AXRE

A GameCube DSP UCode Documentation

Author: DenSinH

July 2, 2021

Contents

This was done using IDA, and the IDA plugin for the GameCube DSP, originally developed by delroth, but later updated by peach AKA wheremyfoodat AKA guccirodakino.

First of all some general functions we might use:

```
#pragma once


#define DMAControl ((volatile u16*)0xffc9)
#define DMALength ((volatile u16*)0xffcb)
#define DMADSPAddr ((volatile u16*)0xffcd)
#define DMAMMAddrHi ((volatile u16*)0xffce)
#define DMAMMAddrLo ((volatile u16*)0xffcf)

#define ToCPUMailHi ((volatile u16*)0xfffc)
#define ToCPUMailLo ((volatile u16*)0xfffd)
#define FromCPUMailHi ((volatile u16*)0xfffe)
#define FromCPUMailLo ((volatile u16*)0xffff)
#define DIRQ ((volatile u16*)0xfffb)

void send_mail(u16 hi, u16 lo) {
    *ToCPUMailHi = hi;
    *ToCPUMailLo = lo;
}

void send_irq() {
    *DIRQ = 1;
}

void wait_for_mail_sent() {
    do { } while ((*ToCPUMailHi) & 0x8000);
}

u32 wait_for_mail_recv() {
    do { } while (!((*FromCPUMailHi) & 0x8000));
    return ((u32)(*FromCPUMailHi) << 16) | *FromCPUMailLo;
}

u32 read_mail_recv() {
    return ((u32)(*FromCPUMailHi) << 16) | *FromCPUMailLo;
}

void dma_to_dmem(u32 mmaddr, u16 src, u16 len) {
    // len in bytes not DSP words!
    (*DMAMMAddrHi) = mmaddr >> 16;
    (*DMAMMAddrLo) = mmaddr;
    (*DMADSPAddr) = src;
    (*DMAControl) = 0;
    (*DMALength) = len;
}

void dma_dmem_to_mmem(u16 dest, u32 mmaddr, u16 len) {
```

```
    // len in bytes not DSP words!
    (*DMAMMAddrHi) = mmaddr >> 16;
    (*DMAMMAddrLo) = mmaddr;
    (*DMADSPAddr) = dest;
    (*DMAControl) = 1;
    (*DMALength) = len;
}

void wait_for_dma_finish() {
    do { } while((*DMAControl) & 4);
}
```

Chapter 1

ROM

The DSP ROM is the public replacement taken from Dolphin. It is fairly simple, probably much simpler than that in the actual DSP.

1.1 Entry

According to dolphin, the reset vector is `0x8000`. I believe this might be a hack though, since games tend to first DMA a short stub of code to the start or IRAM (at `0x0000`), and then ask the DSP to reset.

The replacement DSP ROM starts with

```
ROM:8000 ; =============== S U B R O U T I N E =======================================
ROM:8000
ROM:8000
ROM:8000 rom_start:                          ; CODE XREF: j_rom_start↑j
ROM:8000
ROM:8000 ; FUNCTION CHUNK AT ROM:80C4 SIZE 00000015 BYTES
ROM:8000
ROM:8000                 LRI          $CR, 0xFF
ROM:8002                 LRI          $SR, 0x2000
ROM:8004                 SI           ToCPUMailHi, 0x8071
ROM:8006                 SI           ToCPUMailLo, 0xFEED
ROM:8008
ROM:8008 receive_setup:                      ; CODE XREF: rom_start+19↓j
ROM:8008                                     ; rom_start+24↓j ...
ROM:8008                 CLR          $ACC1
ROM:8009                 CLR          $ACC0
ROM:800A                 CALL         wait_for_mail
ROM:800C                 LR           $AC1.M, FromCPUMailLo
ROM:800E                 LRI          $AC0.M, 0xA001
ROM:8010                 CMP
ROM:8011 ; if (mail.lo != 0xa001) jump -> check_c002
ROM:8011                 JNZ          check_c002
ROM:8013                 CALL         wait_for_mail
ROM:8015                 LR           $IX0, FromCPUMailHi
ROM:8017                 LR           $IX1, FromCPUMailLo
ROM:8019                 JMP          receive_setup
ROM:801B ; ---------------------------------------------------------------------------
ROM:801B
ROM:801B check_c002:                         ; CODE XREF: rom_start+11↑j
ROM:801B                 LRI          $AC0.M, 0xC002
ROM:801D                 CMP
ROM:801E ; if (mail.lo != 0xc002) jump -> check_a002
```

```
ROM:801E                    JNZ             check_a002
ROM:8020                    CALL            wait_for_mail
ROM:8022                    LR              $IX2, FromCPUMailLo
ROM:8024                    JMP             receive_setup
ROM:8026 ; ---------------------------------------------------------------------------
ROM:8026
ROM:8026 check_a002:                                         ; CODE XREF: rom_start+1E↑j
ROM:8026                    LRI             $AC0.M, 0xA002
ROM:8028                    CMP
ROM:8029 ; if (mail.lo != 0xa002) jump -> check_b002
ROM:8029                    JNZ             check_b002
ROM:802B                    CALL            wait_for_mail
ROM:802D                    LR              $IX3, FromCPUMailLo
ROM:802F                    JMP             receive_setup
ROM:8031 ; ---------------------------------------------------------------------------
ROM:8031
ROM:8031 check_b002:                                         ; CODE XREF: rom_start+29↑j
ROM:8031                    LRI             $AC0.M, 0xB002
ROM:8033                    CMP
ROM:8034 ; if (mail.lo != 0xb002) jump -> check_d001
ROM:8034                    JNZ             check_d001
ROM:8036                    CALL            wait_for_mail
ROM:8038                    LR              $AX0.L, FromCPUMailLo
ROM:803A                    JMP             receive_setup
ROM:803C ; ---------------------------------------------------------------------------
ROM:803C
ROM:803C check_d001:                                         ; CODE XREF: rom_start+34↑j
ROM:803C                    LRI             $AC0.M, 0xD001
ROM:803E                    CMP
ROM:803F                    JNZ             receive_setup
ROM:8041                    CALL            wait_for_mail
ROM:8043                    LR              $AR0, FromCPUMailLo
ROM:8045                    JMP             transfer_ucode
ROM:8045 ; End of function rom_start
ROM:8045
ROM:8047
ROM:8047 ; =============== S U B R O U T I N E =======================================
ROM:8047
ROM:8047
ROM:8047 wait_for_dma_finish:                                ; CODE XREF: wait_for_dma_finish+3↓j
ROM:8047                                                     ; sub_808B+6↓p ...
ROM:8047                    LRS             $AC0.M, DMAControl
ROM:8048                    ANDCF           $AC0.M, 4
ROM:804A                    JLZ             wait_for_dma_finish
ROM:804C                    RET
ROM:804C ; End of function wait_for_dma_finish
```

