8bit format (ie Color Indexed) will have a 8x4 pixel tile since 8 * 4 * 1 byte = 32.

So a 32x32 image (like a memory card icon) that is in RGB5 format would be 8 tiles across and 8 tiles down.

17.14.1   CMPR

Like a usual texture, a CMPR-texture is divided on tiles, each 32-bytes to fit a texture cache line. Every tile is sub-tiled into four parts, in zigzag order :

0	1
2	3

The format of the sub-tiles is pretty simple, and looks like DXT1. First two base colors in RGB565, followed by 16 sub-tile texels. Every texel is 2-bit wide, to lookup from four colors : 00, 01, 10 and 11. First two are given already, and last two are interpolated from first ones, by the following rule :

▷ COLOR0 and COLOR1 are base colors.

▷ RGB0 <- unpack RGB565 COLOR0

▷ A0 = 255

▷ RGB1 <- unpack RGB565 COLOR1

▷ A1 = 255
   if COLOR0 > COLOR1

   ▷ RGB2 = (2 * RGB0 + RGB1) / 3
   ▷ A2 = 255

    ▷ RGB3 = (2 * RGB1 + RGB0) / 3

    ▷ A3 = 255

else

    ▷ RGB2 = (RGB0 + RGB1) / 2

    ▷ A2 = 255

    ▷ RGB3 = (2 * RGB1 + RGB0) / 3

    ▷ A3 = 0

18   Appendix

18.1   GCC Quick How To

18.1.1   compile ASM to object:

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-as -c \
-I <DEVKITCUBE>/powerpc-eabi-elf/include -I <additional includes> \
testasm.s -o testasm.o
```

18.1.2   compile C to object:

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-gcc -c \
-I <DEVKITCUBE>/powerpc-eabi-elf/include -I <additional includes> \
-nostdlib testc.c -o testc.o
```

18.1.3   compile C++ to object:

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-g++ -c \
-I <DEVKITCUBE>/powerpc-eabi-elf/include -I <additional includes> \
-nostdlib -fno-exceptions testcpp.cpp -o testcpp.o
```

18.1.4   link objects

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-ld -T ppc-ngcbin.x -o test.elf crt0.o \
<DEVKITCUBE>/lib/gcc-lib/powerpc-eabi-elf/3.3/crtbegin.o \
<DEVKITCUBE>/lib/gcc-lib/powerpc-eabi-elf/3.3/crtend.o \
testasm.o testc.o testcpp.o -lg -lstdc++ -lm -lc -lnosys
```

you only need to link against crtbegin.o/crtend.o if you are using c++, and you only need -lg,-lstdc++,-lc,-lm if you are actually using these libraries (of course:)). however if you do so, linking against -lnosys as well is essential.

18.1.5   remove unneeded sections (debug info etc) from object

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-strip -s test.elf
```

18.1.6   convert object to plain binary

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-objcopy -O binary test.elf test.bin
```

18.1.7   convert absolute address into filename/line number/function

compile with "-g" flag, then use

```
<DEVKITCUBE>/bin/powerpc-eabi-elf-addr2line -f -e test.elf 0x80003100
```

18.1.8 Building a Crosscompiler

configure options:

```
--target=powerpc-eabi-elf

--with-cpu=750

--disable-threads

--enable-languages=c

--disable-shared

--disable-nls

--with-newlib
```

18.1.9 Linker Script

to do

18.1.10 Startup Code

to do

18.2 Boot Process Details

The IPL (Initial Program Loader), or Bootrom, is located inside one Macronix chip (near Flipper, U10) and connected to the EXI bus. When the Gamecube is powered on, bit 25 (IP) in the Machine State Register is set, which means the system exception vector offset is `0xfff00000`. Then a small (about 0x0700 bytes) program called 'BS' will be mapped to `0xfff00100` (the hardware reset vector) and control will be returned to the Gamecube like after a normal reset, which means 'BS' will be started.

18.2.1 BS - Bootstrap 1

▷ copies the Bootstrap 2 code (BS2) from Bootrom to `0x81300000`

▷ disables the IPL decryption logic by clearing bit 17 of `0xcc006800`

▷ sets IP of Machine State Register so exception vectors are pointing to lower memory

▷ jumps to BS2 code

18.2.2 BS2 - Bootstrap 2

BS2 is the Program that loads the game or shows the menus when the gamecube has been powered on without a game inserted. It was written in C, using official SDK libraries, probably earlier than 1.0. __start.c seems to be same as usual, except that there is no OSInit() call (old versions must call OSInit() in main, instead of __start).

note: this has been reversed from a PAL gamecube and looks different on a NTSC one.

