

AXRE

A GameCube DSP UCode Documentation

Author: DenSinH

July 4, 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 DMAMMAAddrHi ((volatile u16*)0xffce)  
#define DMAMMAAddrLo ((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!  
    (*DMAMMAAddrHi) = mmaddr >> 16;  
    (*DMAMMAAddrLo) = 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 of 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:
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:
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:
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:
ROM: 8026          LRI          $A0.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:
ROM: 8031          LRI          $A0.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:
ROM: 803C          LRI          $A0.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_ucose
ROM: 8045 ; End of function rom_start
ROM: 8045
ROM: 8047 ; ===== S U B R O U T I N E =====
ROM: 8047
ROM: 8047
ROM: 8047 wait_for_dma_finish:
ROM: 8047 ; sub_808B+64p ...
ROM: 8047          LRS          $A0.M, DMAControl
ROM: 8048          ANDCF        $A0.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:
ROM: 8078 ; rom_start+13tp ...
ROM: 8078          LRS          $A0.M, FromCPUMailHi
ROM: 8079          ANDCF        $A0.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:
ROM:80C4          ; sub_80B5+5+j
ROM:80C4          MRR          $ACO.M, $IX3
ROM:80C5 transfer the ucode from main mem -> DSP
ROM:80C5          ANDI          $ACO.M, 0xFFFF
ROM:80C7          JZ           jump_to_entry
ROM:80C9          LRIS          $ACO.M, 2
ROM:80CA          SRS           DMAControl, $ACO.M
ROM:80CB          SR            DMAMADDRH, $IX0
ROM:80CD          SR            DMAMADDRL, $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:
ROM:80D5          CLR           $ACC1
ROM:80D6          LR            $AC1.M, DMALength
ROM:80D8          JMPR          $ARO
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; // ACO.M
    u16 entry_point; // ARO
}

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          MO
IRAM:0018          LRI            $CR, 0xFF
IRAM:001A          CLR            $AC0
IRAM:001B          CLR            $AC1
IRAM:001C          LRI            $AC0.M, 0xE80
IRAM:001E          SR             byte_E1B, $AC0.M
IRAM:0020          CLR            $AC0
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:
IRAM:0029          LRS            $AC0.M, ToCPUMailHi
IRAM:002A          ANDF           $AC0.M, 0x8000
IRAM:002C          JLNZ           wait_for_mail
IRAM:002E          JMP            wait_for_babe
IRAM:0030          ; -----
IRAM:0030
IRAM:0030 send_dcd10001_irq:
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          MO
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:
IRAM:0040          LRS          $ACO.M, ToCPUMailHi
IRAM:0041          ANDF         $ACO.M, 0x8000
IRAM:0043          JLNZ         wait_for_mail_sent
IRAM:0045
IRAM:0045 wait_for_babe:
IRAM:0045                                     ; IRAM:0482↓j ...
IRAM:0045          SET16
IRAM:0046          CLR          $ACCO
IRAM:0047          CLR          $ACC1
IRAM:0048          LRI          $AC1.M, 0xBABE
IRAM:004A
IRAM:004A wait_for_babe_loop:
IRAM:004A                                     ; main_entry+40↓j
IRAM:004A          LRS          $ACO.M, FromCPUMailHi
IRAM:004B          ANDCF        $ACO.M, 0x8000
IRAM:004D          JLNZ         wait_for_babe_loop
IRAM:004F          CMP
IRAM:0050          JNZ          wait_for_babe_loop
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          $ACCO
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:
IRAM:0054          LRS          $ACO.M, FromCPUMailHi
IRAM:0055          ANDCF        $ACO.M, 0x8000
IRAM:0057          JLNZ         wait_for_dma_mm_addr
IRAM:0059          LRS          $AC1.M, FromCPUMailLo
IRAM:005A          ANDI         $ACO.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          DMAMADDRH, $ACO.M
IRAM:005D          SRS          DMAMADDRL, $AC1.M
IRAM:005E          SI          DMADSPADDR, 0xC00
IRAM:0060          CLR          $ACCO
IRAM:0061          SRS          DMAControl, $ACO.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:
IRAM:0068                                     ; command_1+1F↓j ...
IRAM:0068          SET16
IRAM:0069          CLR          $ACCO

```

```

IRAM:006A          CLR'L          $ACC1 : $ACO.M, @ $ARO
IRAM:006B          TST           $ACCO
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         $ACCO, $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           $ACCO, $ACC1 ; first word += 0xaff
IRAM:0075          MRR           $AR3, $ACO.M
IRAM:0076          ILRR          $ACO.M, @ $AR3
IRAM:0077          MRR           $AR3, $ACO.M
IRAM:0078          JMPR          $AR3
IRAM:0079 ; -----
IRAM:0079 ; 0x8080FBAD mail (if command does not jump to receive command)
IRAM:0079          SI           ToCPUMailHi, 0xFBAD
IRAM:007B          SI           ToCPUMailLo, 0x8080
IRAM:007D          HALT
IRAM:007E ; -----
IRAM:007E bad_mail:
IRAM:007E                                     ; main_entry+60+j
IRAM:007E          SI           ToCPUMailHi, 0xBAAD
IRAM:0080          SRS           ToCPUMailLo, $ACO.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. There are pointers to what appear to be functions past command 0x11, but these do not return in the way the other commands do (`JMP receive_command`), and would cause the DSP to send 0x8080FBAD in the mail and halt.

Pseudocode for this part could be

```

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

extern u16 data_E1B, data_E31;

void main_entry() {
    // setup status and config registers
    data_E1B = 0xe80;
    data_E31 = 0;

    send_mail(0xdcd1, 0x0000);
    send_irq();
    wait_for_mail_sent();
    goto wait_for_babe;

send_dcd10001_irq:

```

```

    // this part is only used in command f
    send_mail(0xdcd1, 0x0001);
    send_irq();
    wait_for_mail_sent();
wait_for_babe:

    do {
        wait_for_mail_recv();
    } 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();

    // ARO 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 Memory layout

There are different important areas in DMEM:

Start	Length	Description
0x0	0x140	Data buffer section filled with 32 bit values
0x140	0x140	Data buffer section filled with 32 bit values
0x280	0x140	Data buffer section filled with 32 bit values
0x3c0	0x80	Some sort of struct that is only used in command 0x3
0x400	0x3c0	Data buffer similar to those at 0x0, filled with 32 bit values
0x7c0	0x3c0	Data buffer similar to those at 0x0, filled with 32 bit values
0xb80	0x80	Some sort of struct with data used in different commands
0xc00	0xc0?	Command stream
0xcc0	0x20	Data stream referenced by different commands
0xe14	0x1	Some function pointer
0xe15	0x1	Some function pointer
0xe16	0x1	Some data
0xe40	0x4	Some pointers into the stream at 0xcc0. 0xe40 and 0xe41 seem to indicate the “current” position, and 0xe42 and 0xe43 seem to indicate the end (0xce0).
0xe44	0x60	Scratchpad region
0xea4	0x60	Scratchpad region

2.2 Commands

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

2.2.1 Command 0x0

The assembly is at `??`. 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 DMA'd 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
        }
    }
}

```

2.2.2 Command 0x1

Transforms the buffers setup by command 0x0 with data gotten from main memory. Assembly is at ???. Pseudocode for this could be

```

void command_1(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++; // AX0

    i16 scale = *command_stream++; // AX1.L
    transform_buffer(mmaddr, scale, 0x0);
    scale = *command_stream++;
    transform_buffer(mmaddr, scale, 0x400);
    scale = *command_stream++;
    transform_buffer(mmaddr, scale, 0x7c0);
}

void transform_buffer(u32 mmaddr, i16 scale, u16* buffer) {
    dma_to_dmem(0xea4, mmaddr, 0xc0); // bytes, not words
    mmaddr += 0xc0;

    // note: we call transform_buffer_section a total of 4 * 2 + 2 times
    // this function transforms 0x30 u32's
    // that's a total of (4 * 2 + 2) * 0x30 * 2 = 0x3c0 DSP words transformed!

    wait_for_dma_finish();
    for (int i = 0; i < 4; i++) {
        dma_to_dmem(0xea4, mmaddr, 0xc0); // bytes, not words
        mmaddr += 0xc0;
        transform_buffer_section(0xea4, scale, buffer);

        dma_to_dmem(0xea4, mmaddr, 0xc0); // bytes, not words
        mmaddr += 0xc0;
    }
}

```

```

        transform_buffer_section(0xea4, scale, buffer);
    }
    dma_to_dmem(0xea4, mmaddr, 0xc0);
    mmaddr += 0xc0;

    transform_buffer_section(0xe44, scale, buffer);
    transform_buffer_section(0xea4, scale, buffer);
}

void transform_buffer_section(u16* data, i16 scale, u16* &buffer) {
    // data in AR3
    // buffer in AR1, IX1 = -1 to not change AR1 in first read
    // scale in AX1.L
    u32 base = ((*data++) << 16) | (*data++); // AX0
    for (int i = 0; i < 0x30; i++) {
        i32 data_value = ((*data++) << 16) | *(data++);
        i32 buffer_value = ((*buffer) << 16) | *(buffer + 1);
        i32 scaled = (data_value * scale) >> 16;
        scaled += buffer_value;
        *buffer++ = scaled >> 16;
        *buffer++ = scaled;
    }
}

```

2.2.3 Command 0x2

This DMA's a struct of settings from main memory to 0x0b80. It stores pointers to buffer sections to 0x0e08. It also DMA's data to the intermediate section at 0x03c0.

Depending on the data in the DMA'd struct, it either sets some pointers to 0x0ce0 (end of command stream?), or it overwrites the command stream with new data and sets the pointers to addresses relative to 0x0cc0 (command stream start). Assembly is at ??.

And pseudocode could be

```

extern u16* buffer_sections[9]; // DMEM E08

extern u16 data_e14, data_e15, data_e16;
extern u16* data_e40, data_e41, data_e42, data_e43;
extern struct* structb80;

void command_2(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | (*command_stream++);

    // the DMA'd data is not a simple array, but a struct
    dma_to_dmem(structb80, mmaddr, 0xc0);

    buffer_sections = {
        0x0, 0x140, 0x280, 0x400, 0x540, 0x680, 0x7c0, 0x900, 0xa40
    };
    wait_for_dma_finish();

    mmaddr = (structb80[0x27] << 16) | structb80[0x28];
}

```

```

dma_to_dmem(0x3c0, mmaddr, 0x80);

data_e15 = (structb80[0x4]) + 0xb31;
data_e16 = (structb80[0x5]) + 0xb34;
data_e14 = (structb80[0x6]) + 0xb11;

if (structb80[0x1b]) {
    data_e40 = 0xcc0 + structb80[0x1e];
    data_e41 = 0xcc0 + structb80[0x1f];
    data_e42 = 0xce0;
    data_e43 = 0xce0;

    wait_for_dma_finish();

    mmaddr = (structb80[0x1c] << 16) | structb80[0x1d];
    dma_to_dmem(0xcc0, mmaddr, 0x40);
}
else {
    data_e40 = 0xce0; // address
    data_e41 = 0xce0; // address
    data_e42 = 0xce0; // address
    data_e43 = 0xce0; // address
    wait_for_dma_finish();
}
}

```

2.2.4 Command 0x3

This command uses the struct transferred by command 0x2 to transfer and transform other data. The code is quite complex, but here is the assembly. Assembly is at ??.

And pseudocode could be

```

extern u16* buffer_sections[9]; // DMEM E08

extern u16 data_e14, data_e15, data_e16;
extern u16* data_e40, data_e41, data_e42, data_e43;
extern struct* structb80;
extern struct* struct3c0;

void command_3(u16* &command_stream) {
    while (true) {
        u16* buffer_ar0 = &structb80[0x22];
        u16* buffer_ar1 = struct3c0;
        u16* ptr_elc;

        // loop counter stored at OE04
        for (int i = 0; i < 5; i++) {

            const u16 times = *buffer_ar0++;
            for (int j = 0; j < times; j++) {
                structb80[*buffer_ar1++] = *buffer_ar1++;
            }
        }
    }
}

```



```

}

u16* dest_buffer;
if (structb80[0x7] == 1) {
    byte_e1c = data_e42;
    ((void (*)( ))data_e15)();

    dest_buffer = 0xe44;
    i32 step = i16(structb80[0x32]) * i16(structb80[0x33] << 1);
    u32 value0 = structb80[0x32] << 16;
    u32 value1 = value0 + (structb80[0x33] << 16);

    for (int j = 0; j < 16; j++) {
        *dest_buffer++ = value0 >> 16;
        value0 += step;
        *dest_buffer++ = value1 >> 16;
        value1 += step;
    }

    structb80[0x32] = value0;
    u16* src_buffer = 0xe44; // ARO
    dest_buffer = data_e43; // AR3

    for (int j = 0; j < 32; j++) {
        *dest_buffer = (*src_buffer * *dest_buffer) >> 16;
        dest_buffer++;
        src_buffer++;
    }

    ((void (*)( ))data_e14)();

    if (structb80[0x1b]) {
        data_e43 = data_e42;
        if (structb80[0x1e] <= structb80[0x20]) {
            structb80[0x1e]++
        }
        else {
            structb80[0x1e]--;
        }
        data_e40 = data_e43 - 0x20 + structb80[0x1e];

        if (structb80[0x1f] <= structb80[0x21]) {
            structb80[0x1f]++
        }
        else {
            structb80[0x1f]--;
        }
        data_e41 = data_e43 - 0x20 + structb80[0x1f];
    }
    else {
        data_e40 = data_e41 = data_e43 = data_e42;
    }
}

```

```

    }
}

for (int j = 0; j < 9; j++) {
    buffer_sections[j] += 0x40;
}

}

u32 mmaddr;
if (structb80[0x1b]) {
    mmaddr = (structb80[0x1c] << 16) | structb80[0x1d];
    dma_dmem_to_mmem(mmaddr, ptr_e1c, 0x40);
    wait_for_dma_finish();
}

// DMA struct back to main memory
mmaddr = (structb80[0x2] << 16) | structb80[0x3];
dma_dmem_to_mmem(mmaddr, 0xb80, 0xc0);
wait_for_dma_finish();

mmaddr = (structb80[0x0] << 16) | structb80[0x1];
if (!mmaddr) {
    return;
}

if (mmaddr) {
    // same setup as command 2
    dma_to_dmem(structb80, mmaddr, 0xc0);

    buffer_sections = {
        0x0, 0x140, 0x280, 0x400, 0x540, 0x680, 0x7c0, 0x900, 0xa40
    };
    wait_for_dma_finish();

    mmaddr = (structb80[0x27] << 16) | structb80[0x28];
    dma_to_dmem(0x3c0, mmaddr, 0x80);

    data_e15 = (structb80[0x4]) + 0xb31;
    data_e16 = (structb80[0x5]) + 0xb34;
    data_e14 = (structb80[0x6]) + 0xb11;

    if (structb80[0x1b]) {
        data_e40 = 0xcc0 + (structb80[0x1e]);
        data_e41 = 0xcc0 + (structb80[0x1f]);
        data_e42 = 0xce0;
        data_e43 = 0xce0;

        wait_for_dma_finish();

        mmaddr = (structb80[0x1c] << 16) | structb80[0x1d];
        dma_to_dmem(0xcc0, mmaddr, 0x40);
    }
}

```

```

    }
    else {
        data_e40 = 0xce0; // address
        data_e41 = 0xce0; // address
        data_e42 = 0xce0; // address
        data_e43 = 0xce0; // address
        wait_for_dma_finish();
    }
}
}
}

```

2.2.5 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 in this document. Assembly is at ??.

