Hacking a Sega Whitestar Pinball: Focusing on the audio board

Grehack 2015

Pierre Surply

EPITA Systems/Security Laboratory (LSE)

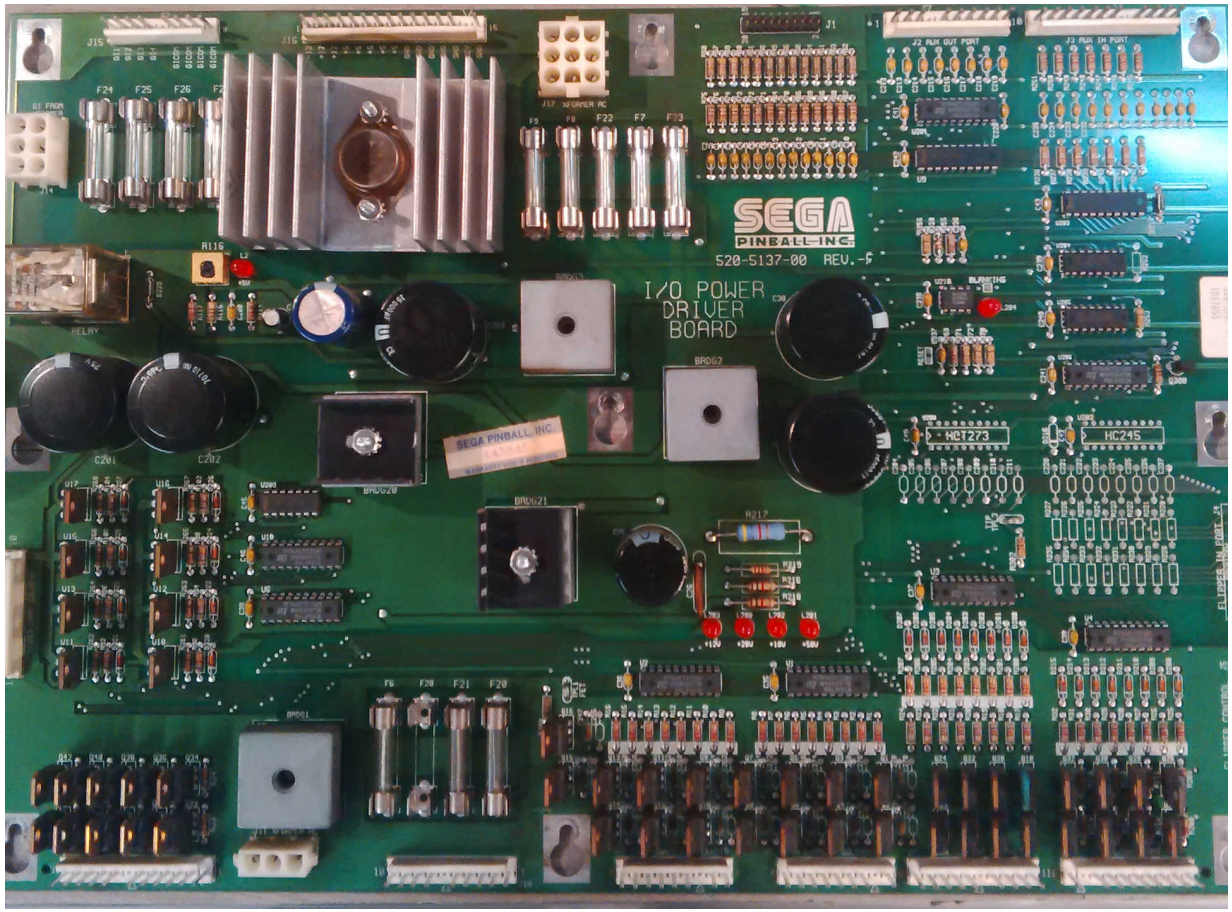
Nov 20, 2015

Sega Whitestar Pinball Overview

Playfield

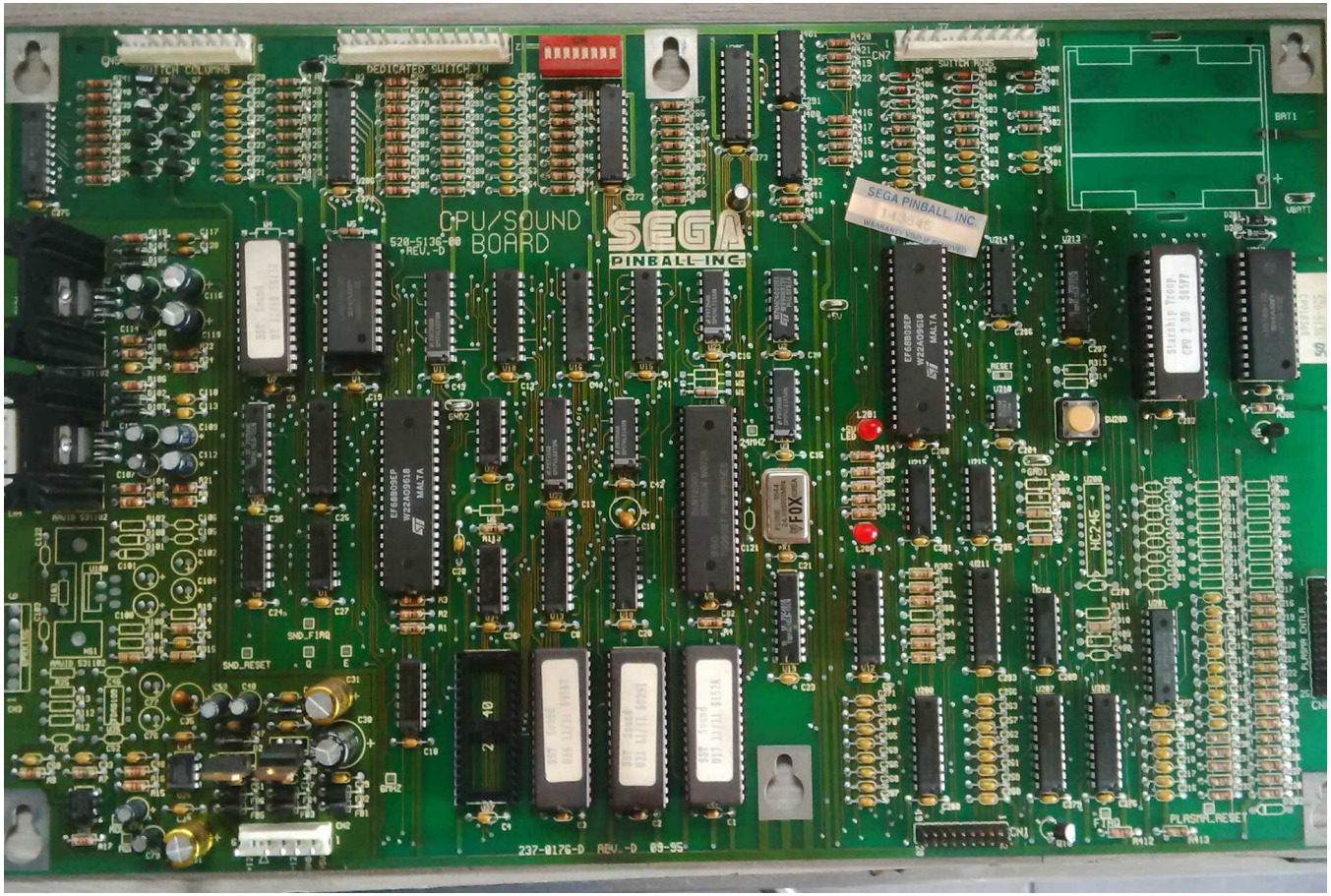

NUKE!
NUKE!
STARSHIP
KLENDATHU
ZEGEMA BEACH
DANTANA
TANGO GRILLA
ANTILLES
KNOW YOUR FOE!
PLANET P
WARRIOR
HOPPER
PLASMA
TANKER
SHOOT TARGETS TO KILL BUGS AND CLEAN PLANE
BALL SAVER