18.2.2.1 short description of start() routine.

```
// 81300000
__start:
    __init_registers() // set stack pointer and static bases (r2, r13)
    __init_hardware() // paired-singles and cache init
    __init_data() // clear bss ?
    .  // here goes Debug Monitor stuff
    .
    .
    DBInit() // debug monitor init :)
    __init_user() // cpp init
    main() // that's actually, IPL (BS2) code
    jmp exit() // halt CPU
```

18.2.2.2 IPL main() reversing

```
// 813006D4
main()
{
    BS2Init();
    OSInit();
    AD16Init();
    AD16WriteReg(0x800);
    DVDInit();
    AD16WriteReg(0x900);
    CARDInit();
    AD16WriteReg(0xa00);
    0x81302104(); // SRAM, real-time clock (check ?)
    __VIInit(0);
    VIInit();
    AD16WriteReg(0xb00);
    0x813004e4(); // setup performance. monitor
    0x8130222c(); // update time-base by SRAM clock
    0x813022c0(); // perform initial DVD actions and fall back into menu
    PADSetSpec(5); // sed PAD type ('spec') to 'production'
    PADInit();
    AD16WriteReg(0xc00);
    BS2Menu(); // here goes intro and main menu...  (BIG one!)
```

```
    OSPanic(__FILE__, __LINE__, "BS2 ERROR >>> SHOULD NEVER REACH HERE");
}
float NaN;
// 8130045C
void BS2Init()
{
    // clear LoMem and OSMem
    memset(0x80000000, 0, 256);
    memset(0x80003000, 0, 256);
    BATInit();
    // set memory size to 24MB
    *0x80000028 = 0x01800000;
    // set console type to default retail 1
    *0x8000002c = 1;
    // upgrade retail
    *0x8000002c += *0xcc00302c >> 28;
    (u32)NaN = -1;
    FPUInit();
}
// 813003A0
void BATInit()
{
    __asm
    {
        isync
        li r4, 0
        mtspr DBAT2L, r4
        mtspr DBAT2U, r4
        mtspr DBAT3L, r4
        mtspr DBAT3U, r4
        mtspr IBAT1L, r4
        mtspr IBAT1U, r4
        mtspr IBAT2L, r4
        mtspr IBAT2U, r4
        mtspr IBAT3L, r4
        mtspr IBAT3U, r4
        isync
```

```
    }
}
// 813003D8
void FPUInit()
{
    // FPU already initialized in __start(),
    // so just invalidate all FPRs.
    __asm
    {
    lfs f0, NaN
    fmr f1, f0
    fmr f2, f0
    fmr f3, f0
    . e
    . t
    . c
    fmr f31, f0
    }
}
// maybe later
0x81302104()
{
    __OSLockSram();
    __OSCheckSram();
    __OSGetRTC();
    OSTickToCalendarTime();
    memset();
    __OSUnlockSram();
    __OSSyncSram();
}
// maybe later
0x813004e4()
{
    OSDisableInterrupts();
    OSGetTick();
    OSGetTick();
    OSGetTick();
```

```
        __div2i();

        __div2i();

        PPCMtpmc1();

        PPCMtmmcr0();

        OSGetTick();

        OSGetTick();

        PPCMtmmcr0();

        PPCMfpmc1();

        __div2i();

        __div2i();

        __div2i();

        OSRestoreInterrupts();

}

// maybe later

0x8130222c()

{

        __OSLockSram();

        __OSGetRTC();

        __OSSetTime();

        __OSUnlockSram();

}

static int BS2State = 0;

// just layer..

0x813022c0()

{

        BS2State = BS2Mach();

}

// 81300A70

// located in __FILE__ = "BS2Mach.c"

int BS2Mach()

{

        static int state = 0;

        BOOL level = OSDisableInterrupts();

        switch(state)

        {

            case 0:

                [r13 - 0x7dc8] = 0x800030d4;
```

```
            state = 1;
        case 1:
            __OSGetSystemTime();
...  some checks
            if(fail) break;
            state = 2;
        // Install DVD cover callback
        case 2:
            if([r13 - 0x7da8] == 0)
            {
                    r3 = [r13 - 0x7dc8]
                    [r3] = 0
                    [r13 - 0x7dc4] = 0
                    [r13 - 0x7dac] = 1
                    DVDLowSetResetCoverCallback(0);
                    DVDReset();
                    [r13 - 0x7da8] = 1
                    (s64)[r13 - 0x7d9c] = __OSGetSystemTime();
                break;
            }
            __OSGetSystemTime();
.
.
.
            if(fail) break;
            DVDLowSetResetCoverCallback(0x813007d8);
            DVDReset();
            state = 3;
        // Read Disk information (ID)
        case 3:
            DVDReadDiskID(0x8145e620 + 64, 0x80000000, 0x813007e4);
            state = 4;
        break;
.
.
.
        // Leave immediately ?
```

```
        case 16:
            break;
        default:
            OSPanic(__FILE__, __LINE__, "BS2 ERROR >> UNKNOWN STATE");
    }
    OSRestoreInterrupts(level);
    return (DVDLowGetCoverStatus() == 1) ?  19 :  step;
}
// 81301154
void BS2Menu()
{
    BS2InitAlloc();
}
static OSHeapHandler BS2Heap;
// 81307EA8
void BS2InitAlloc()
{
    u8 *arenaLo;
    u8 *arenaHi;
    u8 *arenaNew;
    arenaLo = OSGetArenaLo();
    arenaLo = (void *)OSRoundUp32B(arenaLo);
    arenaHi = OSGetArenaHi();
    arenaHi = (void *)OSRoundDown32B(arenaHi);
    arenaNew = OSInitAlloc(0x80800000, arenaHi, 2);
    OSSetArenaLo(arenaHi);
    BS2Heap = OSCreateHeap(arenaLo, arenaHi);
    OSSetCurrentHeap(BS2Heap);
    OSAddToHeap(BS2Heap, arenaNew, 0x81100000);
    BS2CheckAlloc();
}
// 81307F34
void BS2CheckAlloc()
{
    OSCheckHeap(BS2Heap);
}
// 81307F58
```

```
void *OSAlloc(long size)
{
    void *ptr;
    if((ptr = OSAlloc(size)) == 0)
    {
        OSPanic(?);
    }
    return ptr;
}
```