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

```

void command_9(u16* &command_stream) {
    mix_buffers(command_stream, 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();
    mix_buffers(command_stream, 0x400);
}

```

This function `sub_484` DMA's a new buffer from main memory to the buffer passed as argument, and adds it to the current buffer at 0x0. Assembly is at ??.

Pseudocode is

```

void mix_buffers(u16* &command_stream, u16* const buffer) {
    // buffer in IX2
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    u16* dspaddr = buffer;
    dma_to_dmem(dspaddr, mmaddr, 0xc0);
    wait_for_dma_finish();

    u16* buffer_0_3, buffer_0_2 = 0; // AR3/AR2
    u16* _buffer; // ARO

    for (int i = 0; i < 9; i++) {
        _buffer = dspaddr;

        // start DMA for next section
        mmaddr += 0xc0;
    }
}

```

```

    dspaddr += 0x60;
    dma_to_dmem(dspaddr, mmaddr, 0xc0);

    // process current section
    for (int j = 0; j < 0x30; j++) {
        *(u32*)buffer_0 += *(u32*)_buffer;
        _buffer += 2;
        buffer_0 += 2;
    }
}
// process last section
for (int j = 0; j < 0x30; j++) {
    *(u32*)buffer_0 += *(u32*)_buffer;
    _buffer += 2;
    buffer_0 += 2;
}
}

```

2.2.6 Command 0x6

Command 6 simply transfers the buffer at 0x0 back to main memory. Assembly is at ???. Pseudocode is

```

void command_6(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    dma_dmem_to_mmem(mmaddr, 0, 0x780);
    wait_for_dma_finish();
}

```

2.2.7 Command 0x7

clears out 0x140 word section at 0x0000, DMAs 0x140 words of data from main memory to 0xe44 and copies it over to 0x0140 and 0x280, completely filling the buffer at 0x0000. Assembly is at ??.

Pseudocode is

```

void command_7(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;

    // start transfer for short section
    dma_to_dmem(0xe44, mmaddr, 0x20);
    mmaddr += 0x20;
    wait_for_dma_finish();

    // start transfer for rest to process data while DMA is running
    dma_to_dmem(0xe54, mmaddr, 0x260);

    memset(0x280, 0, 0x140); // words
    memcpy(0, 0xe44, 0x140); // words
    memcpy(0x140, 0xe44, 0x140); // words
}

```

2.2.8 Command 0x8

This command is very confusing, since there is either a bug in the UCode or in dolphin and the doc by Duddie. The command saves a main memory address in AX1, then later uses LD extended opcodes to load values into AX1 from the COEF region in memory for some calculation. At the end, it should restore the main memory address, and DMA data to main memory. If dolphin and the doc by Duddie are correct though, this DMA will happen to a pretty random address. I assume there is an error in the way that they describe that the LD extended opcode should happen. Assembly is at ??.

Pseudocode is

```
extern u16* data_E1B;

void command_8(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;
    dma_to_dmem(0xe80, mmaddr, 0x100);
    wait_for_dma_finish();

    mmaddr = ((*command_stream++) << 16) | *command_stream++;
    dma_to_dmem(0x280, mmaddr, 0x280);
    wait_for_dma_finish();

    u16* buffer_280 = 0x280;
    u16* const ptr_E1B = data_E1B; // set by main_entry to 0xe80
    u16* buffer_f00 = 0xf00;

    buffer_280++;
    *ptr_E1B = *buffer_280++;

    // first and last iterations are unrolled in assembly
    for (int i = 0; i < 0xa0; i++) {
        u16* coef = 0x16b4; // in DSP_COEF memory region

        i16 current_coef = *coef++;
        i64 prod = 0;
        for (int j = 0; j < 0x7f; j++) {
            prod += current_coef * current_coef;
            current_coef = *coef++;
        }

        *buffer_f00++ = prod >> 16;
        buffer_280++;
        *ptr_E1B = *buffer_280++;
    }

    *ptr_E1B = buffer_f00;

    buffer_280 = 0x280;
    buffer_f00 = 0xf00;
    u16* buffer_140 = 0x140;
```

```

u16* buffer_000 = 0x000;

i32 value;
for (int i = 0; i < 0xa0; i++) {
    value = EXTS16(*buffer_f00++);
    *(u32*)buffer_280++ = 0;      // increment by 2 as well
    *(u32*)buffer_000++ = value;  // increment by 2 as well
    *(u32*)buffer_140++ = -value; // increment by 2 as well
}

dma_dmem_to_mmem(mmaddr, 0xe80, 0x100);
wait_for_dma_finish();
}

```

2.2.9 Command 0xa - 0xc

These commands immediately return on call.

2.2.10 Command 0xd

This command loads a new command stream to DMEM and resets the `command_stream` pointer. Assembly is at ??.

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;
}

```

2.2.11 Command 0xe

This command DMA's the buffer section at 0x280 to main mem, and then procedurally combines the data from the buffer section at 0x0 and 0x140 and sends that to another main memory address. Assembly is at ??.

Pseudocode for this could be

```

void command_e(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;

    dma_dmem_to_mmem(mmaddr, 0x280, 0x280);
    wait_for_dma_finish();

    mmaddr = ((*command_stream++) << 16) | *command_stream++;

    u16* buffer_400 = 0x400;
    u16* buffer_0 = 0x0;
    u16* buffer_140 = 0x140;

    for (int i = 0; i < 5; i++) {
        u16* buffer_400_loop_start = buffer_400;

```

```

    for (int j = 0; j < 0x20; j++) {
        u32 data_140 = ((*buffer_140++) << 16) | *buffer_140++;
        u32 data_0 = ((*buffer_0++) << 16) | *buffer_0++;
        *buffer_400++ = (u16)data_140; // only bottom bits
        *buffer_400++ = (u16)data_0; // only bottom bits
    }
    dma_dmem_to_mmem(mmaddr, buffer_400_loop_start, 0x80);
    mmaddr += 0x80;
}
wait_for_dma_finish();
}

```

2.2.12 Command 0xf

Resets the DSP. Can be done in different ways, selected by mail that is expected. These ways are:

- Sending 0xdcd10001 and an IRQ to the CPU and going back into the main loop (waiting for 0xbabe mail...). This is a soft reset.
- Some sort of debug reset that allows the CPU to send data back into main memory (potentially to view what went wrong). Then a similar setup is ran as in the ROM. This is a hard reset.
- Jumping to the ROM start. This is a hard reset.
- Jumping to the 0xbabe mail wait loop in the RAM setup. This is a soft reset.

The assembly for this command is in ???. Pseudocode for this command is

```

void (*cmd_f_table[4])() = {
    j_send_dcd10001_irq, debug_reset, j_rom_start, j_wait_for_babe
}

void command_f(u16* &command_stream) {
    send_mail(0xdcd1, 0x0002);
    send_irq();
    u32 mail = wait_for_mail_recv();

    // jump to callback from lower mail
    // the j_* callbacks jump to other parts in the program
    cmd_f_table[mail & 0xffff]();
}

void j_send_dcd10001_irq() {
    // in main_entry (soft reset)
    goto send_dcd10001_irq;
}

// used in ROM setup (calling transfer_ucode)
extern struct setup_data {
    u32 dma_mm_addr; // IX0/IX1
    u16 dma_dsp_addr; // IX2
    u16 dma_length; // IX3
}

```

```

    u16 dma_control;    // ACO.M
    u16 entry_point;    // ARO
}

void debug_reset() {
    u32 mmaddr = wait_for_mail_recv();    // ACO
    u16 length = (u16)wait_for_mail_recv(); // AC1.L
    u16 dspaddr = (u16)wait_for_mail_recv(); // AC1.M
    dma_dmem_to_mmem(mmaddr, dspaddr, length);

    setup_data.dma_mm_addr = wait_for_mail_recv();    // ACO / IX0/1
    setup_data.dma_dsp_addr = (u16)wait_for_mail_recv(); // AC1.L / IX3
    setup_data.dma_length = (u16)wait_for_mail_recv(); // AC1.M / IX2
    setup_data.dma_control = 0;    // ACO.M
    setup_data.entry_point = (u16)wait_for_mail_recv(); // ACO.M / ARO

    mmaddr = wait_for_mail_recv();    // AX0
    length = (u16)wait_for_mail_recv(); // AX1.L
    dspaddr = (u16)wait_for_mail_recv(); // AX1.H

    wait_for_dma_finish();

    // in ROM: (80b5)
    if (length) {
        dma_to_dmem(mmaddr, dspaddr, length);
        wait_for_dma_finish();
    }
    transfer_ucose(); // also calls entry point
}

void j_rom_start() {
    // in ROM 0x8000 (hard reset)
    goto rom_start;
}

void j_wait_for_babe() {
    // in main_entry (softer reset)
    goto wait_for_babe;
}

```

2.2.13 Command 0x10

DMAs the buffer at 0x07c0 to main memory, loads new data into 0x07c0 and mixes it into 0x0000. Assembly is at ??.

Pseudocode for this could be

```

void command_10(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;

    dma_dmem_to_mmem(mmaddr, 0x7c0, 0x500);
    wait_for_dma_finish();
}

```



```

mmaddr = ((*command_stream++) << 16) | *command_stream++;
// DMA first part of buffer
dma_to_dmem(0x7c0, mmaddr, 0x20);
mmaddr += 0x20;
wait_for_dma_finish();

u16* buffer_7c0 = 0x7c0; // AR0
u16* buffer_0 = 0; // AR2/AR3

// DMA the rest while processing
dma_to_dmem(0x7d0, mmaddr, 0x4e0);

// in assembly: 0x4f iterations, 2 dword loads/stores per iteration
// last iteration special case
for (int i = 0; i < 0xa0; i++) {
    u32 data_7c0 = *(u32*)buffer_7c0;
    buffer_7c0 += 2;
    *(u32*)buffer_0 += data_7c0;
    buffer_0 += 2;
}

// in assembly: 0x4f iterations, 2 dword loads/stores per iteration
// last iteration special case
for (int i = 0; i < 0xa0; i++) {
    u32 data_7c0 = *(u32*)buffer_7c0;
    buffer_7c0 += 2;
    *(u32*)buffer_0 = -(*(u32*)buffer_0) + data_7c0;
    buffer_0 += 2;
}
// we have now processed 2 * 0xa0 dwords = 0x280 words of data (entire buffer)
}

```

2.2.14 Command 0x11

The same as command 0x7, except now 0x0 receives the negative 32-bit values from the buffer that is transferred, whereas 0x140 still gets the positive values. Assembly is at ??.

Pseudocode for this could be

```

void command_11(u16* &command_stream) {
    u32 mmaddr = ((*command_stream++) << 16) | *command_stream++;

    dma_to_dmem(0xe44, mmaddr, 0x20);
    mmaddr += 0x20;
    wait_for_dma_finish();

    u16* buffer_e44 = 0xe44;
    u16* buffer_280 = 0x280;
    u16* buffer_0 = 0x0;
    u16* buffer_140 = 0x140;

    // DMA rest of the data to process in parallel

```

```
dma_to_dmem(0xe54, mmaddr, 0x60);

for (int i = 0; i < 0xa0; i++) {
    u32 data_e44 = *(u32*)buffer_e44;
    buffer_e44 += 2;
    *(u32*)buffer_140 = data_e44;
    buffer_140 += 2;
    *(u32*)buffer_0 = -data_e44;
    buffer_0 += 2;
}
// does not wait for DMA to finish (seems like a bad idea)
}
```

Appendix A

Assembly

A.1 Command 0x0

```

command_0:
IRAM:0082          CLR          $ACCO
IRAM:0083 ; load next two words from stream into ac0 and ac1
IRAM:0083          CLR'L        $ACC1 : $ACO.M, @ $ARO
IRAM:0084          SET16'L      $AC1.M : @ $ARO
IRAM:0085 ; store DMA address
IRAM:0085          SRS          DMAMADDRH, $ACO.M
IRAM:0086          SRS          DMAMADDRL, $AC1.M
IRAM:0087 ; DSPADDR = 0xe44
IRAM:0087          LRI          $ACO.M, 0xE44
IRAM:0089          SRS          DMADSPADDR, $ACO.M
IRAM:008A ; DMAControl = 0
IRAM:008A ; to DSP DMEM
IRAM:008A          LRIS        $ACO.M, 0
IRAM:008B          SRS          DMAControl, $ACO.M
IRAM:008C ; length = 0x40 8bit bytes
IRAM:008C          LRI          $ACO.M, 0x40
IRAM:008E          SRS          DMALength, $ACO.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          $ACCO
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        $ACO.M, @ $AR1
IRAM:009D          LRRI        $ACO.L, @ $AR1
IRAM:009E          TST          $ACCO
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, $ACO.M
IRAM:00A4          JMP          cmd0_dmem_140_words_filled
IRAM:00A6 ; -----

```

