Designing Hardware to Improve Your Software

A Use Case in Sound

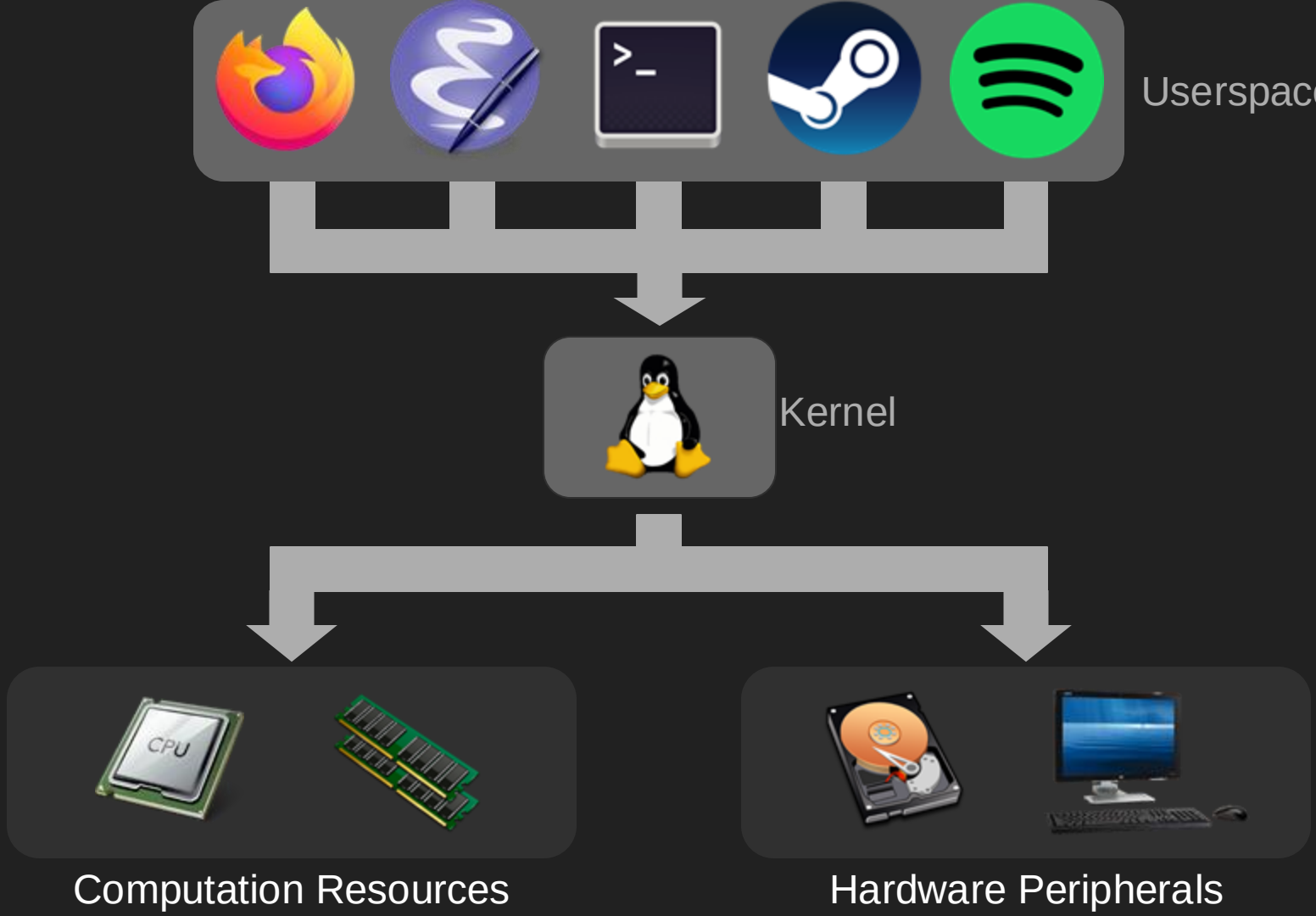
About Me

- Undergrad in EECS @ MIT
- Worked at JetBrains, NVIDIA, and Jump Trading
- Passionate about Linux, audio, and embedded systems

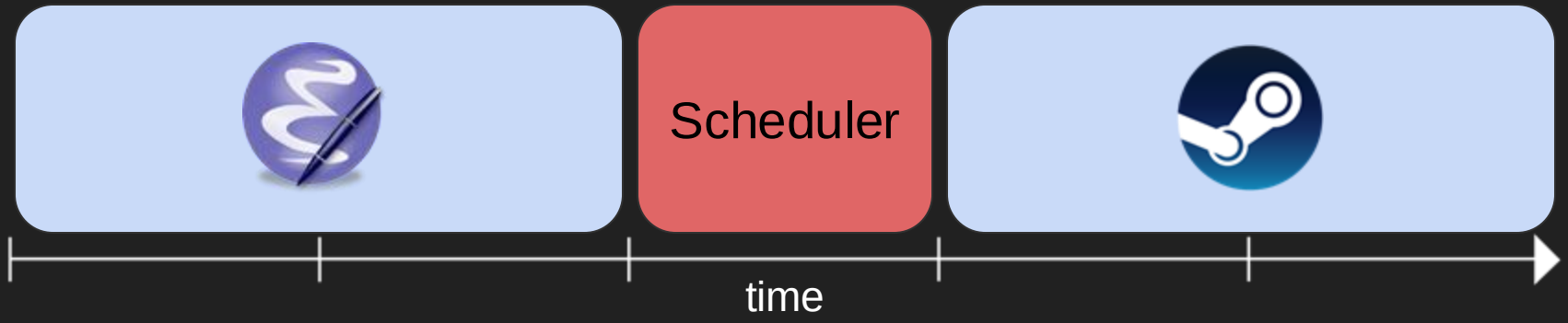
Goals for the talk

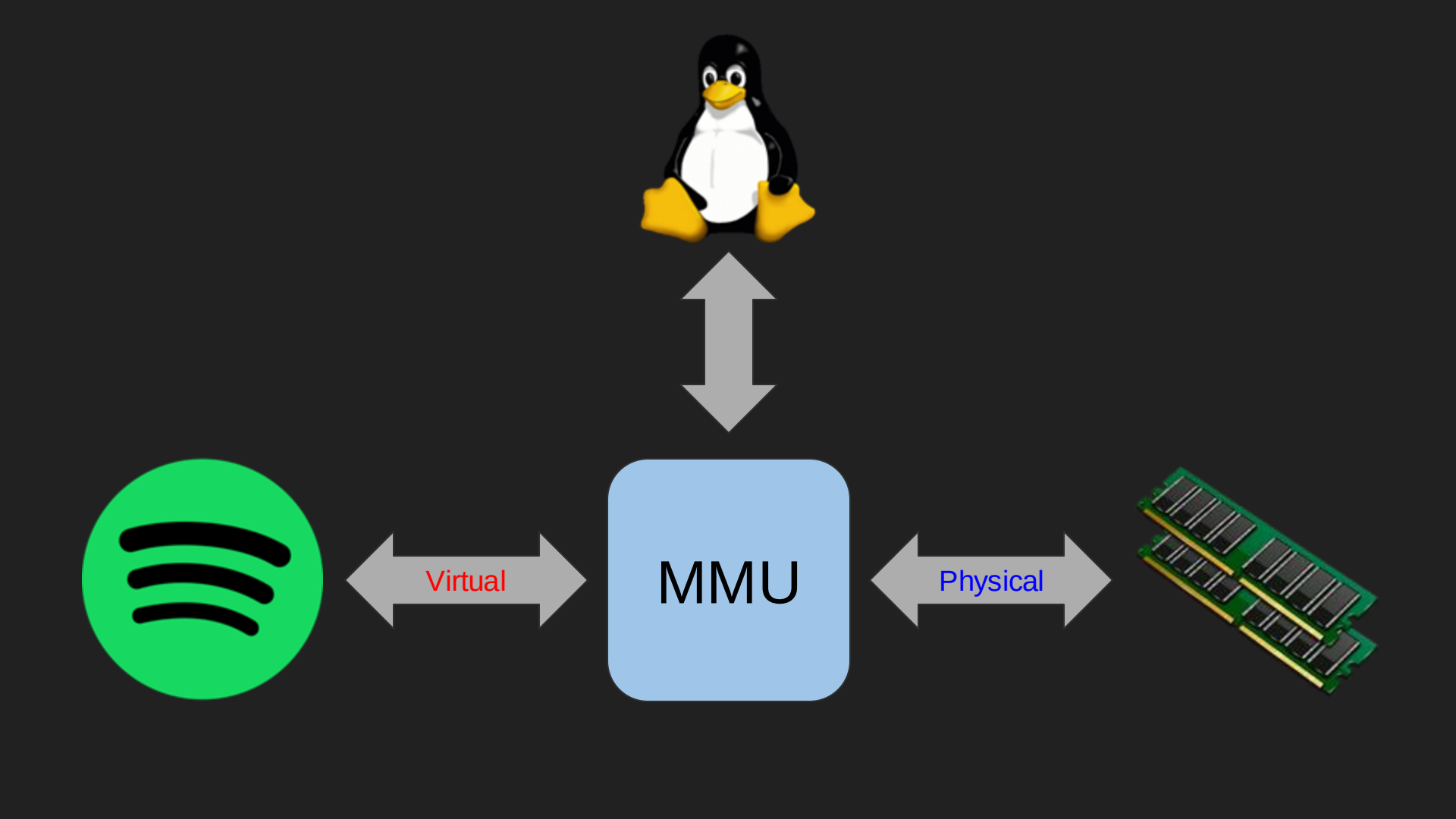
1. How do operating systems work?
2. How does sound work on Linux?
3. How can we build our own hardware with FPGAs?
4. Describe the journey I took of building something complex from scratch