18.2.2.3   Map of IPL Library code

Address	Name	Libray
0x813014C8	DEMOInit (*)	DEMO
0x81307F58	OSAlloc (*)	OS
0x813327BC	PPCMtmmcr0	
0x813327C4	PPCMfpmc1	
0x813327CC	PPCMtpmc1	
0x81332814	OSInit	OS
0x81332EF0	OSInitAlarm	OS
0x81332F3C	OSCreateAlarm	
0x81333688	OSAllocFromHeap	
0x81333784	OSSetCurrentHeap	
0x81333794	OSInitAlloc	
0x81333804	OSCreateHeap	
0x81333870	OSAddToHeap	
0x813338D0	OSCheckHeap	
0x813344C0	OSGetStackPointer	
0x8133491C	OSReport	
0x8133499C	OSPanic	
0x81334AA4	PPCHalt	
0x81334D4C	EXIImm	EXI
0x81335134	EXISync	
0x813353C8	EXIProbeReset	
0x8133570C	EXISelect	
0x81335838	EXIDeselect	
0x81335D6C	EXILock	
0x81335E60	EXIUnlock	
0x81335F54	AD16Init	
0x81336090	AD16WriteReg	
0x813361B0	OSDisableInterrupts	
0x813361C4	OSEnableInterrupts	
0x813361D8	OSRestoreInterrupts	
0x81336DD8	__OSGetRTC	
0x813372B0	__OSLockSram	
0x81337658	__OSUnlockSram	
0x813376A0	__OSSyncSram	
0x813376B0	__OSCheckSram	
0x81338504	OSInitThreadQueue	
0x8133939C	OSGetTick	
0x813393B8	__OSSetTime	
0x8133943C	__OSGetSystemTime	
0x8133963C	OSTicksToCalendarTime	
0x8133AC50	DVDLowGetCoverStatus	DVD
0x8133AB18	DVDLowReset	
0x8133ABD4	DVDLowSetResetCoverCallback	
0x8133B5F0	DVDInit	
0x8133CD18	DVDReadDiskID	
0x8133D0EC	DVDReset	
0x8133DBE0	__VIInit	VI
0x8133DDC8	VIInit	
0x8133E6C0	VIConfigure	
0x8133F0B4	VIGetTvFormat	

Address	Name	Libray
0x8134052c	PADInit	PAD
0x8134092c	PADSetSpec	
0x81343114	CARDInit	
0x813480D4	GXInit	
0x81349148	GXInitFifoBase	
0x81349230	GXSetCPUFifo	
0x81349340	GXSetGPFifo	
0x813494B8	__GXFifoInit	
0x8134B0AC	__GXPEInit	
0x8135A178	__div2i	gcc
0x8135A394	__mod2i	gcc
0x8135B494	vprintf	stdlib

(*) these functions were slightly modified for the IPL.