```

IRAM:00A6
IRAM:00A6 cmd0_BASE_not_0:
IRAM:00A6          ASR16          $ACC1      ; AC1.M ASR16 -> AC1.L
IRAM:00A7 ; BASE to buffer at 0x0000
IRAM:00A7          SRRI          @ $AR2, $ACO.M
IRAM:00A8          SRRI          @ $AR2, $ACO.L
IRAM:00A9 ; loop 0x9f times
IRAM:00A9          BLOOP         $AX1.H, loc_AD
IRAM:00AB ; BASE += INCR
IRAM:00AB
IRAM:00AB          ADD           $ACCO, $ACC1
IRAM:00AC ; store BASE (with INCR added every loop)
IRAM:00AC ; 32 bit value
IRAM:00AC          SRRI          @ $AR2, $ACO.M
IRAM:00AD
IRAM:00AD loc_AD:
IRAM:00AD          SRRI          @ $AR2, $ACO.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:
IRAM:00AE          LRRI          $ACO.M, @ $AR1
IRAM:00AF          LRRI          $ACO.L, @ $AR1
IRAM:00B0          TST           $ACCO
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, $ACO.M
IRAM:00B6          JMP           cmd0_another_140_words_filled
IRAM:00B8 ; -----
IRAM:00B8
IRAM:00B8 loc_B8:
IRAM:00B8          ASR16          $ACC1      ; INCR ac1.m asr16 -> ac2.l
IRAM:00B9 ; store BASE to dest
IRAM:00B9          SRRI          @ $AR2, $ACO.M
IRAM:00BA          SRRI          @ $AR2, $ACO.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           $ACCO, $ACC1
IRAM:00BE          SRRI          @ $AR2, $ACO.M
IRAM:00BF
IRAM:00BF loc_BF:
IRAM:00BF          SRRI          @ $AR2, $ACO.L
IRAM:00C0 ; dest is now 0x280
IRAM:00C0 ; same thing again
IRAM:00C0
IRAM:00C0 cmd0_another_140_words_filled:
IRAM:00C0          LRRI          $ACO.M, @ $AR1
IRAM:00C1          LRRI          $ACO.L, @ $AR1
IRAM:00C2          TST           $ACCO
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 loc_CA:
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 loc_D1:
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:
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 loc_DD:
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:
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:
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 loc_EE:
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                SRR1                @R2, $A0.M
IRAM:00F5
IRAM:00F5 loc_F5:
IRAM:00F5                SRR1                @R2, $A0.L
IRAM:00F6 ; same thing again
IRAM:00F6
IRAM:00F6 cmd0_140_filled_at_540:
IRAM:00F6                LRR1                $A0.M, @R1
IRAM:00F7                LRR1                $A0.L, @R1
IRAM:00F8                TST'L              $ACC0 : $A1.M, @R1
IRAM:00F9                JNZ                loc_FF
IRAM:00FB                LOOP               $AX0.H
IRAM:00FC                SRR1                @R2, $A0.M
IRAM:00FD                JMP                cmd0_140_filled_at_680
IRAM:00FF ; -----
IRAM:00FF
IRAM:00FF loc_FF:
IRAM:00FF                ASR16              $ACC1
IRAM:0100                SRR1                @R2, $A0.M
IRAM:0101                SRR1                @R2, $A0.L
IRAM:0102                BLOOP              $AX1.H, loc_106
IRAM:0104                ADD                $ACC0, $ACC1
IRAM:0105                SRR1                @R2, $A0.M
IRAM:0106
IRAM:0106 loc_106:
IRAM:0106                SRR1                @R2, $A0.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:
IRAM:0107                LRI                $R2, 0x7C0
IRAM:0109                LRR1                $A0.M, @R1
IRAM:010A                LRR1                $A0.L, @R1
IRAM:010B                TST'L              $ACC0 : $A1.M, @R1
IRAM:010C                JNZ                loc_112
IRAM:010E                LOOP               $AX0.H
IRAM:010F                SRR1                @R2, $A0.M
IRAM:0110                JMP                cmd0_140_filled_at_7c0
IRAM:0112 ; -----
IRAM:0112
IRAM:0112 loc_112:
IRAM:0112                ASR16              $ACC1
IRAM:0113                SRR1                @R2, $A0.M
IRAM:0114                SRR1                @R2, $A0.L
IRAM:0115                BLOOP              $AX1.H, loc_119
IRAM:0117                ADD                $ACC0, $ACC1
IRAM:0118                SRR1                @R2, $A0.M
IRAM:0119
IRAM:0119 loc_119:
IRAM:0119                SRR1                @R2, $A0.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:
IRAM:011A                LRR1                $A0.M, @R1
IRAM:011B                LRR1                $A0.L, @R1
IRAM:011C                TST'L              $ACC0 : $A1.M, @R1
IRAM:011D                JNZ                loc_123

```

```

IRAM:011F      LOOP      $AX0.H
IRAM:0120      SRR I     @ $AR2, $AC0.M
IRAM:0121      JMP      cmd0_140_filled_at_900
IRAM:0123      ; -----
IRAM:0123      loc_123:
IRAM:0123      ASR16     $ACC1
IRAM:0124      SRR I     @ $AR2, $AC0.M
IRAM:0125      SRR I     @ $AR2, $AC0.L
IRAM:0126      BLOOP     $AX1.H, loc_12A
IRAM:0128      ADD      $ACC0, $ACC1
IRAM:0129      SRR I     @ $AR2, $AC0.M
IRAM:012A
IRAM:012A      loc_12A:
IRAM:012A      SRR I     @ $AR2, $AC0.L
IRAM:012B      ; dest is now 0xa40
IRAM:012B      ; same thing again
IRAM:012B
IRAM:012B      cmd0_140_filled_at_900:
IRAM:012B      LRR I     $AC0.M, @ $AR1
IRAM:012C      LRR I     $AC0.L, @ $AR1
IRAM:012D      TST'L     $ACC0 : $AC1.M, @ $AR1
IRAM:012E      JNZ      loc_134
IRAM:0130      LOOP     $AX0.H
IRAM:0131      SRR I     @ $AR2, $AC0.M
IRAM:0132      JMP      cmd0_done
IRAM:0134      ; -----
IRAM:0134
IRAM:0134      loc_134:
IRAM:0134      ASR16     $ACC1
IRAM:0135      SRR I     @ $AR2, $AC0.M
IRAM:0136      SRR I     @ $AR2, $AC0.L
IRAM:0137      BLOOP     $AX1.H, loc_13B
IRAM:0139      ADD      $ACC0, $ACC1
IRAM:013A      SRR I     @ $AR2, $AC0.M
IRAM:013B
IRAM:013B      loc_13B:
IRAM:013B      SRR I     @ $AR2, $AC0.L
IRAM:013C      ; dest should end up at 0xb80
IRAM:013C
IRAM:013C      cmd0_done:
IRAM:013C      JMP      receive_command
IRAM:013C      ; End of function command_0

```

A.2 Command 0x1

```

command_1:
IRAM:013E      LRI      $IX1, 0xFFFF
IRAM:0140      ; read main memory address from command stream into AX0 (hi then lo)
IRAM:0140      CLR'L     $ACC0 : $AX0.H, @ $AR0
IRAM:0141      CLR'L     $ACC1 : $AX0.L, @ $AR0
IRAM:0142      ; load scale into AX1.L
IRAM:0142      SET16'L   $AX1.L : @ $AR0
IRAM:0143      ; save main memory address
IRAM:0143      SR       cmd1_mmaddrh_temp_E17, $AX0.H
IRAM:0145      SR       cmd1_mmaddrl_temp_E18, $AX0.L
IRAM:0147      ; this is going to process data in the buffers setup by command 0

```

```

IRAM:0147          LRI          $AR1, 0
IRAM:0149          CALL         transform_buffer
IRAM:014B ; restore mmaddr
IRAM:014B          LR          $AX0.H, cmd1_mmaddrh_temp_E17
IRAM:014D          LR          $AX0.L, cmd1_mmaddrl_temp_E18
IRAM:014F          CLR1L      $ACC1 : $AX1.L, @ $AR0
IRAM:0150          LRI          $AR1, 0x400
IRAM:0152          CALL         transform_buffer
IRAM:0154 ; restore mmaddr
IRAM:0154          LR          $AX0.H, cmd1_mmaddrh_temp_E17
IRAM:0156          LR          $AX0.L, cmd1_mmaddrl_temp_E18
IRAM:0158          CLR1L      $ACC1 : $AX1.L, @ $AR0
IRAM:0159          LRI          $AR1, 0x7C0
IRAM:015B          CALL         transform_buffer
IRAM:015D          JMP          receive_command
IRAM:015D ; End of function command_1

...

transform_buffer:
IRAM:04F1          ; command_1+14tp ...
IRAM:04F1          SET16
IRAM:04F2 ; input ar1: pointer to data transferred by command_0
IRAM:04F2
IRAM:04F2 ; DMA 0xc0 bytes from input mmaddr to E44
IRAM:04F2          LRI          $AX1.H, 0xE44
IRAM:04F4          LRI          $AC1.L, 0xC0
IRAM:04F6          CALL         start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L
IRAM:04F8 ; ac1: mmaddr + 0xc0
IRAM:04F8          ADDAX        $ACC1, $AX0
IRAM:04F9 ; save (new) source address
IRAM:04F9          SR          tf_buffer_mmaddr_temph_E1D, $AC1.M
IRAM:04FB          SR          tf_buffer_mmaddr_templ_E1E, $AC1.L
IRAM:04FD          CLR          $ACC1
IRAM:04FE          CALL         wait_for_dma_finish_0
IRAM:0500 ; REPEAT 4 TIMES
IRAM:0500          BLOOPI      4, loc_52C
IRAM:0502 ; restore mmaddr
IRAM:0502          LR          $AX0.H, tf_buffer_mmaddr_temph_E1D
IRAM:0504          LR          $AX0.L, tf_buffer_mmaddr_templ_E1E
IRAM:0506 ; DMA 0xc0 more bytes
IRAM:0506          LRI          $AX1.H, 0xEA4
IRAM:0508          LRI          $AC1.L, 0xC0
IRAM:050A          CALL         start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L
IRAM:050C ; mmaddr += 0xc0
IRAM:050C          ADDAX        $ACC1, $AX0
IRAM:050D ; save mmaddr
IRAM:050D          SR          tf_buffer_mmaddr_temph_E1D, $AC1.M
IRAM:050F          SR          tf_buffer_mmaddr_templ_E1E, $AC1.L
IRAM:0511          LRI          $AR3, 0xE44
IRAM:0513          CALL         transform_buffer_section
IRAM:0515          CLR          $ACC1
IRAM:0516 ; restore mmaddr
IRAM:0516          LR          $AX0.H, tf_buffer_mmaddr_temph_E1D
IRAM:0518          LR          $AX0.L, tf_buffer_mmaddr_templ_E1E
IRAM:051A ; dma another 0xc0 bytes
IRAM:051A          LRI          $AX1.H, 0xE44
IRAM:051C          LRI          $AC1.L, 0xC0

```



```

IRAM:051E          CALL          start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L
IRAM:0520 ; mmaddr += 0xc0
IRAM:0520          ADDAX         $ACC1, $AX0
IRAM:0521 ; save mmaddr
IRAM:0521          SR           tf_buffer_mmaddr_temph_E1D, $AC1.M
IRAM:0523          SR           tf_buffer_mmaddr_templ_E1E, $AC1.L
IRAM:0525          LRI          $AR3, 0xEA4
IRAM:0527          CALL         transform_buffer_section
IRAM:0529          NOP
IRAM:052A          NOP
IRAM:052B          SET16
IRAM:052C
IRAM:052C loc_52C:
IRAM:052C          CLR          $ACC1
IRAM:052D ; BLOOPI_END
IRAM:052D
IRAM:052D ; restore mmaddr
IRAM:052D          LR          $AX0.H, tf_buffer_mmaddr_temph_E1D
IRAM:052F          LR          $AX0.L, tf_buffer_mmaddr_templ_E1E
IRAM:0531 ; DMA another 0xc0 words
IRAM:0531          LRI          $AX1.H, 0xEA4
IRAM:0533          LRI          $AC1.L, 0xC0
IRAM:0535          CALL         start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L
IRAM:0537 ; mmaddr += 0xc0
IRAM:0537          ADDAX         $ACC1, $AX0
IRAM:0538          LRI          $AR3, 0xEA4
IRAM:053A          CALL         transform_buffer_section
IRAM:053C          LRI          $AR3, 0xEA4
IRAM:053E          CALL         transform_buffer_section
IRAM:0540          RET
IRAM:0540 ; End of function transform_buffer
IRAM:0540
IRAM:0541
IRAM:0541 ; ===== S U B R O U T I N E =====
IRAM:0541
IRAM:0541 start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L:
IRAM:0541 ; CODE XREF: transform_buffer+5tp
IRAM:0541 ; transform_buffer+19tp ...
IRAM:0541          SET16
IRAM:0542          SR          DMAMADDRH, $AX0.H
IRAM:0544          SR          DMAMADDRL, $AX0.L
IRAM:0546          SR          DMADSPADDR, $AX1.H
IRAM:0548          SI          DMAControl, 0
IRAM:054A          SRS          DMALength, $AC1.L
IRAM:054B          RET
IRAM:054B ; End of function start_DMA_to_DSP_mmaddr_AX0_dspaddr_AX1H_len_AC1L
IRAM:054B
IRAM:054C
IRAM:054C ; ===== S U B R O U T I N E =====
IRAM:054C
IRAM:054C transform_buffer_section:
IRAM:054C ; transform_buffer+36tp ...
IRAM:054C          SET40
IRAM:054D          SET15
IRAM:054E          M2
IRAM:054F ; input AR3 is pointer to start of DMA'ed data in command 1

```

```

IRAM:054F ; input AR1 is pointer to start of DMA'ed data in command 0
IRAM:054F ; load 2 words (base)
IRAM:054F ; AX1.L = scale (from cmd1)
IRAM:054F ; IX1 = 0xffff (-1)
IRAM:054F LRR1 $AX0.H, @ $AR3
IRAM:0550 LRR1 $AX0.L, @ $AR3
IRAM:0551 ; ac0 = (i16(base)) * scale;
IRAM:0551 ; prod = (i16(base >> 16)) * scale;
IRAM:0551 MULX $AX0.L, $AX1.L
IRAM:0552 MULX MV $AX0.H, $AX1.L, $ACCO
IRAM:0553 ; REPEAT 0x30 = 48 times
IRAM:0553 BLOOPI 0x30, loc_55A
IRAM:0555 ; load word from AR1 stream to AC1.ml, don't change AR1
IRAM:0555 ; ac0 = (ac0 >> 16) + prod
IRAM:0555 ; fixed point?
IRAM:0555 ASR16 L $ACCO : $AC1.M, @ $AR1
IRAM:0556 ADDP L $ACCO : $AC1.L, @ $AR1
IRAM:0557 ; load new word from AR3 data stream
IRAM:0557 LRR1 $AX0.H, @ $AR3
IRAM:0558 ; ac1 += ac0
IRAM:0558 ; load new AX0.L from AR3 stream
IRAM:0558 ADD L $ACC1, $ACCO : $AX0.L, @ $AR3
IRAM:0559 ; same product as above the loop
IRAM:0559 ; *(u32*)ar1++ = ac1.ml
IRAM:0559 ; this overwrites the previous value
IRAM:0559 MULX S $AX0.L, $AX1.L : @ $AR1, $AC1.M
IRAM:055A
IRAM:055A loc_55A:
IRAM:055A MULX MV S $AX0.H, $AX1.L, $ACCO : @ $AR1, $AC1.L
IRAM:055B ; BLOOPI_END
IRAM:055B RET
IRAM:055B ; End of function transform_buffer_section

```

A.3 Command 0x2

```

; ===== S U B R O U T I N E =====
IRAM:01BC
IRAM:01BC
IRAM:01BC command_2:
IRAM:01BC CLR $ACCO
IRAM:01BD ; read mmaddr from command stream
IRAM:01BD CLR L $ACC1 : $ACO.M, @ $ARO
IRAM:01BE SET16 L $AC1.M : @ $ARO
IRAM:01BF ; start DMA to DSP DMEM 0xb80 of length 0xc0
IRAM:01BF ; this probably holds some settings or a struct
IRAM:01BF SRS DMAMADDRH, $ACO.M
IRAM:01C0 SRS DMAMADDRL, $AC1.M
IRAM:01C1 SI DMADSPADDR, 0xb80
IRAM:01C3 SI DMAControl, 0
IRAM:01C5 SI DMALength, 0xc0
IRAM:01C7 LRI $AR2, buffer_sections_E08
IRAM:01C9 ; store addresses of buffer sections to DMEM 0xe08
IRAM:01C9 LRI $AC1.M, 0
IRAM:01CB SRRI @ $AR2, $AC1.M
IRAM:01CC LRI $AC1.M, 0x140
IRAM:01CE SRRI @ $AR2, $AC1.M
IRAM:01CF LRI $AC1.M, 0x280

```