...

```
ROM:8078 ; =============== S U B R O U T I N E =======================================
ROM:8078
ROM:8078
ROM:8078 wait_for_mail:                                      ; CODE XREF: rom_start+A↑p
ROM:8078                                                     ; rom_start+13↑p ...
ROM:8078                    LRS             $AC0.M, FromCPUMailHi
ROM:8079                    ANDCF           $AC0.M, 0x8000
ROM:807B                    JLNZ            wait_for_mail
```

```
ROM:807D                 RET
ROM:807D ; End of function wait_for_mail

...

ROM:80C4 ; =============== S U B R O U T I N E =======================================
ROM:80C4 transfer_ucode:                                ; CODE XREF: rom_start+45↑j
ROM:80C4                                                ; sub_80B5+5↑j
ROM:80C4                 MRR            $AC0.M, $IX3
ROM:80C5 transfer the ucode from main mem -> DSP
ROM:80C5                 ANDI           $AC0.M, 0xFFFF
ROM:80C7                 JZ             jump_to_entry
ROM:80C9                 LRIS           $AC0.M, 2
ROM:80CA                 SRS            DMAControl, $AC0.M
ROM:80CB                 SR             DMAMMADDRH, $IX0
ROM:80CD                 SR             DMAMMADDRL, $IX1
ROM:80CF                 SR             DMADSPADDR, $IX2
ROM:80D1                 SR             DMALength, $IX3
ROM:80D3                 CALL           wait_for_dma_finish
ROM:80D5 ; jump to entrypoint
ROM:80D5 ; for MK5/AX: 0x0010
ROM:80D5
ROM:80D5 jump_to_entry:                                 ; CODE XREF: rom_start+C7↑j
ROM:80D5                 CLR            $ACC1
ROM:80D6                 LR             $AC1.M, DMALength
ROM:80D8                 JMPR           $AR0
ROM:80D8 ; END OF FUNCTION CHUNK FOR rom_start
```

The first thing it does is send the CPU `0x8071FEED` in the mail. Then it waits for the mail to be sent. It loads some registers with the values it receives. These values hold info on how to load the actual ucode from main memory. Once it has all the info it needs, it does a DMA and jumps to the entry point.

Pseudocode for this is

```
struct setup_data {
    u32 dma_mm_addr;  // IX0/IX1
    u16 dma_dsp_addr;  // IX2
    u16 dma_length;  // IX3
    u16 dma_control;  // AC0.M
    u16 entry_point;  // AR0
}

void rom_start() {
    // setup config and status reg
    while (true) {
        u16 mail_lo = wait_for_mail_recv();
        if (mail_lo == 0xa001) {
            setup_data.dma_mm_addr = wait_for_mail_recv();
        }
        else if (mail_lo == 0xc002) {
            setup_data.dma_dsp_addr = wait_for_mail_recv();  // low word
        }
        else if (mail_lo == 0xa002) {
            setup_data.dma_length = wait_for_mail_recv();  // low
        }
        else if (mail_lo == 0xb002) {
            setup_data.dma_control = wait_for_mail_recv();  // low
        }
```

```
        else if (mail_lo == 0xd001) {
            setup_data.entry_point = wait_for_mail_recv();  // low
            transfer_ucode();
        }
    }
}

void transfer_ucode() {
    (*DMAControl) = setup_data.dma_control;
    (*DMAMMAddrHi) = setup_data.dma_mm_addr >> 16;
    (*DMAMMAddrLo) = setup_data.dma_mm_addr;
    (*DMADSPAddr) = setup_data.dma_dsp_addr;
    wait_for_dma_finish();
    goto setup_data.entry_point;
}
```

Chapter 2

UCode

The main interesting part of the DSP's workings is the actual UCode itself. The main entrypoint (for Mortal Kombat 5 at least), is at `0x10`. The main thing it does is waiting for mail, and then processing a stream of commands (at 00 in DMEM).