18.2.3 Apploader

The Apploader provides functions to the bootrom that load the game (using bootrom read DVD functions). The bootrom calls the Init function, then the Main function in a loop, then the Closing function. At first, the BIOS calls the Apploader entrypoint with r3, r4, and r5 pointing to a free space for a 32 bit value.

```
// info based on Luigi Mansion appldr.bin file
// (built date is 17 Dec 2001).
// Apploader Entrypoint
// Input values :
// r3 = Address where to put the address of the Init function
// r4 = Address where to put the address of the Main Loading function
// r5 = Address where to put the address of the Closing function
// Return values :
// none
//
// file:[0010-0013] = 0x81200288 (apploader entrypoint)
void Entrypoint(r3, r4, r5)
{
    [r3] = 0x81200290 // Init
    [r4] = 0x81200580 // Main
    [r5] = 0x81200D50 // Close
}
// Init function
// Input values :
// ?
```

```
// Return values :
// none
void Init(void (*OSReport)(char *fmt, ...))
{
    // clear some important memory areas
    memset(OSAppLdr + 32, 0, 32);
    memset(&OSAppLdr.DolImage, 0, sizeof(DolImage));
    [+0x140] = 0
    OSAppLdr.pass = 0
    [+0x148] = 0
    OSAppLdr.OSReport = OSReport // save report callback
    OSAppLdr.OSReport("Apploader Initialized.  $ Revision:  28 $\n");
    OSAppLdr.OSReport("This Apploader built %s %s\n", __DATE__, __TIME__);
}
// Main Loader function
//
// Input values :
// r3 = Address where to put the Memory destination of the disk read
// r4 = Address where to put the Size of the disk read
// r5 = Address where to put the Starting position of the disk read
//
// Return value:
// r3 = 0 if everything is already loaded
// = 1 (or !=0) if main function should be called again
//
// at 0x81200580
// helper functions (below)
u32 DOLSize(void);
// 0x812013E0 seems to be a big structure, like that :
struct OSAppLdr
{
    // untouched
    u32 SecondTimeForThePart;
    u8 [28]
    u8 [32] // "BB2" structure ?
    DolImage DolImage; // main DOL executable header
    // flags or something
```

```
    u32 +0x140
    u32 pass; // 0...12
    u32 +0x148
    // report routine itself is placed somewhere in bootrom
    void (*OSReport)(char *fmt, ...);
    // flags or something
    u32 +0x150
    u32 +0x154
    u32 +0x158
    u32 +0x15C
    u8 [32]
} OSAppLdr; // 0x174 total
int Main(r3, r4, r5)
{
    int pass = OSAppLdr.pass;
    if(pass <= 12)
    {
        switch(pass)
        {
            // read "BB2" structure (DVD offset at 0x0420)
            case 0:
            case 1:
                // "BB2" structure ?
                // 0420-0424 offset of main executable DOL
                // 0424-0427 offset of the FST
                // 0428-042B size of FST
                // 042C-042F maximum size of FST
                [r3] = OSAppLdr + 32
                [r4] = 32
                [r5] = 0x420
                OSAppLdr.pass = 2
                DCInvalidateRange([r3], [r4])
            break;
            // check "BB2" structure FST sizes
            case 2:
                FSTLength = [OSAppLdr + 32 + 8]
                FSTMaxLength = [OSAppLdr + 32 + 12]
```

```
                if(FSTLength > FSTMaxLength)
                {
                    OSAppLdr.OSReport(
                    "APPLOADER ERROR >>> FSTLength(%d) in BB2 is greater \
                    than FSTMaxLength(%d)\n", FSTLength, FSTMaxLength);
                    PPCHalt();
                }
                [r3] = OSAppLdr + 0x160
                [r4] = 32
                [r5] = 0x440
                OSAppLdr.pass = 3
                DCInvalidateRange([r3], [r4])
            break;
            case 3:
                [0x800000E8] = [OSAppLdr + 0x160] // word
            break;
            case 4:
            // load main DOL header (256 bytes)
            case 5:
                [r3] = &OSAppLdr.DolImage
                [r4] = 256
                [r5] = [OSAppLdr + 32] // from BB2
                OSAppLdr.pass = 6
                DCInvalidateRange([r3], [r4])
            break;
            case 6:
                totalSize = DOLSize();
                maxSize = [[800000F4] + 0x28]; // PadSpec ?
                if((totalSize > maxSize) && maxSize)
                {
                    OSAppLdr.OSReport(
                    "APPLOADER ERROR >>> Total size of text/data sections \
                    of the dol file are too big (%d(0x%08x) bytes).  Currently \
                    the limit is set as %d(0x%08x) bytes\n", totalSize, maxSize);
                    PPCHalt();
                }
.
```

.
.

```
            case 7:
            case 8:
            case 9:
            case 10:
            case 11:
            case 12:
                if(SecondTimeForThePart == TRUE)
                {
                    OSAppLdr.OSReport(
                    "Failed assertion SecondTimeForThePart == TRUE");
                    PPCHalt();
                }
```
.
.
.
```
            break;
              }
        return 1;
    }
    else
    {
        return 0;
    }
}
// helper functions
// at 0x81200338
u32 DOLSize(void)
{
    DolImage *dol = &OSAppLdr.DolImage;
    u32 totalBytes = 0;
    int i;
    for(i=0; i<DOL_MAX_TEXT; i++)
    {
        if(dol->textData[i])
        {
```

```
        // aligned to 32 byte boundary

        totalBytes += (dol->textLen[i] + 31) & ~31;

        }

    }

    for(i=0; i<DOL_MAX_DATA; i++)

    {

        if(dol->dataData[i])

        {

            // aligned to 32 byte boundary

            totalBytes += (dol->dataLen[i] + 31) & ~31;

        }

    }

    return totalBytes;

}

// Closing function

//

// Return value:  r3 = entry point

//

// at 0x81200D50

u32 Close(void)

{

    // provide entrypoint of main DOL executable to IPL

    return OSAppLdr.DolImage.entry;

}
```