```

IRAM:01D1          SRRI          @R2, $A1.M
IRAM:01D2          LRI           $A1.M, 0x400
IRAM:01D4          SRRI          @R2, $A1.M
IRAM:01D5          LRI           $A1.M, 0x540
IRAM:01D7          SRRI          @R2, $A1.M
IRAM:01D8          LRI           $A1.M, 0x680
IRAM:01DA          SRRI          @R2, $A1.M
IRAM:01DB          LRI           $A1.M, 0x7C0
IRAM:01DD          SRRI          @R2, $A1.M
IRAM:01DE          LRI           $A1.M, 0x900
IRAM:01E0          SRRI          @R2, $A1.M
IRAM:01E1          LRI           $A1.M, 0xA40
IRAM:01E3          SRRI          @R2, $A1.M
IRAM:01E4          CALL          wait_for_dma_finish_0
IRAM:01E6 ; load address from DMA'ed settings and start DMA to DSP 0x3c0
IRAM:01E6 of length 0x80
IRAM:01E6          LR           $A0.M, loc_BA6+1
IRAM:01E8          LR           $A1.M, DMEM_BA8
IRAM:01EA          SRS          DMAMADDRH, $A0.M
IRAM:01EB          SRS          DMAMADDRL, $A1.M
IRAM:01EC          SI           DMADSPADDR, 0x3C0
IRAM:01EE          SI           DMAControl, 0
IRAM:01F0          SI           DMALength, 0x80
IRAM:01F2          CLR          $ACC0
IRAM:01F3          CLR          $ACC1
IRAM:01F4 ; load offset from DMA'ed data and copy value from 0xb31 + offset to E15
IRAM:01F4          LR           $A0.M, DMEM_B84
IRAM:01F6          LRI          $A1.M, 0xB31
IRAM:01F8          ADD          $ACC0, $ACC1
IRAM:01F9          MRR          $R3, $A0.M
IRAM:01FA          ILRR         $A0.M, @R3
IRAM:01FB          SR           DMEM_E15, $A0.M
IRAM:01FD ; load offset from DMA'ed data and copy value from 0xb34 + offset to E16
IRAM:01FD          LR           $A0.M, DMEM_B85
IRAM:01FF          LRI          $A1.M, 0xB34
IRAM:0201          ADD          $ACC0, $ACC1
IRAM:0202          MRR          $R3, $A0.M
IRAM:0203          ILRR         $A0.M, @R3
IRAM:0204 ; load offset from DMA'ed data and copy value from 0xb11 + offset to E14
IRAM:0204          SR           DMEM_E16, $A0.M
IRAM:0206          LR           $A0.M, DMEM_B86
IRAM:0208          LRI          $A1.M, 0xB11
IRAM:020A          ADD          $ACC0, $ACC1
IRAM:020B          MRR          $R3, $A0.M
IRAM:020C          ILRR         $A0.M, @R3
IRAM:020D          SR           DMEM_E14, $A0.M
IRAM:020F ; if [B9B] == 0: jump
IRAM:020F          CLR          $ACC0
IRAM:0210          LR           $A0.M, DMEM_B9B
IRAM:0212          TST          $ACC0
IRAM:0213          JZ           b9b_zero
IRAM:0215 ; else
IRAM:0215          CLR          $ACC1
IRAM:0216 ; store offsets relative to cc0 (command stream start) to E40/41/42/43
IRAM:0216          LR           $A1.M, loc_B9E
IRAM:0218          ADDI         $A1.M, 0xCC0
IRAM:021A          SR           cmd2_DMEM_E40_start, $A1.M
IRAM:021C          LR           $A1.M, loc_B9F

```

```

IRAM:021E      ADDI      $AC1.M, 0xCC0
IRAM:0220      SR        cmd2_DMEM_E41_end, $AC1.M
IRAM:0222      LRI       $AC1.M, 0xCE0
IRAM:0224      SR        cmd2_DMEM_E42, $AC1.M
IRAM:0226      SR        cmd2_DMEM_E43, $AC1.M
IRAM:0228      CALL      wait_for_dma_finish_0
IRAM:022A      ; load DMA address from transferred data and start DMA to DSP DMEM CCO of length 0x40
IRAM:022A      LR        $AC0.M, DMEM_B9C
IRAM:022C      SRS       DMAMADDRH, $AC0.M
IRAM:022D      LR        $AC0.M, DMEM_B9D
IRAM:022F      SRS       DMAMADDRH, $AC0.M
IRAM:0230      SI        DMADSPADDR, 0xCC0
IRAM:0232      SI        DMAControl, 0
IRAM:0234      SI        DMALength, 0x40
IRAM:0236      CALL      wait_for_dma_finish_0
IRAM:0238      JMP       receive_command
IRAM:023A      ; -----
IRAM:023A      ; store end of command stream (?) to E40/41/42/43
IRAM:023A      b9b_zero:
IRAM:023A      LRI       $AC1.M, 0xCE0
IRAM:023C      SR        cmd2_DMEM_E42, $AC1.M
IRAM:023E      SR        cmd2_DMEM_E40_start, $AC1.M
IRAM:0240      SR        cmd2_DMEM_E41_end, $AC1.M
IRAM:0242      SR        cmd2_DMEM_E43, $AC1.M
IRAM:0244      CALL      wait_for_dma_finish_0
IRAM:0246      JMP       receive_command
IRAM:0246      ; End of function command_2

```

A.4 Command 0x3

```

; ===== S U B R O U T I N E =====
IRAM:0248
IRAM:0248
IRAM:0248      command_3:
IRAM:0248      ; command_3+1C94j
IRAM:0248      ; DATA XREF: ...
IRAM:0248      SET16
IRAM:0249      ; save command_stream pointer
IRAM:0249      SR        cmd3_temp_command_stream, $ARO
IRAM:024B      ; ARO holds pointer to address in region where cmd2 DMAs to
IRAM:024B      ; AR1 holds pointer to start of region cmd2 DMAs to (second DMA)
IRAM:024B
IRAM:024B      ; ar0: buffer_ba2
IRAM:024B      ; ar1: buffer_3c0
IRAM:024B      LRI       $AR0, 0xBA2
IRAM:024D      LRI       $AR1, 0x3C0
IRAM:024F      LRIS      $AC0.M, 5
IRAM:0250      SR        cmd3_loop_counter, $AC0.M
IRAM:0252      CLR       $ACC1
IRAM:0253      ; load loop length from buffer_ba2
IRAM:0253
IRAM:0253      cmd3_loop_5_start:
IRAM:0253      CLR1L      $ACC0 : $AX0.H, @ $ARO
IRAM:0254      LRI       $AC1.M, 0xB80
IRAM:0256      BLOOP     $AX0.H, loc_25B
IRAM:0258      ; dest = *(buffer_3c0++) + 0xb80

```

```

IRAM:0258                LRR1                $ACO.M, @ $AR1
IRAM:0259                ADDL                $ACCO, $ACC1 : $AX1.L, @ $AR1
IRAM:025A                MRR                $AR2, $ACO.M
IRAM:025B ; *dest = *(buffer_3c0++)
IRAM:025B
IRAM:025B loc_25B:
IRAM:025B                SRR                @ $AR2, $AX1.L
IRAM:025C ; BLOOP END
IRAM:025C
IRAM:025C ; save buffer_3c0 end pointer to E05
IRAM:025C ; save buffer_ba2 end pointer to E06 (should just be ba3)
IRAM:025C                LRI                $AR3, cmd3_temp_AR1
IRAM:025E                SRR1               @ $AR3, $AR1
IRAM:025F                SRR1               @ $AR3, $AR0
IRAM:0260 ; check flag in struct from command 2
IRAM:0260                LR                $ACO.M, cmd3_flag_B87
IRAM:0262                CMPIS             $ACO.M, 1
IRAM:0263                JZ                cmd3_struct_flag_1
IRAM:0265                JMP                cmd3_struct_flag_0
IRAM:0267 ; -----
IRAM:0267 if [b87] == 1
IRAM:0267
IRAM:0267 cmd3_struct_flag_1:
IRAM:0267                LR                $ACO.M, cmd2_DMEM_E42
IRAM:0269 ; load pointer setup by command 2
IRAM:0269 ; load value from E15 (from struct, setup by command 2)
IRAM:0269                SR                byte_E1C, $ACO.M
IRAM:026B ; call pointer
IRAM:026B                LR                $AR3, DMEM_E15
IRAM:026D                CALLR             $AR3
IRAM:026E ; reset state
IRAM:026E                SET16
IRAM:026F                M2
IRAM:0270                CLR                $ACCO
IRAM:0271                CLR                $ACC1
IRAM:0272 ; load data from struct
IRAM:0272                LR                $ACO.M, loc_BB3
IRAM:0274                LR                $AC1.M, loc_BB2
IRAM:0276 ; ac1.m = [bb3] + [bb2]
IRAM:0276 ; ax0.l = [bb2]
IRAM:0276 ; ax1.h = [bb3] << 1
IRAM:0276 ; ac0.m = [bb2]
IRAM:0276 ; ax0.l = 0x8000
IRAM:0276                MRR                $AX0.L, $AC1.M
IRAM:0277                ADD                $ACC1, $ACCO
IRAM:0278                ASL                $ACCO, 1
IRAM:0279                SET15MV           $AX1.H : $ACO.M
IRAM:027A                MRR                $ACO.M, $AX0.L
IRAM:027B                LRI                $AX0.L, 0x8000
IRAM:027D ; load pointer to e44
IRAM:027D                LRI                $AR0, byte_E44
IRAM:027F ; prod = ax0.l * ax1.l
IRAM:027F ; *buffer_e44++ = ac0.m
IRAM:027F ; repeatedly:
IRAM:027F ;     ac0 += prod
IRAM:027F ;     prod = ax0.l * ax1.l
IRAM:027F ;     *buffer_e44++ = ac1.m
IRAM:027F ;     ac1 += prod

```

```

IRAM:027F ;      prod = ax0.l * ax1.l
IRAM:027F ;      *buffer_e44++ = ac0.m
IRAM:027F      MULX'S      $AX0.L, $AX1.H : @ $AR0, $ACO.M
IRAM:0280      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0281      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0282      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0283      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0284      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0285      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0286      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0287      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0288      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0289      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:028A      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:028B      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:028C      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:028D      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:028E      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:028F      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0290      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0291      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0292      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0293      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0294      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0295      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0296      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0297      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:0298      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:0299      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:029A      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:029B      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:029C      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:029D      MULXAC'S     $AX0.L, $AX1.H, $ACC1 : @ $AR0, $ACO.M
IRAM:029E      MULXAC'S     $AX0.L, $AX1.H, $ACCO : @ $AR0, $AC1.M
IRAM:029F ; store final resulting ac0.m in struct
IRAM:029F      SR          loc_BB2, $ACO.M
IRAM:02A1      SET40
IRAM:02A2 ; pointer to buffer at 0xe44 again
IRAM:02A2 ; load second word stored by command 2
IRAM:02A2      LRI          $AR0, byte_E44
IRAM:02A4      LR           $AR1, cmd2_DMEN_E43
IRAM:02A6      MRR          $AR3, $AR1
IRAM:02A7      LRRI         $AX0.H, @ $AR1
IRAM:02A8      LRRI         $AX0.L, @ $AR0
IRAM:02A9 ; AC[1 - d] = prod
IRAM:02A9 ; prod = AXd.l * AXd.h
IRAM:02A9 ; AX[1 - d].h = *buffer_pointed_by_e43
IRAM:02A9 ; AX[1 - d].l = *buffer_e44++
IRAM:02A9 ; *buffer_pointed_by_e43++ = AC[1 - d].m // buffer_pointed_by_e43 in both ar1 and ar3
IRAM:02A9      MUL'L        $AX0.L, $AX0.H : $AX1.H, @ $AR1
IRAM:02AA      LRRI         $AX1.L, @ $AR0
IRAM:02AB      MULMV'L      $AX1.L, $AX1.H, $ACCO : $AX0.H, @ $AR1
IRAM:02AC      NX'LS        $AX0.L : $ACO.M
IRAM:02AD      MULMV'L      $AX0.L, $AX0.H, $ACC1 : $AX1.H, @ $AR1
IRAM:02AE      NX'LS        $AX1.L : $AC1.M
IRAM:02AF      MULMV'L      $AX1.L, $AX1.H, $ACCO : $AX0.H, @ $AR1
IRAM:02B0      NX'LS        $AX0.L : $ACO.M
IRAM:02B1      MULMV'L      $AX0.L, $AX0.H, $ACC1 : $AX1.H, @ $AR1

```


IRAM: 02B2	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02B3	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02B4	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02B5	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02B6	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02B7	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02B8	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02B9	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02BA	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02BB	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02BC	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02BD	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02BE	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02BF	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02C0	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02C1	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02C2	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02C3	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02C4	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02C5	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02C6	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02C7	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02C8	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02C9	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02CA	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02CB	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02CC	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02CD	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02CE	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02CF	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02D0	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02D1	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02D2	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02D3	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02D4	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02D5	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02D6	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02D7	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02D8	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02D9	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02DA	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02DB	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02DC	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02DD	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02DE	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02DF	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02E0	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02E1	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02E2	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02E3	MULMV'L	\$AX1.L, \$AX1.H, \$ACCO : \$AX0.H, @ \$AR1
IRAM: 02E4	NX'LS	\$AX0.L : \$AC0.M
IRAM: 02E5	MULMV'L	\$AX0.L, \$AX0.H, \$ACC1 : \$AX1.H, @ \$AR1
IRAM: 02E6	NX'LS	\$AX1.L : \$AC1.M
IRAM: 02E7	MULMV	\$AX1.L, \$AX1.H, \$ACCO
IRAM: 02E8 ; last step is different		
IRAM: 02E8	MOVP'S	\$ACC1 : @ \$AR3, \$AC0.M
IRAM: 02E9	SRRI	@ \$AR3, \$AC1.M
IRAM: 02EA ; call data from command 2		

```

IRAM:02EA          LR          $AR3, DMEM_E14
IRAM:02EC ; reset state
IRAM:02EC          SET40
IRAM:02ED          SET15
IRAM:02EE          M2
IRAM:02EF          CALLR      $AR3
IRAM:02F0          CLR        $ACCO
IRAM:02F1 ; load data from struct
IRAM:02F1          LR          $AC0.M, DMEM_B9B
IRAM:02F3          TST        $ACCO
IRAM:02F4          JZ         cmd3_struct_data_0
IRAM:02F6 ; transfer data from command 2
IRAM:02F6          LR          $AC0.M, cmd2_DMEM_E42
IRAM:02F8          SR         cmd2_DMEM_E43, $AC0.M
IRAM:02FA          CLR        $ACCO
IRAM:02FB          CLR        $ACC1
IRAM:02FC          LR          $AC0.M, loc_B9E
IRAM:02FE          LR          $AC1.M, loc_BA0
IRAM:0300          CMP
IRAM:0301 ; if [b9e] <= [ba0]:
IRAM:0301 ;     [b9e]++
IRAM:0301 ; else:
IRAM:0301 ;     [b9e]--
IRAM:0301          JLE        loc_306
IRAM:0303          DECM       $AC0.M
IRAM:0304          JMP        loc_309
IRAM:0306 ; -----
IRAM:0306 loc_306:
IRAM:0306          JZ         loc_309
IRAM:0308          INCM       $AC0.M
IRAM:0309 loc_309:
IRAM:0309          ; command_3:loc_306†j
IRAM:0309          SR         loc_B9E : $AC0.M,
IRAM:030B ; [e40] = [e43] - 0x20 + [b9e] // the incr/decr [b9e]
IRAM:030B          LR          $AC1.M, cmd2_DMEM_E43
IRAM:030D          ADDIS      $AC1.M, 0xE0
IRAM:030E          ADD        $ACCO, $ACC1
IRAM:030F          SR         cmd2_DMEM_E40_start, $AC0.M
IRAM:0311          CLR        $ACCO
IRAM:0312          CLR        $ACC1
IRAM:0313 ; if [b9f] <= [ba1]:
IRAM:0313 ;     [b9f]++
IRAM:0313 ; else:
IRAM:0313 ;     [b9f]--
IRAM:0313          LR          $AC0.M, loc_B9F
IRAM:0315          LR          $AC1.M, loc_BA1
IRAM:0317          CMP
IRAM:0318          JLE        loc_31D
IRAM:031A          DECM       $AC0.M
IRAM:031B          JMP        loc_320
IRAM:031D ; -----
IRAM:031D loc_31D:
IRAM:031D          JZ         loc_320
IRAM:031F          INCM       $AC0.M
IRAM:0320

```