IO Board

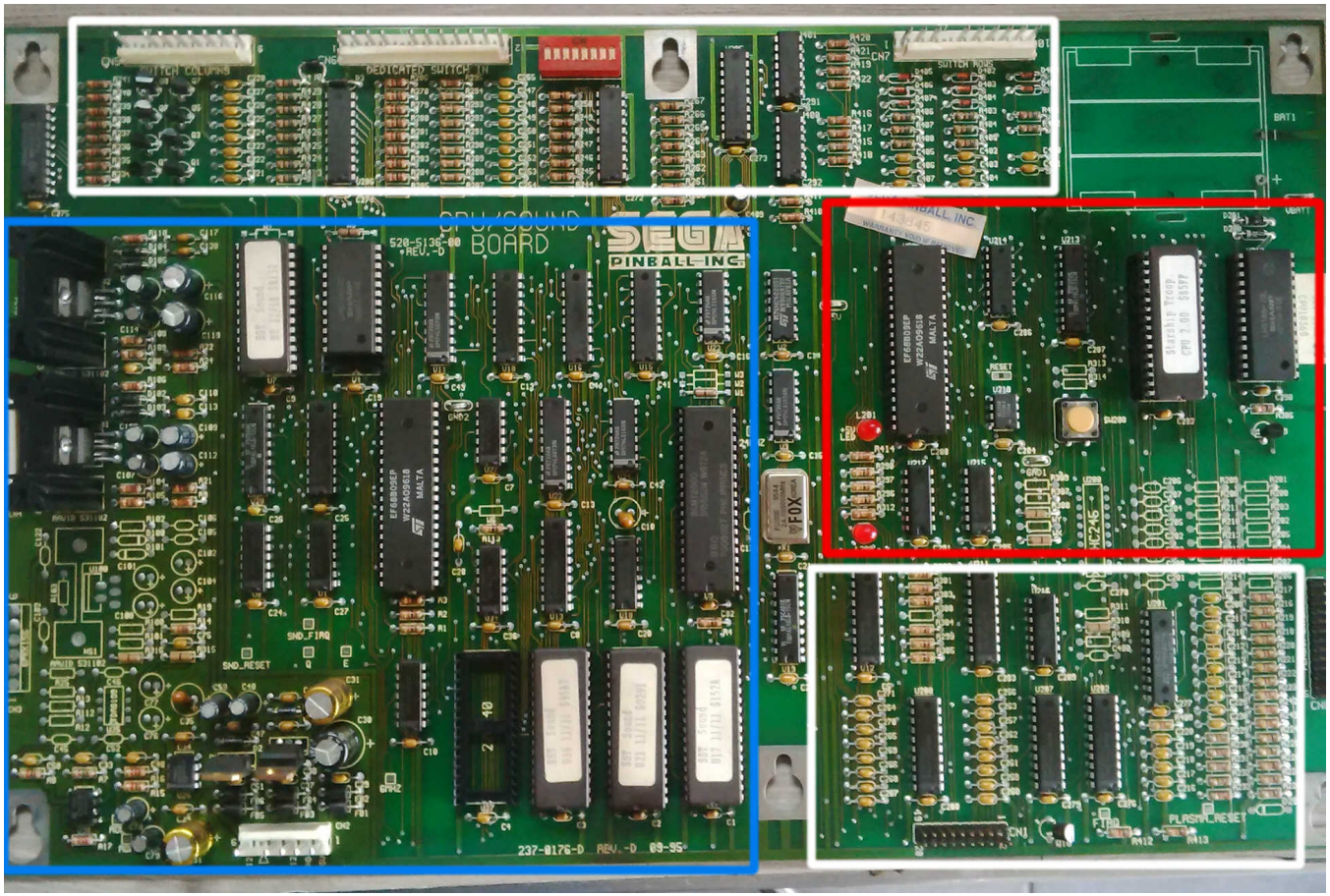


LSE Security System

CPU/Sound Board

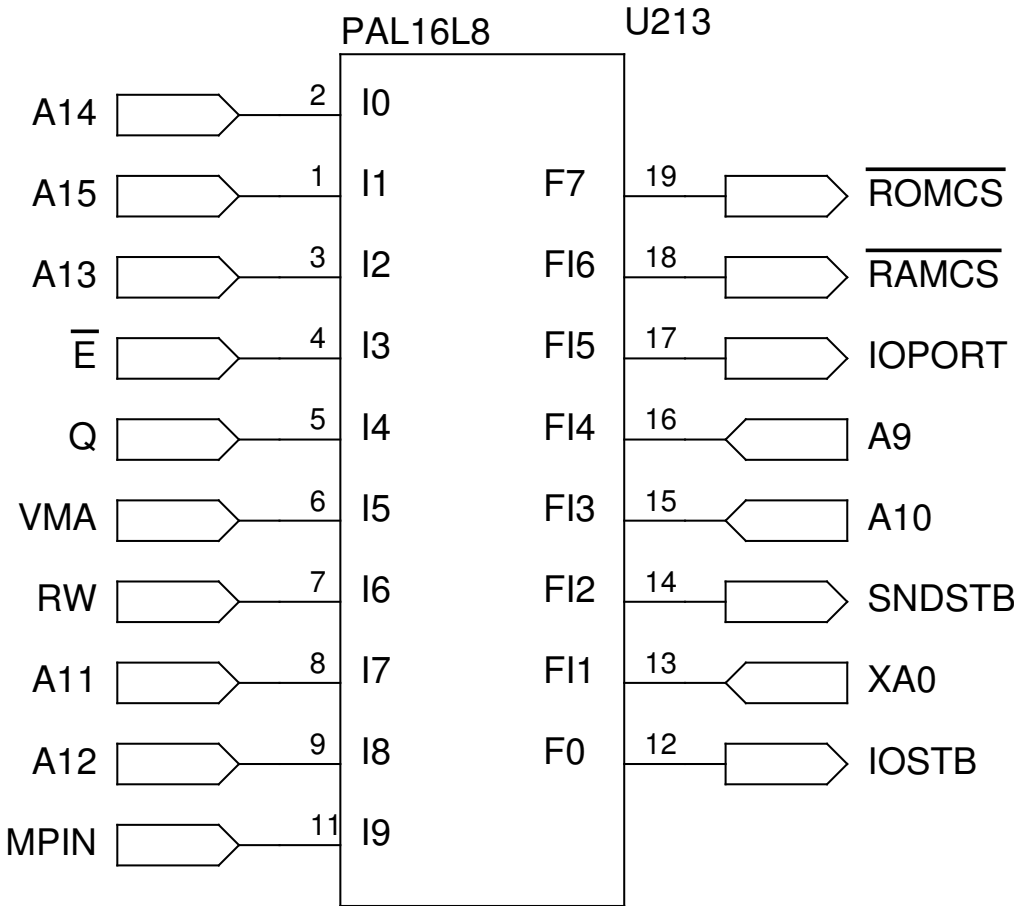


CPU/Sound Board

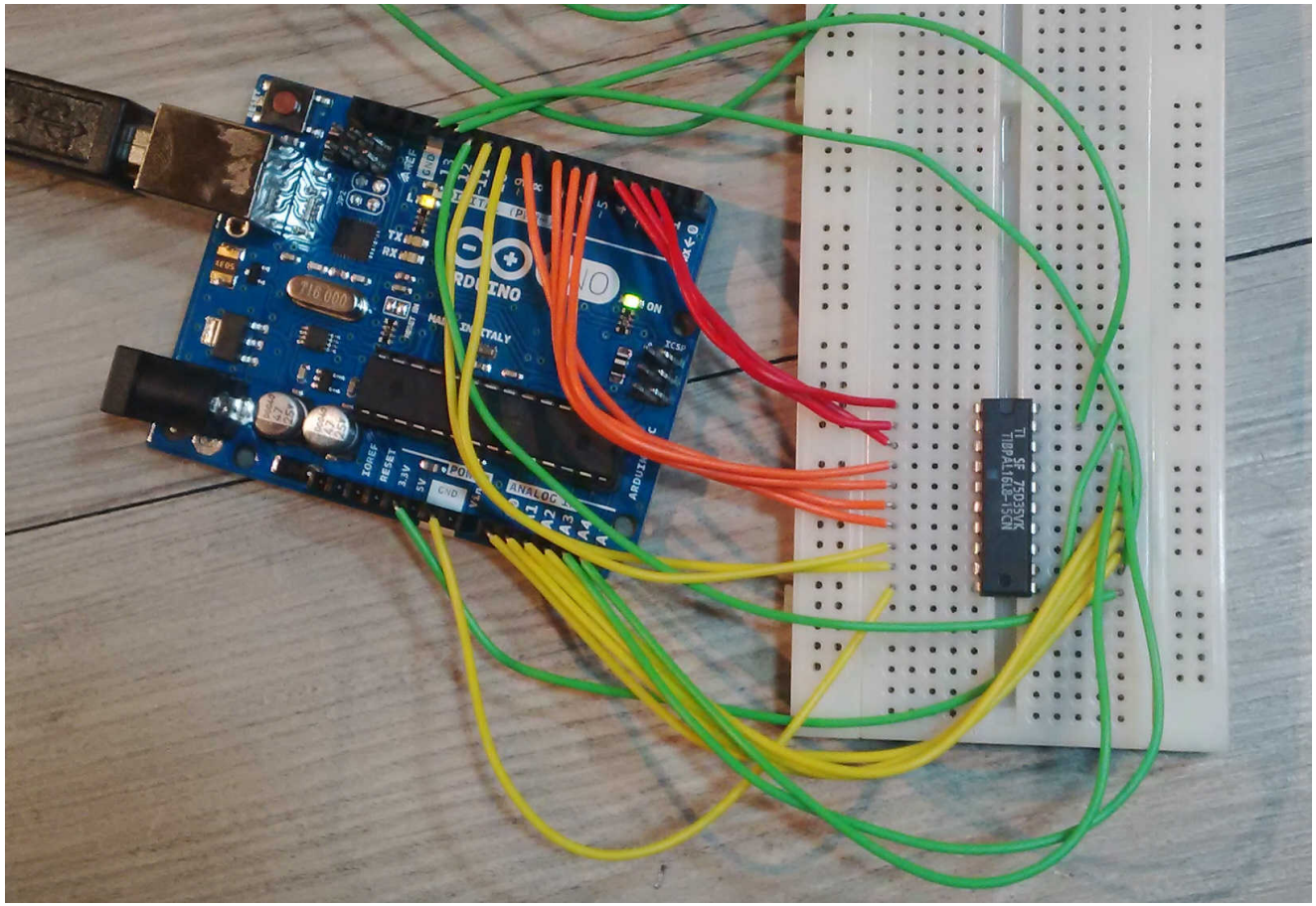


Main CPU

Main CPU Address Space



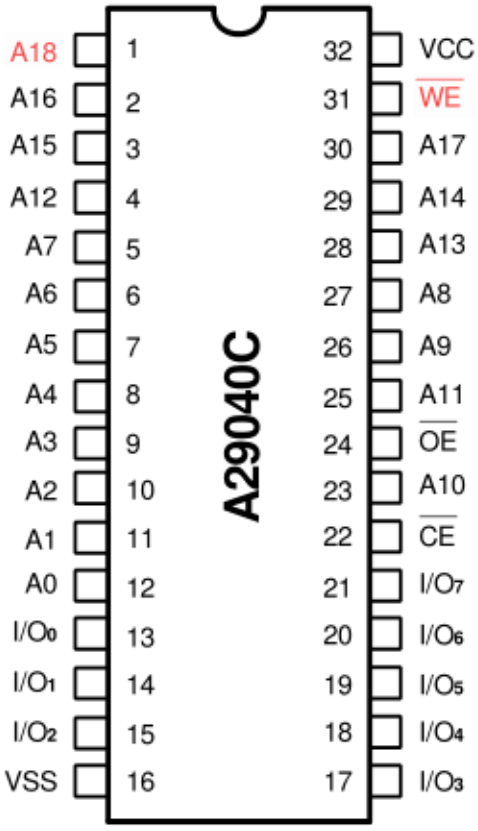
Main CPU Address Space

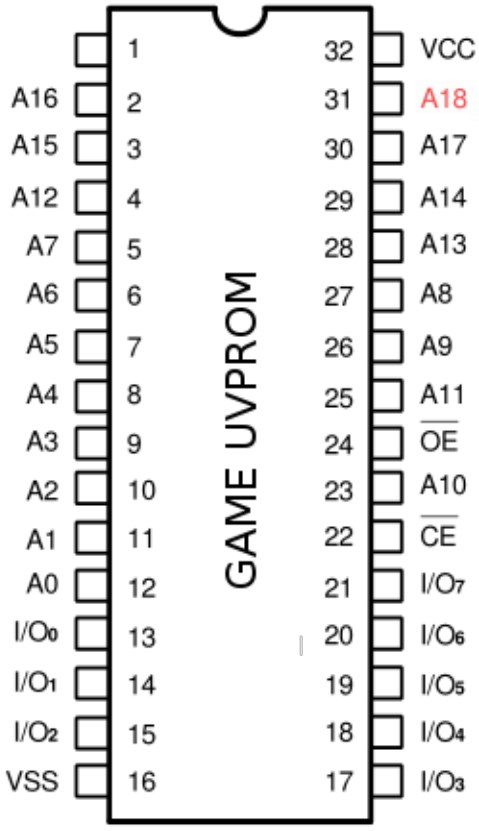


Main CPU Address Space

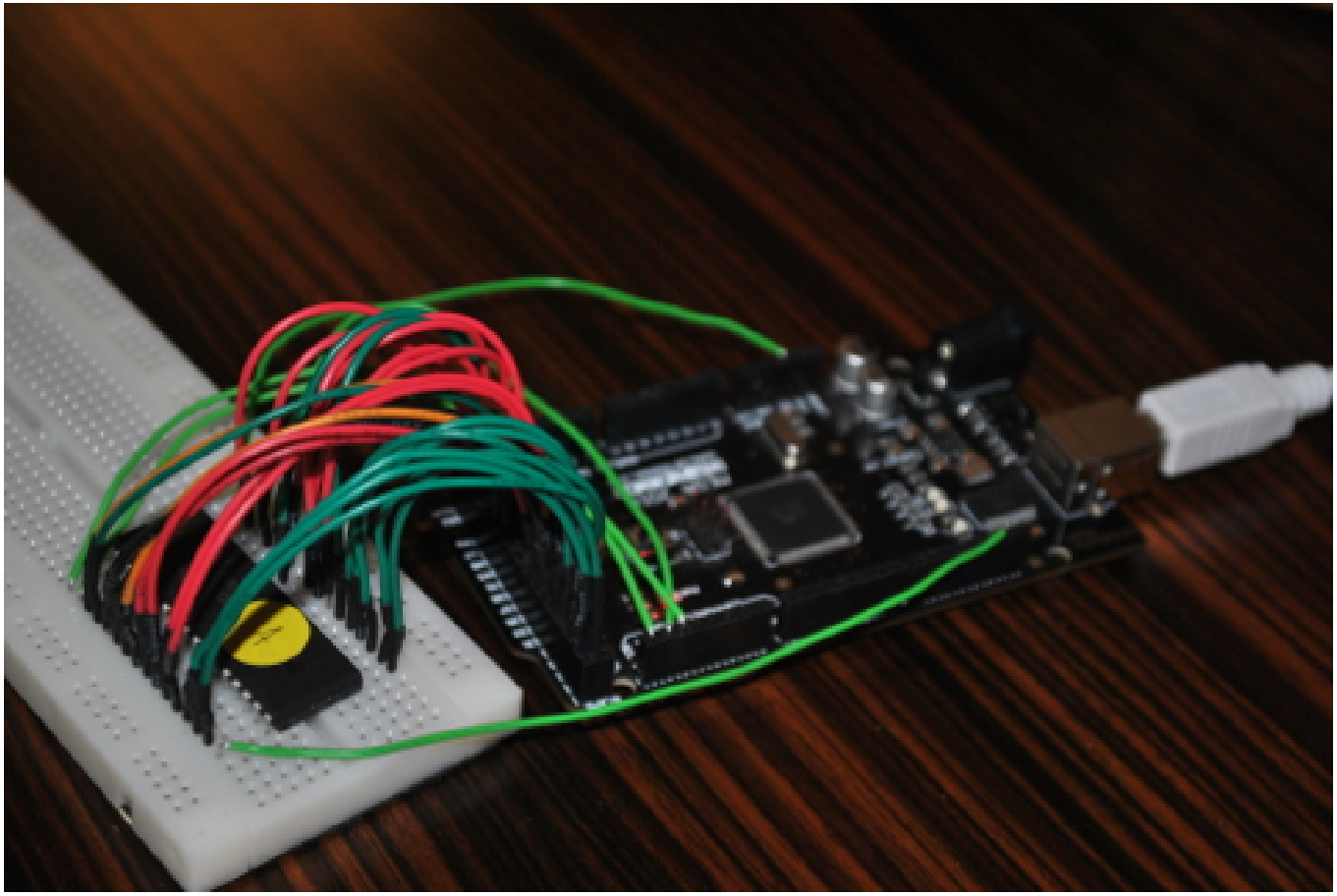
```
GAL16V8
U213


A15  A14   A13 /E     Q   VMA RW     A11    A12    GND
MPIN IOSTB XA0 SNDSTB A10 A9  IOPORT /RAMCS /ROMCS VCC

/ROMCS.T = A15 + A14 + IOPORT
/ROMCS.E = /E
RAMCS.T = A15 + A14 + A13 + A12 * A11 * A10 * A9 * /RW * /MPIN
/RAMCS.E = /E
IOPORT.T = A15 + A14 + /A13 + A12 + A11 + XA0
IOPORT.E = /E
IOSTB.T = /A15 * /A14 * A13 * /A11
IOSTB.E = /E

DESCRIPTION:
    Sega Whitestar Pinball
    U213 (Address space decoding)
    (Extracted using Quine-McCluskey method)
```