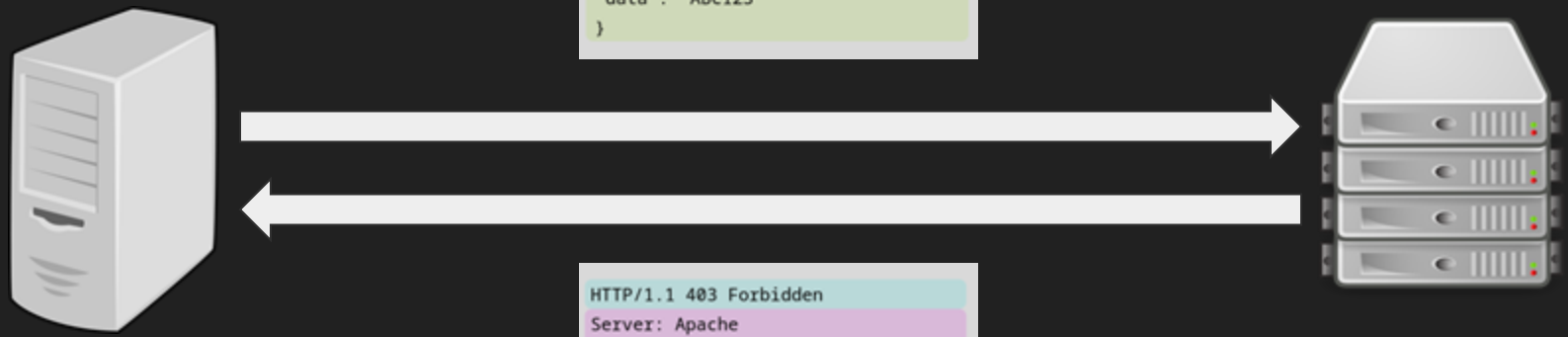

CPU
Computation Resources
Hardware Peripherals


Userspace
Kernel
CPU
Computation Resources
Hardware Peripherals


CPU


Scheduler
time


Virtual
MMU
Physical


POST / HTTP/1.1
Host: developer.mozilla.org
User-Agent: curl/8.6.0
Accept: */*
Content-Type: application/json
Content-Length: 345
{
"data": "ABC123"
}
HTTP/1.1 403 Forbidden
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Content-Length: 678
Content-Type: text/html
Cache-Control: no-store
<!DOCTYPE html>
<html lang="en">
(more data…)

```
long square(long x) {
    return x * x;
}

int main() {
    long result = square(4);
    // ... do something with return value ...
}
```

```
square:
    // Move argument into return register
    mov rax, rdi

    // Square argument, leaving result in rax
    imul rax, rax

    // Pop return address, jump back to caller
    ret

main:
    // Correctly place argument, per ABI
    mov rdi, 4

    // Push return address, jump to function
    call square

    // ... do something with return value ...
```

```c
#include <stdlib.h> // for exit()

int main() {
    exit(42);
}
```

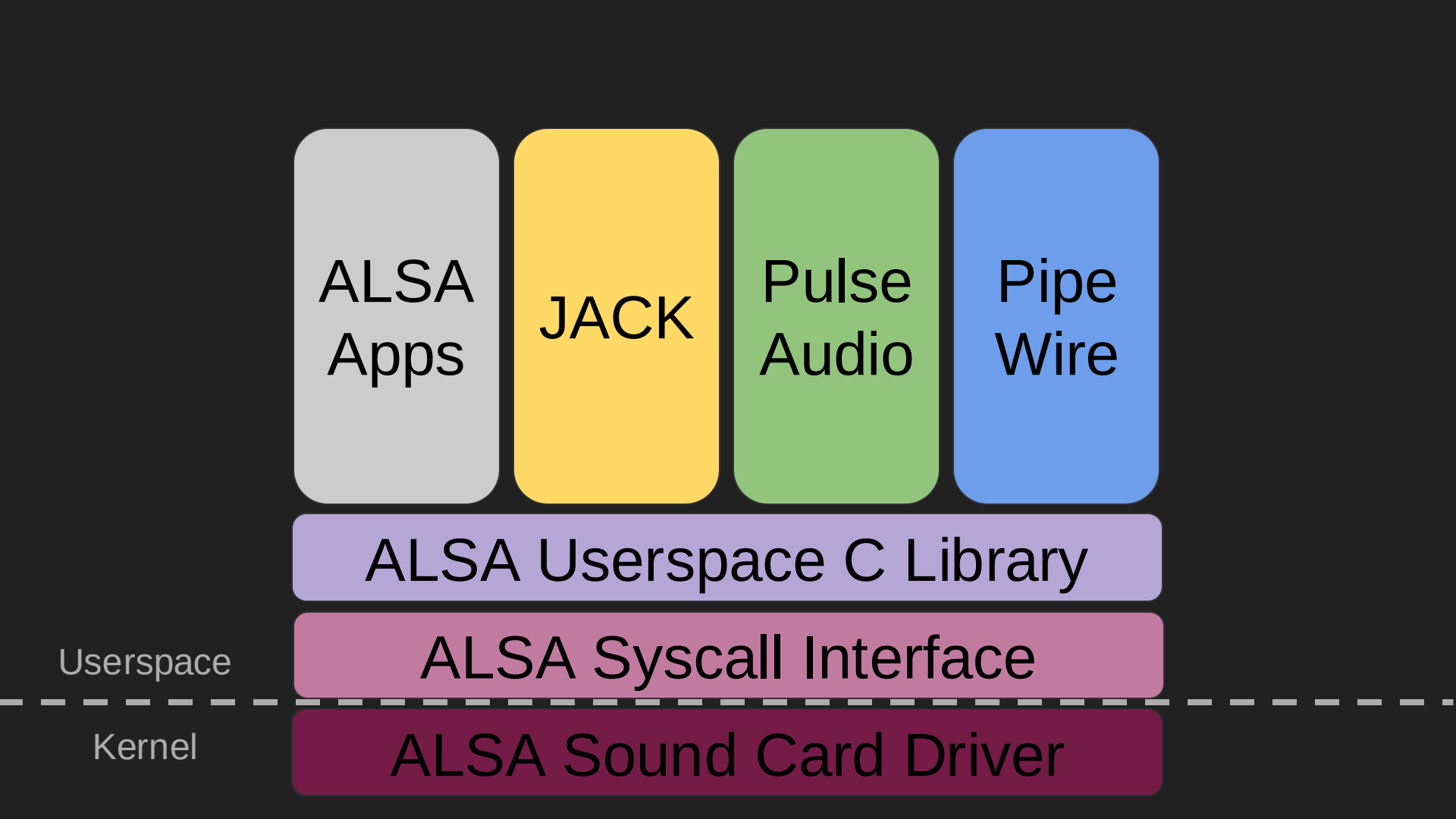
```asm
main:
    // Select syscall (see /usr/include/unistd_64.h)
    mov rax, 60

    // Syscall argument: exit code
    mov rdi, 42

    // Trigger syscall, transfer control to kernel
    syscall
```