The start of the UCode looks like this:

```
main_entry: ; 0x10
IRAM:0010                SBSET        2
IRAM:0011                SBSET        3
IRAM:0012                SBCLR        4
IRAM:0013                SBSET        5
IRAM:0014                SBSET        6
IRAM:0015                SET16
IRAM:0016                CLR15
IRAM:0017                M0
IRAM:0018                LRI          $CR, 0xFF
IRAM:001A                CLR          $ACC0
IRAM:001B                CLR          $ACC1
IRAM:001C                LRI          $AC0.M, 0xE80
IRAM:001E                SR           byte_E1B, $AC0.M
IRAM:0020                CLR          $ACC0
IRAM:0021                SR           byte_E31, $AC0.M
IRAM:0023 ; send initial mail (0x8000dcd1)
IRAM:0023                SI           ToCPUMailHi, 0xDCD1
IRAM:0025                SI           ToCPUMailLo, 0
IRAM:0027                SI           DIRQ, 1
IRAM:0029
IRAM:0029 wait_for_mail:                      ; CODE XREF: main_entry+1C↓j
IRAM:0029                LRS          $AC0.M, ToCPUMailHi
IRAM:002A                ANDF         $AC0.M, 0x8000
IRAM:002C                JLNZ         wait_for_mail
IRAM:002E                JMP          mail_sent
IRAM:0030 ; ---------------------------------------------------------------------------
IRAM:0030
IRAM:0030 send_dcd10001_irq:                  ; CODE XREF: j_send_dcd10001_irq↓j
IRAM:0030                SBSET        2
IRAM:0031                SBSET        3
IRAM:0032                SBCLR        4
IRAM:0033                SBSET        5
IRAM:0034                SBSET        6
IRAM:0035                SET16
IRAM:0036                CLR15
IRAM:0037                M0
IRAM:0038                LRI          $CR, 0xFF
```

```
IRAM:003A                   SI              ToCPUMailHi, 0xDCD1
IRAM:003C                   SI              ToCPUMailLo, 1
IRAM:003E                   SI              DIRQ, 1
IRAM:0040
IRAM:0040 wait_for_mail_sent:                        ; CODE XREF: main_entry+33↓j
IRAM:0040                   LRS             $AC0.M, ToCPUMailHi
IRAM:0041                   ANDF            $AC0.M, 0x8000
IRAM:0043                   JLNZ            wait_for_mail_sent
IRAM:0045
IRAM:0045 mail_sent:                               ; CODE XREF: main_entry+1E↑j
IRAM:0045                                            ; IRAM:0482↓j ...
IRAM:0045                   SET16
IRAM:0046                   CLR             $ACC0
IRAM:0047                   CLR             $ACC1
IRAM:0048                   LRI             $AC1.M, 0xBABE
IRAM:004A
IRAM:004A wait_for_babe:                             ; CODE XREF: main_entry+3D↓j
IRAM:004A                                            ; main_entry+40↓j
IRAM:004A                   LRS             $AC0.M, FromCPUMailHi
IRAM:004B                   ANDCF           $AC0.M, 0x8000
IRAM:004D                   JLNZ            wait_for_babe
IRAM:004F                   CMP
IRAM:0050                   JNZ             wait_for_babe
IRAM:0052 ; AX1.H contains the low part of the babe mail
IRAM:0052 ; this holds the DMA length
IRAM:0052                   LRS             $AX1.H, FromCPUMailLo
IRAM:0053                   CLR             $ACC0
IRAM:0054 ; wait for DMA mm address to be sent over mail
IRAM:0054 ; mail lo -> ac1 -> addr lo
IRAM:0054 ; mail hi -> ac0 -> addr hi
IRAM:0054
IRAM:0054 wait_for_dma_mm_addr:                      ; CODE XREF: main_entry+47↓j
IRAM:0054                   LRS             $AC0.M, FromCPUMailHi
IRAM:0055                   ANDCF           $AC0.M, 0x8000
IRAM:0057                   JLNZ            wait_for_dma_mm_addr
IRAM:0059                   LRS             $AC1.M, FromCPUMailLo
IRAM:005A                   ANDI            $AC0.M, 0x7FFF
IRAM:005C ; start the DMA
IRAM:005C ; length from babe mail
IRAM:005C ; mm address from second mail
IRAM:005C ; DMA control 0: to DSP DMEM
IRAM:005C                   SRS             DMAMMADDRH, $AC0.M
IRAM:005D                   SRS             DMAMMADDRL, $AC1.M
IRAM:005E                   SI              DMADSPADDR, 0xC00
IRAM:0060                   CLR             $ACC0
IRAM:0061                   SRS             DMAControl, $AC0.M ; set DMA control to 0
IRAM:0062                   MRR             $AC1.M, $AX1.H
IRAM:0063                   SRS             DMALength, $AC1.M
IRAM:0064                   CALL            wait_for_dma_finish_0
IRAM:0066                   LRI             $AR0, 0xC00
IRAM:0068
IRAM:0068 ; at the start of the commands:
IRAM:0068 ; ar0: word* cmd_stream_ptr
IRAM:0068
IRAM:0068 receive_command:                           ; CODE XREF: command_0:cmd0_done↓j
IRAM:0068                                            ; command_1+1F↓j ...
IRAM:0068                   SET16
IRAM:0069                   CLR             $ACC0
```

```
IRAM:006A                  CLR'L          $ACC1 : $AC0.M, @$AR0
IRAM:006B                  TST            $ACC0
IRAM:006C ; check current stream word
IRAM:006C ; jump if less than (top bit set, invalid command)
IRAM:006C                  JL             bad_mail
IRAM:006E                  LRIS           $AX0.H, 0x12
IRAM:006F                  CMPAR          $ACC0, $AX0.H
IRAM:0070 ; jump if word > 0x12
IRAM:0070                  JG             bad_mail
IRAM:0072 ; ar3 : addr = word + 0xaff // command_jump_table
IRAM:0072 ; ar3 : ac0.m : call_addr = [addr++]
IRAM:0072 ; jump call_addr
IRAM:0072                  LRI            $AC1.M, 0xAFF ; command_jump_table
IRAM:0074                  ADD            $ACC0, $ACC1 ; first word += 0xaff
IRAM:0075                  MRR            $AR3, $AC0.M
IRAM:0076                  ILRR           $AC0.M, @$AR3
IRAM:0077                  MRR            $AR3, $AC0.M
IRAM:0078                  JMPR           $AR3
IRAM:0079 ; ---------------------------------------------------------------------------
IRAM:0079 ; 0x8080FBAD mail [UNUSED]
IRAM:0079                  SI             ToCPUMailHi, 0xFBAD
IRAM:007B                  SI             ToCPUMailLo, 0x8080
IRAM:007D                  HALT
IRAM:007E ; ---------------------------------------------------------------------------
IRAM:007E
IRAM:007E bad_mail:                                      ; CODE XREF: main_entry+5C↑j
IRAM:007E                                                ; main_entry+60↑j
IRAM:007E                  SI             ToCPUMailHi, 0xBAAD
IRAM:0080                  SRS            ToCPUMailLo, $AC0.M
IRAM:0081                  HALT
IRAM:0081 ; End of function main_entry
```