```

IRAM:0320 loc_320:
IRAM:0320 ; command_3:loc_31D1j
IRAM:0320 SR loc_B9F : $ACO.M,
IRAM:0322 ; [e41] = [e43] - 0x20 + [b9f]
IRAM:0322 LR $AC1.M, cmd2_DMEN_E43
IRAM:0324 ADDIS $AC1.M, 0xE0
IRAM:0325 ADD $ACC0, $ACC1
IRAM:0326 SR cmd2_DMEN_E41_end, $ACO.M
IRAM:0328 JMP cmd3_struct_flag_0
IRAM:032A ; -----
IRAM:032A cmd3_struct_data_0:
IRAM:032A LR $ACO.M, cmd2_DMEN_E42
IRAM:032C ; [e40] = [e41] = [e43] = [e42]
IRAM:032C SR cmd2_DMEN_E40_start, $ACO.M
IRAM:032E SR cmd2_DMEN_E41_end, $ACO.M
IRAM:0330 SR cmd2_DMEN_E43, $ACO.M
IRAM:0332 if [b87] != 1
IRAM:0332 cmd3_struct_flag_0:
IRAM:0332 ; command_3+E01j
IRAM:0332 CLR $ACC0
IRAM:0333 ; reset state
IRAM:0333 SET16
IRAM:0334 CLRP
IRAM:0335 CLR $ACC1
IRAM:0336 MRR $PROD.M2, $ACO.M
IRAM:0337 LRIS $ACO.M, 0x40
IRAM:0338 ; prod.m = 0x40
IRAM:0338 ; ac1.m = 0x40
IRAM:0338 ; ar0 = ar3 = 0xE08
IRAM:0338 MRR $PROD.M1, $ACO.M
IRAM:0339 LRI $AR3, buffer_sections_E08
IRAM:033B MRR $AR0, $AR3
IRAM:033C MRR $AC1.M, $PROD.M1
IRAM:033D ; ax0.h = *buffer_sections_e08++;
IRAM:033D ; first step is slightly different
IRAM:033D ; repeatedly:
IRAM:033D ; ac0.hm = prod.m (=0x40) + ax0.h
IRAM:033D ; ax1.h = *buffer_sections_e08;
IRAM:033D ; *buffer_sections_e08 = ac1.m;
IRAM:033D ; buffer_sections_e08++; // both ARO and AR3
IRAM:033D ; ac1.hm = prod.m (=0x40) + ax1.h
IRAM:033D ; ax0.h = *buffer_sections_e08;
IRAM:033D ; *buffer_sections_e08 = ac0.m;
IRAM:033D ; buffer_sections_e08++; // both ARO and AR3
IRAM:033D LRRI $AX0.H, @ $AR0
IRAM:033E ADDPAXZ'L $ACC0, $AX0 : $AX1.H, @ $AR0
IRAM:033F ADDPAXZ'LS $ACC1, $AX1 : $AX0.H, $ACO.M
IRAM:0340 ADDPAXZ'LS $ACC0, $AX0 : $AX1.H, $AC1.M
IRAM:0341 ADDPAXZ'LS $ACC1, $AX1 : $AX0.H, $ACO.M
IRAM:0342 ADDPAXZ'LS $ACC0, $AX0 : $AX1.H, $AC1.M
IRAM:0343 ADDPAXZ'LS $ACC1, $AX1 : $AX0.H, $ACO.M
IRAM:0344 ADDPAXZ'LS $ACC0, $AX0 : $AX1.H, $AC1.M
IRAM:0345 ADDPAXZ'LS $ACC1, $AX1 : $AX0.H, $ACO.M
IRAM:0346 ADDPAXZ'S $ACC0, $AX0 : @ $AR3, $AC1.M
IRAM:0347 SRRI @ $AR3, $ACO.M

```

```

IRAM:0348 ; ac0.m = (*buffer_e04++) - 1;
IRAM:0348 ; ar1 = *buffer_e04++;
IRAM:0348 ; ar0 = *buffer_e04++;
IRAM:0348 LRI $AR3, cmd3_loop_counter
IRAM:034A CLR $ACCO
IRAM:034B CLR'L $ACC1 : $ACO.M, @ $AR3
IRAM:034C LRRI $AR1, @ $AR3 ; data_E05
IRAM:034D LRRI $AR0, @ $AR3 ; cmd3_temp_AR0
IRAM:034E DECM $ACO.M
IRAM:034F SR cmd3_loop_counter, $ACO.M
IRAM:0351 ; while (loop_counter)
IRAM:0351 JNZ cmd3_loop_5_start
IRAM:0353 SET16
IRAM:0354 CLR $ACCO
IRAM:0355 LR $ACO.M, DMEM_B9B
IRAM:0357 TST $ACCO
IRAM:0358 ; if ([b9b] == 0)
IRAM:0358 JZ cmd3_b9b_zero
IRAM:035A ; DMA to MMEM from address stored in [e1c]
IRAM:035A LR $ACO.M, DMEM_B9C
IRAM:035C LR $ACO.L, DMEM_B9D
IRAM:035E SRS DMAMADDRH, $ACO.M
IRAM:035F SRS DMAMADDRL, $ACO.L
IRAM:0360 CLR $ACCO
IRAM:0361 LR $ACO.M, byte_E1C
IRAM:0363 SRS DMADSPADDR, $ACO.M
IRAM:0364 SI DMAControl, 1
IRAM:0366 SI DMALength, 0x40
IRAM:0368 CALL wait_for_dma_finish_0
IRAM:036A ; same sort of setup as in command 2
IRAM:036A
IRAM:036A cmd3_b9b_zero:
IRAM:036A CLR $ACCO
IRAM:036B CLR $ACC1
IRAM:036C LR $ACO.M, loc_B82
IRAM:036E LR $AC1.M, loc_B83
IRAM:0370 SRS DMAMADDRH, $ACO.M
IRAM:0371 SRS DMAMADDRL, $AC1.M
IRAM:0372 SI DMADSPADDR, 0xB80
IRAM:0374 SI DMAControl, 1
IRAM:0376 SI DMALength, 0xC0
IRAM:0378 CALL wait_for_dma_finish_0
IRAM:037A CLR $ACCO
IRAM:037B LR $ACO.M, loc_B80
IRAM:037D LR $ACO.L, loc_B81
IRAM:037F TST $ACCO
IRAM:0380 JNZ loc_386
IRAM:0382 ; restore command_stream pointer
IRAM:0382 LR $AR0, cmd3_temp_command_stream
IRAM:0384 JMP receive_command
IRAM:0386 ; -----
IRAM:0386
IRAM:0386 loc_386:
IRAM:0386 SRS DMAMADDRH, $ACO.M
IRAM:0387 SRS DMAMADDRL, $ACO.L
IRAM:0388 SI DMADSPADDR, 0xB80
IRAM:038A SI DMAControl, 0
IRAM:038C SI DMALength, 0xC0

```

IRAM:038E	LRI	\$AR2, buffer_sections_E08
IRAM:0390	LRI	\$AC1.M, 0
IRAM:0392	SRRI	@\$AR2, \$AC1.M
IRAM:0393	LRI	\$AC1.M, 0x140
IRAM:0395	SRRI	@\$AR2, \$AC1.M
IRAM:0396	LRI	\$AC1.M, 0x280
IRAM:0398	SRRI	@\$AR2, \$AC1.M
IRAM:0399	LRI	\$AC1.M, 0x400
IRAM:039B	SRRI	@\$AR2, \$AC1.M
IRAM:039C	LRI	\$AC1.M, 0x540
IRAM:039E	SRRI	@\$AR2, \$AC1.M
IRAM:039F	LRI	\$AC1.M, 0x680
IRAM:03A1	SRRI	@\$AR2, \$AC1.M
IRAM:03A2	LRI	\$AC1.M, 0x7C0
IRAM:03A4	SRRI	@\$AR2, \$AC1.M
IRAM:03A5	LRI	\$AC1.M, 0x900
IRAM:03A7	SRRI	@\$AR2, \$AC1.M
IRAM:03A8	LRI	\$AC1.M, 0xA40
IRAM:03AA	SRRI	@\$AR2, \$AC1.M
IRAM:03AB	CALL	wait_for_dma_finish_0
IRAM:03AD	LR	\$AC0.M, loc_BA6+1
IRAM:03AF	LR	\$AC1.M, DMEM_BA8
IRAM:03B1	SRS	DMAMADDRH, \$AC0.M
IRAM:03B2	SRS	DMAMADDRL, \$AC1.M
IRAM:03B3	SI	DMADSPADDR, 0x3C0
IRAM:03B5	SI	DMAControl, 0
IRAM:03B7	SI	DMALength, 0x80
IRAM:03B9	CLR	\$ACCO
IRAM:03BA	CLR	\$ACC1
IRAM:03BB	LR	\$AC0.M, DMEM_B84
IRAM:03BD	LRI	\$AC1.M, 0xB31
IRAM:03BF	ADD	\$ACCO, \$ACC1
IRAM:03C0	MRR	\$AR3, \$AC0.M
IRAM:03C1	ILRR	\$AC0.M, @\$AR3
IRAM:03C2	SR	DMEM_E15, \$AC0.M
IRAM:03C4	LR	\$AC0.M, DMEM_B85
IRAM:03C6	LRI	\$AC1.M, 0xB34
IRAM:03C8	ADD	\$ACCO, \$ACC1
IRAM:03C9	MRR	\$AR3, \$AC0.M
IRAM:03CA	ILRR	\$AC0.M, @\$AR3
IRAM:03CB	SR	DMEM_E16, \$AC0.M
IRAM:03CD	LR	\$AC0.M, DMEM_B86
IRAM:03CF	LRI	\$AC1.M, 0xB11
IRAM:03D1	ADD	\$ACCO, \$ACC1
IRAM:03D2	MRR	\$AR3, \$AC0.M
IRAM:03D3	ILRR	\$AC0.M, @\$AR3
IRAM:03D4	SR	DMEM_E14, \$AC0.M
IRAM:03D6	CLR	\$ACCO
IRAM:03D7	LR	\$AC0.M, DMEM_B9B
IRAM:03D9	TST	\$ACCO
IRAM:03DA	JZ	loc_403
IRAM:03DC	CLR	\$ACC1
IRAM:03DD	LR	\$AC1.M, loc_B9E
IRAM:03DF	ADDI	\$AC1.M, 0xCC0
IRAM:03E1	SR	cmd2_DMEM_E40_start, \$AC1.M
IRAM:03E3	LR	\$AC1.M, loc_B9F
IRAM:03E5	ADDI	\$AC1.M, 0xCC0
IRAM:03E7	SR	cmd2_DMEM_E41_end, \$AC1.M

```

IRAM:03E9          LRI          $AC1.M, 0xCE0
IRAM:03EB          SR          cmd2_DMEM_E42, $AC1.M
IRAM:03ED          SR          cmd2_DMEM_E43, $AC1.M
IRAM:03EF          CALL        wait_for_dma_finish_0
IRAM:03F1          LR          $AC0.M, DMEM_B9C
IRAM:03F3          SRS          DMAMADDRH, $AC0.M
IRAM:03F4          LR          $AC0.M, DMEM_B9D
IRAM:03F6          SRS          DMAMADDRL, $AC0.M
IRAM:03F7          SI          DMADSPADDR, 0xCC0
IRAM:03F9          SI          DMAControl, 0
IRAM:03FB          SI          DMALength, 0x40
IRAM:03FD          CALL        wait_for_dma_finish_0
IRAM:03FF          ; restore command_stream pointer
IRAM:03FF          LR          $AR0, cmd3_temp_command_stream
IRAM:0401          JMP          command_3
IRAM:0403          ; -----
IRAM:0403          loc_403:
IRAM:0403          LRI          $AC1.M, 0xCE0
IRAM:0405          SR          cmd2_DMEM_E42, $AC1.M
IRAM:0407          SR          cmd2_DMEM_E40_start, $AC1.M
IRAM:0409          SR          cmd2_DMEM_E41_end, $AC1.M
IRAM:040B          SR          cmd2_DMEM_E43, $AC1.M
IRAM:040D          CALL        wait_for_dma_finish_0
IRAM:040F          LR          $AR0, cmd3_temp_command_stream
IRAM:0411          JMP          command_3

```

A.5 Command 0x4, 0x5 and 0x9

```

IRAM:0413          command_4:
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 mix_buffers with 0x400
IRAM:0414          LRI          $IX2, 0x400
IRAM:0416          CLR          $ACCO
IRAM:0417          CLR'L       $ACC1 : $AC0.M, @ $AR0
IRAM:0418          LRRl        $AC0.L, @ $AR0
IRAM:0419          SRS          DMAMADDRH, $AC0.M
IRAM:041A          SRS          DMAMADDRL, $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        mix_buffers
IRAM:0425          JMP          receive_command

IRAM:0484          mix_buffers:
IRAM:0484          ; command_4+10tp ...
IRAM:0484          SET16
IRAM:0485          ; IX2 holds some address:
IRAM:0485          ; 0x400 for cmd4
IRAM:0485          ; 0x7c0 for cmd9/5
IRAM:0485          ; read main memory address from command stream
IRAM:0485          ; store in AC1.ml and AX0.hl
IRAM:0485          LRRl        $AC1.M, @ $AR0

```