Replacing ROM





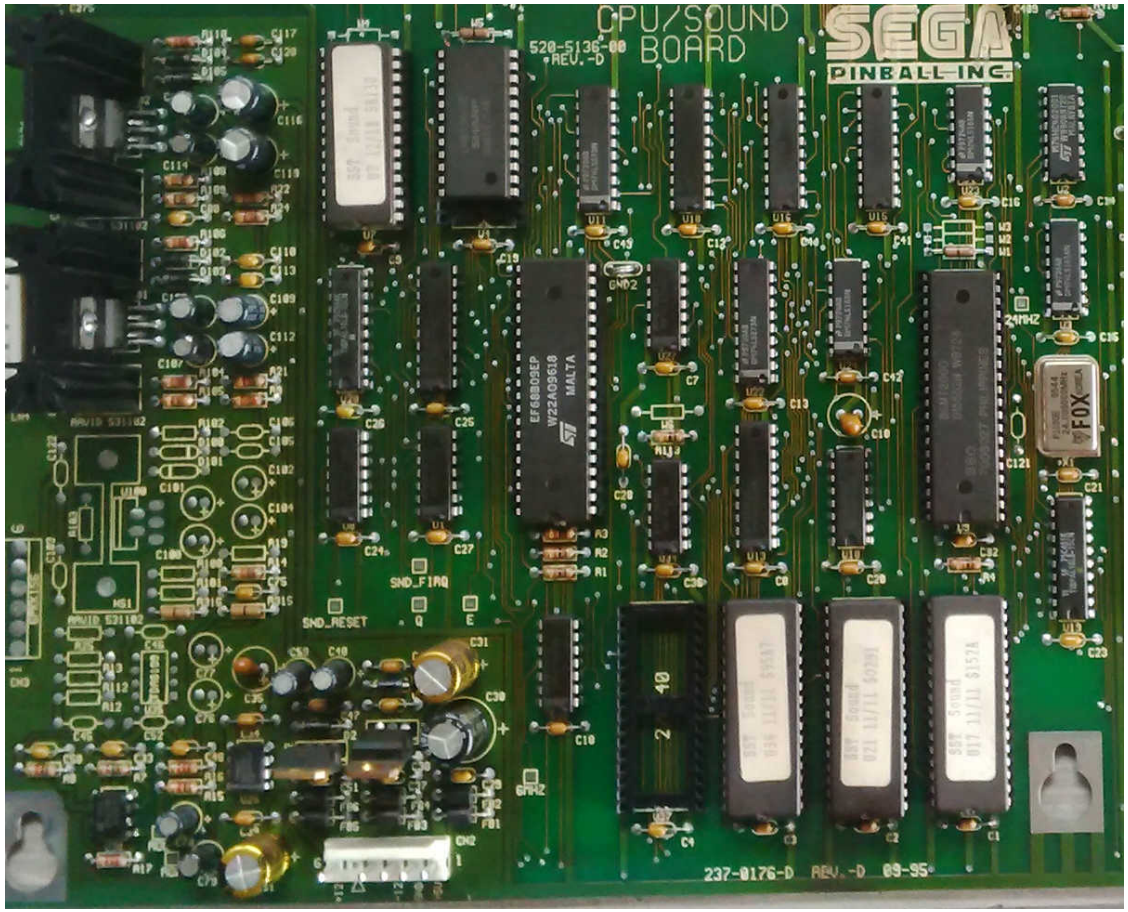
Replacing ROM



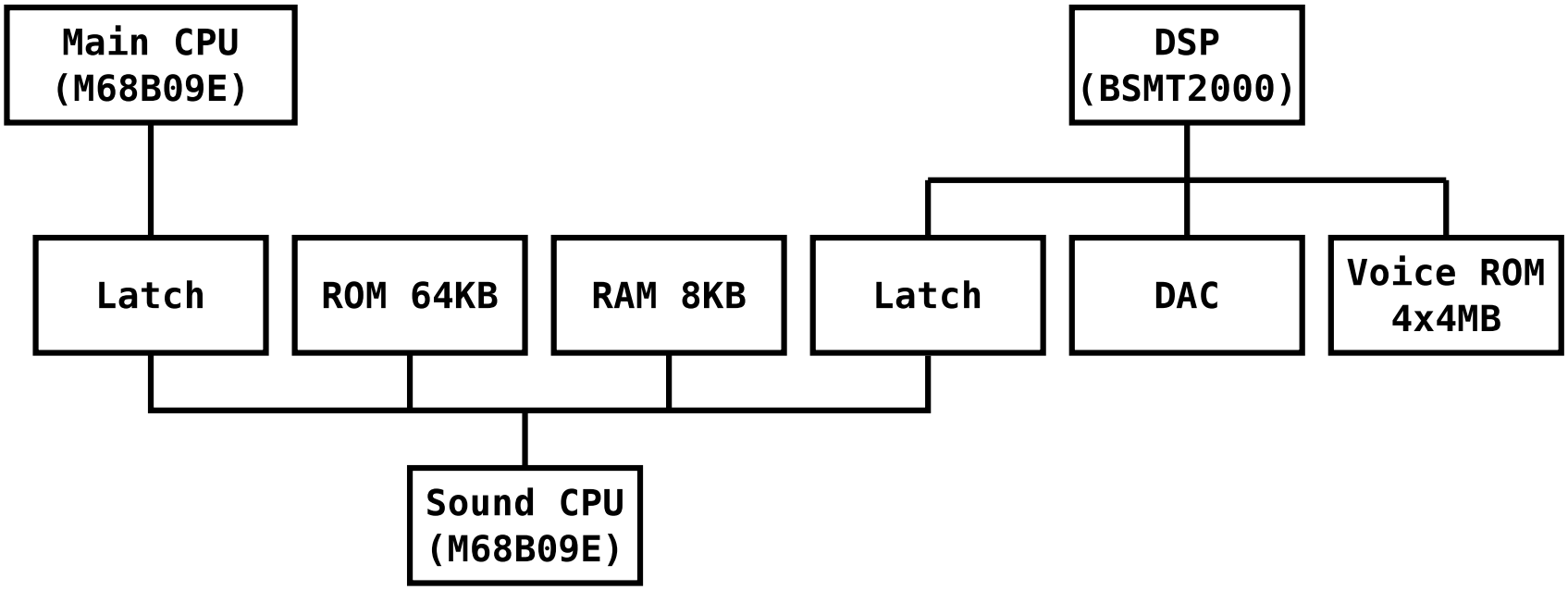
Main CPU Programming

Sound Board

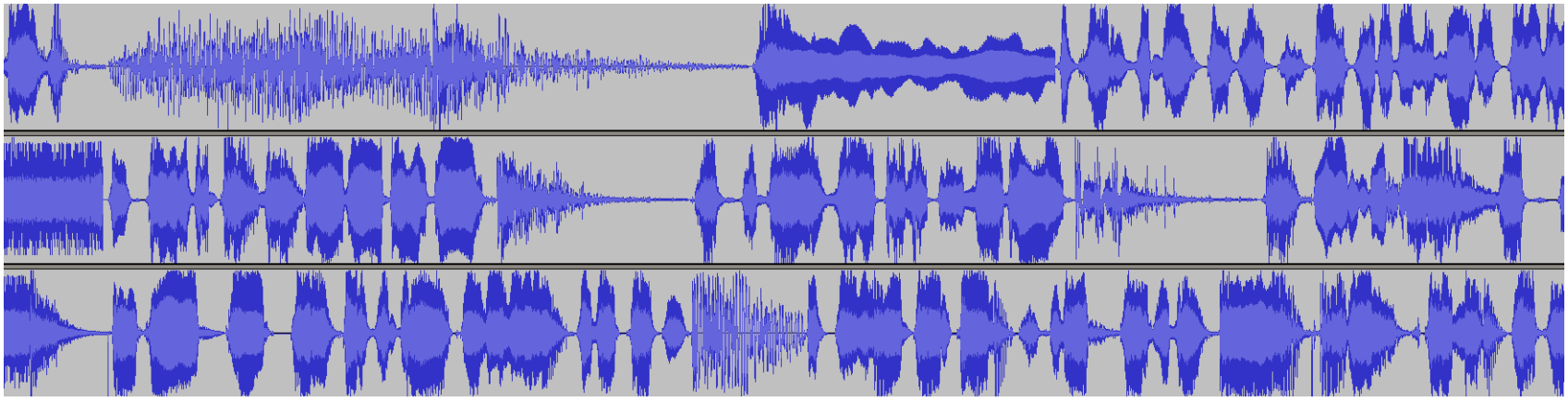
Sound Board


CPU/SOUND
BOARD
520-5136-00
REV.-D
SEGA
PINBALL-INC.
SND_FIRQ
SND_RESET
24MHZ
237-0176-D REV.-D 09-95

Sound Board Block Diagram



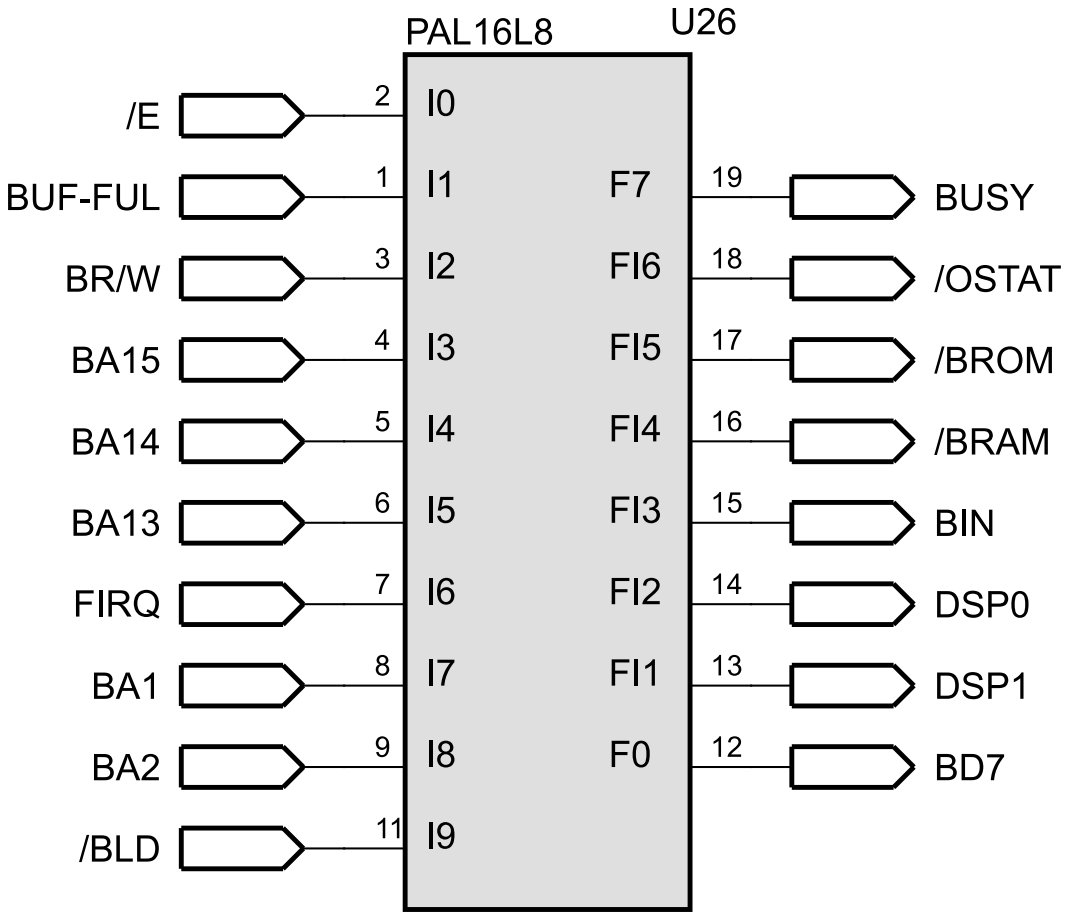
Voices EEPROM



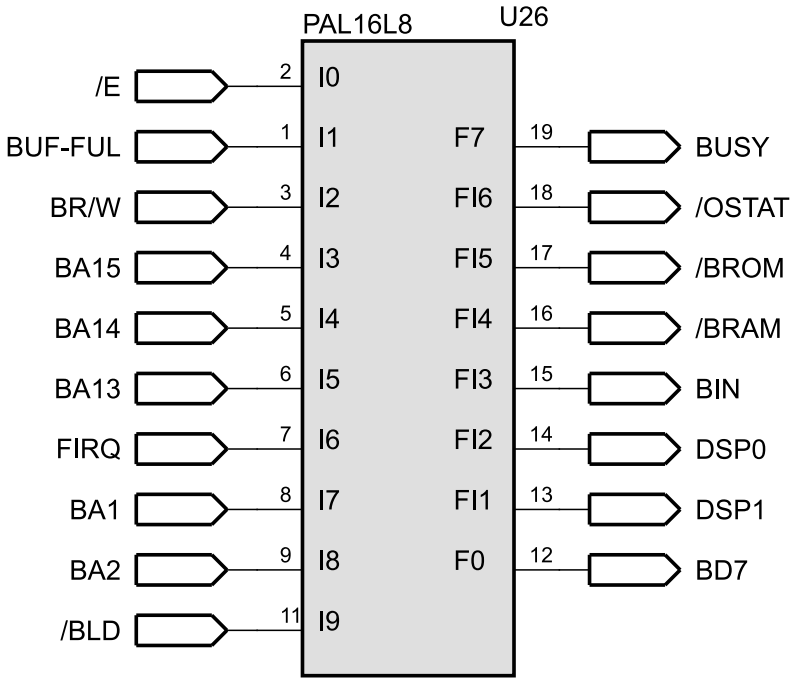
- 8-bit PCM @ 8kHz

Sound CPU

Sound CPU Address Space

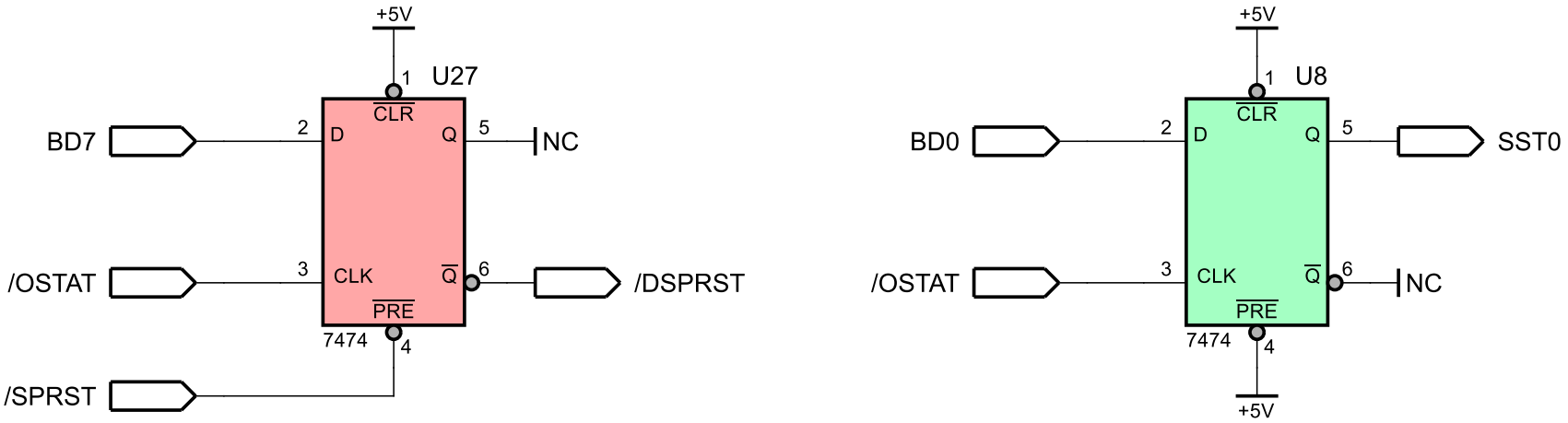


Sound CPU Address Space



- `0x0000-0x1FFF`: RAM
- `0x2000`: Status Register (`OSTAT` signal)
- `0x2002`: Main CPU / Sound CPU Command Register (`BIN` signal)
- `0x2006`: DSP Status (`BLD` signal)
- During *read* operation:
  - `0x4000-0xFFFF`: ROM
- During *write* operation:
  - `0x6000`: DSP Command (MSB)
  - `0xA000-0xA0FF`: DSP Command (LSB)

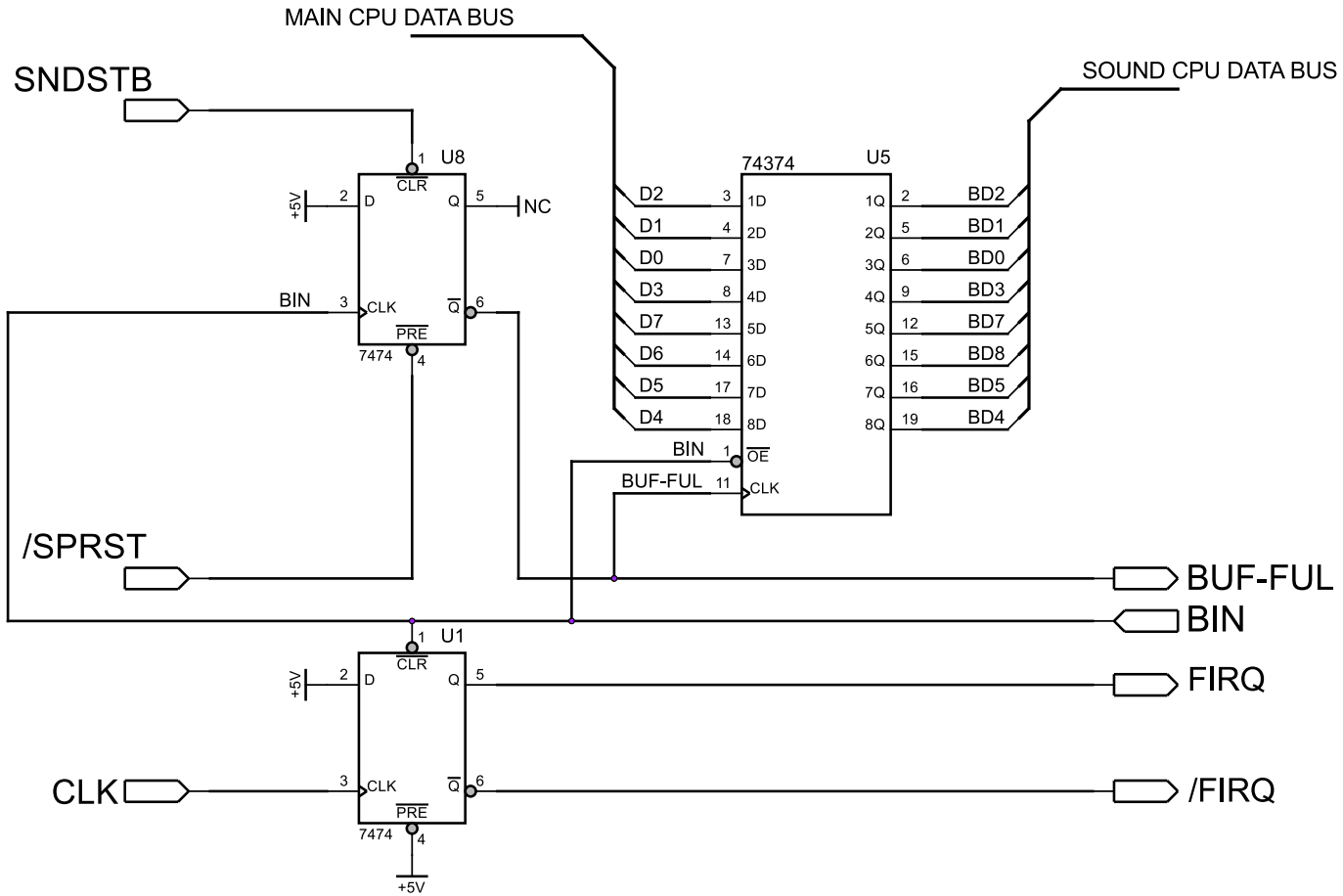
Sound CPU Wiring




```
lda     #$80
sta     IO_STATUS ;; Reset DSP

cla
anda    #1
sta     IO_STATUS ;; Indicate to Main CPU that audio card
                  ;; is ready
```

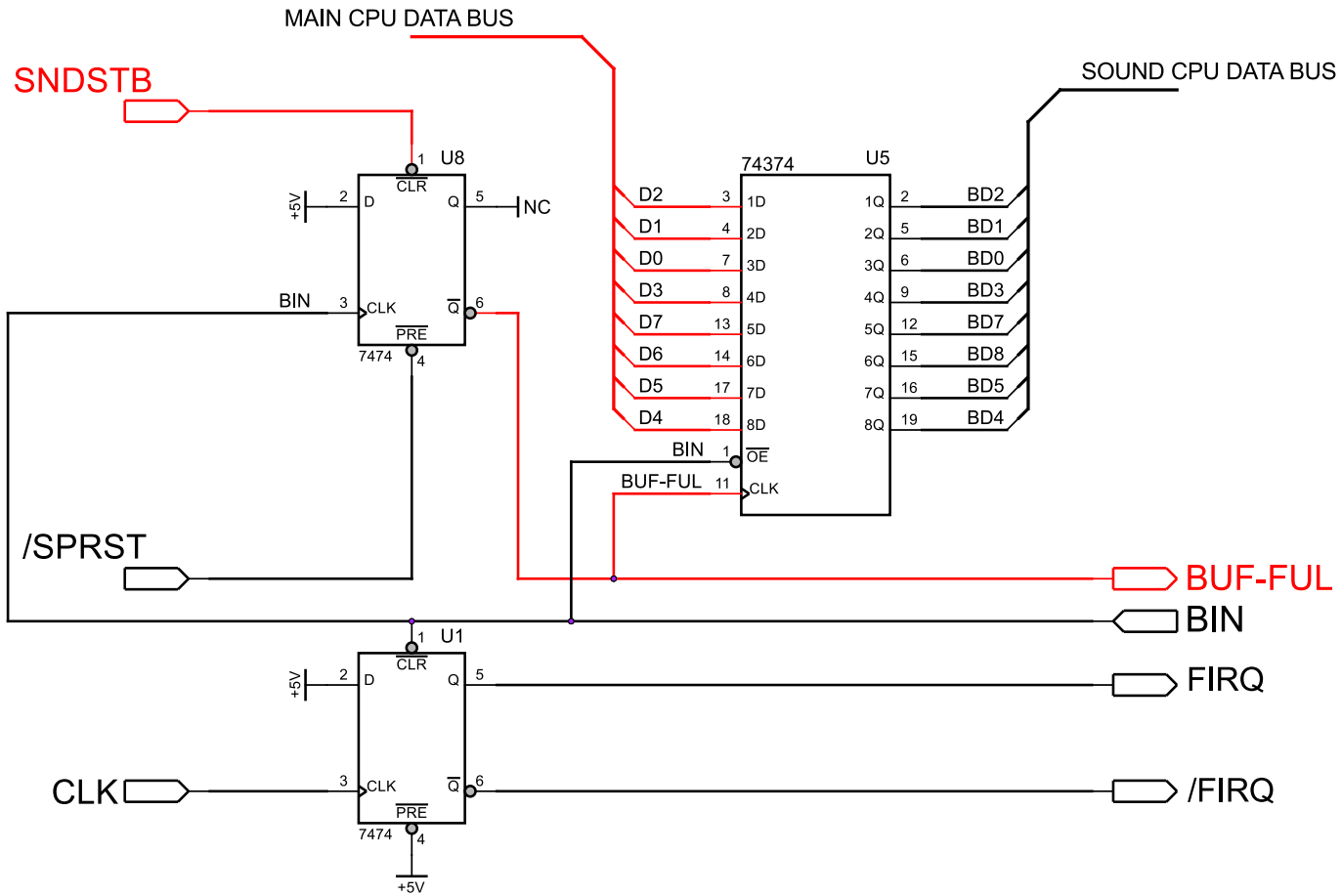
Main CPU / Sound CPU Interface

- Main CPU Command (8bit)