18.2.4   Main DOL executable

18.3   Game and Maker Codes

18.3.1   Gamecodes

offset	size	Description			
1	1	System ID			
		value	id	Description	
		`0x47`	`G`	Gamecube (standard value)	
		`0x44`	`D`	used by Legend Of Zelda: Ocarina Of Time (Master Quest) Might be a indicator for emulated/ported/promotional titles.	
		`0x55`	`U`	used by GBA-Player Boot CD	
2-3	2	Game ID/serial Number			
4	1	Country/Region Code			
		value	id	Country	
		`0x45`	`E`	USA/NTSC	
		`0x50`	`P`	Europe/PAL	
		`0x4a`	`J`	Japan/NTSC	
		`0x55`	`U`	used by the European version of The Legend Of Zelda: Ocarina Of Time (Master Quest)	

18.3.2   Game Serial ID

Characters	Description	
3	System ID 'DOL'	
4	Gamecode	
3	Country ID	
	ID	Country
	`USA`	guess what :)
	`NOE`	Nintendo of Europe
	`NOA`	Nintendo of America
	`JPN`	Japan

for example

▷ `DOL-GZLE-USA` (Zelda)

▷ `DOL-GNHE-USA` (NHL Hitz 20-20)

▷ `DOL-GTEP-NOE` (1080ř Avalanche)

▷ `DL-DOL-GFZJ-JPN` (F-Zero GX)

18.3.3   Makercodes

The ID (2 Bytes ASCII) belongs to the publisher, not the developer. Hence, even though Rare developed Star Fox Adventures, and Retro Studios developed Metroid Prime, they both have the Vendor ID of Nintendo (01).
It is unknown how vendor IDs are allocated; However, all IDs thus far seem to be alphanumeric. If this is accurate, then as a result the maximum number of unique vendors is 1,296. Vendor IDs seem to be region-independent.

ID	Vendor
01	Nintendo
08	Capcom
41	Ubisoft
4F	Eidos
51	Acclaim
52	Activision
5D	Midway
5G	Hudson
64	Lucas Arts
69	Electronic Arts
6S	TDK Mediactive
8P	Sega
A4	Mirage Studios
AF	Namco
B2	Bandai
DA	Tomy
EM	Konami

18.4   Macronix Chip IDs

	`MX ff t mm b p r s`		
MX	MX', vendor id		
ff	2 digits, device family		
	17	auto focus controller	
	23	mask rom	
	25	spi serial flash memory	
	26	mtp eeprom	
	27	eeprom	
	28	flash memory	
	29	flash memory (single voltage)	
	53	memory card (smc)	
	67	flash memory	
	69	flash memory + sram (stacked chip)	
	88	digital camera/flat panel display controller	
	89	flat panel display controller	
	92	sound generator	
	93	single chip answering machine/digital recorder controller	
	97	isdn controller	
	98	network	
	99	bluetooth	
t	1 character, device type		
	c	cmos	
	f	flash	
	l	low-voltage	
	w	srw	
	v	2.2v	
	u	1.8v	
	x	1.5v	
	vw	2.25v+srw	
mm	2 to 4 digits, mode/density		
	004	4M, x8 Boot Block	
	040	4M, x8 Equal Sector	
	400	4M, x8/x16 Boot Block	
b	1 character, bootblock type (rom only)		
	t	top	
	b	bottom	

	MX ff t mm b p r s	
p	1 character, package type	

p	plastic dip	
m	plastic sop	
q	plastic plcc	
t	tsop normal	
d	qeramic dip	
x8	0.8mm ball pitch, bga, csp	
x	csp	

r	1 character, temperature range	

c	commercial	
i	industrial	
m	military	

s	1 character, speed	

45	45ns	
55	55ns	
70	70ns	
85	85ns	
90	90ns	
10	100ns	
12	120ns	
15	150ns	
20	200ns	
25	250ns	

18.5   chip simelarities

▷ mx25L4001 (serial flash rom in nintendo memory card 59)

    ▷ datasheet was never available at Macronixă but take a look at their 3-volt SPI Flash ROMs (the MX25LXX02 series) for a general pinout withoutăthe Unknown pin 24.