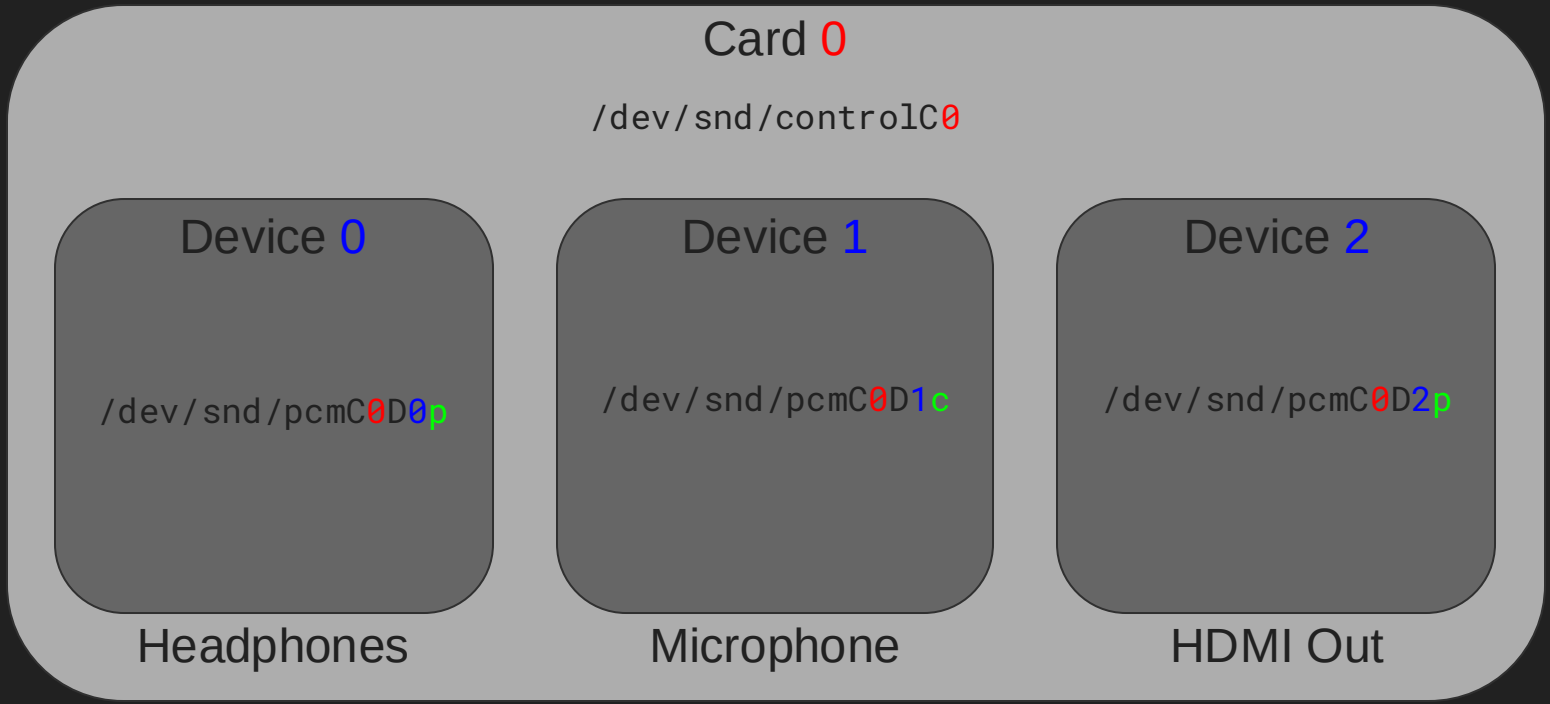
Register	Argument User Space	Argument Kernel Space
%rax	Not Used	System Call Number
%rdi	Arguement 1	Arguement 1
%rsi	Arguement 2	Arguement 2
%rdx	Arguement 3	Arguement 3
%r10	Not Used	Arguement 4
%r8	Arguement 5	Arguement 5
%r9	Arguement 6	Arguement 6
%rcx	Arguement 4	Destroyed
%r11	Not Used	Destroyed

Advanced Linux Sound Architecture

- Created by Jaroslav Kysela in 1998
- Merged into Linux 2.5 in 2002
- Framework for writing sound card drivers
- Userspace API for interacting with sound cards

ALSA


ALSA Apps
JACK
Pulse Audio
Pipe Wire
ALSA Userspace C Library
ALSA Syscall Interface
Userspace
Kernel
ALSA Sound Card Driver

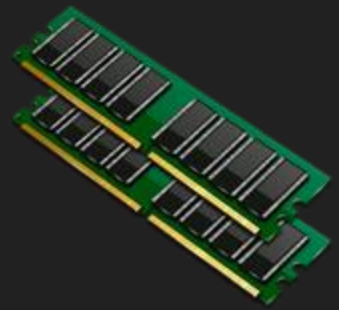

Card 0
/dev/snd/controlC0
Device 0
/dev/snd/pcmC0D0p
Device 1
/dev/snd/pcmC0D1c
Device 2
/dev/snd/pcmC0D2p
Headphones
Microphone
HDMI Out

What do you mean “my sound card is just a bunch of files”?

Well, not just your sound card. Everything is.

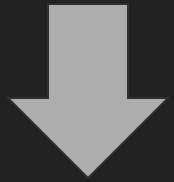
`/dev/input/by-id/<id>`

`/dev/sda`

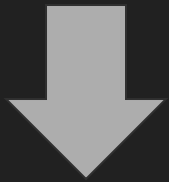


`/dev/mem`

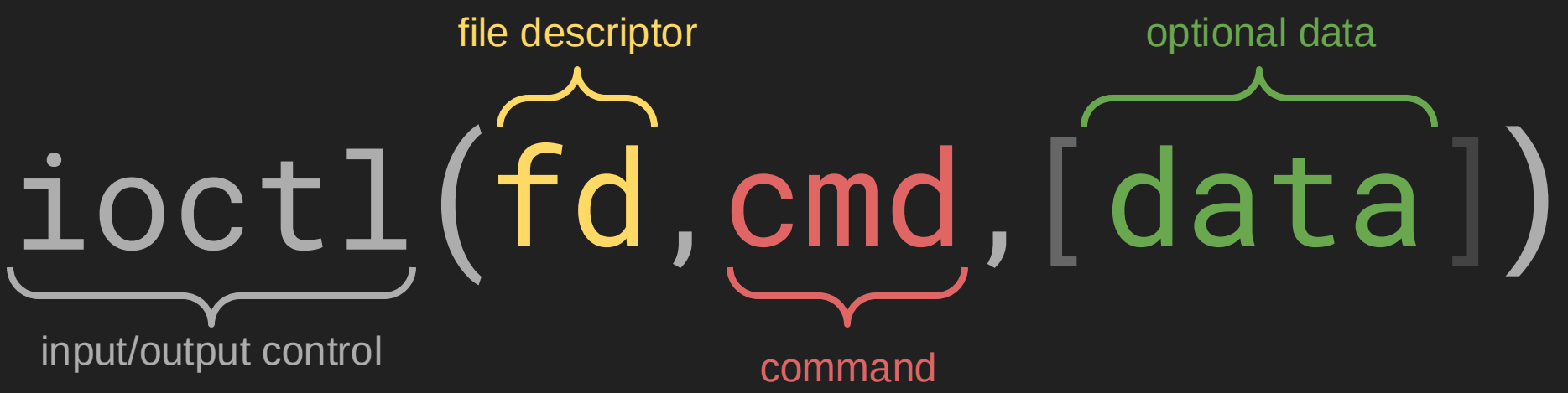
```
open(path, flags)
```



“File Descriptor”



```
close(fd)
```


file descriptor
optional data
ioctl(fd, cmd, [data])
input/output control
command

(Excerpt from `/usr/include/sound/asound.h`)

```
#define SNDRV_PCM_IOCTL_WRITEN_FRAMES   _IOW('A', 0x52, struct snd_xfern)
```

01	00000000011000	01000001	01010010

1075331410

(in decimal)

◆ snd_pcm_open()

```
int snd_pcm_open ( snd_pcm_t **        pcmp,
                   const char *        name,
                   snd_pcm_stream_t    stream,
                   int                 mode
                 )
```

Opens a PCM.

Parameters

- **pcmp** Returned PCM handle
- **name** ASCII identifier of the PCM handle
- **stream** Wanted stream
- **mode** Open mode (see **SND_PCM_NONBLOCK**, **SND_PCM_ASYNC**)

```
// Open card
ioctl("/dev/snd/controlC0", SNDRV_CTL_IOCTL_CARD_INFO)
ioctl("/dev/snd/controlC0", SNDRV_CTL_IOCTL_PVERSION)
ioctl("/dev/snd/controlC0", SNDRV_CTL_IOCTL_PCM_PREFER_SUBDEVICE)

// Open PCM
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_INFO)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_PVERSION)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_USER_PVERSION)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_TTSTAMP)
```

◆ snd_pcm_hw_params_any()

```
int snd_pcm_hw_params_any ( snd_pcm_t *           pcm,
                            snd_pcm_hw_params_t * params
                          )
```

Fill params with a full configuration space for a PCM.

Parameters

pcm	PCM handle
params	Configuration space

The configuration space will be filled with all possible ranges for the PCM device.