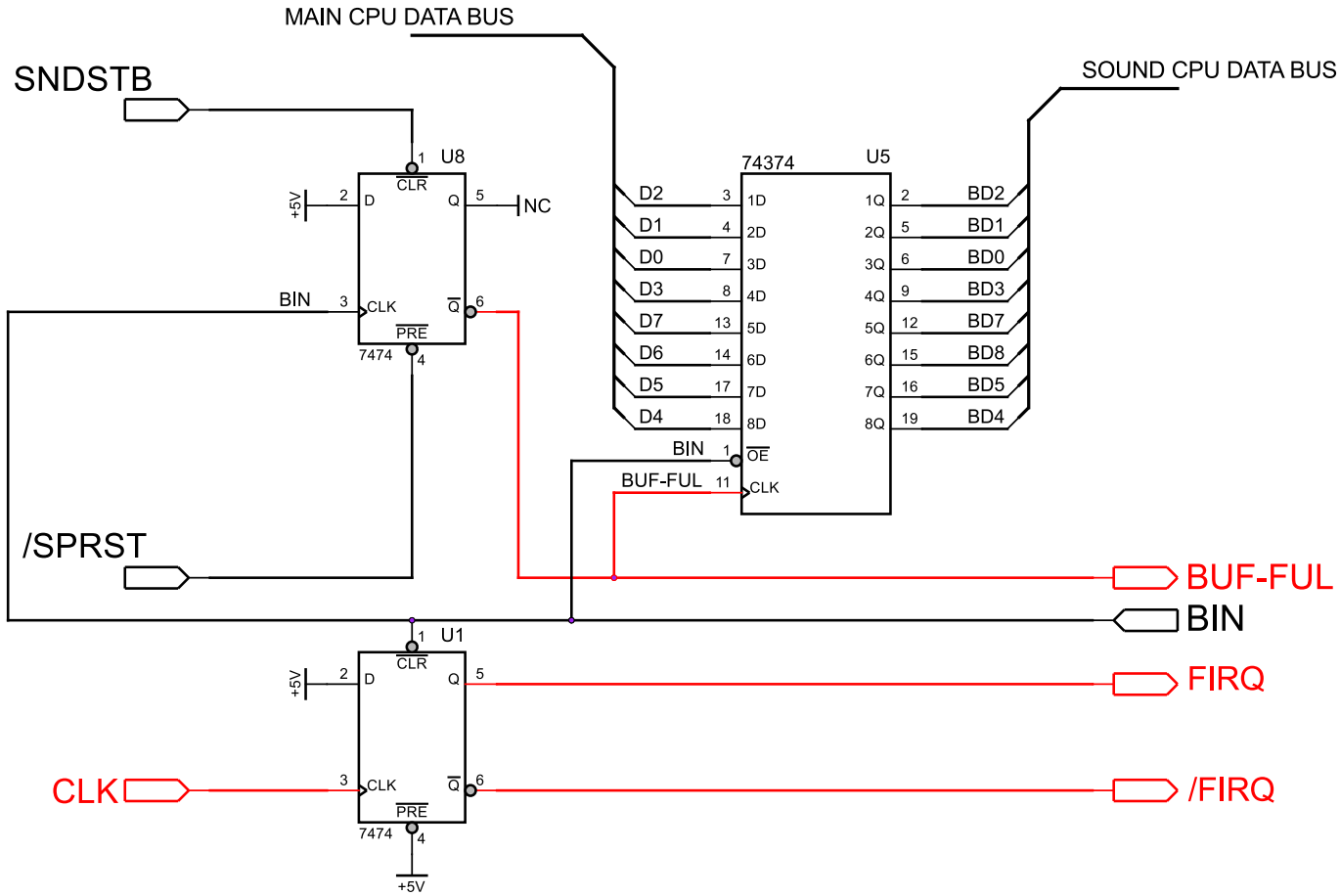


Main CPU / Sound CPU Interface

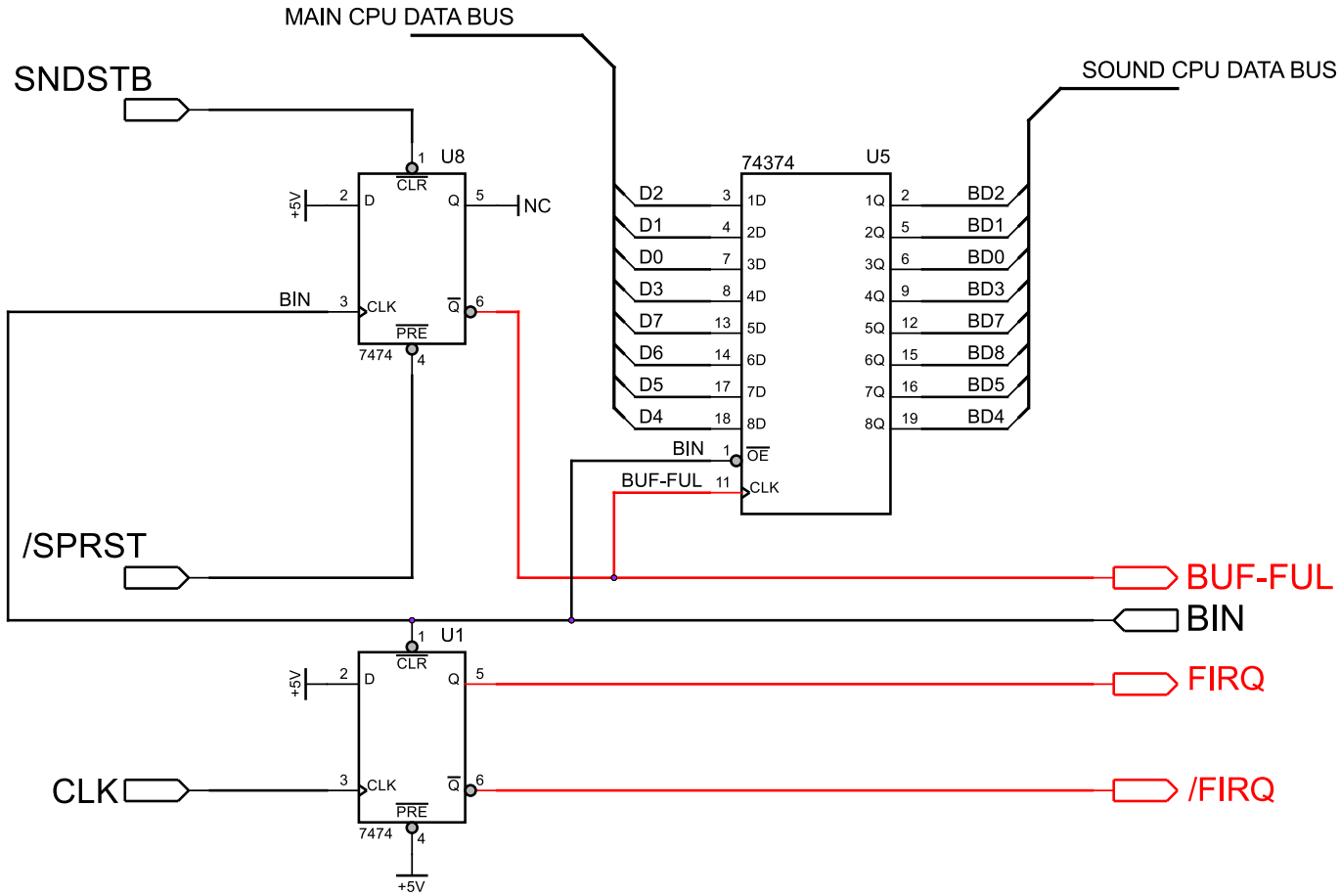
```
lda #CMD                ;; a <- CMD
sta SNDSTB              ;; [SNDSTB] <- a
```



Main CPU / Sound CPU Interface

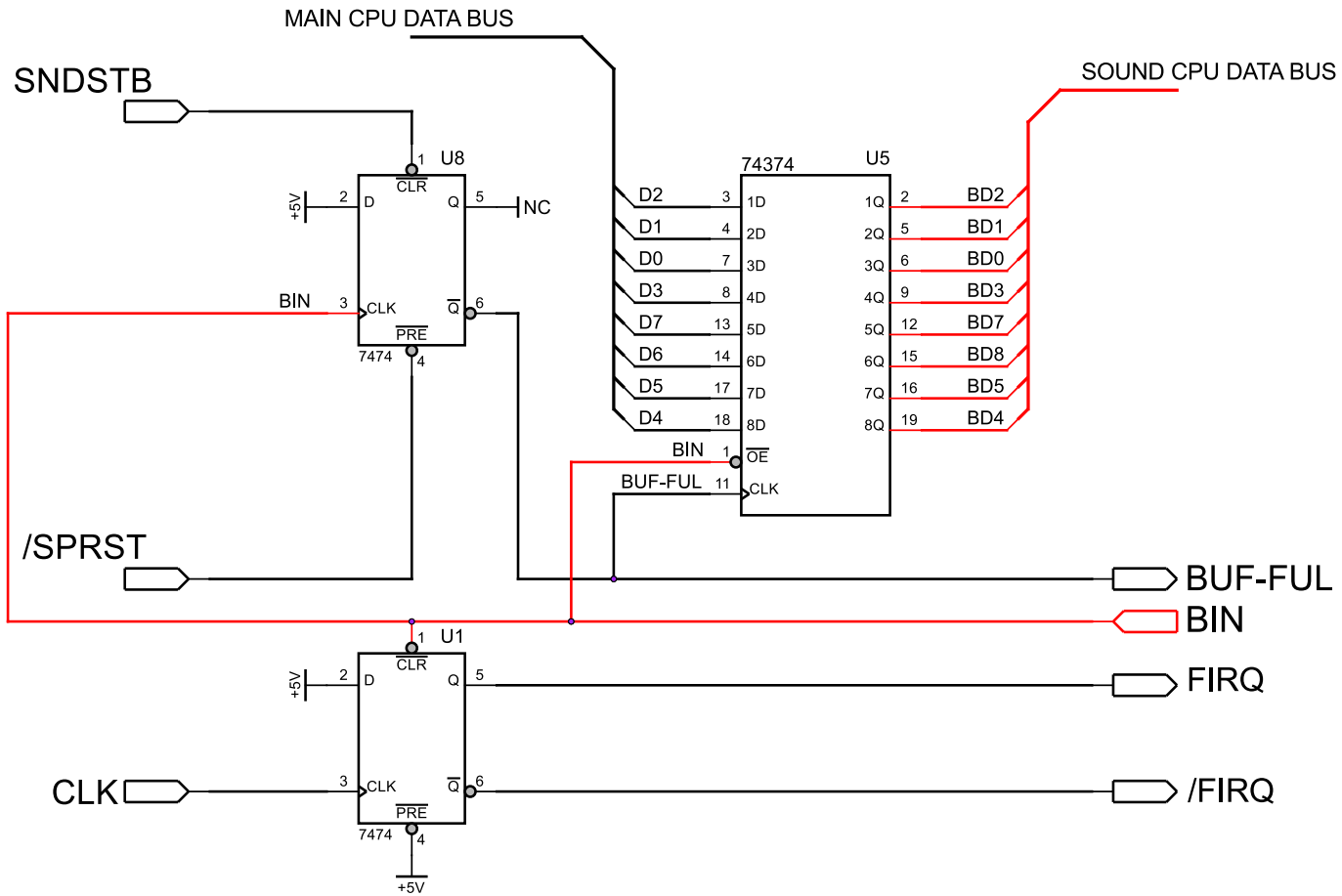


Main CPU / Sound CPU Interface



Main CPU / Sound CPU Interface

```
lda BIN                    ;; a <- [BIN]
```

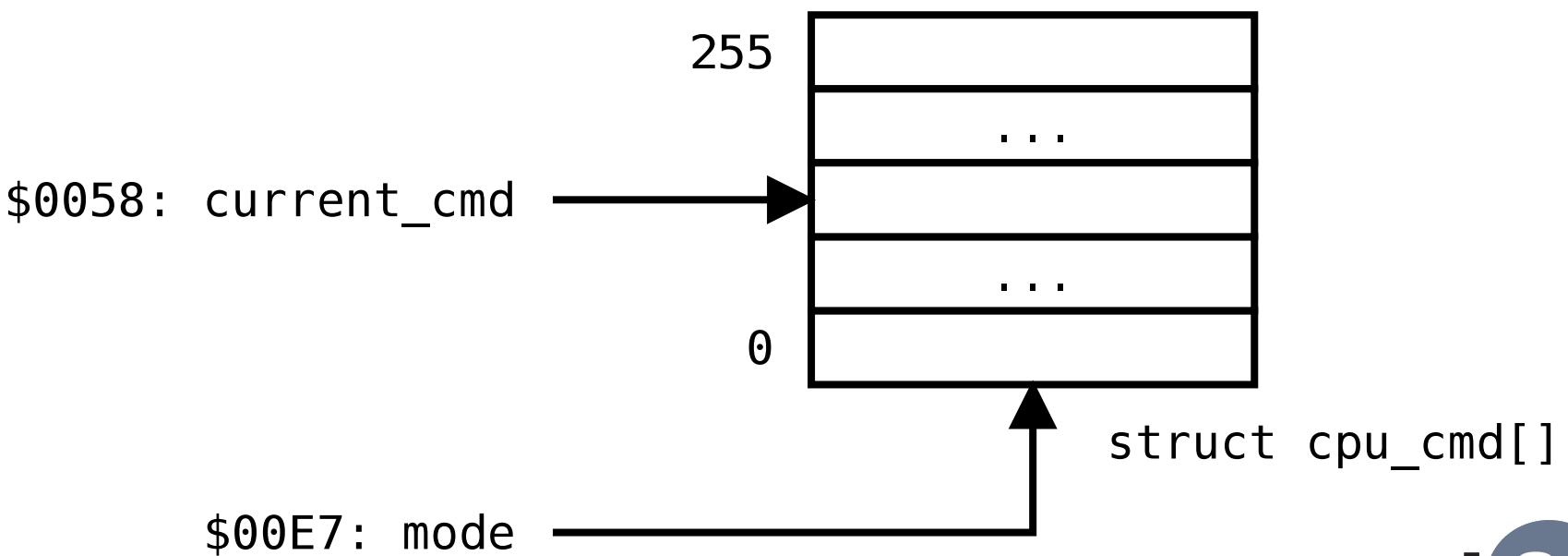


FIRQ Handler

```
struct cmd_ring_buffer {
    uint8_t begin;
    uint8_t end;
    uint8_t data[16];
};
```

Main Loop

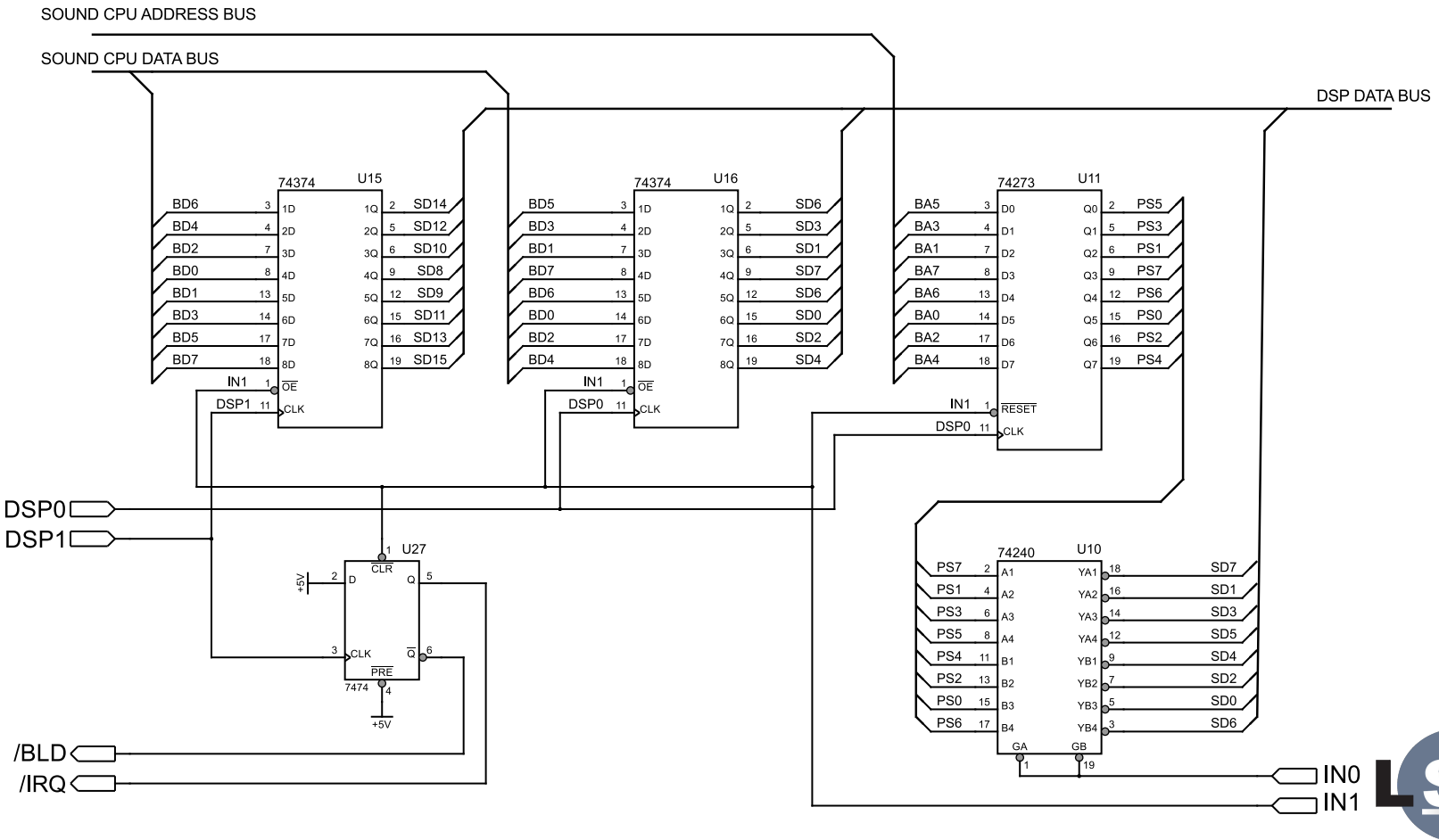
```
struct cpu_cmd {
    uint8_t     callback_idx;
    uint8_t     unk0;
    uint16_t    mask;
    void        **data;
};
```