```

IRAM:0486          LRR1          $AC1.L, @ $AR0
IRAM:0487          MRR          $AX0.H, $AC1.M
IRAM:0488          MRR          $AX0.L, $AC1.L
IRAM:0489 ; start DMA from main memory address to DSP DME0 0x400 length 0xc0
IRAM:0489          SRS          DMAMADDRH, $AC1.M
IRAM:048A          SRS          DMAMADDRL, $AC1.L
IRAM:048B          CLR          $ACC1
IRAM:048C          MRR          $AC1.L, $IX2
IRAM:048D          SRS          DMADSPADDR, $AC1.L
IRAM:048E          LRIS        $AC0.M, 0
IRAM:048F          SRS          DMAControl, $AC0.M
IRAM:0490          CLR          $ACCO
IRAM:0491          LRI          $AC0.L, 0xc0
IRAM:0493          SRS          DMALength, $AC0.L
IRAM:0494 ; save command stream pointer
IRAM:0494          MRR          $IX1, $AR0
IRAM:0495 ; ar1: 0xe44
IRAM:0495 ; ac0: mmaddr + 0xc0
IRAM:0495          LRI          $AR1, 0xE44
IRAM:0497          ADDAX        $ACCO, $AX0
IRAM:0498 ; *(u32*)0xe44 = ac0 (mmaddr + 0xc0)
IRAM:0498          SRR1        @ $AR1, $AC0.M
IRAM:0499          SRR1        @ $AR1, $AC0.L
IRAM:049A          LRIS        $AX1.H, 0
IRAM:049B          LRI          $AX1.L, 0x60
IRAM:049D ac1 = 0x400 + 0x60
IRAM:049D          ADDAX        $ACC1, $AX1
IRAM:049E *0xe46 = ac1
IRAM:049E          SRR1        @ $AR1, $AC1.L
IRAM:049F ar1 = 0xe44
IRAM:049F ar0 = 0x400
IRAM:049F ar2 = ar3 = 0
IRAM:049F          LRI          $AR1, 0xE44
IRAM:04A1          MRR          $AR0, $IX2
IRAM:04A2          LRI          $AR3, 0
IRAM:04A4          MRR          $AR2, $AR3
IRAM:04A5
IRAM:04A5 cmdf_wait_for_dma_finish:
IRAM:04A5          LRS          $AC1.M, DMAControl
IRAM:04A6          ANDF        $AC1.M, 4
IRAM:04A8          JLNZ        cmdf_wait_for_dma_finish
IRAM:04AA REPEAT 9 TIMES
IRAM:04AA          BLOOPI      9, loc_4DA
IRAM:04AC          SET16
IRAM:04AD read mmaddr from 0xe44
IRAM:04AD          recall: we stored one right before this
IRAM:04AD          LRR1        $AX0.H, @ $AR1
IRAM:04AE          LRR1        $AX0.L, @ $AR1
IRAM:04AF AX0 still holds initial MMADDR
IRAM:04AF          MOVAX        $ACC1, $AX0
IRAM:04B0          SRS          DMAMADDRH, $AC1.M
IRAM:04B1          SRS          DMAMADDRL, $AC1.L
IRAM:04B2          CLR          $ACC1
IRAM:04B3 read dsp address from 0xe44 buffer
IRAM:04B3          recal: we stored one right before this
IRAM:04B3          LRR1        $AC1.L, @ $AR1
IRAM:04B4          SRS          DMADSPADDR, $AC1.L
IRAM:04B5          SI          DMAControl, 0

```

```

IRAM:04B7 DMA length 0xc0
IRAM:04B7 CLR ACC0
IRAM:04B8 LRI AC0.L, 0xC0
IRAM:04BA SRS DMALength, AC0.L
IRAM:04BB restore ar1 pointer to address data
IRAM:04BB LRI AR1, 0xE44
IRAM:04BD ac0 = mmaddr + 0xc0
IRAM:04BD ADDAX ACC0, AX0
IRAM:04BE store mmaddr + 0xc0 again (keep incrementing)
IRAM:04BE SRRI @AR1, AC0.M
IRAM:04BF SRRI @AR1, AC0.L
IRAM:04C0 store dspaddr + 0x60 again (keep incrementing)
IRAM:04C0 LRIS AX1.H, 0
IRAM:04C1 LRIS AX1.L, 0x60
IRAM:04C2 ADDAX ACC1, AX1
IRAM:04C3 SRRI @AR1, AC1.L
IRAM:04C4 restore ar1 pointer to address data
IRAM:04C4 LRI AR1, 0xE44
IRAM:04C6 SET40
IRAM:04C7 ; ar0 holds 0x400 in first iteration
IRAM:04C7 ; ar3 holds 0 in first iteration
IRAM:04C7 NX'LD AX0.H : AX1.H, @SAR0
IRAM:04C8 NX'LD AX0.L : AX1.L, @SAR0
IRAM:04C9 MOVAX ACC0, AX1
IRAM:04CA ADDAX ACC0, AX0
IRAM:04CB ; REPEAT 0x17 TIMES
IRAM:04CB BLOOPI 0x17, loc_4D4
IRAM:04CD NX'LD AX0.H : AX1.H, @SAR0
IRAM:04CE NX'LD AX0.L : AX1.L, @SAR0
IRAM:04CF MOVAX' S ACC1, AX1 : @SAR2, AC0.M
IRAM:04D0 ADDAX' S ACC1, AX0 : @SAR2, AC0.L
IRAM:04D1 NX'LD AX0.H : AX1.H, @SAR0
IRAM:04D2 NX'LD AX0.L : AX1.L, @SAR0
IRAM:04D3 MOVAX' S ACC0, AX1 : @SAR2, AC1.M
IRAM:04D4
IRAM:04D4 loc_4D4:
IRAM:04D4 ADDAX' S ACC0, AX0 : @SAR2, AC1.L
IRAM:04D5 ; BLOOPI END
IRAM:04D5 NX'LD AX0.H : AX1.H, @SAR0
IRAM:04D6 NX'LD AX0.L : AX1.L, @SAR0
IRAM:04D7 MOVAX' S ACC1, AX1 : @SAR2, AC0.M
IRAM:04D8 ADDAX' S ACC1, AX0 : @SAR2, AC0.L
IRAM:04D9 SRRI @SAR2, AC1.M
IRAM:04DA
IRAM:04DA loc_4DA:
IRAM:04DA SRRI @SAR2, AC1.L
IRAM:04DB ; BLOOPI END
IRAM:04DB NX'LD AX0.H : AX1.H, @SAR0
IRAM:04DC NX'LD AX0.L : AX1.L, @SAR0
IRAM:04DD MOVAX ACC0, AX1
IRAM:04DE ADDAX ACC0, AX0
IRAM:04DF BLOOPI 0x17, loc_4E8
IRAM:04E1 NX'LD AX0.H : AX1.H, @SAR0
IRAM:04E2 NX'LD AX0.L : AX1.L, @SAR0
IRAM:04E3 MOVAX' S ACC1, AX1 : @SAR2, AC0.M
IRAM:04E4 ADDAX' S ACC1, AX0 : @SAR2, AC0.L
IRAM:04E5 NX'LD AX0.H : AX1.H, @SAR0
IRAM:04E6 NX'LD AX0.L : AX1.L, @SAR0

```

```

IRAM:04E7                MOVAXTS      $ACCO, $AX1 : @T$AR2, $AC1.M
IRAM:04E8
IRAM:04E8 loc_4E8:
IRAM:04E8                ADDAXTS      $ACCO, $AX0 : @T$AR2, $AC1.L
IRAM:04E9                NXTLD      $AX0.H : $AX1.H, @T$AR0
IRAM:04EA                NXTLD      $AX0.L : $AX1.L, @T$AR0
IRAM:04EB                MOVAXTS      $ACC1, $AX1 : @T$AR2, $AC0.M
IRAM:04EC                ADDAXTS      $ACC1, $AX0 : @T$AR2, $AC0.L
IRAM:04ED                SRRTI      @T$AR2, $AC1.M
IRAM:04EE                SRRTI      @T$AR2, $AC1.L
IRAM:04EF ; restore command stream pointer
IRAM:04EF                MRR      $AR0, $IX1
IRAM:04F0                RET

```

A.6 Command 0x6

```

IRAM:0165 command_6:
IRAM:0165                CLR      $ACCO
IRAM:0166                SET16
IRAM:0167 DMA 0x780 bytes from DSP DMEM[0] to main mem address from command stream
IRAM:0167                LRRTI      $ACO.M, @T$AR0
IRAM:0168                LRRTI      $ACO.L, @T$AR0
IRAM:0169                SRS      DMAMADDRH, $ACO.M
IRAM:016A                SRS      DMAMADDRL, $ACO.L
IRAM:016B                SI      DMADSPADDR, 0
IRAM:016D                SI      DMAControl, 1
IRAM:016F                SI      DMALength, 0x780
IRAM:0171                CALL     wait_for_dma_finish_0
IRAM:0173                JMP      receive_command
IRAM:0173 ; End of function command_6

```

A.7 Command 0x7

```

command_7:
IRAM:0574                CLR      $ACCO
IRAM:0575                CLRTL      $ACC1 : $ACO.M, @T$AR0
IRAM:0576                SET16TL      $ACO.L : @T$AR0
IRAM:0577 ; DMA 0x20 bytes from mmaddr read from command stream to buffer at 0xe44
IRAM:0577                SRS      DMAMADDRH, $ACO.M
IRAM:0578                SRS      DMAMADDRL, $ACO.L
IRAM:0579                SI      DMADSPADDR, 0xE44
IRAM:057B                SI      DMAControl, 0
IRAM:057D                CLR      $ACC1
IRAM:057E                LRIS     $AC1.L, 0x20
IRAM:057F                SRS      DMALength, $AC1.L
IRAM:0580 ; mmaddr += 0x20
IRAM:0580                ADD      $ACCO, $ACC1
IRAM:0581 ; save command_stream pointer
IRAM:0581                MRR      $IX0, $AR0
IRAM:0582                LRI      $AR0, 0x280
IRAM:0584                LRI      $AR1, 0
IRAM:0586                LRI      $AR2, 0x140
IRAM:0588                LRI      $AR3, 0xE44
IRAM:058A                LRIS     $AX0.H, 0
IRAM:058B ; wait for DMA to finish
IRAM:058B
IRAM:058B loc_58B:

```



```

IRAM:058B          LRS          $AC1.M, DMAControl
IRAM:058C          ANDF         $AC1.M, 4
IRAM:058E          JLNZ         loc_58B
IRAM:0590          ; DMA 0x260 bytes from mmaddr to 0xe54 (contiguous with previous section)
IRAM:0590          SRS          DMAMADDRH, $AC0.M
IRAM:0591          SRS          DMAMADDRL, $AC0.L
IRAM:0592          SI           DMADSPADDR, 0xE54
IRAM:0594          SI           DMAControl, 0
IRAM:0596          SI           DMALength, 0x260
IRAM:0598          LRI          $AC1.M, 0xA0
IRAM:059A          SET40
IRAM:059B          ; REPEAT 0xA0 = 160 TIMES
IRAM:059B          ; initially:
IRAM:059B          ; AR0-AR3: sections in 0x3c0 buffer at 0x0000
IRAM:059B          ; AR0 = 0x280
IRAM:059B          ; AR1 = 0
IRAM:059B          ; AR2 = 0x140
IRAM:059B          ; AR3 = 0xe44 // buffered data
IRAM:059B          BLOOP        $AC1.M, loc_5A4
IRAM:059D          LRRI         $AC0.M, @ $AR3
IRAM:059E          ; clear out section at 0x280
IRAM:059E          ; copy words from 0xe44 to 0x000 and 0x140
IRAM:059E          SRRI         @ $AR0, $AX0.H
IRAM:059F          LRRI         $AC0.L, @ $AR3
IRAM:05A0          SRRI         @ $AR0, $AX0.H
IRAM:05A1          SRRI         @ $AR2, $AC0.M
IRAM:05A2          SRRI         @ $AR2, $AC0.L
IRAM:05A3          SRRI         @ $AR1, $AC0.M
IRAM:05A4
IRAM:05A4 loc_5A4:
IRAM:05A4          SRRI         @ $AR1, $AC0.L
IRAM:05A5          ; BLOOP END
IRAM:05A5
IRAM:05A5          ; restore command_stream pointer
IRAM:05A5          MRR          $AR0, $IX0
IRAM:05A6          JMP          receive_command

```

A.8 Command 0x8

```

command_8:
IRAM:0B37          SET16
IRAM:0B38          CLR          $ACCO
IRAM:0B39          ; read mmaddr from command stream
IRAM:0B39          ; DMA 0x100 bytes to 0xe80 from mmaddr
IRAM:0B39          CLR'L       $ACC1 : $AC0.M, @ $AR0
IRAM:0B3A          LRRI         $AC0.L, @ $AR0
IRAM:0B3B          SRS          DMAMADDRH, $AC0.M
IRAM:0B3C          SRS          DMAMADDRL, $AC0.L
IRAM:0B3D          SI           DMADSPADDR, 0xE80
IRAM:0B3F          SI           DMAControl, 0
IRAM:0B41          SI           DMALength, 0x100
IRAM:0B43          ; save mmaddr in AX1
IRAM:0B43          MRR          $AX1.H, $AC0.M
IRAM:0B44          MRR          $AX1.L, $AC0.L
IRAM:0B45          CLR          $ACCO
IRAM:0B46          ; wait for DMA to finish
IRAM:0B46

```



```

IRAM:0B46 loc_B46:
IRAM:0B46          LRS          $AC0.M, DMAControl
IRAM:0B47          ANDF         $AC0.M, 4
IRAM:0B49          JLNZ         loc_B46
IRAM:0B4B ; read another mmaddr from command stream and DMA 0x280 bytes to 0x280
IRAM:0B4B          LRRI         $AC0.M, @ $AR0
IRAM:0B4C          LRRI         $AC0.L, @ $AR0
IRAM:0B4D          SRS          DMAMADDRH, $AC0.M
IRAM:0B4E          SRS          DMAMADDRL, $AC0.L
IRAM:0B4F          SI           DMADSPADDR, 0x280
IRAM:0B51          SI           DMAControl, 0
IRAM:0B53          SI           DMALength, 0x280
IRAM:0B55 ; save command stream pointer
IRAM:0B55          MRR          $IX0, $AR0
IRAM:0B56 ; ar0: data_ptr, pointer to DMA data
IRAM:0B56          LRI          $AR0, 0x280
IRAM:0B58          LR           $AR1, byte_E1B ; written to in main_entry (0xe80)
IRAM:0B5A ; IX1 = 0
IRAM:0B5A ; WR1 = 0x7f
IRAM:0B5A ; AR2 = 0xf00
IRAM:0B5A ; IX3 = AR3 = 0x16b4 (in DSP_COEF)
IRAM:0B5A ; AR1: buffer_f00, wrap every 0x80 bytes
IRAM:0B5A ; AR3: coef
IRAM:0B5A          LRI          $IX1, 0
IRAM:0B5C          LRI          $WR1, 0x7F
IRAM:0B5E          LRI          $AR2, 0xF00
IRAM:0B60          LRI          $AR3, 0x16B4
IRAM:0B62          MRR          $IX3, $AR3
IRAM:0B63          CLR          $AC0
IRAM:0B64 ; wait for DMA to finish
IRAM:0B64
IRAM:0B64 loc_B64:
IRAM:0B64          LRS          $AC0.M, DMAControl
IRAM:0B65          ANDF         $AC0.M, 4
IRAM:0B67          JLNZ         loc_B64
IRAM:0B69          SET40
IRAM:0B6A ; u32 data = *(u32*)data_ptr++;
IRAM:0B6A          M2'L         $AC1.M : @ $AR0
IRAM:0B6B          CLR15'L      $AC1.L : @ $AR0
IRAM:0B6C          LSL16        $ACC1
IRAM:0B6D ; *ptr_E1B = data.lo
IRAM:0B6D          SRR          @ $AR1, $AC1.M
IRAM:0B6E ; AX0.H = AX1.L = *coef++
IRAM:0B6E          CLRP'L       $AX0.H : $AX1.L, @ $AR3
IRAM:0B6F ; prod = 0
IRAM:0B6F          LOOPI        0x7E
IRAM:0B70 ; WHAT IS AX0.L INITIALLY??
IRAM:0B70
IRAM:0B70 ; repeat 0x7e:
IRAM:0B70 ;     prod += AX0.L * AX0.H
IRAM:0B70 ;     AX0.H = AX1.L = *coef++
IRAM:0B70          MADD'L       $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3 ; REPEAT 0x7e = 12
IRAM:0B71 ; prod += AX0.L * AX0.H
IRAM:0B71 ; AX0.H = AX1.L = *coef
IRAM:0B71 ; coef += 0x16b4 (this doesnt matter because ar3 is reloaded after this)
IRAM:0B71          MADD'L       $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3
IRAM:0B72 ; ac0 = prod + AX0.L * AX0.H
IRAM:0B72 ; ac1.ml = *(u32*)data_ptr++

```