Note that the configuration space may be constrained by the currently installed configuration on the PCM device. To remove any constrains, free the configuration with **snd_pcm_hw_free** first.

```
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HW_REFINE)
```

• snd_pcm_hw_params()

```
int snd_pcm_hw_params ( snd_pcm_t *                pcm,
                        snd_pcm_hw_params_t *      params
                      )
```

Install one PCM hardware configuration chosen from a configuration space and snd_pcm_prepare it.

Parameters

pcm PCM handle

params Configuration space definition container

Returns

0 on success otherwise a negative error code

The configuration is chosen fixing single parameters in this order: first access, first format, first subformat, min channels, min rate, min period time, max buffer size, min tick time. If no mutually compatible set of parameters can be chosen, a negative error code will be returned.

After this call, snd_pcm_prepare() is called automatically and the stream is brought to SND_PCM_STATE_PREPARED state.

The hardware parameters cannot be changed when the stream is running (active). The software parameters can be changed at any time.

The configuration space will be updated to reflect the chosen parameters.

```
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HW_REFINE)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HW_PARAMS)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_PREPARE)
```

◆ snd_pcm_sw_params()

```
int snd_pcm_sw_params ( snd_pcm_t *              pcm,
                        snd_pcm_sw_params_t *    params
                      )
```

Install PCM software configuration defined by params.

Parameters

pcm PCM handle

params Configuration container

Returns

0 on success otherwise a negative error code

The software parameters can be changed at any time. The hardware parameters cannot be changed when the stream is running (active).

The function is thread-safe when built with the proper option.

```
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_SW_PARAMS)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_PREPARE)
```

◆ snd_pcm_writei()

```
snd_pcm_sframes_t snd_pcm_writei ( snd_pcm_t *          pcm,
                                   const void *         buffer,
                                   snd_pcm_uframes_t    size
                                 )
```

Write interleaved frames to a PCM.

Parameters

pcm	PCM handle
buffer	frames containing buffer
size	frames to be written

Returns

a positive number of frames actually written otherwise a negative error code

```
// Start filling up audio buffer
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_WRITEN_FRAMES)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HWSYNC)

// Once enough data is sent, begin playback
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_START)

// Continue sending audio data as playback continues
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_WRITEN_FRAMES)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HWSYNC)
```

◆ snd_pcm_drain()

int snd_pcm_drain (**snd_pcm_t** * pcm)

Stop a PCM preserving pending frames.

Parameters

pcm PCM handle

Returns

0 on success otherwise a negative error code

```
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_SW_PARAMS)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_DRAIN)
```

◆ snd_pcm_close()

int snd_pcm_close (**snd_pcm_t** * pcm)

close PCM handle

Parameters

pcm PCM handle

Returns

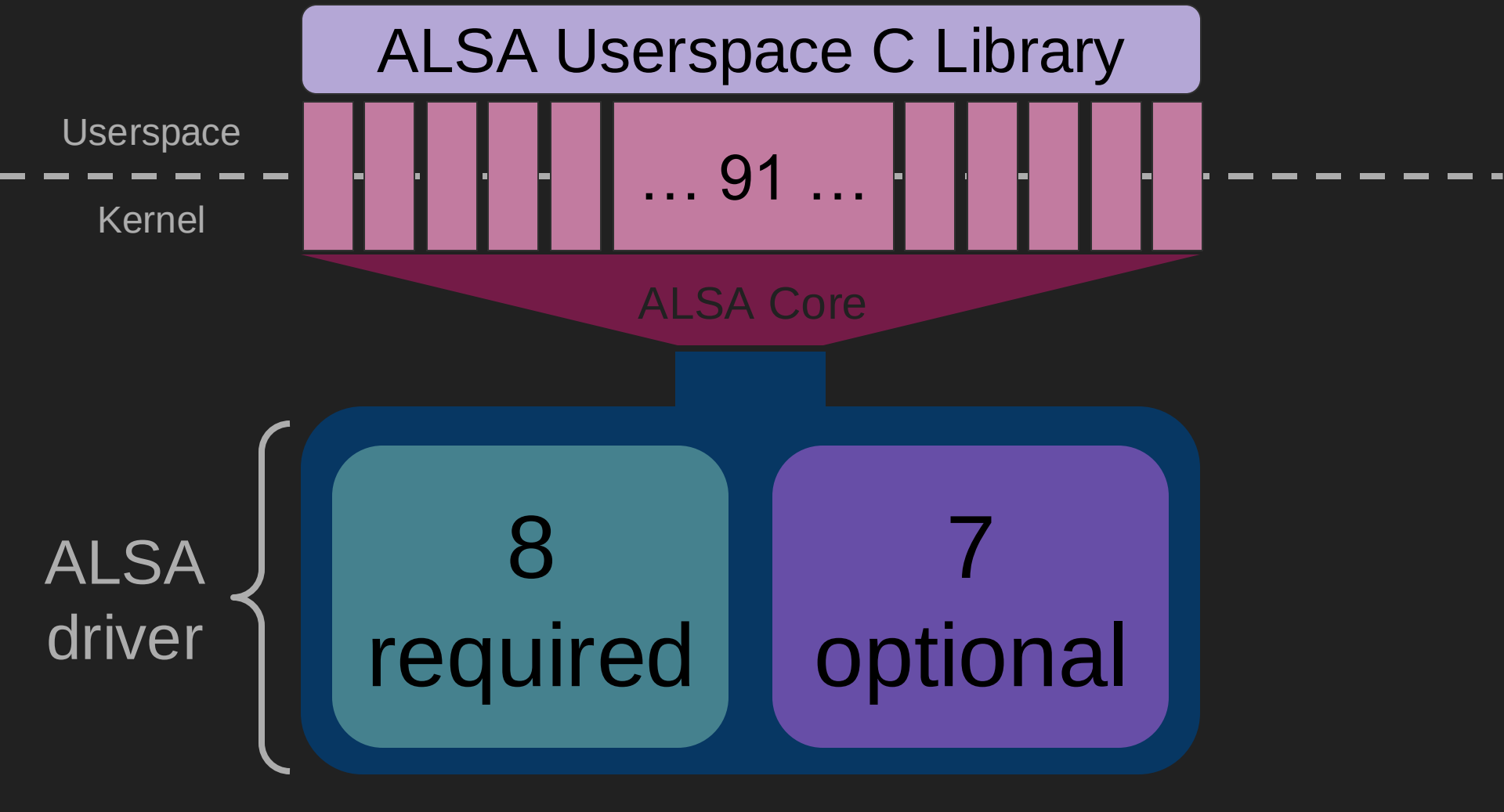
0 on success otherwise a negative error code

Closes the specified PCM handle and frees all associated resources.

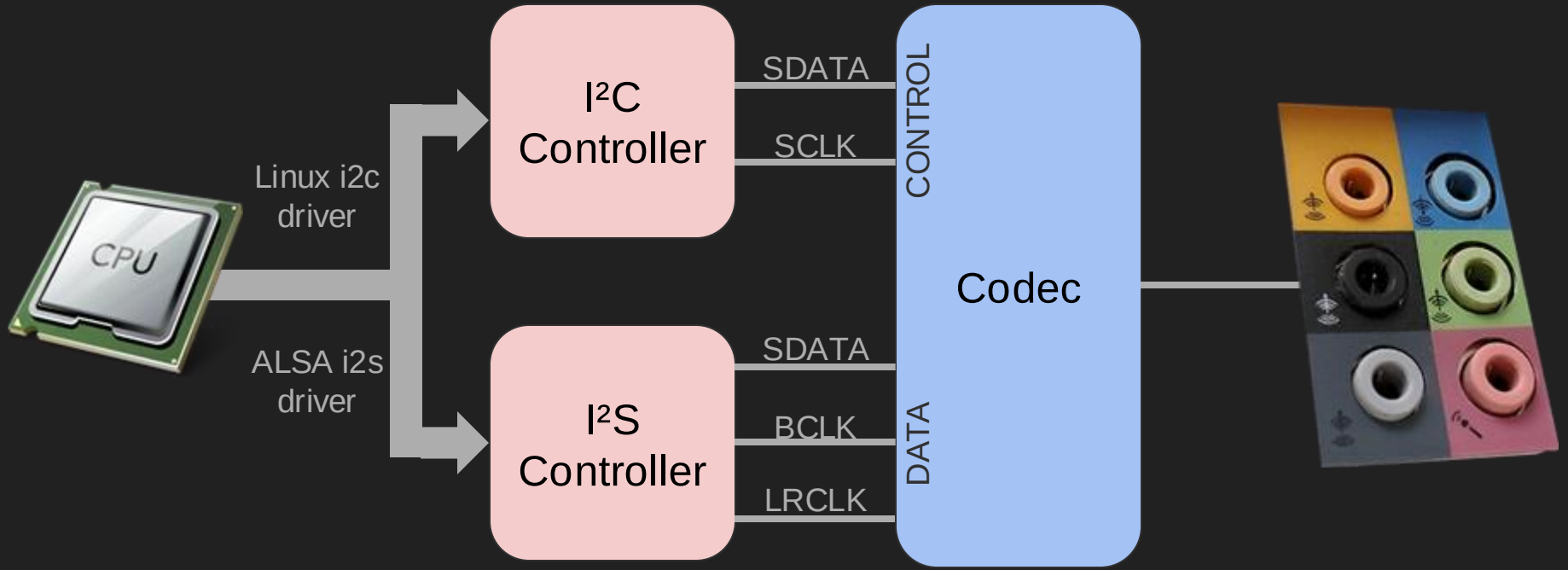
```
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_DROP)
ioctl("/dev/snd/pcmC0D0p", SNDRV_PCM_IOCTL_HW_FREE)
```