The `command_jump_table` is a table with commands `0x0` through `0x11`, though the bounds check also allows for a command `0x12` to exist.

Pseudocode for this part could be

```
// at 0xaff
extern void (*)(u16* &command_stream) command_jump_table[0x12];


void main_entry() {
    // setup status and config registers
    // todo: write to byte_E1B and byte_E31
    send_mail(0xdcd1, 0x0000);
    send_irq();
    wait_for_mail_sent();

    do { } while ((*FromCPUMailHi) != 0xbabe);
    u16 dma_len = (*FromCPUMailLo);
    u32 dma_mmaddr = wait_for_mail_recv() & 0x7fff'ffff;
    dma_to_dmem(0xc00, dma_mmaddr, dma_len);
    wait_for_dma_finish();

    // AR0 holds the command stream pointer at the start of every command
    u16* command_stream = 0xc00;
    // receive_command
```

```
    while (true) {
        u16 command = *command_stream++;
        if ((i16)command < 0) {
            send_mail(0xBAAD, command);
            exit();  // halt
        }
        if (command > 0x12) {
            send_mail(0xBAAD, command);
            exit();  // halt
        }
        command_jump_table[command]();
    }
}
```

2.1 Commands

The commands all return with a `JMP receive_command`, save for command `0xf`, which does some sort of reset.