```

IRAM:0B72          MADDL          $AX0.L, $AX0.H : $AC1.M, @ $AR0
IRAM:0B73          MOVPL          $ACCO : $AC1.L, @ $AR0
IRAM:0B74 ; ac1 <= 16
IRAM:0B74 ; *buffer_f00++ = ac0.m
IRAM:0B74          LSL16S          $ACC1 : @ $AR2, $ACO.M
IRAM:0B75          SRR              @ $AR1, $AC1.M
IRAM:0B76 ; REPEAT 0x9e = 158 TIMES
IRAM:0B76          BLOOPI          0x9E, loc_B80
IRAM:0B78 ; coef = 0x16b4
IRAM:0B78          MRR              $AR3, $IX3
IRAM:0B79 ; same thing as before
IRAM:0B79          CLRPLLD        $AX0.H : $AX1.L, @ $AR3
IRAM:0B7A          LOOPI          0x7E
IRAM:0B7B          MADDLLD        $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3
IRAM:0B7C          MADDLLDN       $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3
IRAM:0B7D          MADDL          $AX0.L, $AX0.H : $AC1.M, @ $AR0
IRAM:0B7E          MOVPL          $ACCO : $AC1.L, @ $AR0
IRAM:0B7F          LSL16S          $ACC1 : @ $AR2, $ACO.M
IRAM:0B80          SRR              @ $AR1, $AC1.M
IRAM:0B81 ; BLOOPI END
IRAM:0B81          MRR              $AR3, $IX3
IRAM:0B82 ; restore coef = 0x16b4
IRAM:0B82 ; same thing as before
IRAM:0B82          CLRPLLD        $AX0.H : $AX1.L, @ $AR3
IRAM:0B83          LOOPI          0x7E
IRAM:0B84          MADDLLD        $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3
IRAM:0B85          ; command_3+17C|LLDN       $AX0.L, $AX0.H : $AX0.H, $AX1.L, @ $AR3
IRAM:0B86          MADD            $AX0.L, $AX0.H
IRAM:0B87          MOVP            $ACCO
IRAM:0B88          SRR1           @ $AR2, $ACO.M
IRAM:0B89 ; store end of 0xf00 buffer to e1b
IRAM:0B89          SR              byte_E1B, $AR1
IRAM:0B8B          LRI              $AR0, 0x280
IRAM:0B8D          LRI              $AR3, 0xF00
IRAM:0B8F          LRI              $AR1, 0
IRAM:0B91          LRI              $AR2, 0x140
IRAM:0B93          LRI              $WR1, 0xFFFF
IRAM:0B95          CLR              $ACC1
IRAM:0B96          CLR              $ACCO
IRAM:0B97          SET40
IRAM:0B98 REPEAT 0xa0 160 TIMES
IRAM:0B98          BLOOPI          0xA0, loc_BA0
IRAM:0B9A ; ac1.ml = EXT16(*buffer_f00++)
IRAM:0B9A ; *buffer_280++ = 0
IRAM:0B9A ; *buffer_280++ = 0
IRAM:0B9A          LRR1            $AC1.M, @ $AR3
IRAM:0B9B          ASR16S          $ACC1 : @ $AR0, $ACO.M
IRAM:0B9C          SRR1           @ $AR0, $ACO.M
IRAM:0B9D ; store ac1.ml to *buffer_0++, *buffer_0++
IRAM:0B9D          SRR1           @ $AR1, $AC1.M
IRAM:0B9E          NEGS            $ACC1 : @ $AR1, $AC1.L
IRAM:0B9F ; store -ac1.ml to *buffer_140++, *buffer_140++
IRAM:0B9F          SRR1           @ $AR2, $AC1.M
IRAM:0BA0          SRR1           @ $AR2, $AC1.L
IRAM:0BA1 ; BLOOPI END
IRAM:0BA1          SET16
IRAM:0BA2 ; restore mmaddr


```

```

IRAM:0BA2          MRR          $ACO.M, $AX1.H
IRAM:0BA3          MRR          $ACO.L, $AX1.L
IRAM:0BA4 ; DMA DMEM 0xe80 back to main memory
IRAM:0BA4          SRS          DMAMADDRH, $ACO.M
IRAM:0BA5          SRS          DMAMADDRL, $ACO.L
IRAM:0BA6
IRAM:0BA6 loc_BA6:
IRAM:0BA6          ; command_3+165tr
IRAM:0BA6          SI          DMADSPADDR, 0xE80
IRAM:0BA8
IRAM:0BA8 DMEM_BA8:
IRAM:0BA8          ; command_3+167tr
IRAM:0BA8          SI          DMAControl, 1
IRAM:0BAA          SI          DMALength, 0x100
IRAM:0BAC          CALL        wait_for_dma_finish_0
IRAM:0BAE ; restore command_stream pointer
IRAM:0BAE          MRR          $ARO, $IX0
IRAM:0BAF          JMP         receive_comman

```

A.9 Command 0xd

```

IRAM:01A9 command_d:
IRAM:01A9          SET16'L     $ACO.M : @ $ARO
IRAM:01AA ; load main memory address and length from command stream
IRAM:01AA          CLR'L      $ACC1 : $ACO.L, @ $ARO
IRAM:01AB          LRRl       $AC1.M, @ $ARO
IRAM:01AC ; DMA to command stream address
IRAM:01AC          SRS          DMAMADDRH, $ACO.M
IRAM:01AD          SRS          DMAMADDRL, $ACO.L
IRAM:01AE          SI          DMADSPADDR, 0xC00
IRAM:01B0          SI          DMAControl, 0
IRAM:01B2          ADDIS       $AC1.M, 3
IRAM:01B3          ANDI        $AC1.M, 0xFFFF0
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         $ARO, 0xC00
IRAM:01BA          JMP         receive_command

```

A.10 Command 0xe

```

IRAM:043B command_e:
IRAM:043B          CLR15
IRAM:043C          M2
IRAM:043D          CLR        $ACCO
IRAM:043E ; read mmaddr from command stream and DMA 0x280 bytes from 0x280
IRAM:043E          CLR'L      $ACC1 : $ACO.M, @ $ARO
IRAM:043F          LRRl       $AC1.M, @ $ARO
IRAM:0440          SRS          DMAMADDRH, $ACO.M
IRAM:0441          SRS          DMAMADDRL, $AC1.M
IRAM:0442          SI          DMADSPADDR, 0x280
IRAM:0444          SI          DMAControl, 1
IRAM:0446          SI          DMALength, 0x280
IRAM:0448 ; load mmaddr AX0.HL from command stream
IRAM:0448          SET40'L     $AX0.H : @ $ARO
IRAM:0449          CLR'L      $ACCO : $AX0.L, @ $ARO

```

```

IRAM:044A          LRI          $AR1, 0x400 ; u16* buffer_400
IRAM:044C          LRI          $AR3, 0 ; u16* buffer_0
IRAM:044E          LRI          $AR2, 0x140 ; u16* buffer_140
IRAM:0450          LRI          $AX1.L, 0x80
IRAM:0452          CALL         wait_for_dma_finish_0
IRAM:0454 ; REPEAT 5 TIMES
IRAM:0454          BLOOPI       5, loc_46C
IRAM:0456          MRR          $AX1.H, $AR1 ; buffer400_loop_start
IRAM:0457 ; REPEAT 0x20 TIMES
IRAM:0457          BLOOPI       0x20, loc_45E
IRAM:0459 ; ac0.ml = *(u32*)buffer_140++;
IRAM:0459 ; ac0 <= 16;
IRAM:0459 ; ac1.ml = *(u32*)buffer_0++;
IRAM:0459 ; ac1 <= 16;
IRAM:0459 ; *buffer_400++ = ac0.m
IRAM:0459 ; *buffer_400++ = ac1.m
IRAM:0459          CLR1L       $ACC1 : $ACO.M, @ $AR2
IRAM:045A          LRR1         $ACO.L, @ $AR2
IRAM:045B          LSL161L     $ACCO : $AC1.M, @ $AR3
IRAM:045C          LRR1         $AC1.L, @ $AR3
IRAM:045D          LSL161S     $ACC1 : @ $AR1, $ACO.M
IRAM:045E
IRAM:045E loc_45E:
IRAM:045E          CLR1S       $ACCO : @ $AR1, $AC1.M
IRAM:045F ; BLOOPI END
IRAM:045F          CLR          $ACC1
IRAM:0460          MOVAX        $ACCO, $AX0 ; mmaddr
IRAM:0461          SRS          DMAMADDRH, $ACO.M
IRAM:0462          SRS          DMAMADDRL, $ACO.L
IRAM:0463          MRR          $AC1.M, $AX1.H ; buffer400_loop_start
IRAM:0464          SRS          DMADSPADDR, $AC1.M
IRAM:0465          LRIS        $AC1.M, 1
IRAM:0466          SRS          DMAControl, $AC1.M
IRAM:0467          MRR          $AC1.M, $AX1.L
IRAM:0468          SRS          DMALength, $AC1.M
IRAM:0469          ADDAXL       $ACCO, $AX1.L ; 0x80
IRAM:046A          MRR          $AX0.H, $ACO.M ; mmaddr += 0x80
IRAM:046B          MRR          $AX0.L, $ACO.L
IRAM:046C
IRAM:046C loc_46C:
IRAM:046C          CLR          $ACCO
IRAM:046D ; BLOOPI END
IRAM:046D
IRAM:046D cmde_wait_for_dma_finish:
IRAM:046D          LRS          $ACO.M, DMAControl
IRAM:046E          ANDF         $ACO.M, 4
IRAM:0470          JLNZ         cmde_wait_for_dma_finish
IRAM:0472          JMP          receive_command
IRAM:0472 ; End of function command_e

```

A.11 Command 0xf

```

command_f:
IRAM:047A
IRAM:047A ; FUNCTION CHUNK AT IRAM:0C91 SIZE 0000000D BYTES
IRAM:047A
IRAM:047A          SI          ToCPUMailHi, 0xDCD1

```

```

IRAM:047C all jump table entries are some form of resetting
IRAM:047C
IRAM:047C send mail dcd10002 and trigger IRQ
IRAM:047C             SI             ToCPUMailLo, 2
IRAM:047E             SI             DIRQ, 1
IRAM:0480             JMP            cmdf_jump_table_select
...

IRAM:0C8D cmd_f_table      .word j_send_dcd10001_irq, debug_reset, j_rom_start, j_wait_for_babe_0
IRAM:0C91 ; -----
IRAM:0C91 ; START OF FUNCTION CHUNK FOR command_f
IRAM:0C91 cmdf_jump_table_select:
IRAM:0C91             SET16
IRAM:0C92             CLR            $ACCO
IRAM:0C93             CLR            $ACC1
IRAM:0C94             CALL           cmdf_wait_for_cpu_mail_ac0m
IRAM:0C96             LRS            $AC1.M, FromCPUMailLo
IRAM:0C97             LRI            $AC0.M, cmd_f_table
IRAM:0C99             ADD            $ACCO, $ACC1
IRAM:0C9A             MRR            $AR3, $AC0.M
IRAM:0C9B             ILRR           $AC1.M, @ $AR3
IRAM:0C9C             MRR            $AR3, $AC1.M
IRAM:0C9D             JMPR           $AR3
IRAM:0C9D ; END OF FUNCTION CHUNK FOR command_f
IRAM:0C9E ; -----
IRAM:0C9E             HALT
IRAM:0C9F
IRAM:0C9F ; ===== S U B R O U T I N E =====
IRAM:0C9F
IRAM:0C9F ; Attributes: thunk
IRAM:0C9F
IRAM:0C9F j_send_dcd10001_irq:
IRAM:0C9F             JMP            send_dcd10001_irq
IRAM:0C9F ; End of function j_send_dcd10001_irq
IRAM:0C9F
IRAM:0CA1 ; -----
IRAM:0CA1             HALT
IRAM:0CA2
IRAM:0CA2 ; ===== S U B R O U T I N E =====
IRAM:0CA2
IRAM:0CA2 debug_reset:
IRAM:0CA2             CLR            $ACCO
IRAM:0CA3             CLR            $ACC1
IRAM:0CA4             CALL           cmdf_wait_for_cpu_mail_ac0m
IRAM:0CA6             LRS            $AC0.L, FromCPUMailLo
IRAM:0CA7             CALL           wait_for_mail_from_cpu_ac1m
IRAM:0CA9             LRS            $AC1.L, FromCPUMailLo
IRAM:0CAA             CALL           wait_for_mail_from_cpu_ac1m
IRAM:0CAC             LRS            $AC1.M, FromCPUMailLo
IRAM:0CAD             SRS            DMAMADDRH, $AC0.M
IRAM:0CAE             SRS            DMAMADDRL, $AC0.L
IRAM:0CAF             SI             DMAControl, 1
IRAM:0CB1             SRS            DMADSPADDR, $AC1.M
IRAM:0CB2             SRS            DMALength, $AC1.L
IRAM:0CB3             CLR            $ACCO

```