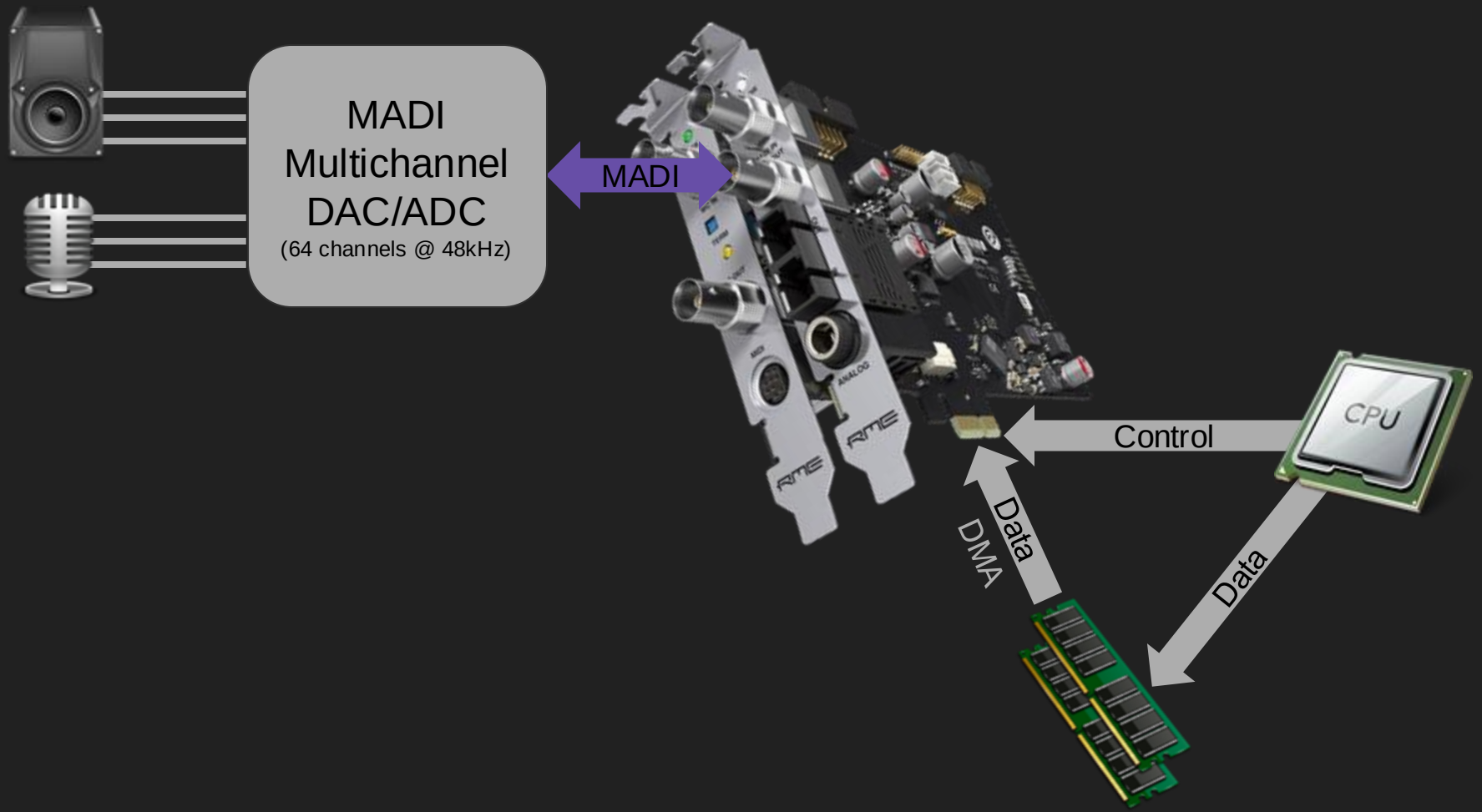
How many ioctl’s does ALSA define?

91


ALSA Userspace C Library
Userspace
Kernel
... 91 ...
ALSA Core
ALSA driver
8 required
7 optional

```
struct snd_pcm_ops {
    int (*open)(struct snd_pcm_substream *substream);
    int (*close)(struct snd_pcm_substream *substream);
    int (*ioctl)(struct snd_pcm_substream * substream,
            unsigned int cmd, void *arg);
    int (*hw_params)(struct snd_pcm_substream *substream,
            struct snd_pcm_hw_params *params);
    int (*hw_free)(struct snd_pcm_substream *substream);
    int (*prepare)(struct snd_pcm_substream *substream);
    int (*trigger)(struct snd_pcm_substream *substream, int cmd);
    int (*sync_stop)(struct snd_pcm_substream *substream);
    snd_pcm_uframes_t (*pointer)(struct snd_pcm_substream *substream);
    int (*get_time_info)(struct snd_pcm_substream *substream,
            struct timespec64 *system_ts, struct timespec64 *audio_ts,
            struct snd_pcm_audio_tstamp_config *audio_tstamp_config,
            struct snd_pcm_audio_tstamp_report *audio_tstamp_report);
    int (*fill_silence)(struct snd_pcm_substream *substream, int channel,
                unsigned long pos, unsigned long bytes);
    int (*copy)(struct snd_pcm_substream *substream, int channel,
            unsigned long pos, struct iov_iter *iter, unsigned long bytes);
    struct page *(*page)(struct snd_pcm_substream *substream,
                unsigned long offset);
    int (*mmap)(struct snd_pcm_substream *substream, struct vm_area_struct *vma);
    int (*ack)(struct snd_pcm_substream *substream);
};
```


CPU
Linux i2c driver
ALSA i2s driver
I²C Controller
I²S Controller
SDATA
SCLK
SDATA
BCLK
LRCLK
CONTROL
DATA
Codec


MADI
Multichannel
DAC/ADC
(64 channels @ 48kHz)
MADI
CPU
Control
Data
DMA
Data

What if we want more?

Field Programmable Gate Array

```vhdl
library ieee;
    use ieee.std_logic_1164.all;

entity blink_led is
    port (
        i_clk : in  std_logic;
        o_led : out std_logic;
    );
end blink_led;

architecture behavioral of blink_led is
    constant CLKS_PER_SEC : natural   := 100000000;
    signal   counter      : natural   := 0;
    signal   led          : std_logic := '0';
begin

    blink_proc : process(i_clk)
    begin
        if rising_edge(i_clk) then
            if counter < CLKS_PER_SEC then
                counter <= counter + 1;
            else
                led <= not led;
                counter <= 0;
            end if;
        end if;
    end process;

    o_led <= led;

end behavioral;
```