2.1.1 Command `0x0`

The assembly looks like

```
command_0:                              ; DATA XREF: IRAM:command_jump_table↓o
IRAM:0082                 CLR             $ACC0
IRAM:0083 ; load next two words from stream into ac0 and ac1
IRAM:0083                 CLR'L           $ACC1 : $AC0.M, @$AR0
IRAM:0084                 SET16'L         $AC1.M : @$AR0
IRAM:0085 ; store DMA address
IRAM:0085                 SRS             DMAMMADDRH, $AC0.M
IRAM:0086                 SRS             DMAMMADDRL, $AC1.M
IRAM:0087 ; DSPADDR = 0xe44
IRAM:0087                 LRI             $AC0.M, 0xE44
IRAM:0089                 SRS             DMADSPADDR, $AC0.M
IRAM:008A ; DMAControl = 0
IRAM:008A ; to DSP DMEM
IRAM:008A                 LRIS            $AC0.M, 0
IRAM:008B                 SRS             DMAControl, $AC0.M
IRAM:008C ; length = 0x40 8bit bytes
IRAM:008C                 LRI             $AC0.M, 0x40
IRAM:008E                 SRS             DMALength, $AC0.M
IRAM:008F ; setup registers and wait for DMA
IRAM:008F                 LRI             $AR1, 0xE44
IRAM:0091                 LRI             $AR2, 0
IRAM:0093                 LRI             $AX1.H, 0x9F
IRAM:0095                 LRI             $AX0.H, 0x140
IRAM:0097                 CLR             $ACC0
IRAM:0098                 CLR             $ACC1
IRAM:0099                 SET40
IRAM:009A                 CALL            wait_for_dma_finish_0
IRAM:009C ; Load 2 words from 0x40 byte stream (BASE)
IRAM:009C                 LRRI            $AC0.M, @$AR1
IRAM:009D                 LRRI            $AC0.L, @$AR1
IRAM:009E                 TST             $ACC0
IRAM:009F ; load third word from stream (INCR)
IRAM:009F                 LRRI            $AC1.M, @$AR1
```

```
IRAM:00A0 ; if BASE is not 0: jump
IRAM:00A0                   JNZ            cmd0_BASE_not_0 ; AC1.M ASR16 -> AC1.L
IRAM:00A2 ; zero out 0x140 words at the start of ARAM (AR2 set to 0)
IRAM:00A2 ; for (i = 0; i < 0x140; i++) *dest++ = 0;
IRAM:00A2                   LOOP           $AX0.H
IRAM:00A3                   SRRI           @$AR2, $AC0.M
IRAM:00A4                   JMP            cmd0_dmem_140_words_filled
IRAM:00A6 ; ---------------------------------------------------------------------
IRAM:00A6
IRAM:00A6 cmd0_BASE_not_0:                         ; CODE XREF: command_0+1E↑j
IRAM:00A6                   ASR16          $ACC1     ; AC1.M ASR16 -> AC1.L
IRAM:00A7 ; BASE to buffer at 0x0000
IRAM:00A7                   SRRI           @$AR2, $AC0.M
IRAM:00A8                   SRRI           @$AR2, $AC0.L
IRAM:00A9 ; loop 0x9f times
IRAM:00A9                   BLOOP          $AX1.H, loc_AD
IRAM:00AB ; BASE += INCR
IRAM:00AB
IRAM:00AB                   ADD            $ACC0, $ACC1
IRAM:00AC ; store BASE (with INCR added every loop)
IRAM:00AC ; 32 bit value
IRAM:00AC                   SRRI           @$AR2, $AC0.M
IRAM:00AD
IRAM:00AD loc_AD:                                    ; CODE XREF: command_0+27↑j
IRAM:00AD                   SRRI           @$AR2, $AC0.L
IRAM:00AE ; dest is now 0x140
IRAM:00AE ; load 2 more words from the DMA'ed stream (new BASE)
IRAM:00AE
IRAM:00AE cmd0_dmem_140_words_filled:                ; CODE XREF: command_0+22↑j
IRAM:00AE                   LRRI           $AC0.M, @$AR1
IRAM:00AF                   LRRI           $AC0.L, @$AR1
IRAM:00B0                   TST            $ACC0
IRAM:00B1 ; and another INCR word
IRAM:00B1                   LRRI           $AC1.M, @$AR1
IRAM:00B2 ; if BASE != 0: jump
IRAM:00B2                   JNZ            loc_B8    ; INCR ac1.m asr16 -> ac2.l
IRAM:00B4 ; zero out another 0x140 words if BASE is 0
IRAM:00B4                   LOOP           $AX0.H
IRAM:00B5                   SRRI           @$AR2, $AC0.M
IRAM:00B6                   JMP            cmd0_another_140_words_filled
IRAM:00B8 ; ---------------------------------------------------------------------
IRAM:00B8
IRAM:00B8 loc_B8:                                    ; CODE XREF: command_0+30↑j
IRAM:00B8                   ASR16          $ACC1     ; INCR ac1.m asr16 -> ac2.l
IRAM:00B9 ; store BASE to dest
IRAM:00B9                   SRRI           @$AR2, $AC0.M
IRAM:00BA                   SRRI           @$AR2, $AC0.L
IRAM:00BB ; for (int i = 0; i < 0x9f; i++, BASE += INCR) {
IRAM:00BB ;       *dest++ = BASE >> 16;
IRAM:00BB ;       *dest++ = (word)BASE
IRAM:00BB ; }
IRAM:00BB                   BLOOP          $AX1.H, loc_BF
IRAM:00BD                   ADD            $ACC0, $ACC1
IRAM:00BE                   SRRI           @$AR2, $AC0.M
IRAM:00BF