▷ mx25L4004 (serial flash rom in datel memory card)

    ▷ mx25L4004 - 4Mbit, 4Mx1 serial flash rom (datasheet was, but is no longer available at Macronix :/)

▷ mx98730ec (eth controller in bba)

    ▷ mx98728ec - single chip 10/100 base generic MAC interface

    ▷ (mx98726, mx98728)

▷ Gekko

    ▷ ibm PowerPc750CXe

    ▷ (PowerPc740, PowerPc750, PowerPc750CX)

▷ (inside flipper)

    ▷ (big ???) mx92L832 - 32 poly phony sound generator

    ▷ (big ???) mx96037 - 16bit DSP Controller

    ▷ (mx93011a,...)

▷ mx 8013108-M rtnc-dol 1r6022a1 (rtc/ipl)

▷ MoSys (MS3M23B-5 A) 12MB 1-T SRAM (main memory)

▷ NEC (D4891281G5 0125XU621) 16MB ARAM (auxiliary/dsp memory)

18.6   Easter Eggs

▷ To hear a different sound when the console boots hold the Z button down once you turn the console on and as the square bounces down you will hear the noises of kids.

▷ Hold Z then about one second later hold A to hear another sound.

▷ If you have 4 controllers (and 5 hands) this one will work: Before you turn the Gamecube on hold down Z on each controller, then turn the system on (with your 5th hand of course). You will be greeted by a ninja yell.

▷ holding B on first controller, then powering on will let you switch a pal gamecube into 60Hz mode

18.7   Terms and Acronyms

▷ AA

Antialiasing. Rendering method, that makes polygon edges seem less sharpen, combining colors of nearby pixels.

▷ AD16

Mysterious EXI device.

▷ AI

Audio Interface. Hardware responsible for DMA playback of PCM buffer and DVD ADPCM streaming sound. AI hardware cannot mix sound channels or set channel volume for PCM DMA playback. These operations and more advanced sound effects are produced by DSP.

▷ Apploader

Small program on DVD to load main DOL executable.

▷ AR, ARAM

Auxiliary (Audio) Memory. 16 MB of slow (comapred to RAM) DRAM. Used for raw DSP sound data and as temporary space for textures. ARAM has DMA communication channel with main memory (RAM). Development boards has "ARAM Expansion" (additional 4, 16 or 32 MBs).

▷ BAT

Block Address Translation, PPC MMU translation mechanism. There are DBAT and IBAT special-purpose registers for data and instruction address translation respectively.

▷ BBA

Broad-Band Adapter, GC's 10BaseT Ethernet Adapter.

▷ BS

Bootstrap Stage (from analogy with UNIX). Very first code, executed after GC hard reset.

▷ BS2

Bootstrap Stage 2. Same as IPL.

▷ CR

PPC Condition Register, stores result of integer comare operation, for conditional branch decision.

▷ CRT

C Run Time. C/C++ program environment (libraries and startup calls).

▷ DOL

Gamecube application (custom executable file format).

▷ Dolphin

Early development work name of Gamecube.

▷ Dolphin OS

Gamecube OS. Single user, single process, multithreaded. Linked together with any GC application ("hard-linked"), as library.

▷ DSP

Digital Signal Processor. Used to produce advanced sound on GC. DSP is integrated with GP in Flipper chip and has its own ROM. Developed by Macronix.

▷ DI, DVD

DVD hardware interface. GC DVD is actually microcontroller, based on MN-102 CPU with proprietary firmware ROM. DVD is protected by non-standard barcodes and data encryption, which is decrypted on-the-fly by DVD controller. GC DVD cannot be read on usual PC hardware. Whole GC DVD stuff is developed by Matsushita.

▷ EFB

Embedded Framebuffer. 2MB of fast 1T-SRAM memory located inside Flipper. Used by GP's pixel engine to draw pixels. Later copied into XFB, for final TV-output.

▷ EXI

Expansion Interface. Gamecube peripherial devices bus, sort of USB architecture. Developed by Macronix. Devices drived by EXI: memory cards, broad-band adapter, real-time clock, bootrom, SRAM.

▷ FIFO

First-In-First-Out buffer to send GP commands and create GP command lists.

▷ Flipper

Gamecube Northbridge+Peripheral Hardware+Graphics Processor+Audio DSP.

▷ FPR

Floating Point Register. Gekko has 32 64-bit FPRs, named f0-f31.

▷ FPSCR

Floating Point Status and Control Register.

▷ JTAG

Hardware debug interface to CPU. You can connect some wires to CPU pins, to overwhelm it. Gekko has full support of IEEE 1149-1a-1993 JTAG standard.

▷ GC, GCN, NGC

Nintendo Gamecube.

▷ GCM

Gamecube Master Data (official term). GC DVD Image files.

▷ Gekko

Gamecube CPU, PowerPC 750-derivative processor with FPU extensions, called "Paired Single".