```

IRAM:0CB4          CLR          $ACC1
IRAM:0CB5          CALL         cmdf_wait_for_cpu_mail_ac0m
IRAM:0CB7          LRS          $AC0.L, FromCPUMailLo
IRAM:0CB8          MRR          $IX0, $AC0.M
IRAM:0CB9          MRR          $IX1, $AC0.L
IRAM:0CBA          CALL         wait_for_mail_from_cpu_ac1m
IRAM:0CBC          LRS          $AC1.L, FromCPUMailLo
IRAM:0CBD          CALL         wait_for_mail_from_cpu_ac1m
IRAM:0CBF          LRS          $AC1.M, FromCPUMailLo
IRAM:0CC0          MRR          $IX2, $AC1.M
IRAM:0CC1          MRR          $IX3, $AC1.L
IRAM:0CC2          CLR          $ACCO
IRAM:0CC3          CALL         cmdf_wait_for_cpu_mail_ac0m
IRAM:0CC5          LRS          $AC0.M, FromCPUMailLo
IRAM:0CC6          MRR          $AR0, $AC0.M
IRAM:0CC7          CLR          $ACC1
IRAM:0CC8          CALL         wait_for_mail_from_cpu_ac1m
IRAM:0CCA          LRS          $AX0.L, FromCPUMailLo
IRAM:0CCB          MRR          $AX0.H, $AC1.M
IRAM:0CCC          CALL         cmdf_wait_for_cpu_mail_ac0m
IRAM:0CCE          LRS          $AX1.L, FromCPUMailLo
IRAM:0CCF          CALL         cmdf_wait_for_cpu_mail_ac0m
IRAM:0CD1          LRS          $AX1.H, FromCPUMailLo
IRAM:0CD2
IRAM:0CD2 cmdf_wait_for_dma_finish:
IRAM:0CD2          LRS          $AC0.M, DMAControl
IRAM:0CD3          ANDF         $AC0.M, 4
IRAM:0CD5          JLNZ         cmdf_wait_for_dma_finish
IRAM:0CD7          JMP          sub_80B5
IRAM:0CD7 ; End of function debug_reset
IRAM:0CD7
IRAM:0CD7 ; -----
IRAM:0CD9          .word      0x21 ; !
IRAM:0CDA
IRAM:0CDA ; ===== S U B R O U T I N E =====
IRAM:0CDA
IRAM:0CDA ; Attributes: thunk
IRAM:0CDA
IRAM:0CDA j_rom_start:
IRAM:0CDA          JMP          rom_start
IRAM:0CDA ; End of function j_rom_start
IRAM:0CDA
IRAM:0CDC ; -----
IRAM:0CDC          HALT
IRAM:0CDD
IRAM:0CDD ; ===== S U B R O U T I N E =====
IRAM:0CDD
IRAM:0CDD ; Attributes: thunk
IRAM:0CDD
IRAM:0CDD j_wait_for_babe_0:
IRAM:0CDD          JMP          wait_for_babe
IRAM:0CDD ; End of function j_wait_for_babe_0
IRAM:0CDD
IRAM:0CDF ; -----
IRAM:0CDF          HALT
IRAM:0CEO
IRAM:0CEO ; ===== S U B R O U T I N E =====
IRAM:0CEO

```

```

IRAM:0CE0
IRAM:0CE0 cmdf_wait_for_cpu_mail_ac0m:
IRAM:0CE0 ; debug_reset+2tp ...
IRAM:0CE0 LRS $AC0.M, FromCPUMailHi
IRAM:0CE1 ANDCF $AC0.M, 0x8000
IRAM:0CE3 JLNZ cmdf_wait_for_cpu_mail_ac0m
IRAM:0CE5 RET
IRAM:0CE5 ; End of function cmdf_wait_for_cpu_mail_ac0m
IRAM:0CE5
IRAM:0CE6
IRAM:0CE6 ; ===== S U B R O U T I N E =====
IRAM:0CE6
IRAM:0CE6 wait_for_mail_from_cpu_ac1m:
IRAM:0CE6 ; debug_reset+8tp ...
IRAM:0CE6 LRS $AC1.M, FromCPUMailHi
IRAM:0CE7 ANDCF $AC1.M, 0x8000
IRAM:0CE9 JLNZ wait_for_mail_from_cpu_ac1m
IRAM:0CEB RET
IRAM:0CEB ; End of function wait_for_mail_from_cpu_ac1m

...

ROM:80B5 sub_80B5:
ROM:80B5
ROM:80B5 ; FUNCTION CHUNK AT ROM:80C4 SIZE 00000015 BYTES
ROM:80B5
ROM:80B5 SET16
ROM:80B6 CLR $AC0
ROM:80B7 MRR $AC0.M, $AX1.L
ROM:80B8 ANDI $AC0.M, 0xFFFF
ROM:80BA JZ transfer_ucode
ROM:80BC LRIS $AC0.M, 0
ROM:80BD SRS DMAControl, $AC0.M
ROM:80BE SRS DMAMADDRH, $AX0.H
ROM:80BF SRS DMAMADDRL, $AX0.L
ROM:80C0 SRS DMADSPADDR, $AX1.H
ROM:80C1 SRS DMALength, $AX1.L
ROM:80C2 CALL wait_for_dma_finish
ROM:80C2 ; End of function sub_80B5
ROM:80C2
ROM:80C4 ; START OF FUNCTION CHUNK FOR rom_start
ROM:80C4 ; ADDITIONAL PARENT FUNCTION sub_80B5
ROM:80C4
ROM:80C4 transfer_ucode:
ROM:80C4 ; sub_80B5+5tpj
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 DMAMADDRH, $IX0
ROM:80CD SR DMAMADDRL, $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:
ROM:80D5          CLR          $ACC1
ROM:80D6          LR           $AC1.M, DMALength
ROM:80D8          JMPR         $ARO
ROM:80D8 ; END OF FUNCTION CHUNK FOR rom_start

```

A.12 Command 0x10

```

IRAM:0BB1 command_10:
IRAM:0BB1          SET16
IRAM:0BB2
IRAM:0BB2 loc_BB2:
IRAM:0BB2          ; command_3+57tr
IRAM:0BB2          CLR          $ACCO
IRAM:0BB3
IRAM:0BB3 loc_BB3:
IRAM:0BB3          CLR'L        $ACC1 : $ACO.M, @ $ARO
IRAM:0BB4 ; DMA buffer 0x7c0 to main memory address from command stream
IRAM:0BB4          LRR1         $ACO.L, @ $ARO
IRAM:0BB5          SRS          DMAMADDRH, $ACO.M
IRAM:0BB6          SRS          DMAMADDRL, $ACO.L
IRAM:0BB7
IRAM:0BB7 DMEM_BB7:
IRAM:0BB7          ; sub_C50+4tr
IRAM:0BB7          SI           DMADSPADDR, 0x7C0
IRAM:0BB9          SI           DMAControl, 1
IRAM:0BBB          SI           DMALength, 0x500
IRAM:0BBD          CALL         wait_for_dma_finish_0
IRAM:0BBF ; DMA 0x20 bytes from main memory to buffer 0x7c0 from address in command stream
IRAM:0BBF          CLR          $ACCO
IRAM:0BC0          CLR'L        $ACC1 : $ACO.M, @ $ARO
IRAM:0BC1          LRR1         $ACO.L, @ $ARO
IRAM:0BC2          SRS          DMAMADDRH, $ACO.M
IRAM:0BC3          SRS          DMAMADDRL, $ACO.L
IRAM:0BC4          SI           DMADSPADDR, 0x7C0
IRAM:0BC6          SI           DMAControl, 0
IRAM:0BC8          CLR          $ACC1
IRAM:0BC9          LRIS         $AC1.L, 0x20
IRAM:0BCA          SRS          DMALength, $AC1.L
IRAM:0BCB ; mmaddr += 0x20, save in IX0
IRAM:0BCB          ADD          $ACCO, $ACC1
IRAM:0BCC          MRR          $IX0, $ARO
IRAM:0BCD ; ar0: buffer_7c0
IRAM:0BCD ; ar3, ar2: buffer_0
IRAM:0BCD          LRI          $ARO, 0x7C0
IRAM:0BCF          LRI          $AR3, 0
IRAM:0BD1          MRR          $AR2, $AR3
IRAM:0BD2          LRIS         $AX0.H, 0
IRAM:0BD3
IRAM:0BD3 cmd10_wait_for_dma_finish:
IRAM:0BD3          LRS          $AC1.M, DMAControl
IRAM:0BD4          ANDF         $AC1.M, 4
IRAM:0BD6          JLNZ         cmd10_wait_for_dma_finish
IRAM:0BD8 ; DMA another 0x20 bytes (same staggering as other commands)
IRAM:0BD8          SRS          DMAMADDRH, $ACO.M

```



```

IRAM:0BD9          SRS          DMAMMADDR, $ACO.L
IRAM:0BDA
IRAM:0BDA loc_BDA:
IRAM:0BDA          ; sub_C50:loc_C71+r ...
IRAM:0BDA          SI
IRAM:0BDC
IRAM:0BDC loc_BDC:
IRAM:0BDC          ; sub_C50+29+r ...
IRAM:0BDC          SI          DMAControl, 0
IRAM:0BDE          SI          DMALength, 0x4E0
IRAM:0BE0          SET40
IRAM:0BE1 ; load dwords from buffer_7c0
IRAM:0BE1 ; add them to words from buffer_0
IRAM:0BE1 ; store them to buffer_0
IRAM:0BE1          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BE2          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BE3          MOVAX          $ACCO, $AX1
IRAM:0BE4          ADDAX          $ACCO, $AX0
IRAM:0BE5 ; REPEAT 0x4F TIMES
IRAM:0BE5          BLOOPI          0x4F, loc_BEE
IRAM:0BE7          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BE8          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BE9          MOVAX' S          $ACC1, $AX1 : @ $AR2, $ACO.M
IRAM:0BEA          ADDAX' S          $ACC1, $AX0 : @ $AR2, $ACO.L
IRAM:0BEB          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BEC          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BED          MOVAX' S          $ACCO, $AX1 : @ $AR2, $AC1.M
IRAM:0BEE
IRAM:0BEE loc_BEE:
IRAM:0BEE          ADDAX' S          $ACCO, $AX0 : @ $AR2, $AC1.L
IRAM:0BEF ; BLOOPI END
IRAM:0BEF          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BF0          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BF1          MOVAX' S          $ACC1, $AX1 : @ $AR2, $ACO.M
IRAM:0BF2          ADDAX' S          $ACC1, $AX0 : @ $AR2, $ACO.L
IRAM:0BF3          SRRI          @ $AR2, $AC1.M
IRAM:0BF4          SRRI          @ $AR2, $AC1.L
IRAM:0BF5 ; same thing as above starts here, except theres a negative
IRAM:0BF5          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BF6          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BF7          MOVAX          $ACCO, $AX0
IRAM:0BF8          NEG          $ACCO
IRAM:0BF9          ADDAX          $ACCO, $AX1
IRAM:0BFA ; REPEAT 0x4F TIMES
IRAM:0BFA          BLOOPI          0x4F, loc_C05
IRAM:0BFC          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0BFD          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0BFE          MOVAX' S          $ACC1, $AX0 : @ $AR2, $ACO.M
IRAM:0BFF          NEG          $ACC1
IRAM:0C00          ADDAX' S          $ACC1, $AX1 : @ $AR2, $ACO.L
IRAM:0C01          NX'LD          $AX0.H : $AX1.H, @ $ARO
IRAM:0C02          NX'LD          $AX0.L : $AX1.L, @ $ARO
IRAM:0C03          MOVAX' S          $ACCO, $AX0 : @ $AR2, $AC1.M
IRAM:0C04          NEG          $ACCO
IRAM:0C05
IRAM:0C05 loc_C05:
IRAM:0C05          ADDAX' S          $ACCO, $AX1 : @ $AR2, $AC1.L
IRAM:0C06 ; BLOOPI END

```

IRAM:0C06	NX'LD	\$AX0.H : \$AX1.H, @ \$ARO
IRAM:0C07	NX'LD	\$AX0.L : \$AX1.L, @ \$ARO
IRAM:0C08	MOVAX'S	\$ACC1, \$AX0 : @ \$AR2, \$ACO.M
IRAM:0C09	NEG	\$ACC1
IRAM:0C0A	ADDAX'S	\$ACC1, \$AX1 : @ \$AR2, \$ACO.L
IRAM:0C0B	SRRI	@ \$AR2, \$AC1.M
IRAM:0C0C	SRRI	@ \$AR2, \$AC1.L
IRAM:0C0D	MRR	\$ARO, \$IX0
IRAM:0C0E	JMP	receive_command

A.13 Command 0x11

IRAM:0175	command_11:	
IRAM:0175	CLR	\$ACCO
IRAM:0176	CLR'L	\$ACC1 : \$ACO.M, @ \$ARO
IRAM:0177	SET16'L	\$ACO.L : @ \$ARO
IRAM:0178	<i>; DMA 0x20 bytes to 0xe44 from main memory address from command stream</i>	
IRAM:0178	SRS	DMAMADDRH, \$ACO.M
IRAM:0179	SRS	DMAMADDRL, \$ACO.L
IRAM:017A	SI	DMADSPADDR, 0xE44
IRAM:017C	SI	DMAControl, 0
IRAM:017E	CLR	\$ACC1
IRAM:017F	LRIS	\$AC1.L, 0x20
IRAM:0180	SRS	DMALength, \$AC1.L
IRAM:0181	<i>; ac0 = mmaddr + 0x20</i>	
IRAM:0181	ADD	\$ACCO, \$ACC1
IRAM:0182	<i>; save command stream pointer</i>	
IRAM:0182	MRR	\$IX0, \$ARO
IRAM:0183	<i>; ar0: dest280 = 0x280</i>	
IRAM:0183	<i>; ar1 = 0</i>	
IRAM:0183	<i>; ar2: dest140 = 0x140</i>	
IRAM:0183	<i>; ar3: srce44 = 0xe44</i>	
IRAM:0183	LRI	\$ARO, 0x280
IRAM:0185	LRI	\$AR1, 0
IRAM:0187	LRI	\$AR2, 0x140
IRAM:0189	LRI	\$AR3, 0xE44
IRAM:018B	LRIS	\$AX0.H, 0
IRAM:018C		
IRAM:018C	<i>; cmd11_wait_for_dma_finish:</i>	
IRAM:018C	LRS	\$AC1.M, DMAControl
IRAM:018D	ANDF	\$AC1.M, 4
IRAM:018F	JLNZ	cmd11_wait_for_dma_finish
IRAM:0191	<i>; DMA 0x260 bytes from mmaddr + 0x20 to DSP DMEM 0xe54 (contiguous)</i>	
IRAM:0191	SRS	DMAMADDRH, \$ACO.M
IRAM:0192	SRS	DMAMADDRL, \$ACO.L
IRAM:0193	SI	DMADSPADDR, 0xE54
IRAM:0195	SI	DMAControl, 0
IRAM:0197	SI	DMALength, 0x260
IRAM:0199	LRI	\$AC1.M, 0xA0
IRAM:019B	SET40	
IRAM:019C	<i>; REPEAT 0xa0 = 160 TIMES</i>	
IRAM:019C	BLOOP	\$AC1.M, loc_1A5
IRAM:019E	<i>; ac0.ml = *(u32*)srce44++</i>	
IRAM:019E	LRRI	\$ACO.M, @ \$AR3
IRAM:019F	<i>; *dest280++ = 0</i>	
IRAM:019F	SRRI	@ \$ARO, \$AX0.H
IRAM:01A0	LRRI	\$ACO.L, @ \$AR3

```

IRAM:01A1 ; *dest280++ = 0
IRAM:01A1          SRRI          @ $AR0, $AX0.H
IRAM:01A2 ; *dest140++ = ac0.ml
IRAM:01A2          SRRI          @ $AR2, $AC0.M
IRAM:01A3          NEG 'S        $ACCO : @ $AR2, $AC0.L
IRAM:01A4 ; *dest0++ = -ac0.ml
IRAM:01A4          SRRI          @ $AR1, $AC0.M
IRAM:01A5
IRAM:01A5 loc_1A5:
IRAM:01A5          SRRI          @ $AR1, $AC0.L
IRAM:01A6 ; BLOOP END
IRAM:01A6
IRAM:01A6 ; restore command stream pointer
IRAM:01A6          MRR           $AR0, $IX0
IRAM:01A7          JMP           receive_command

```