IRAM:00BF loc_BF:                                    ; CODE XREF: command_0+39↑j
IRAM:00BF                   SRRI           @$AR2, $AC0.L
IRAM:00C0 ; dest is now 0x280
```

```
IRAM:00C0 ; same thing again
IRAM:00C0
IRAM:00C0 cmd0_another_140_words_filled:         ; CODE XREF: command_0+34↑j
IRAM:00C0                 LRRI            $AC0.M, @$AR1
IRAM:00C1                 LRRI            $AC0.L, @$AR1
IRAM:00C2                 TST             $ACC0
IRAM:00C3                 LRRI            $AC1.M, @$AR1
IRAM:00C4                 JNZ             loc_CA
IRAM:00C6                 LOOP            $AX0.H
IRAM:00C7                 SRRI            @$AR2, $AC0.M
IRAM:00C8                 JMP             cmd0_another_140_words_filled_1
IRAM:00CA ; ---------------------------------------------------------------------------
IRAM:00CA
IRAM:00CA loc_CA:                                ; CODE XREF: command_0+42↑j
IRAM:00CA                 ASR16           $ACC1
IRAM:00CB                 SRRI            @$AR2, $AC0.M
IRAM:00CC                 SRRI            @$AR2, $AC0.L
IRAM:00CD                 BLOOP           $AX1.H, loc_D1
IRAM:00CF                 ADD             $ACC0, $ACC1
IRAM:00D0                 SRRI            @$AR2, $AC0.M
IRAM:00D1
IRAM:00D1 loc_D1:                                ; CODE XREF: command_0+4B↑j
IRAM:00D1                 SRRI            @$AR2, $AC0.L
IRAM:00D2 ; At this point, 3 * 0x140 = 0x3c0 words are filled at the start of DMEM
IRAM:00D2 ; ar2: dest = 0x400 // skip 0x40 bytes
IRAM:00D2
IRAM:00D2 cmd0_another_140_words_filled_1:       ; CODE XREF: command_0+46↑j
IRAM:00D2                 LRI             $AR2, 0x400
IRAM:00D4 ; again, load BASE and INCR
IRAM:00D4                 LRRI            $AC0.M, @$AR1
IRAM:00D5                 LRRI            $AC0.L, @$AR1
IRAM:00D6                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:00D7                 JNZ             loc_DD
IRAM:00D9                 LOOP            $AX0.H
IRAM:00DA                 SRRI            @$AR2, $AC0.M
IRAM:00DB                 JMP             cmd0_140_filled_at_400
IRAM:00DD ; ---------------------------------------------------------------------------
IRAM:00DD
IRAM:00DD loc_DD:                                ; CODE XREF: command_0+55↑j
IRAM:00DD                 ASR16           $ACC1
IRAM:00DE                 SRRI            @$AR2, $AC0.M
IRAM:00DF                 SRRI            @$AR2, $AC0.L
IRAM:00E0                 BLOOP           $AX1.H, loc_E4
IRAM:00E2                 ADD             $ACC0, $ACC1
IRAM:00E3                 SRRI            @$AR2, $AC0.M
IRAM:00E4
IRAM:00E4 loc_E4:                                ; CODE XREF: command_0+5E↑j
IRAM:00E4                 SRRI            @$AR2, $AC0.L
IRAM:00E5 ; again load BASE and INCR and fill 140 words
IRAM:00E5
IRAM:00E5 cmd0_140_filled_at_400:                ; CODE XREF: command_0+59↑j
IRAM:00E5                 LRRI            $AC0.M, @$AR1
IRAM:00E6                 LRRI            $AC0.L, @$AR1
IRAM:00E7                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:00E8                 JNZ             loc_EE
IRAM:00EA                 LOOP            $AX0.H
IRAM:00EB                 SRRI            @$AR2, $AC0.M
IRAM:00EC                 JMP             cmd0_140_filled_at_540
```

```
IRAM:00EE ; ---------------------------------------------------------------------------
IRAM:00EE
IRAM:00EE loc_EE:                                       ; CODE XREF: command_0+66↑j
IRAM:00EE                 ASR16           $ACC1
IRAM:00EF                 SRRI            @$AR2, $AC0.M
IRAM:00F0                 SRRI            @$AR2, $AC0.L
IRAM:00F1                 BLOOP           $AX1.H, loc_F5
IRAM:00F3                 ADD             $ACC0, $ACC1
IRAM:00F4                 SRRI            @$AR2, $AC0.M
IRAM:00F5
IRAM:00F5 loc_F5:                                       ; CODE XREF: command_0+6F↑j
IRAM:00F5                 SRRI            @$AR2, $AC0.L
IRAM:00F6 ; same thing again
IRAM:00F6
IRAM:00F6 cmd0_140_filled_at_540:                       ; CODE XREF: command_0+6A↑j
IRAM:00F6                 LRRI            $AC0.M, @$AR1
IRAM:00F7                 LRRI            $AC0.L, @$AR1
IRAM:00F8                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:00F9                 JNZ             loc_FF
IRAM:00FB                 LOOP            $AX0.H
IRAM:00FC                 SRRI            @$AR2, $AC0.M
IRAM:00FD                 JMP             cmd0_140_filled_at_680
IRAM:00FF ; ---------------------------------------------------------------------------
IRAM:00FF