▷ GP, GX

Graphics Processor, the major part of Flipper chip. GP is fixed point state-machine. Developed by ArtX team.

▷ GPR

General Purpose Register. Gekko has 32 32-bit GPRs, named r0-r31. r1 often used as stack pointer (sp).

▷ GX

Software library, developed by Nintendo and ArtX, to drive GP hardware. Has many crossways with OpenGL (but more advanced).

▷ HW2

Common name of GC hardware. Number state for revision ("2" is production board).

▷ IPL

Initial Program Loader. Graphics shell, used to load game from DVD.

▷ MC

Memory Card, EXI device.

▷ MI

Flipper memory interface, plays role of "Nothbridge".

▷ MMU

PPC Memory Management Unit. Translates virtual address to physical. MMU has two translation mechanisms: block address translation and page table translation. Address translation for data access and instruction fetch is processed separately in DMMU and IMMU.

▷ MSR

Machine State Register. CPU status and control register.

▷ MX

Macronix Ltd. chips index. GC has many hardware parts, developed by Macronix, like DSP, EXI and bootrom chip.

▷ PC

Program Counter. PowerPC architecture does not define such term, but everyone is using it anyway, instead "CIA" (Current Instruction Address).

▷ PCM

Pulse Code Modulation, method commonly used in digital sound hardware. PCM sound parameters are: playback rate, bits per sample, sample format. GC AI can playback 32000/48000 Hz, 16-bit big-endian stereo samples via DMA.

▷ PI

Peripheral Interface. Set of hardware registers to control interrupts and hardware reset. There also "PI FIFO": hardware-driven FIFO buffer in RAM.

▷ PM

PowerPC Performance Monitor. Set of PPC special purpose registers used for speed profiling of applications.

▷ PPC

IBM PowerPC Architecture.

▷ PTE

Page Table Entry. Page table record, used to translate virtual address to physical.

▷ RAM

Main memory. GC has 24 MB of fast 1T-SRAM. Development boards has RAM extended up to 48 MB. Developed by MoSys.

▷ ROM

Read-only memory. GC has following ROMs: 2 MB encrypted bootrom, 128 KB DVD firmware, 4 KB DSP DROM, 8 KB DSP IROM.

▷ RSW

"Reset Switch", same as reset button. Seems early development GC models were equipped by switch, insted programmable reset button.

▷ RTC

Real-time clock. EXI device, counting seconds since 00:00 AM 2000.

▷ SDK

Software Development Kit. Full set of compilers, libraries and documentation for development on specified platform. Gamecube SDK contain development tools, like sound and texture convertors, and set of libraries for OS and hardware. Compiler is provided by Metrowerk's CodeWarrior. There also huge development board and paper documentation.

▷ SI

Serial Interface. Hardware responsible for communication with serial devices, such as GC controller and keyboard via serial I/O buffer.

▷ SPR

Special-purpose register. Set of registers, dedicated to operating system. Gekko has about 60 SPRs.

▷ SR

Segment Registers, used by MMU for virtual address translation.

▷ SRAM

Small amount of battery backuped memory for OS misc settings.

▷ TLB

PPC MMU Translation Lookaside Buffer, used to keep recently used page address translations. Gekko has 128 two-way set associative TLB for each MMU (DMMU and IMMU).

▷ VI

Video Interface. Hardware responsible for TV-out of framebuffer (XFB), and generating VBlank interrupt (actually can be configured to interrupt CPU at any beam location). Has support for light-gun, antialiasing of XFB by tap-filters and progressive video mode (480p).

▷ WBUF

Gekko Write Gather Buffer. Small cache for burst memory transactions. Used together with graphics
FIFO to send GP commands.

▷ XFB

External Framebuffer, located in main memory. Used for final TV-out by VI.

19 References

▷ U.S. Pat. 5,680,534 (Video game/video graphics program fabricating system and method with superimpose control)

▷ U.S. Pat. 6,411,301; 6,452,600; 6,466,218; 6,697,074 (Graphics system interface) - GX Info

▷ U.S. Pat. 6,421,058 (Graphics command stream for calling a display object in a graphics system) - a lot of GX Info

▷ U.S. Pat. 6,424,348; 6,456,290; 6,489,963 (Application program interface for a graphics system) - GX Info

▷ U.S. Pat. 6,457,128 (Optical disk. An optical disk barcode forming method, an optical disk reproduction apparatus, a marking forming apparatus, a method of forming a laser marking on an optical disk, and a method of manufacturing an optical disk.)