Hardware Description Language

Synthesis

Bitstream

Programming

Programming Language

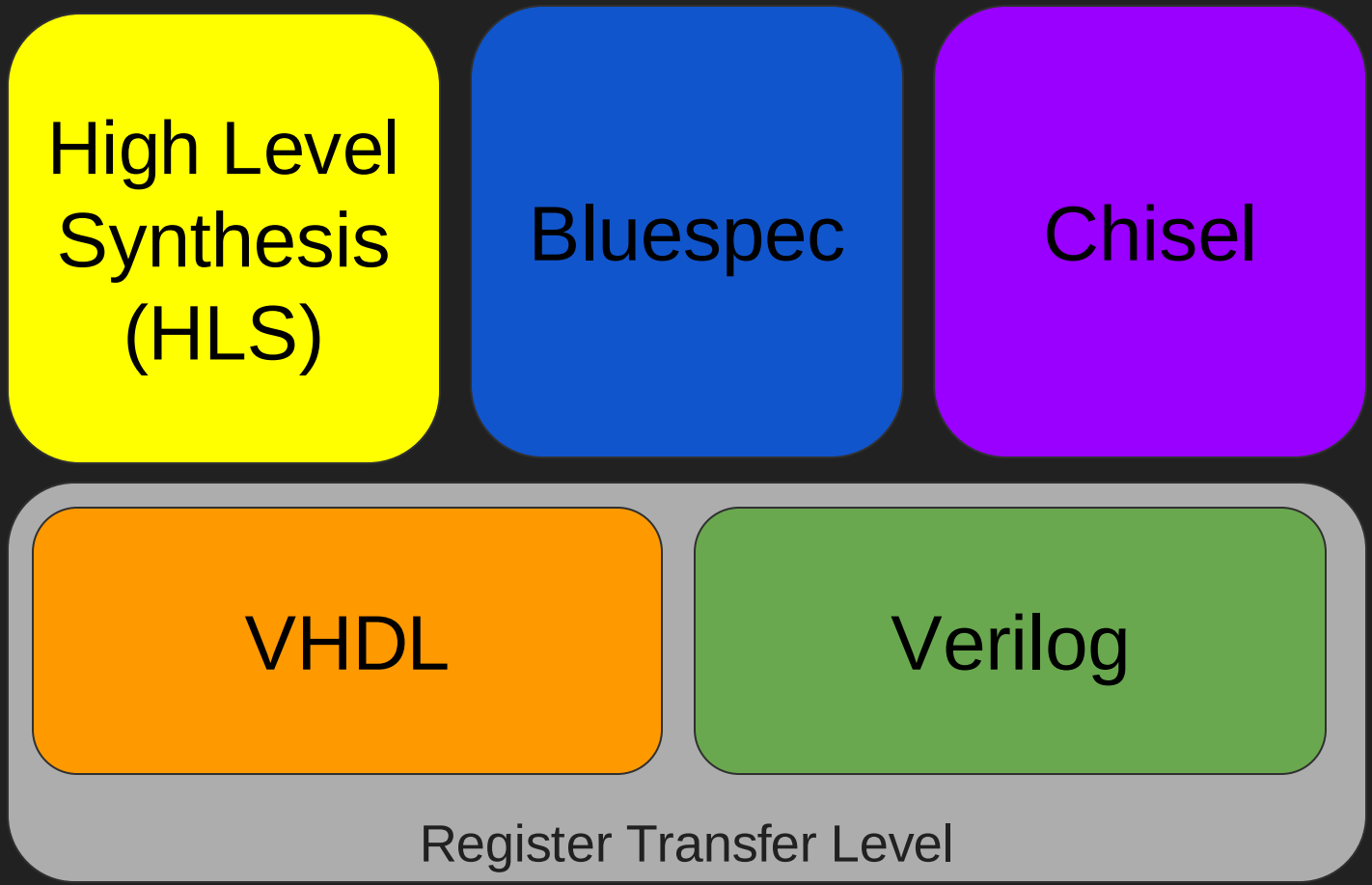
- Program begins at a specific point
- Execution proceeds step-by-step
- Data is shared by putting it into CPU registers or RAM
- Program exits once it finishes its work

Inherently serial

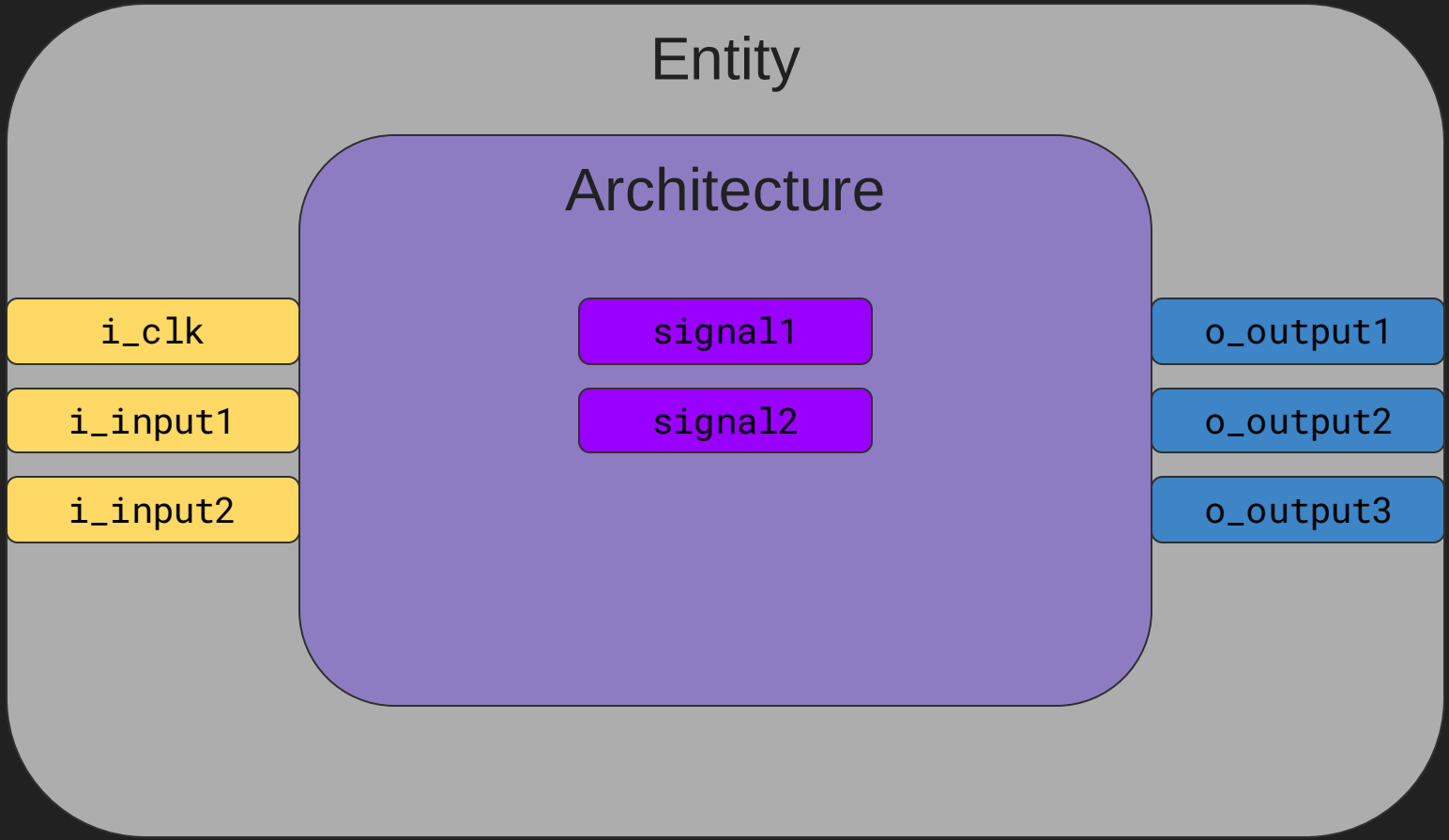
Hardware Description Language

- Entire design starts simultaneously
- All logic executes simultaneously
- Data is shared by running “wires” to every place where the data needs to be read
- Design runs forever

Inherently parallel


High Level Synthesis (HLS)
Bluespec
Chisel
VHDL
Verilog
Register Transfer Level

VHDL Crash Course


Entity
Architecture
i_clk
i_input1
i_input2
signal1
signal2
o_output1
o_output2
o_output3

```vhdl
library ieee;
    use ieee.std_logic_1164.all;

entity my_entity is
    port (
        i_clk     : in  std_logic;
        i_input1  : in  std_logic;
        i_input2  : in  std_logic;
        o_output1 : out std_logic;
        o_output2 : out std_logic;
        o_output3 : out std_logic;
    );
end my_entity;

architecture my_architecture of my_entity is
    signal signal1 : std_logic := '0';
    signal signal2 : std_logic := '0';
begin
    -- Implementation goes here
end my_architecture;
```

Dataflow Modelling

- Combinatorial logic drives outputs
- Uses concurrent signal assignment statements

```vhdl
library ieee;
    use ieee.std_logic_1164.all;

entity full_adder is
    port (
        i_a     : in  std_logic;
        i_b     : in  std_logic;
        i_carry : in  std_logic;
        o_sum   : out std_logic;
        o_carry : out std_logic;
    );
end full_adder;

architecture dataflow of full_adder is
begin
    o_sum   <= i_a xor i_b xor i_carry;

    o_carry <= (i_a and i_b)     or
               (i_a and i_carry) or
               (i_b and i_carry);
end dataflow;
```

```vhdl
library ieee;
    use ieee.std_logic_1164.all;

-- Assume this is defined in another package:
-- type State_t is (INIT, WAIT_FOR_DATA, PROCESSING, DONE);

entity state_display is
    port (
        i_state : in  State_t;
        o_leds  : out std_logic_vector(3 downto 0);
    );
end state_display;

architecture dataflow of state_display is
begin
    with state select
        o_leds <= "1000" when STATE = INIT,
                  "0100" when STATE = WAIT_FOR_DATA,
                  "0010" when STATE = PROCESSING,
                  "0001" when STATE = DONE,
                  "0000" when others;
end dataflow;
```