Digital Signal Processor

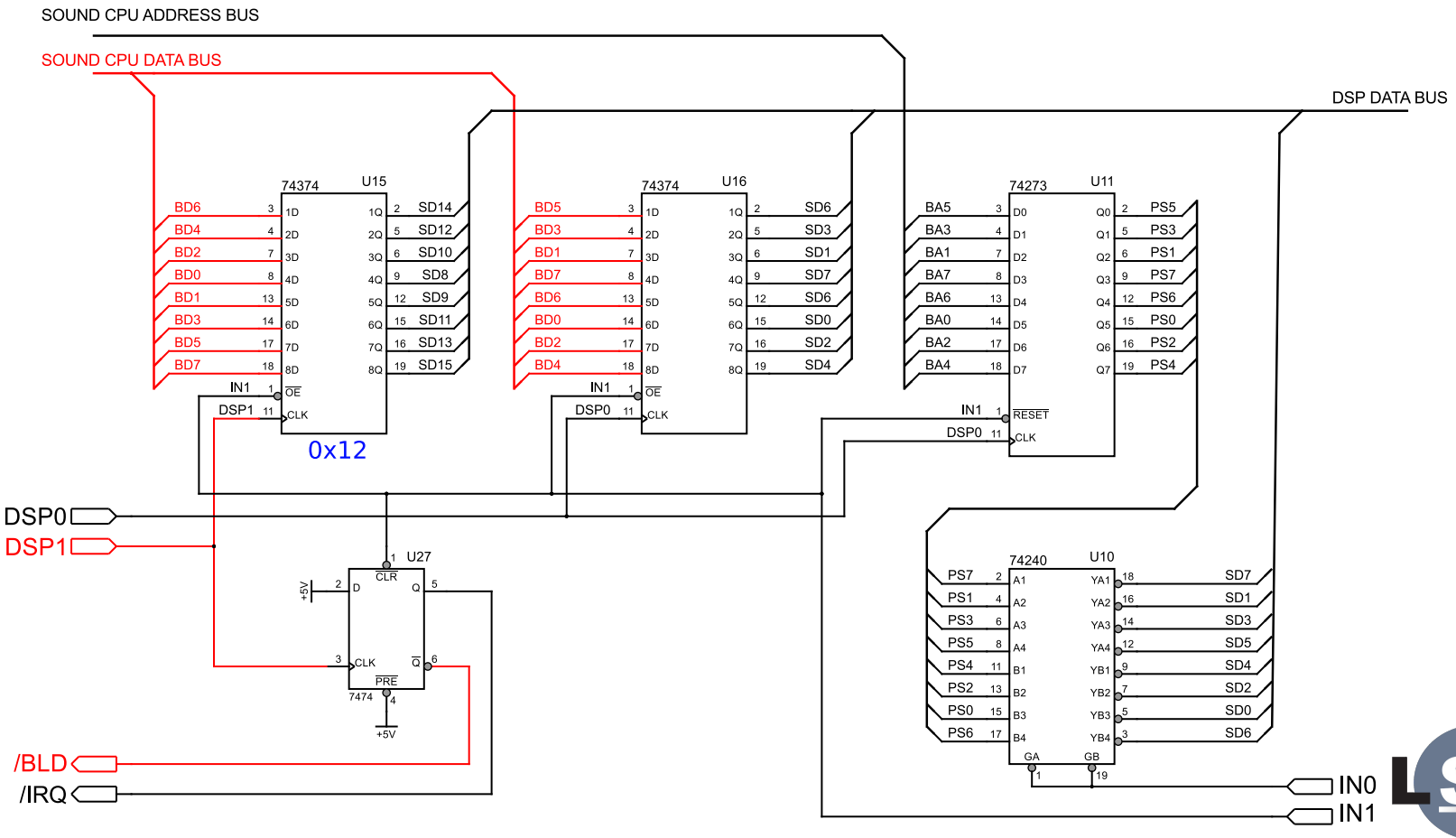
Sound CPU / DSP Interface

- Data (16bit): `0x1234`
- Address (8bit): `0x56`



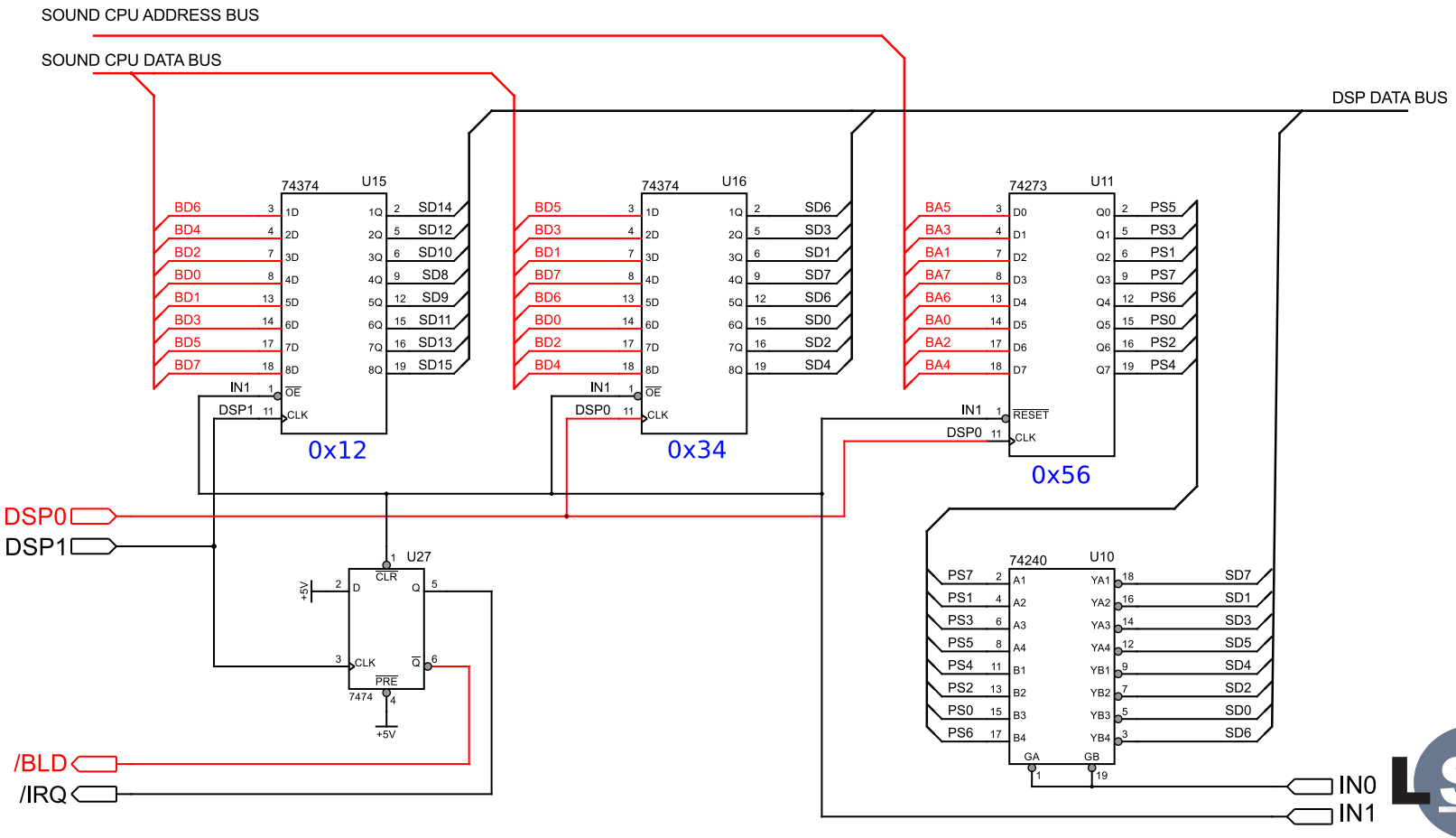
Sound CPU / DSP Interface

```
lda #$12                ;; a <- $12
sta DSP1                ;; [DSP1] <- a
```



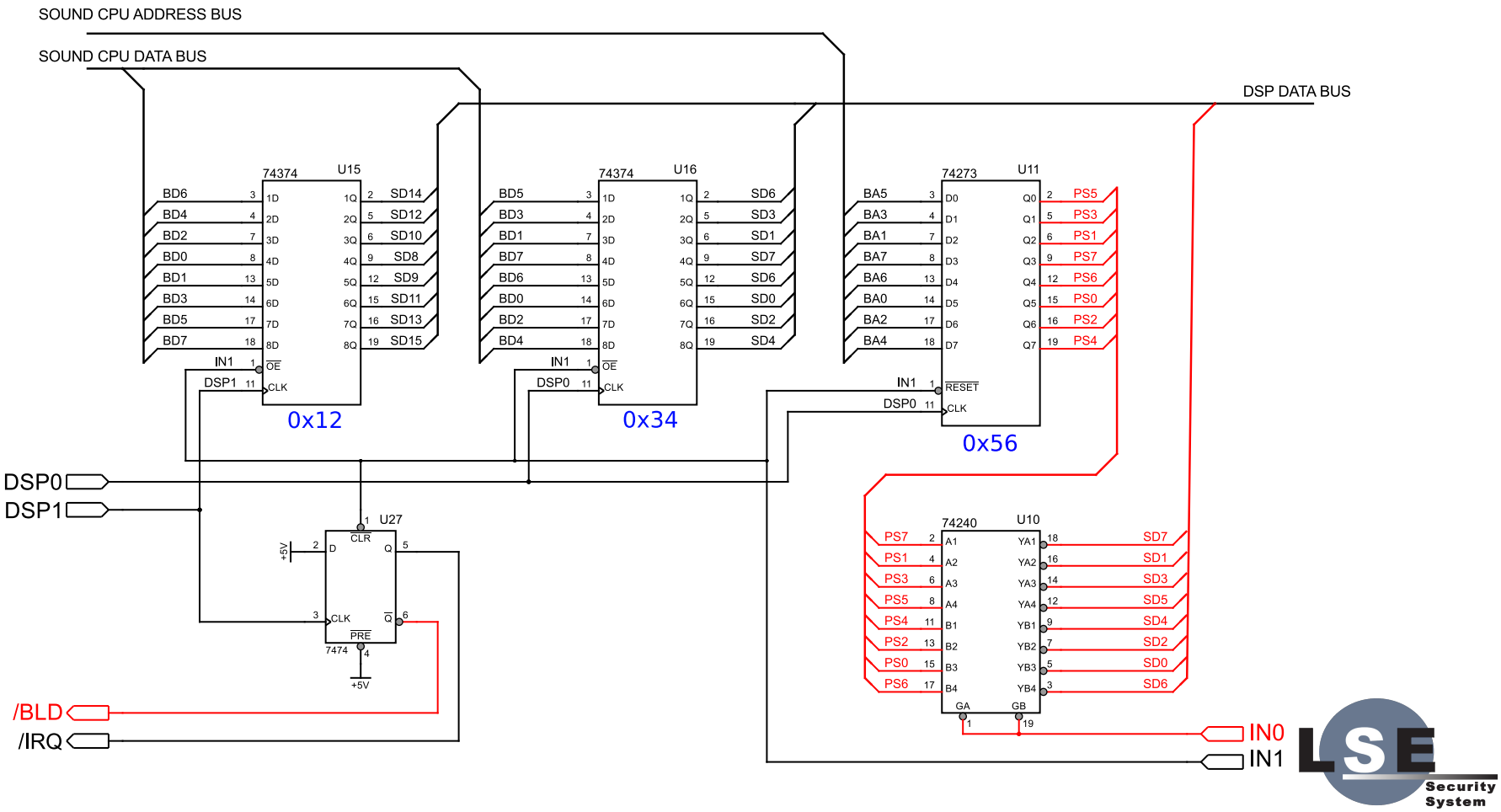
Sound CPU / DSP Interface

```
lda #$34                ;; a <- $34
sta DSP0 + $56          ;; [DSP0 + $56] <- a
```



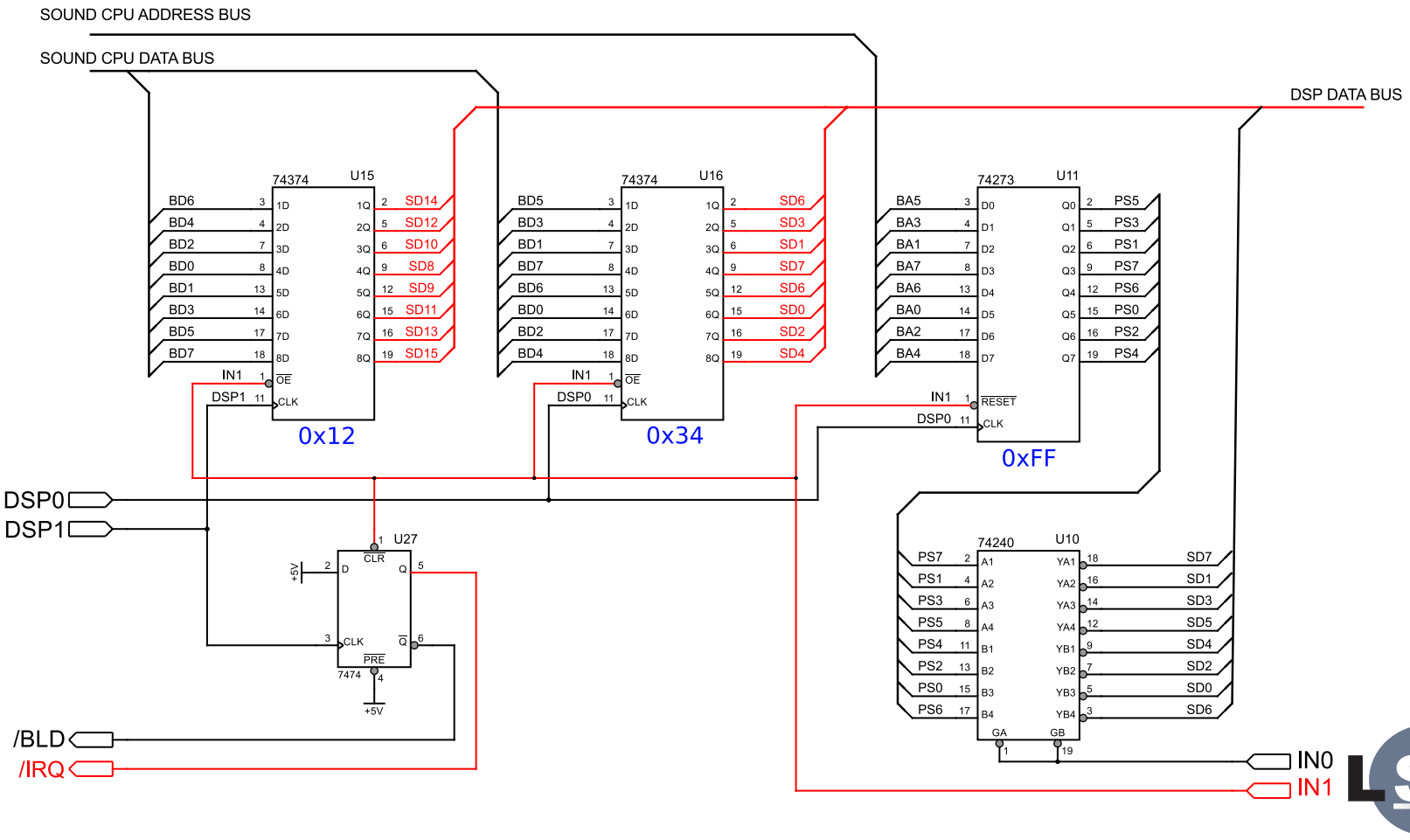
Sound CPU / DSP Interface

```
IN dma, IN0      ;; DATA[dma] <- $0056
```

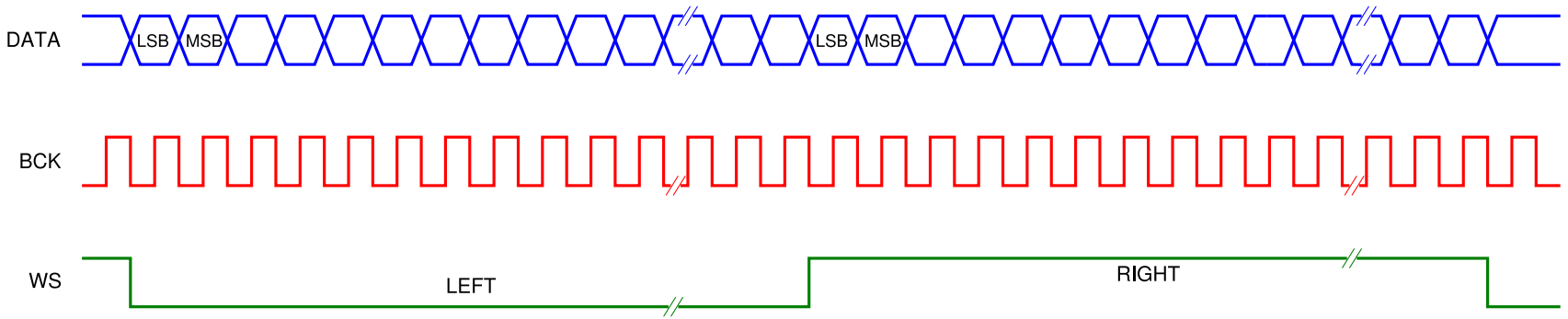


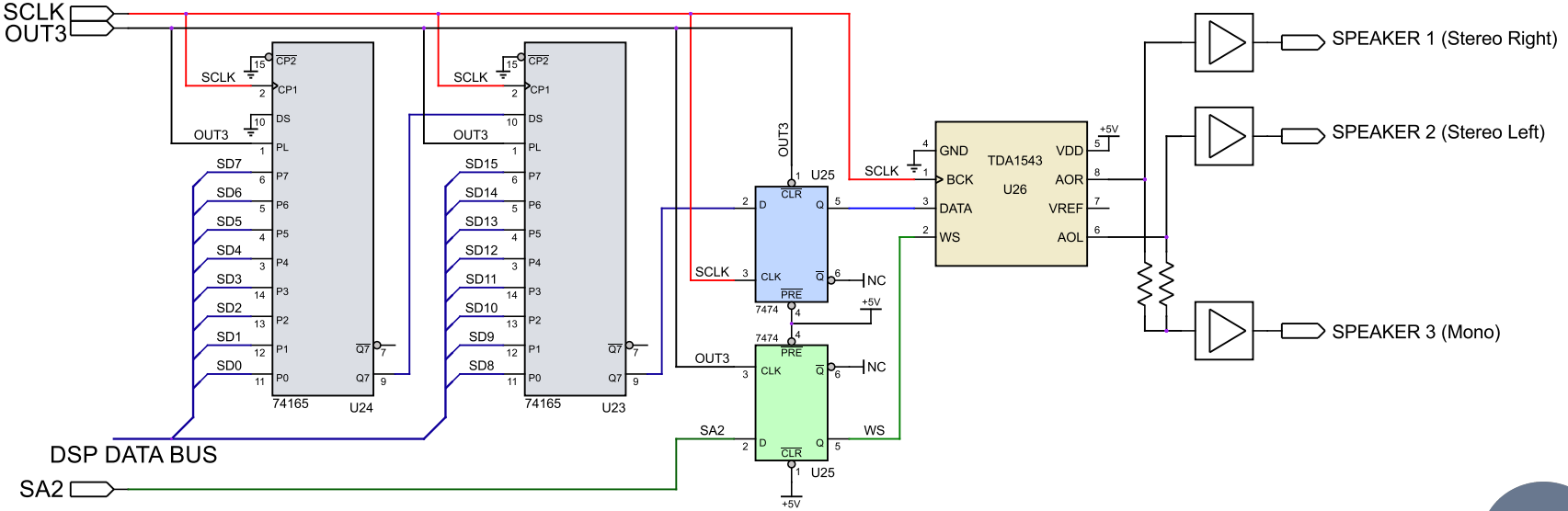
Sound CPU / DSP Interface

```
IN dma, IN1        ;; DATA[dma] <- $1234
```