IRAM:00FF loc_FF:                                       ; CODE XREF: command_0+77↑j
IRAM:00FF                 ASR16           $ACC1
IRAM:0100                 SRRI            @$AR2, $AC0.M
IRAM:0101                 SRRI            @$AR2, $AC0.L
IRAM:0102                 BLOOP           $AX1.H, loc_106
IRAM:0104                 ADD             $ACC0, $ACC1
IRAM:0105                 SRRI            @$AR2, $AC0.M
IRAM:0106
IRAM:0106 loc_106:                                      ; CODE XREF: command_0+80↑j
IRAM:0106                 SRRI            @$AR2, $AC0.L
IRAM:0107 ; at this point, dest is already 0x7c0, not sure why the DSP loads it directly
IRAM:0107 ; going to do the same thing yet again
IRAM:0107
IRAM:0107 cmd0_140_filled_at_680:                       ; CODE XREF: command_0+7B↑j
IRAM:0107                 LRI             $AR2, 0x7C0
IRAM:0109                 LRRI            $AC0.M, @$AR1
IRAM:010A                 LRRI            $AC0.L, @$AR1
IRAM:010B                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:010C                 JNZ             loc_112
IRAM:010E                 LOOP            $AX0.H
IRAM:010F                 SRRI            @$AR2, $AC0.M
IRAM:0110                 JMP             cmd0_140_filled_at_7c0
IRAM:0112 ; ---------------------------------------------------------------------------
IRAM:0112
IRAM:0112 loc_112:                                      ; CODE XREF: command_0+8A↑j
IRAM:0112                 ASR16           $ACC1
IRAM:0113                 SRRI            @$AR2, $AC0.M
IRAM:0114                 SRRI            @$AR2, $AC0.L
IRAM:0115                 BLOOP           $AX1.H, loc_119
IRAM:0117                 ADD             $ACC0, $ACC1
IRAM:0118                 SRRI            @$AR2, $AC0.M
IRAM:0119
IRAM:0119 loc_119:                                      ; CODE XREF: command_0+93↑j
IRAM:0119                 SRRI            @$AR2, $AC0.L
```

```
IRAM:011A ; going to do the same thing again
IRAM:011A ; dest is now 0x900
IRAM:011A
IRAM:011A cmd0_140_filled_at_7c0:                     ; CODE XREF: command_0+8E↑j
IRAM:011A                 LRRI            $AC0.M, @$AR1
IRAM:011B                 LRRI            $AC0.L, @$AR1
IRAM:011C                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:011D                 JNZ             loc_123
IRAM:011F                 LOOP            $AX0.H
IRAM:0120                 SRRI            @$AR2, $AC0.M
IRAM:0121                 JMP             cmd0_140_filled_at_900
IRAM:0123 ; ---------------------------------------------------------------------------
IRAM:0123
IRAM:0123 loc_123:                                    ; CODE XREF: command_0+9B↑j
IRAM:0123                 ASR16           $ACC1
IRAM:0124                 SRRI            @$AR2, $AC0.M
IRAM:0125                 SRRI            @$AR2, $AC0.L
IRAM:0126                 BLOOP           $AX1.H, loc_12A
IRAM:0128                 ADD             $ACC0, $ACC1
IRAM:0129                 SRRI            @$AR2, $AC0.M
IRAM:012A
IRAM:012A loc_12A:                                    ; CODE XREF: command_0+A4↑j
IRAM:012A                 SRRI            @$AR2, $AC0.L
IRAM:012B ; dest is now 0xa40
IRAM:012B ; same thing again
IRAM:012B
IRAM:012B cmd0_140_filled_at_900:                     ; CODE XREF: command_0+9F↑j
IRAM:012B                 LRRI            $AC0.M, @$AR1
IRAM:012C                 LRRI            $AC0.L, @$AR1
IRAM:012D                 TST'L           $ACC0 : $AC1.M, @$AR1
IRAM:012E                 JNZ             loc_134
IRAM:0130                 LOOP            $AX0.H
IRAM:0131                 SRRI            @$AR2, $AC0.M
IRAM:0132                 JMP             cmd0_done
IRAM:0134 ; ---------------------------------------------------------------------------
IRAM:0134
IRAM:0134 loc_134:                                    ; CODE XREF: command_0+AC↑j
IRAM:0134                 ASR16           $ACC1
IRAM:0135                 SRRI            @$AR2, $AC0.M
IRAM:0136                 SRRI            @$AR2, $AC0.L
IRAM:0137                 BLOOP           $AX1.H, loc_13B
IRAM:0139                 ADD             $ACC0, $ACC1
IRAM:013A                 SRRI            @$AR2, $AC0.M
IRAM:013B
IRAM:013B loc_13B:                                    ; CODE XREF: command_0+B5↑j
IRAM:013B                 SRRI            @$AR2, $AC0.L
IRAM:013C ; dest should end up at 0xb80
IRAM:013C
IRAM:013C cmd0_done:                                  ; CODE XREF: command_0+B0↑j
IRAM:013C                 JMP             receive_command
IRAM:013C ; End of function command_0
```