Behavioral Modelling

- Event-driven state machine drives outputs
- Uses process statements

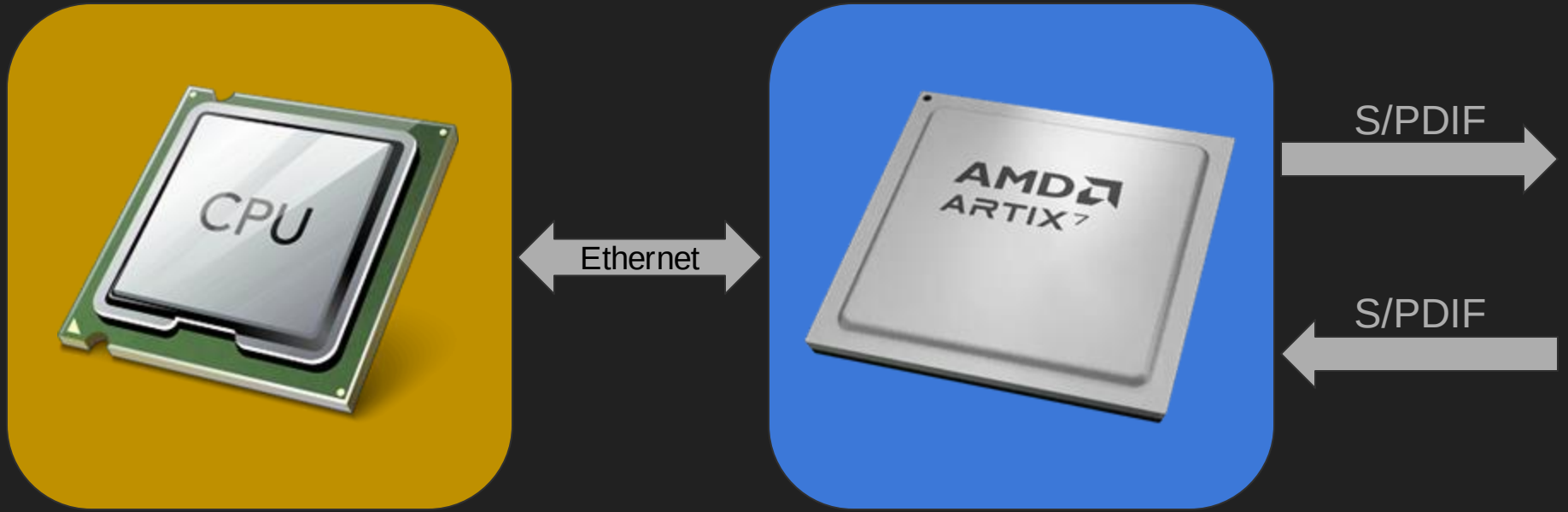
```vhdl
library ieee;
    use ieee.std_logic_1164.all;

entity blink_led is
    port (
        i_clk : in  std_logic;
        o_led : out std_logic;
    );
end blink_led;

architecture behavioral of blink_led is
    constant CLKS_PER_SEC : natural   := 100000000;
    signal   counter      : natural   := 0;
    signal   led          : std_logic := '0';
begin

    blink_proc : process(i_clk)
    begin
        if rising_edge(i_clk) then
            if counter < CLKS_PER_SEC then
                counter <= counter + 1;
            else
                led <= not led;
                counter <= 0;
            end if;
        end if;
    end process;

    o_led <= led;

end behavioral;
```

Structural Modelling

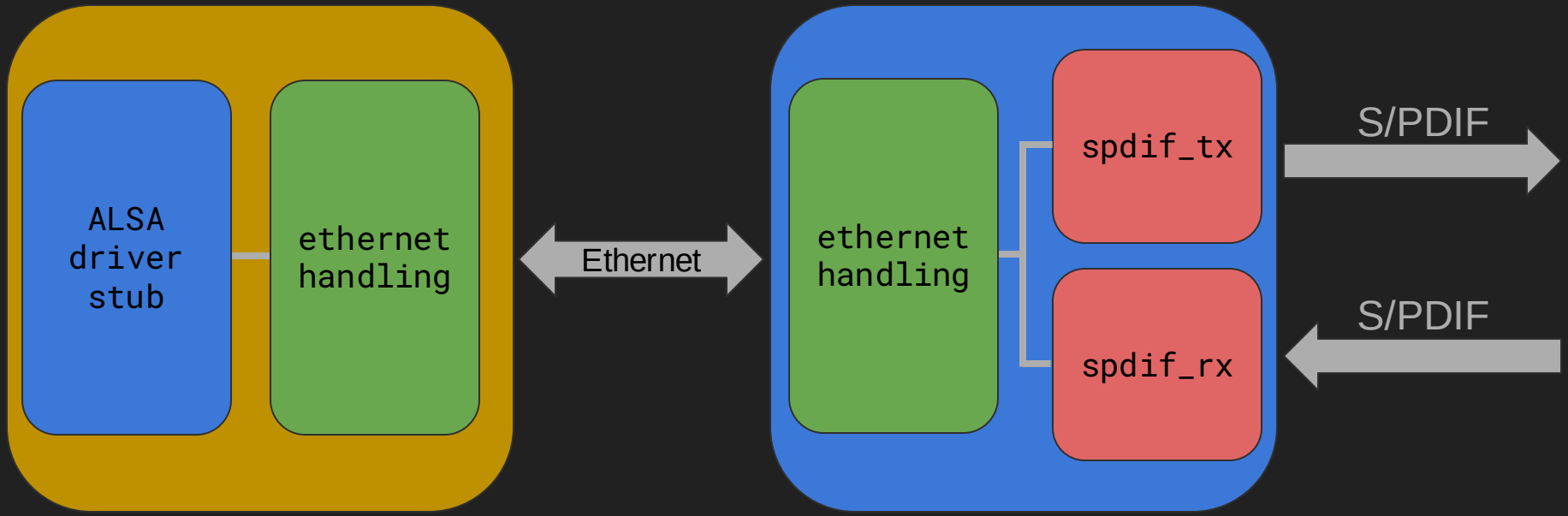
- Instantiated sub-entities drive outputs
- Uses component instantiation statements

```
library ieee;
    use ieee.std_logic_1164.all;

-- Bring in component definitions from our package
library work;
    use work.all;

entity two_bit_adder is
    port (
        i_a     : in  std_logic_vector(0 to 1);
        i_b     : in  std_logic_vector(0 to 1);
        o_sum   : out std_logic_vector(0 to 1);
        o_carry : out std_logic;
    );
end full_adder;

architecture structural of top is
    signal carry : std_logic := '0';
begin
    first_full_adder : work.full_adder
        port map (
            i_a => i_a(0),
            i_b => i_b(0),
            i_carry => '0',
            o_sum   => o_sum(0),
            o_carry => carry
        );

    second_full_adder : work.full_adder
        port map (
            i_a     => i_a(1),
            i_b     => i_b(1),
            i_carry => carry,
            o_sum   => o_sum(1),
            o_carry => o_carry
        );
end structural;
```