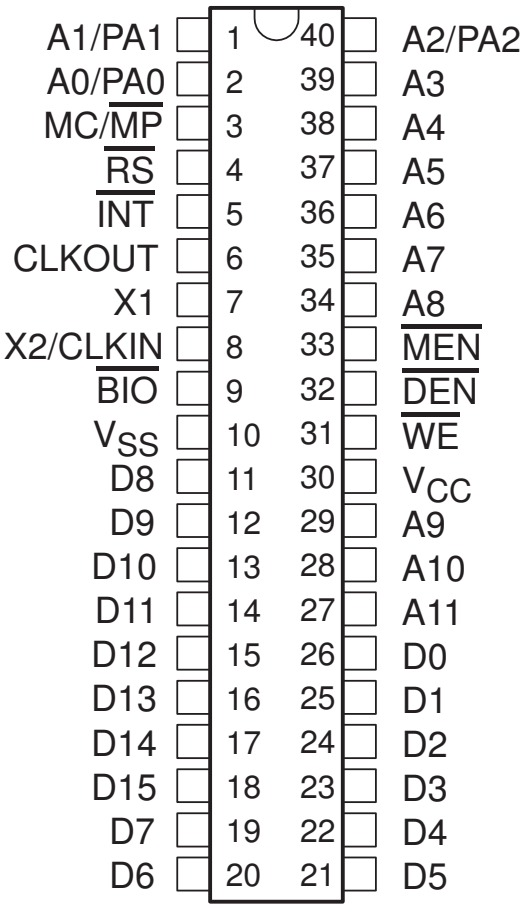


DSP / DAC Interface



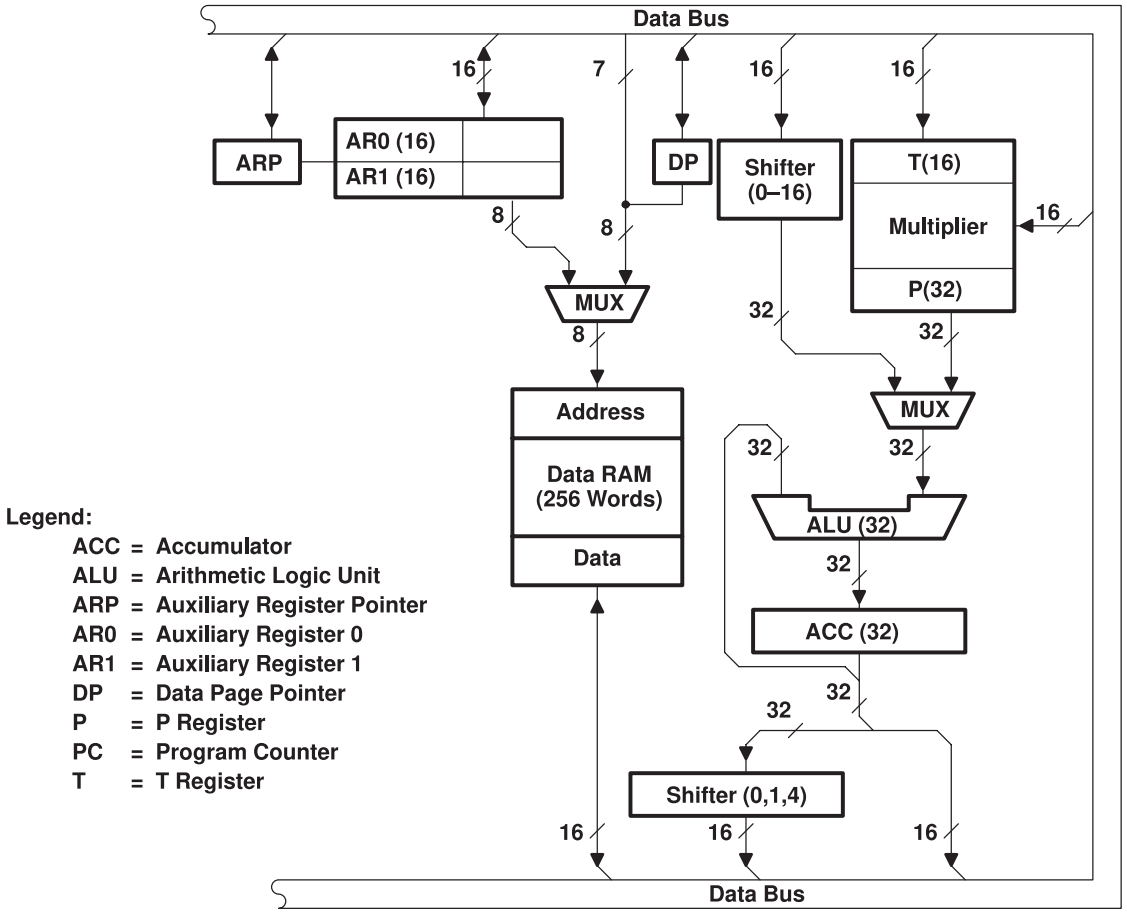


BSMT2000

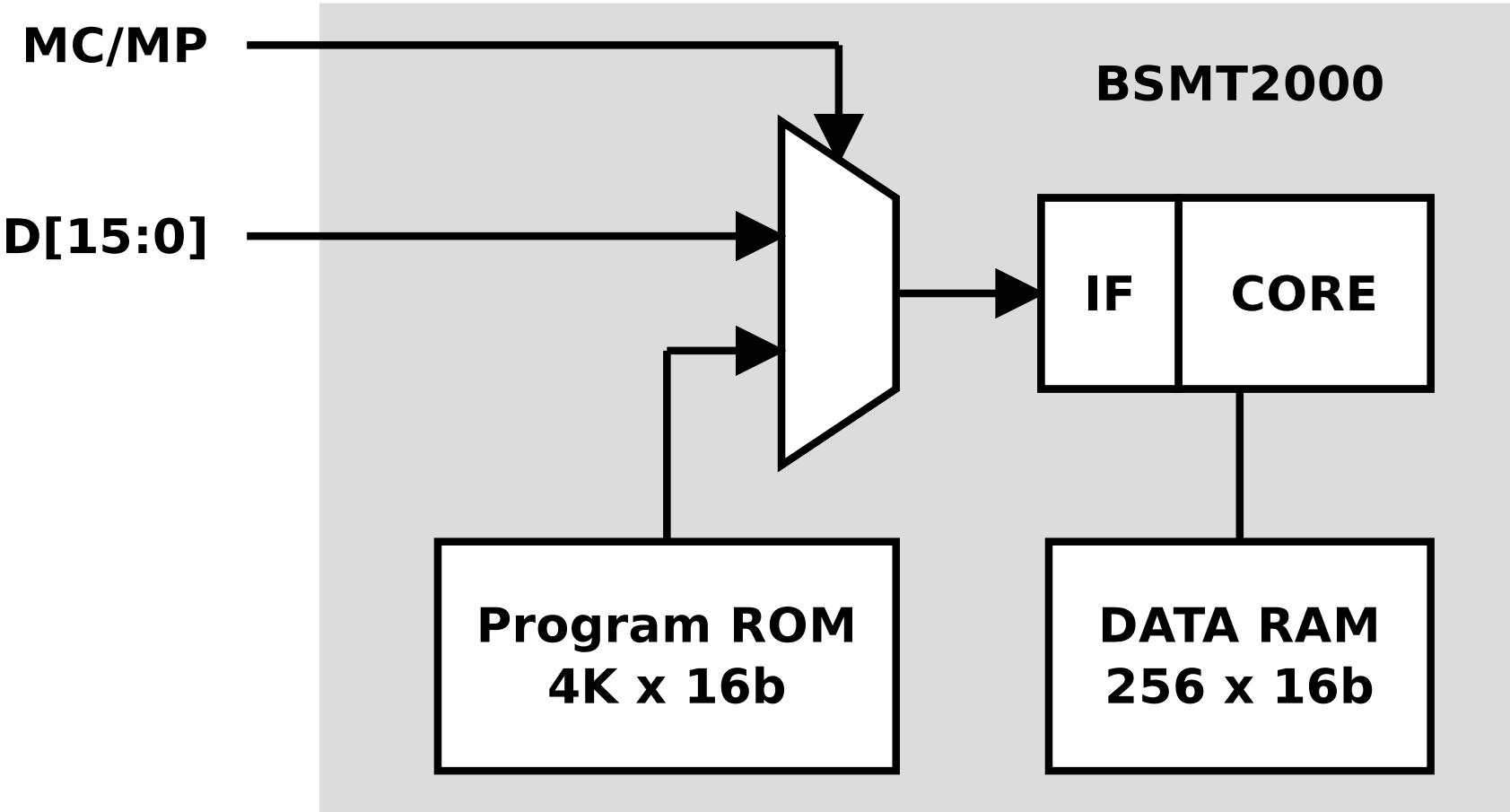


- *Brian Schmidt's Mouse Trap*
- Used in many arcade machines from 1991 to 2003
- Masked ROM `TMS320C15`
- DSP from Texas Instruments

TMS320C15 Block Diagram

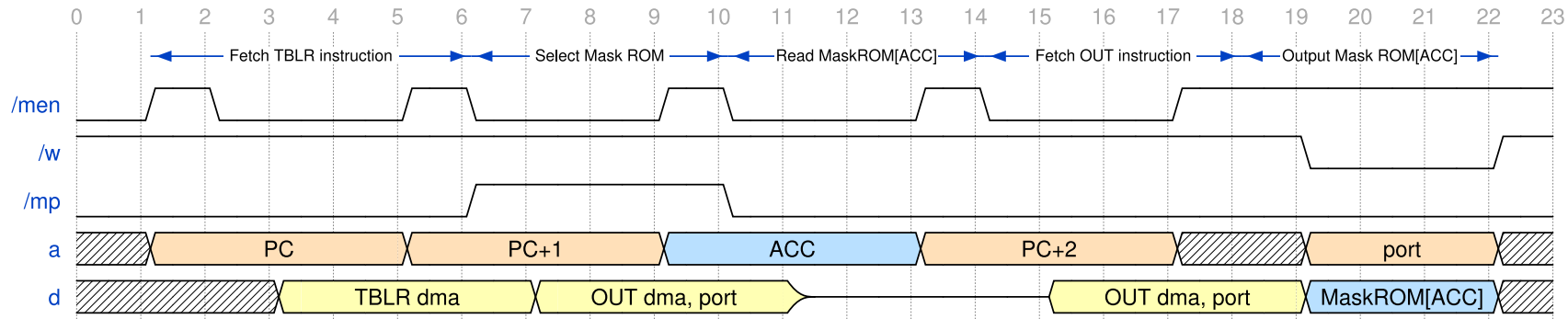


BSMT2000

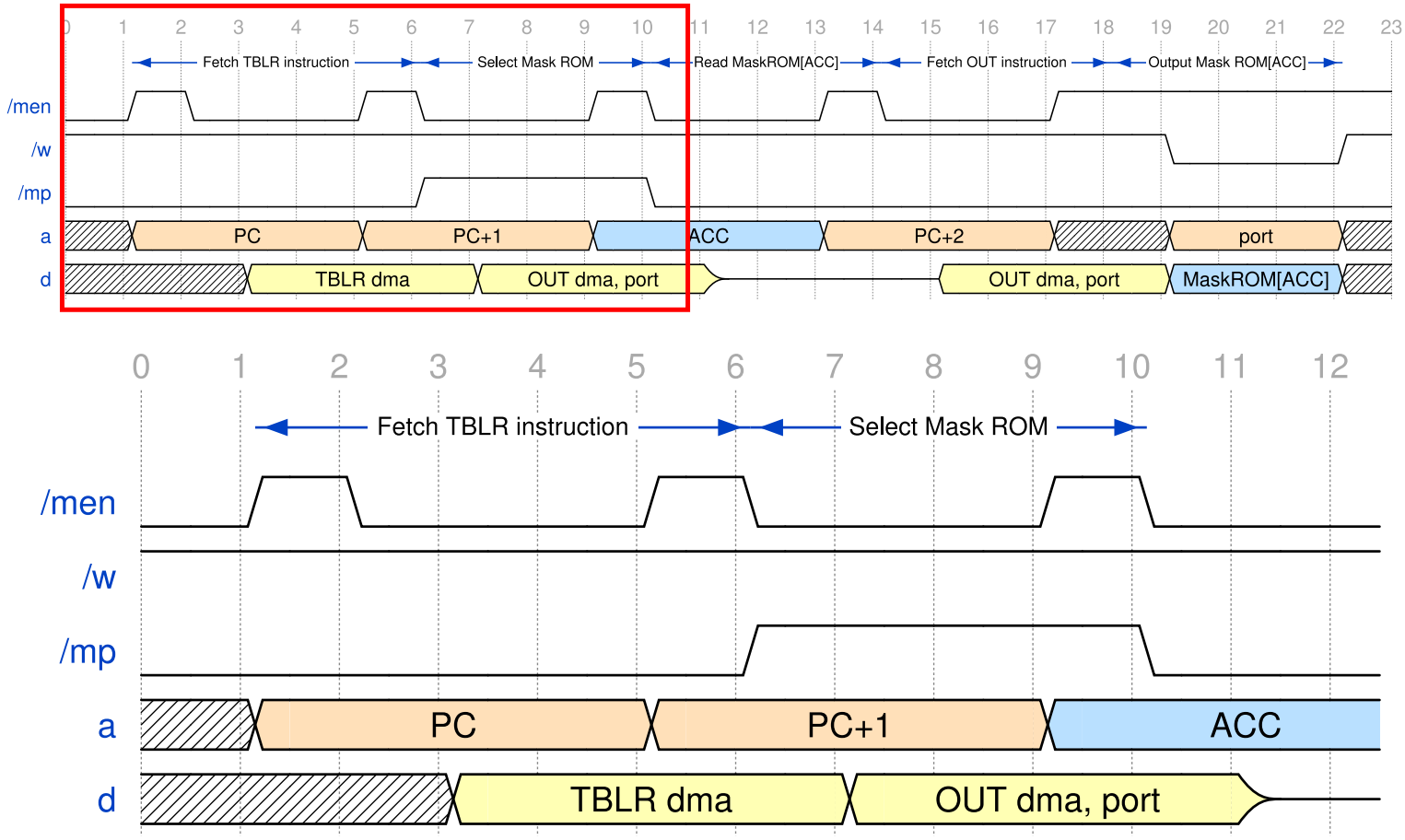

MC/MP
D[15:0]
BSMT2000
IF
CORE
Program ROM
4K x 16b
DATA RAM
256 x 16b

Dumping BSMT2000 Mask ROM

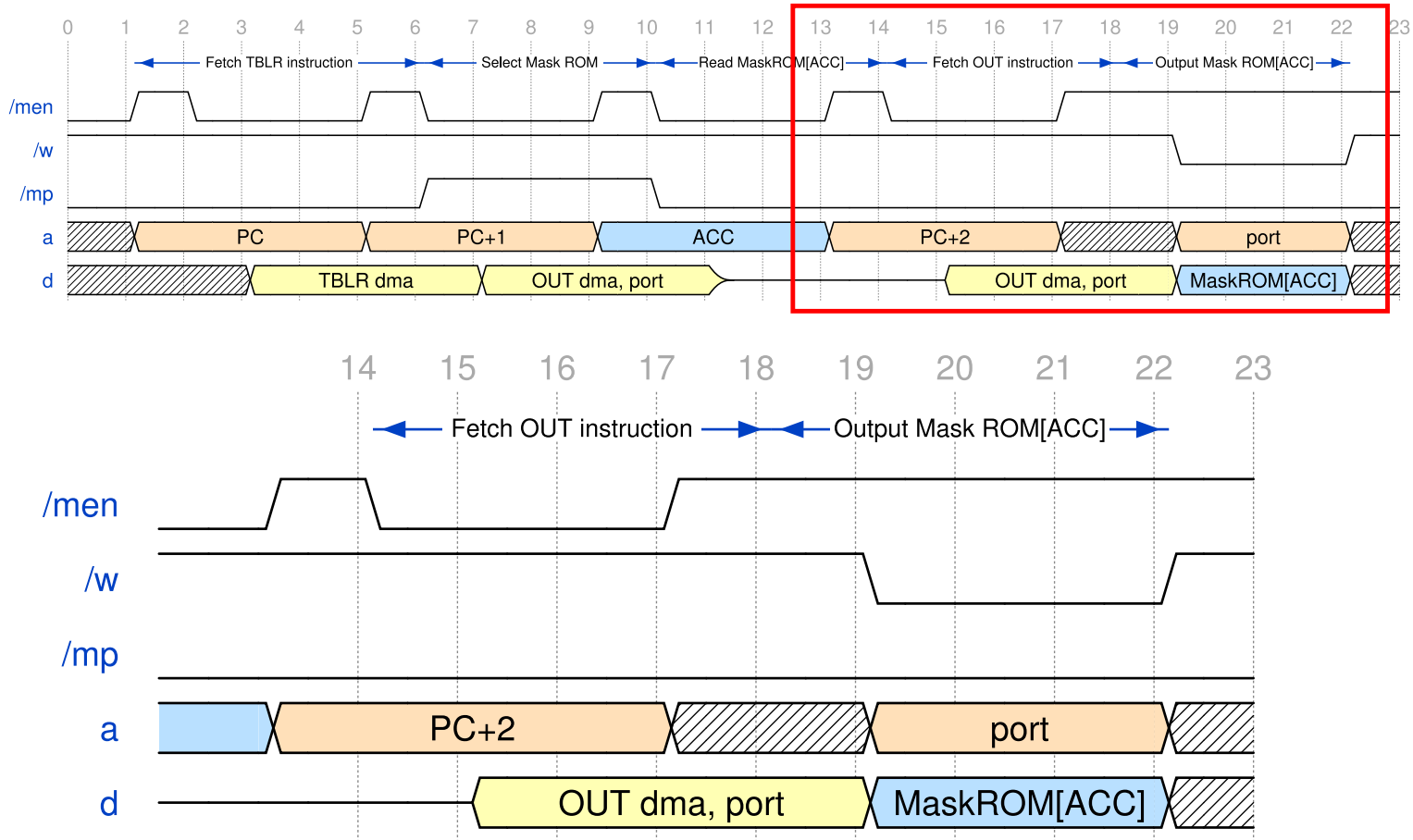
```
TBLR dma          ;; DATA[dma] <- PROG[ACC]
OUT dma, port     ;; IO[port] <- DATA[dma]
```