The point of this is to fill 3 regions of memory with either 0's, or incrementing values. Which of the 2 depends on the values from a `0x40` byte stream DMAd from main memory.

Note that we are reading a `base` and an `incr` 9 times from the stream, which would amount to 9 * `0x6` = `0x36` bytes, so the DMA transfers 4 bytes too many.

I suspect that the incrementing values are a main memory address and strides.

The address regions 0x0000 - 0x03c0, 0x0400 - 0x07c0 and 0x07c0 - 0x0b80 will be used in most other commands.

Pseudocode for this could be

```
void command_0(u16* &command_stream) {
    u16 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    dma_to_dmem(0xe44, mmaddr, 0x40);

    u16* stream = 0xe44;  // AR1
    u16* buffer = 0;  // AR2
    // constants 0x9f and 0x140 in AX0/1.H
    wait_for_dma_finish();
    u32 base;
    i16 incr;
    foreach (u16* buffer in {0x0000, 0x0400, 0x07c0}) {
        // unrolled in the assembly
        for (int i = 0; i < 3; i++) {
            // unrolled in the assembly
            base = ((*stream++) << 16) | *stream++;
            incr = *stream++;
            if (base) {
                int j = 0;
                do {
                    *buffer = *base;
                    base += incr;
                    j++;
                } while (j < 0x140);
            }
            else {
                memset(buffer, 0, 0x140);  // in words, not bytes
            }
        }
    }
}
```

2.1.2 Command 0x1

2.1.3 Command 0x2

2.1.4 Command 0x3

2.1.5 Command 0x4, 0x5 and 0x9

These commands are all very similar. Command 0x9 only calls sub_484 with a pointer to the buffer at 0x7c0, while 0x4 and 0x5 DMA the buffers at 0x400 and 0x7c0 respectively, before also calling sub_484 with their respective buffers as arguments. Since they are so similar, I will only put the assembly for command 0x4 here.

```
IRAM:0413 command_4:                         ; DATA XREF: IRAM:command_jump_table↓o
IRAM:0413                 SET16
IRAM:0414 ; DMA 0x780 bytes to main mem from DSP DMEM 0x400
IRAM:0414 ; MMADDR read from command stream
IRAM:0414 ; then call sub_484 with 0x400
IRAM:0414                 LRI           $IX2, 0x400
```

```
IRAM:0416                    CLR           $ACC0
IRAM:0417                    CLR'L         $ACC1 : $AC0.M, @$AR0
IRAM:0418                    LRRI          $AC0.L, @$AR0
IRAM:0419                    SRS           DMAMMADDRH, $AC0.M
IRAM:041A                    SRS           DMAMMADDRL, $AC0.L
IRAM:041B                    MRR           $AC0.M, $IX2
IRAM:041C                    SRS           DMADSPADDR, $AC0.M
IRAM:041D                    SI            DMAControl, 1
IRAM:041F                    SI            DMALength, 0x780
IRAM:0421                    CALL          wait_for_dma_finish_0
IRAM:0423                    CALL          sub_484
IRAM:0425                    JMP           receive_command
```

And the pseudocode for `0x4` and `0x5` is the same, except `0x5` uses `0x7c0` instead of `0x400`:

```
void sub_484(u16* buffer);  // in in IX2

void command_9(u16* &command_stream) {
    sub_484(0x7c0);
}

void command_4(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    // 0x780 bytes, so precisely 0x3c0 words
    dma_dmem_to_mmem(mmaddr, 0x400, 0x780);
    wait_for_dma_finish();
    sub_484(0x400);
}
```

2.1.6 Command `0x6`

2.1.7 Command `0x7`

2.1.8 Command `0x8`

2.1.9 Command `0xa - 0xc`

These commands immediately return on call.

2.1.10 Command `0xd`

This command loads a new command stream to DMEM and resets the `command_stream` pointer.

```
IRAM:01A9 command_d:                                       ; DATA XREF: IRAM:command_jump_table↓o
IRAM:01A9                    SET16'L       $AC0.M : @$AR0
IRAM:01AA ; load main memory address and length from command stream
IRAM:01AA                    CLR'L         $ACC1 : $AC0.L, @$AR0
IRAM:01AB                    LRRI          $AC1.M, @$AR0
IRAM:01AC ; DMA to command stream address
IRAM:01AC                    SRS           DMAMMADDRH, $AC0.M
IRAM:01AD                    SRS           DMAMMADDRL, $AC0.L
IRAM:01AE                    SI            DMADSPADDR, 0xC00
IRAM:01B0                    SI            DMAControl, 0
IRAM:01B2                    ADDIS         $AC1.M, 3
IRAM:01B3                    ANDI          $AC1.M, 0xFFF0
```

```
IRAM:01B5 ; round to 16 byte blocks
IRAM:01B5 ; DMALen = (len_from_stream + 3) & 0xfff0
IRAM:01B5                    SRS            DMALength, $AC1.M
IRAM:01B6                    CALL           wait_for_dma_finish_0
IRAM:01B8                    LRI            $AR0, 0xC00
IRAM:01BA                    JMP            receive_command
```

Pseudocode for this could be

```
void command_d(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    u16 len = *command_stream++;
    dma_to_dmem(0xc00, mmaddr, (len + 3) & 0xfff0);
    wait_for_dma_finish();
    command_stream = 0xc00;
}
```

2.1.11 Command 0xe

2.1.12 Command 0xf

2.1.13 Command 0x10

2.1.14 Command 0x11