▷ U.S. Pat. 6,468,160; 6,712,704 (Security system for video game system with hard disk drive and internet access capability)

▷ U.S. Pat. 6,591,019 (3D transformation matrix compression and decompression)

▷ U.S. Pat. 6,606,689 (Method and apparatus for pre-caching data in audio memory) - ARAM Info

▷ U.S. Pat. 6,609,977 (External interfaces for a 3D graphics system) - Lots of Register Info

▷ U.S. Pat. 6,618,048 (3D graphics rendering system for performing Z value clamping in near-Z range to maximize scene resolution of visually important Z components)

▷ U.S. Pat. 6,636,214 (Method and apparatus for dynamically reconfiguring the order of hidden surface processing based on rendering mode) - Some GX Stuff

▷ U.S. Pat. 6,639,595 (Achromatic lighting in a graphics system and method) - Some GX Info

▷ U.S. Pat. 6,580,430 (Method and apparatus for providing improved fog effects in a graphics system) - some GX Info

▷ U.S. Pat. 6,643,744 (Method and apparatus for pre-fetching audio data) - Some ARAM Info

▷ U.S. Pat. 6,664,958 (Z-Texturing) - GX Info

▷ U.S. Pat. 6,664,962 (Shadow mapping in a low cost graphics system) - GX Info

▷ U.S. Pat. 6,707,458 (Method and apparatus for texture tiling in a graphics system) - GX Info

▷ U.S. Pat. 6,681,296; 6,859,862 (Method and apparatus for software management of on-chip cache)

▷ U.S. Pat. 6,700,586 (Low cost graphics with stitching processing hardware support for skeletal animation)

▷ U.S. Pat. 6,701,424 (Method and apparatus for efficient loading and storing of vectors)

▷ U.S. Pat. 6,717,577 (Vertex cache for 3D computer graphics)

▷ U.S. Pat. 6,571,328 (Method and apparatus for obtaining a scalar value directly from a vector register)

▷ Macronix Product Catalog

▷ mx98726 Datasheet

▷ mx98728 Datasheet

   ▷ mx25L4004 Datasheet

   ▷ PowerPC 740/PowerPC 750 RISC Microprocessor User's Manual

   ▷ PowerPC 750CX/ 750CXe RISC Microprocessor User's Manual

   ▷ Standard ECMA-268 (80 mm DVD - Read-Only Disk)

   ▷ MN102H60G/60K/F60G/F60K LSI User Manual

19.1   Sources

   ▷ `http://www.uspto.gov`

   ▷ `http://www.macronix.com`

   ▷ `http://www.ibm.com`

   ▷ `http://www.s3.com/s3tc` (no more available)

   ▷ `http://www.ecma-international.org/publications/standards/Ecma-268.htm`

   ▷ `https://www.semicon.panasonic.co.jp`

20   Credits

besides freely available datasheets and patents, this document was created based on information provided by the following people. if you think you are missing in this list, please keep me informed so i can add you immediately.

titanik/crazy nation	'Gamecube Low-level Info' in CZN 'Gamecube Source pack #1'
duke/napalm	duke@napalm-x.com initial "gcinfo.txt" some invaluable information (you know who you are)
costis	costis@gbaemu.com, gcdev.com http://www.gcdev.com hardware introduction text (posted on some website...uhm :)) gcspec.html additional info in sram checksum, video regs
org	kvzorganic@mail.ru additional apploader info / apploader RE IPL RE, boot process details info on Gekko specific opcodes tons of other info (cheers mate)
torlus	gcc config
???	www.gc-nfo.com some file-format info
Crowtrobo	ctr-snd.txt memory card info
Azimer	http://www.apollo64.com some additional VI info
tmbinc	debugmo.de http://debugmo.de released some sources that helped to close the one or other gap driving force behind GX reversing IPL encryption reversing/IPL replacement info
shagkur	shagkur@gmx.net GX reversing, additional sources/infos
Timothy Wilson	theimp@iinet.net.au compiled some valuable info concerning memory cards
Authors of GClib	gclib.sf.net cross-checking against the Source helped to make sure no bad errors sneaked in
GC-Linux Team	gc-linux.sf.net another valueable source for code that has been cross-checked against
Aaron Kaluszka	megabyte@kontek.net some image format info
Monk	monk@mad.scientist.com TPL Fileformat details
Steven Looman	steven@krx.nl keyboard scancodes, comments on adapters
thakis	http://www.amnoid.de/gc/ lots of additonal fileformat info, proofreading&spellchecking
Alexander Wold (micropal)	http://cube.iu.hio.no/~s104086/ additional rtc/ipl pinout info

moreover, many thanks must go to everyone who helped making this document more consistant and error free by proofreading and pointing out mistakes, in particular tmbinc, org, hubb, Aaron Kaluszka,

Skywalker, Jihad, xor37h, costis, CrowTrobo, mist, ionic, Briii, Desktopman, Spike Grobstein, Steven Looman, Anders Montonen, Monk, Josiah "afnom" Burroughs, Scream|CT, thakis ... (please check the changelog for details)