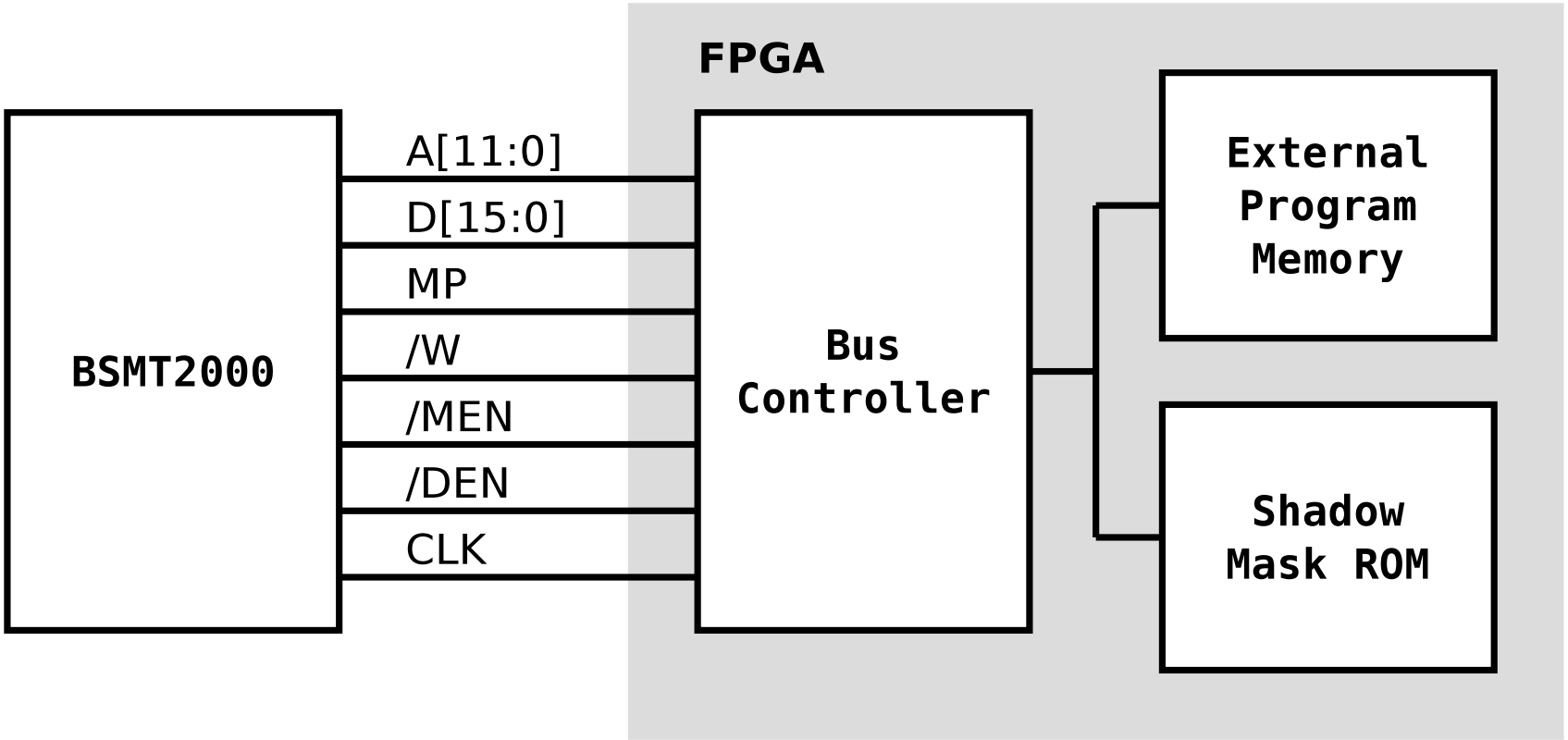
Dumping BSMT2000 Mask ROM



Dumping BSMT2000 Mask ROM



BSMT2000 Testbench Block Diagram

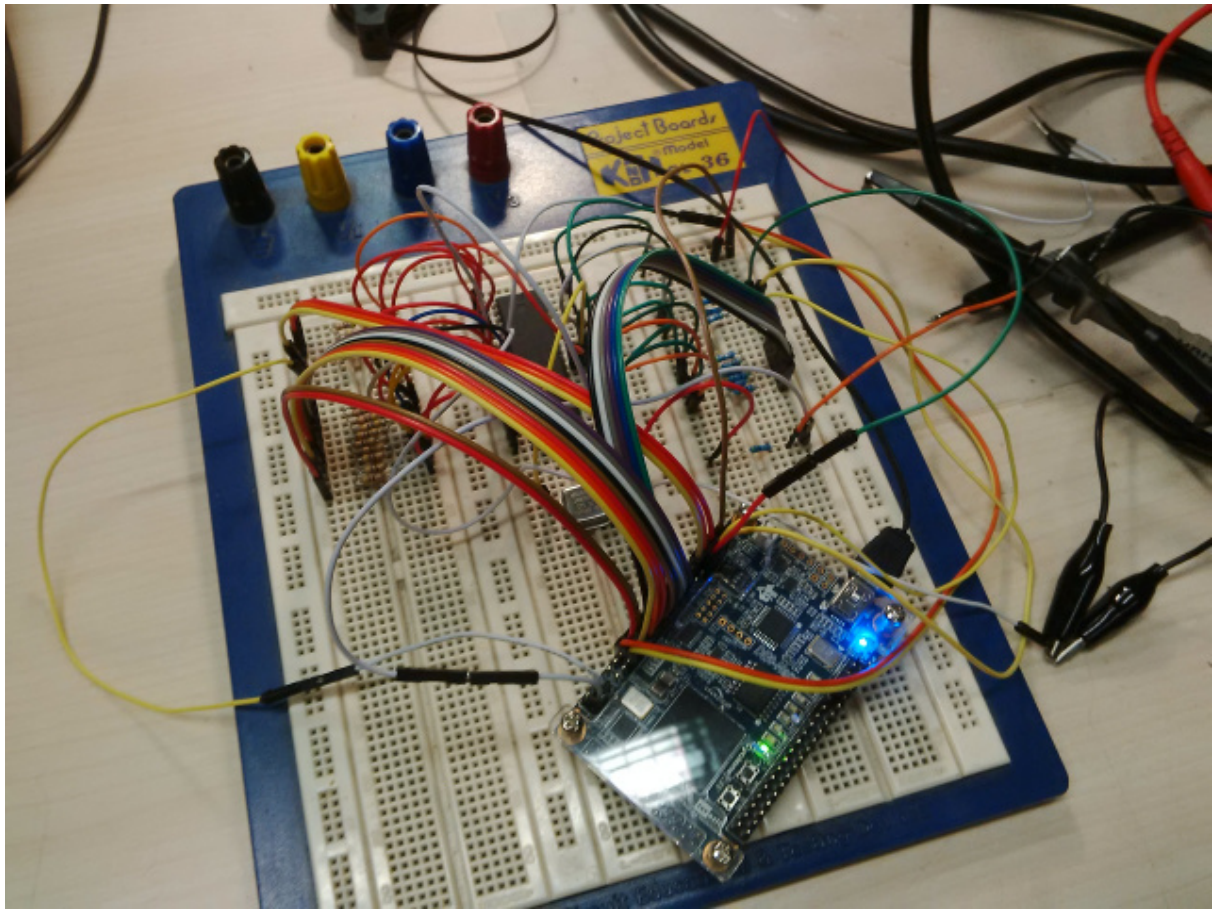


Dumping BSMT2000 Mask ROM

```
        LACK 1          ;; ACC <- 1
        SACL 0          ;; DATA[0] <- ACC
        LT 0            ;; T <- DATA[0]
        MPYK 1          ;; P <- 1 x T
        ZAC             ;; ACC <- 0

loop:   TBLR 0          ;; DATA[0] <- PROG[ACC]
        SACL 1          ;; DATA[1] <- ACC
        OUT 1, 1        ;; IO[1] <- DATA[1]
        OUT 0, 0        ;; IO[0] <- DATA[0]
        APAC            ;; ACC <- ACC + P
        B loop
```

LSE Security System

BSMT2000 Testbench



LSE Security System

BSMT2000 Address Space

- `IN 0`: Sound CPU command address
- `IN 1`: Sound CPU command data
- `IN 2`: EEPROM data
- `OUT 0`: EEPROM address
- `OUT 1`: EEPROM bank
- `OUT 3`: Sample out (Left)
- `OUT 7`: Sample out (Right)

BSMT2000 Initialization

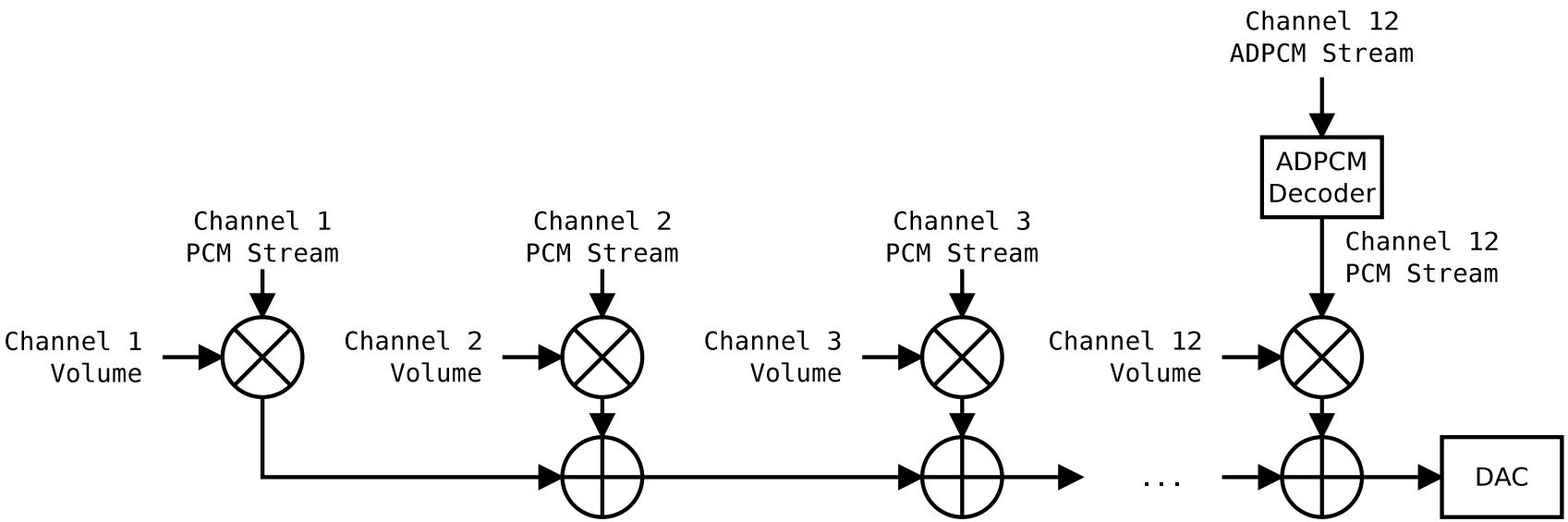
```
OSTAT   EQU $2000
DSP1    EQU $6000
DSP0    EQU $A000

init_dsp:
            ;; Reset DSP
            lda #$80
            sta OSTAT   ;; Set DSPRST

            ;; Compute command address according to the desired DSP mode
            ldb #$FE    ;; select mode 1
            ldx #DSP0
            abx         ;; x <- b + x

            ;; Select DSP mode by writing 0 at DSP0 + ~mode
            clra
            sta #DSP1   ;; MSB
            sta ,x      ;; LSB

            ;; Start DSP
            sta OSTAT   ;; Clear DSPRST
            rts
```

DSP Main Loop

```
ZAC                 ;; ACC <- 0
LT VOLUME1          ;; T <- DATA[VOLUME1]
MPY SAMPLE1         ;; P <- T * DATA[SAMPLE1]
LTA VOLUME2         ;; ACC <- ACC + P; T <- DATA[VOLUME2]
MPY SAMPLE2         ;; P <- T * DATA[SAMPLE2]
...
LTA VOLUME12        ;; ACC <- ACC + P; T <- DATA[VOLUME12]
MPY SAMPLE12        ;; P <- T * DATA[SAMPLE12]
APAC                ;; ACC <- ACC + P
SACH 0, TMP         ;; DATA[TMP] <- ACC[31:16]
OUT DAC, TMP        ;; IO[DAC] <- DATA[TMP]
```

LSE Security System

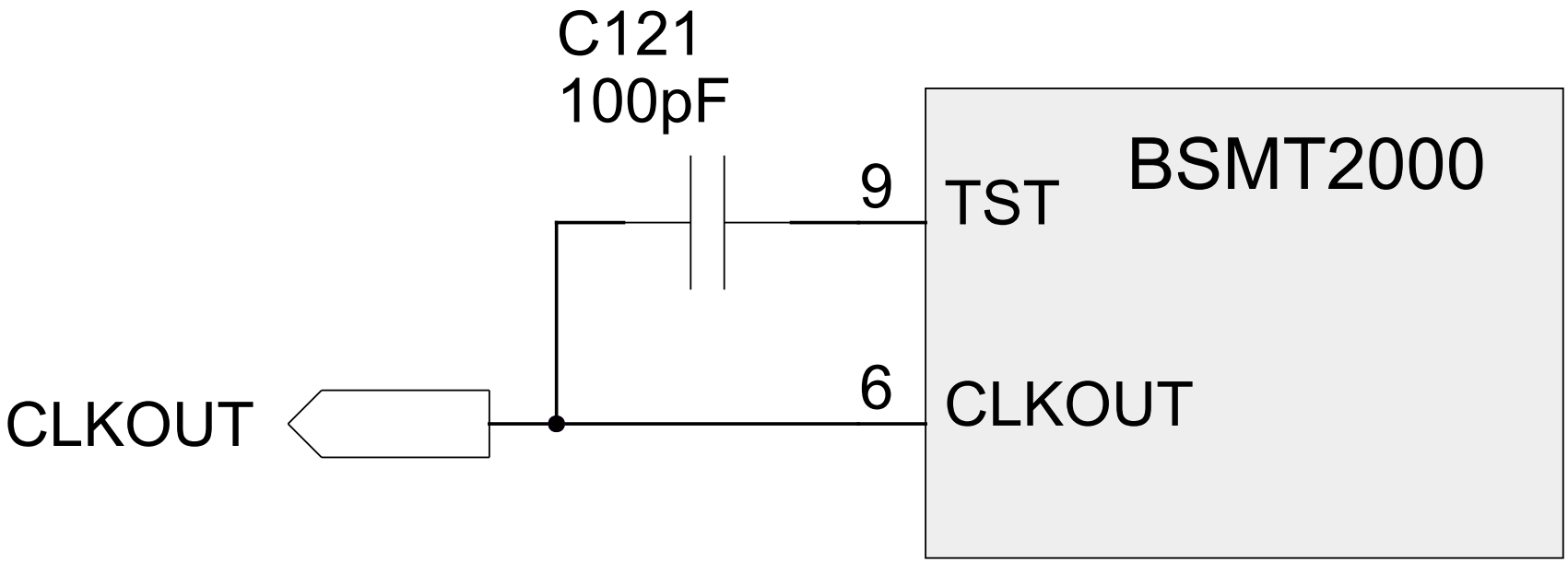
Mixing Audio Streams



Sound CPU Commands Handling

```
        BIOZ fetch          ;; Jump to ‘fetch‘ if TST pin
                            ;; is active

        NOP                 ;; Burn CPU cycles
        NOP                 ;;
        NOP                 ;;
        B next

fetch:  IN 0, 60            ;; DATA[60] <- IO[0]
        LAR AR0, 60         ;; AR0 <- DATA[60]
        IN 1, *             ;; DATA[AR0] <- IO[1]

next:
```