Vision



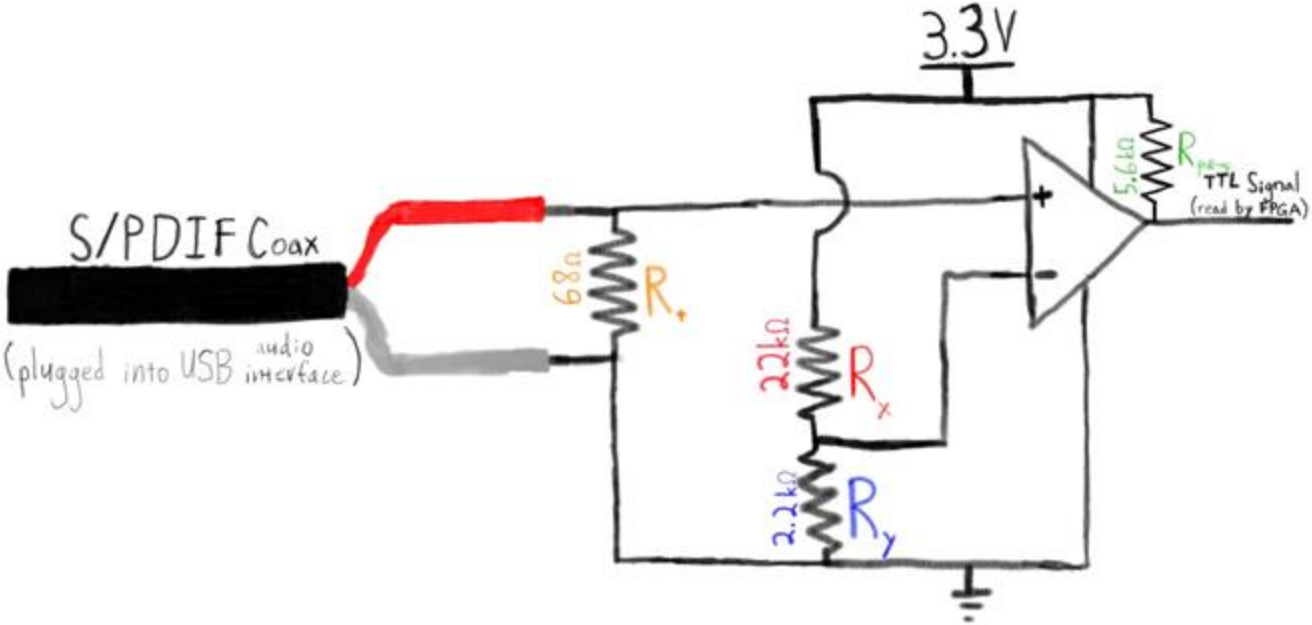
Vision



Phase 1
Phase 2
Phase 3

Phase 1 Gameplan

- Buy a book on VHDL
- Download vivado & build command-line workflow
  - Requires writing TCL scripts for IP generation, synthesis, and programming
- Write `spdif_tx` and `spdif_rx` entities in VHDL
- Create circuits to shift S/PDIF coax from TTL <-> 1V


3.3V
S/PDIF Coax
(plugged into USB audio interface)
68Ω
R_+
22kΩ
R_x
2.2kΩ
R_y
5.6kΩ
$R_{pull-up}$
TTL Signal
(read by FPGA)
+
-


Scope
200 ns/div
Samples
200 (2040)
Sample rate
100 MS/s
Trigger
-800 mV
Simple edge
Auto
Waveform
64
of 64
Instruments
Auto setup
Open
Save
Print
Full
pico
Technology
Views
Math channels
Serial decoding
Masks
Actions
Send feedback
2.0
V
1.6
1.2
0.8
0.4
0.0
-0.4
-0.8
-1.2
-1.6
-2.0
0.0 µs
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0


Stopped
Scope
500 ns/div
Samples
500 (2016)
Sample rate
100 MS/s
Trigger
800 mV
S/PDIF
0 %
Simple edge
Auto
Waveform
64
of 64
Instruments
Auto setup
Open
Save
Print
Full
pico
Technology
S/PDIF
DC
x1
±5 V
TTL
DC
x1
±5 V
Gen
Off
More...
Views
Measurements
Math channels
DeepMeasure
Serial decoding
Reference waveforms
Masks
Rulers
Actions
Notes
Send feedback
About PicoScope 7
5.0
V
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
0.0 µs
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Stopped
Scope
200 ns/div
Samples
200 (2040)
Sample rate
100 MS/s
Trigger
900 mV
S/PDIF
-1 %
Simple edge
Auto
Waveform
64
of 64
Instruments
Auto setup
Open
Save
Print
Full
pico
Technology
S/PDIF
DC
x1
±5 V
TTL
DC
x1
±5 V
Gen
Off
More...
Views
Measurements
Math channels
DeepMeasure
Serial decoding
Reference waveforms
Masks
Rulers
Actions
Notes
Send feedback
About PicoScope 7
5.0
V
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
0.2 µs
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

#12

#2

Joined Nov 30, 2010

18,224

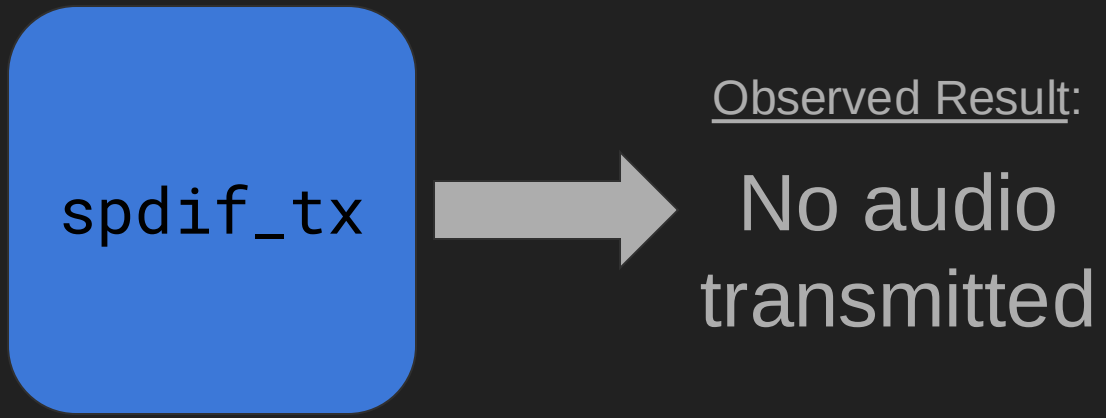
Apr 22, 2016

One way to achieve fractional gain is to use 2 resistors and no amplifier. Another way is to use the inverting configuration and make the feedback resistor smaller than the input resistor. If you don't like the polarity inversion, add another inverting gain stage with a gain of one.


Stopped
Scope
1 µs/div
Samples
1000 (2000)
Sample rate
100 MS/s
Trigger
3.1 V
TTL
-1 %
Simple edge
Auto
Waveform
64
of 64
Instruments
Auto setup
Open
Save
Print
Full
pico
Technology
TTL
DC
x1
±5 V
Amplifier Out
DC
x1
±5 V
Gen
Off
More...
Views
Measurements
Math channels
DeepMeasure
Serial decoding
Reference waveforms
Masks
Rulers
Actions
Notes
Send feedback
About PicoScope 7
5.0
V
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
1.0 µs
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0


Stopped
Scope
500 ns/div
Samples
500 (2016)
Sample rate
100 MS/s
Trigger
3.2 V
Simple edge
Auto
Waveform
64
of 64
Instruments
Auto setup
Open
Save
Print
Full
pico
Technology
A
DC
x1
±5 V
B
DC
x1
±5 V
Gen
Off
More...
Views
Measurements
Math channels
DeepMeasure
Serial decoding
Reference waveforms
Masks
Rulers
Actions
Notes
Send feedback
About PicoScope 7
5.0
V
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
0.5 µs
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


NE5532P
OPA2604AP
Slow (0.5 kHz)
Fast (0.5 MHz)

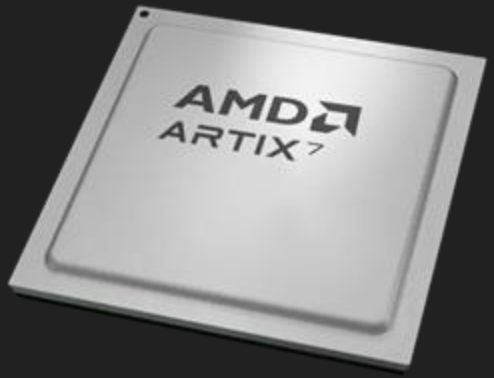

spdif_tx
Observed Result:
No audio transmitted

Phase 2 Gameplan

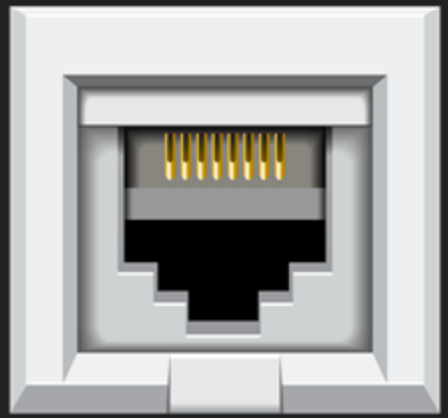
- Buy a book on Linux device drivers
- Find an example ALSA driver online
  - `dummy.c`, ~1200 lines of C code without even a single comment
- Refactor into multiple source files
- Study the code until I understood it

Phase 3 Gameplan

- Build ability to send/recv ethernet from FPGA



Ethernet PHY



MII GMII

RMII RGMII

Phase 3 Gameplan

- Build ability to send/recv ethernet from FPGA
- Build ability to send/recv ethernet from driver
- Design wire protocol
- Build out device handshake & heartbeating
- Copy audio data into ethernet frames
- Build audio buffer in FPGA to store samples
- Connect audio buffer to `spdif_rx` and `spdif_tx`

Demo


SNL
ANY QUESTIONS?