TST pin wiring



- `CLKOUT` = `CLKIN` / 4 = 6MHz

BSMT2000 data memory layout

- `0x0-0xA`: Channel playback positions
- `0x16-0x20`: Channel rates
- `0x21-0x2B`: Sample limits
- `0x2C-0x36`: Sample loops
- `0x37-0x41`: Sample bank
- `0x42-0x4C`: Channel right volume
- `0x4D-0x57`: Channel left volume
- `0x58-0x62`: Sample data
- `0xFF`: Scratch

LSE Security System

Back to Sound CPU firmware

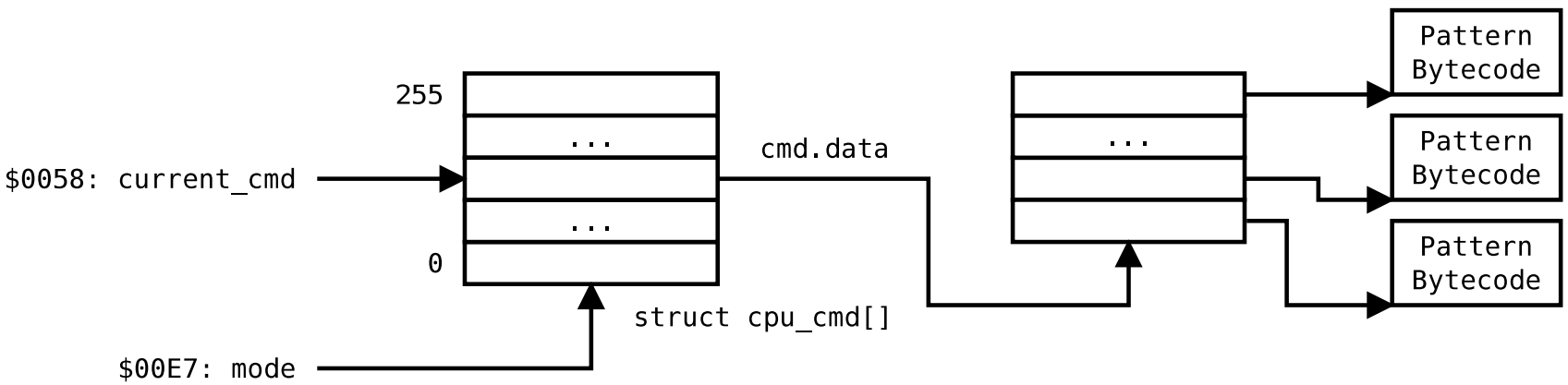
DSP operations

```
#define MAX_CHAN 12

struct dsp_ops {
    void (*set_fixed_volume[MAX_CHAN])();
    void (*set_rate[MAX_CHAN])();
    void (*set_default_rate[MAX_CHAN])();
    void (*stop_playing[MAX_CHAN])();
    void (*load_pcm_sample[MAX_CHAN])();

    void (*op5[MAX_CHAN])();
    void (*op6[MAX_CHAN])();
    void (*op7[MAX_CHAN])();
    void (*op8[MAX_CHAN])();
    void (*op9[MAX_CHAN])();
};
```

LSE Security System

Main CPU commands

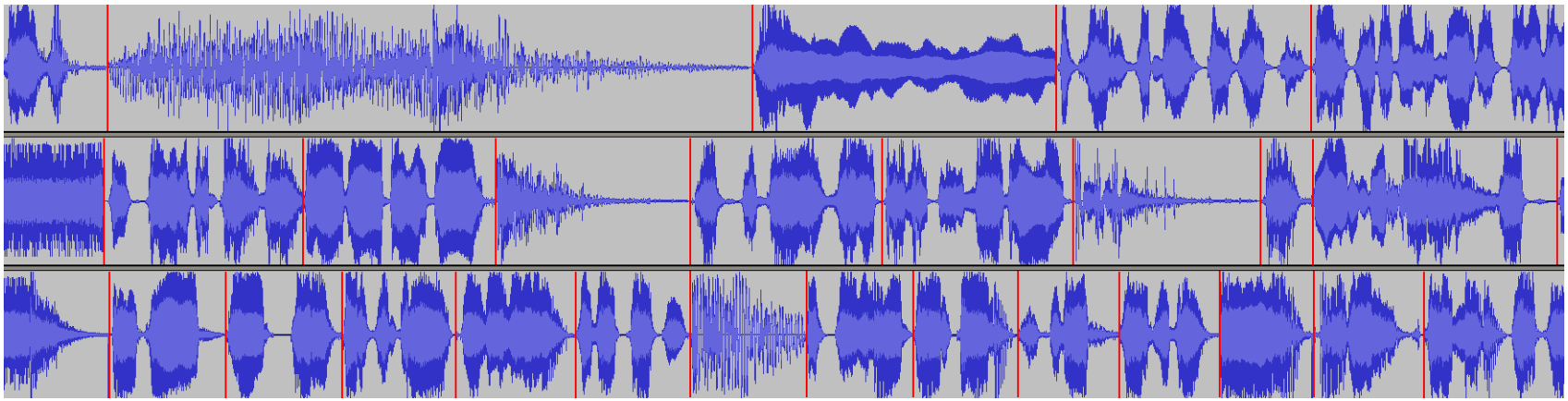
```
struct cpu_cmd {
    uint8_t     callback_idx;
    uint8_t     unk0;
    uint16_t    mask;
    void        **data;
};
```



Example : Play PCM Sample

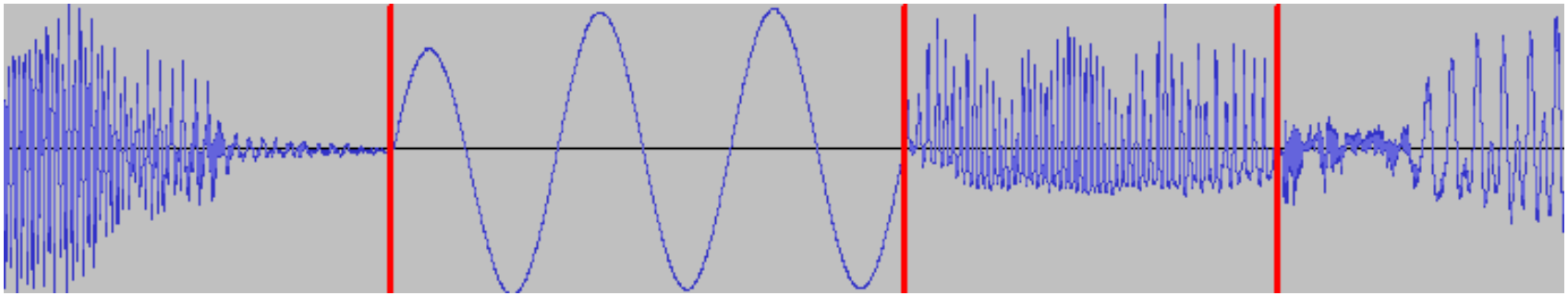
```
    ;; PCM sample description
818B: 00 00         ;; pcm.base
                    ;; sample starts at 0x0000

818D: 47 AC         ;; pcm.limit
                    ;; sample finishes at 0x47AC

818F: 47 86         ;; pcm.loop_start
                    ;; sample playing must loop at 0x4786

818F: 3C

818F: 03            ;; pcm.bank
                    ;; sample is located on bank 3 of U17 EEPROM

    ;; Explosion pattern bytecode
91DE: 05 81 8B      ;; load pcm sample described at 0x818B into channel
91E1: 09 01         ;; set channel volume
91E3: 01 1D 01 6D   ;; set channel rate, start sample playing
                    ;; and wait 7425 ticks (0x1D01) => 2.53 seconds
91E7: 0F            ;; free the channel and stop sample playing
```

PCM Samples

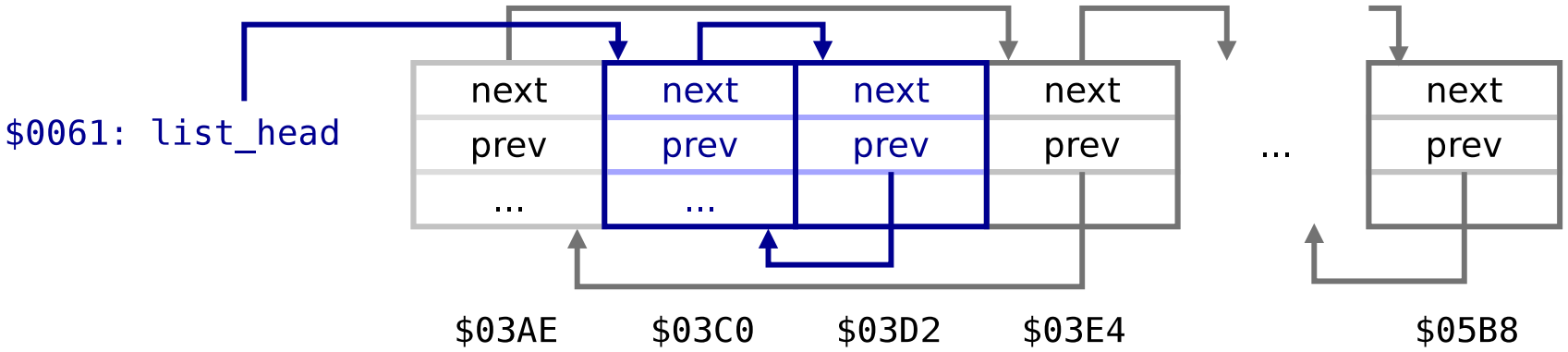


```
struct pcm_sample {
    uint16_t    base;
    uint16_t    limit;
    uint16_t    loop_start;
    uint8_t     unk;
    uint8_t     bank;
};
```

PCM Samples

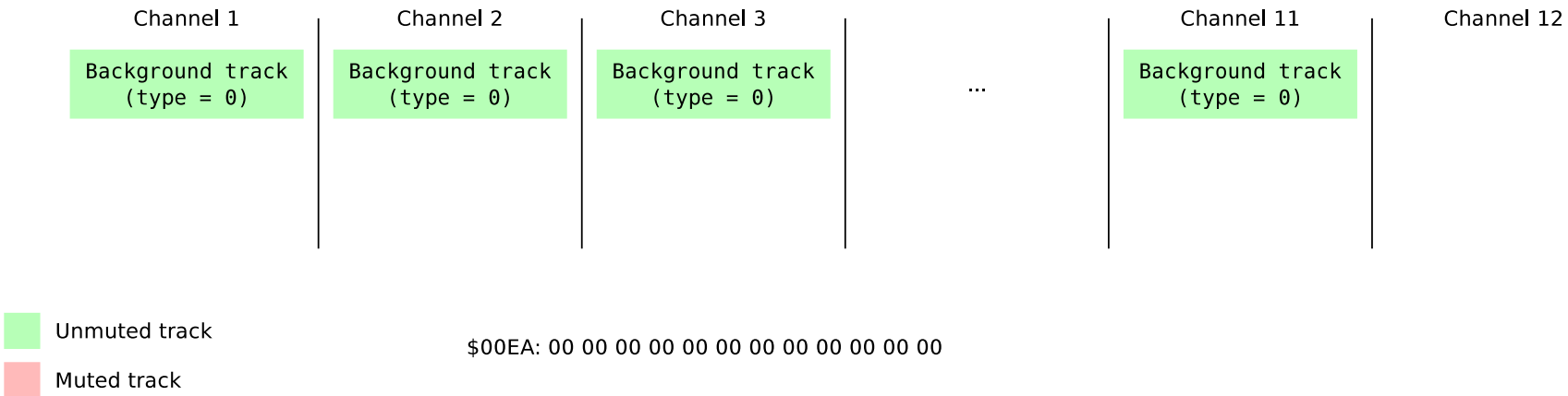


Track Allocation

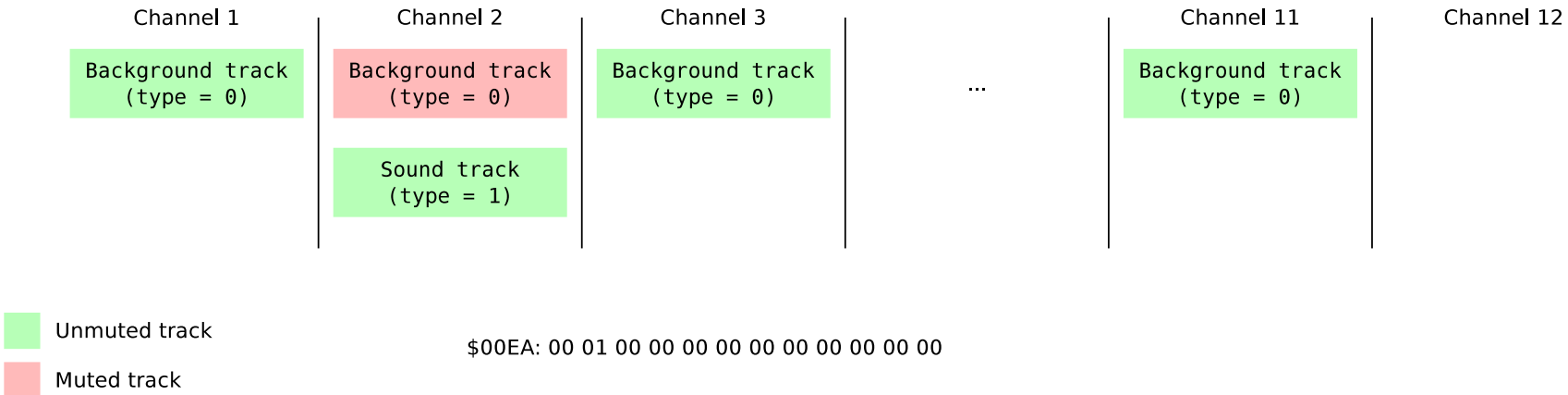



```
struct track {
    struct track    *next;
    struct track    *prev;

    void            *instruction_pointer;   // Address of the next bytecode
                                            // instruction

    uint16_t        counter;                // Used for operation timing
    uint16_t        last_timestamp;

    uint8_t         next_instruction;
    uint8_t         type;                   // 0: Background track
                                            // 1: Foreground track

    uint8_t         channel_id;
    ...
};
```

Track Types

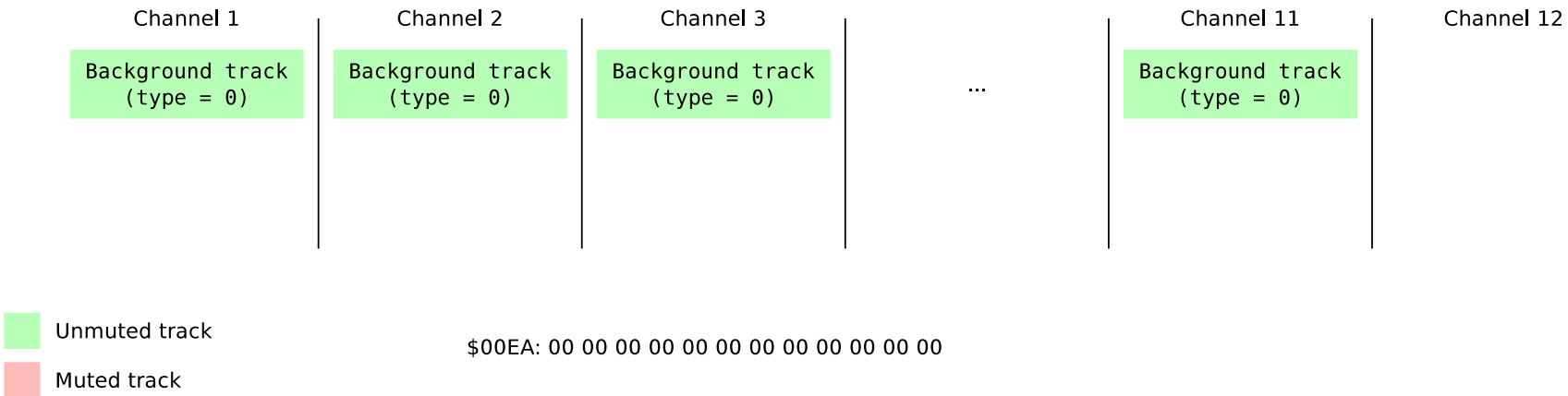

Channel 1
Background track (type = 0)
Channel 2
Background track (type = 0)
Channel 3
Background track (type = 0)
...
Channel 11
Background track (type = 0)
Channel 12
Unmuted track
Muted track
$00EA: 00 00 00 00 00 00 00 00 00 00 00 00

Track Types




```
uint8_t *channels_types = (void *) 0x00EA;

if (track.type == channels_types[current_channel])
    dsp_ops[current_channel]();
```

Track Types


Channel 1
Background track (type = 0)
Channel 2
Background track (type = 0)
Channel 3
Background track (type = 0)
...
Channel 11
Background track (type = 0)
Channel 12
Unmuted track
Muted track
$00EA: 00 00 00 00 00 00 00 00 00 00 00 00

Q&A

- IRC: `Ptishell@irc.rezosup.org`
- Mail: `surply@lse.epita.fr`
- Twitter: `@Ptishell`

LSE Security System