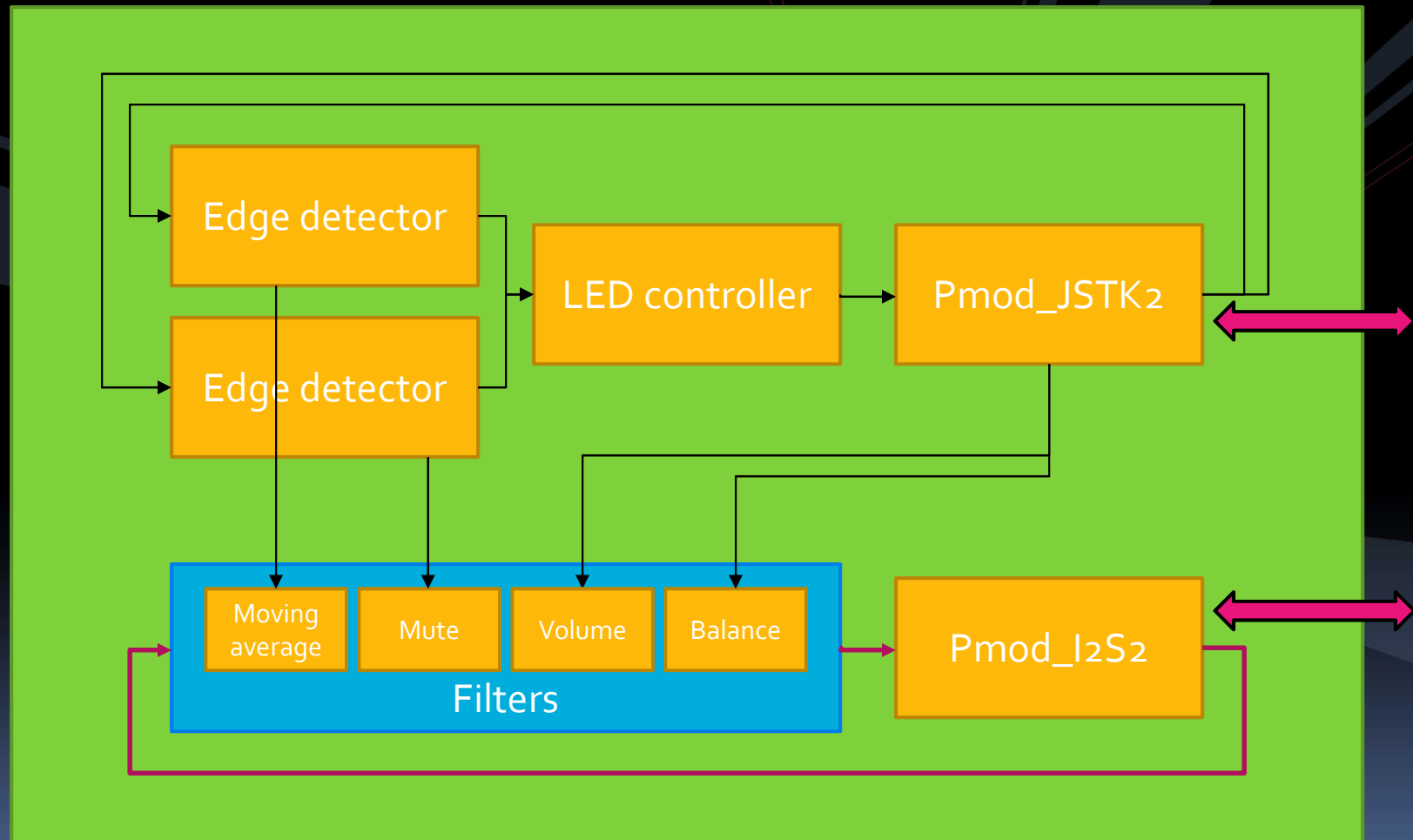


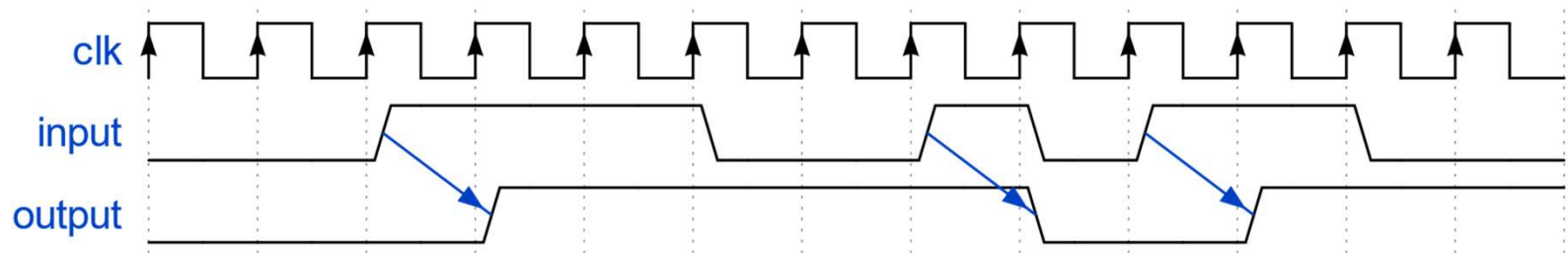
LAB 4 SOLUTION

Top Construction (overview)



Edge detector

This module “detects” a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition (selectable by generic). At each transition, it toggles the output.



Edge detector implementation

This can be done by sampling the input at each clock cycle and comparing the current value with the old one, sampled at the last clock cycle.

```
input_signal_old  <= input_signal;  
  
if input_signal_old = '0' and input_signal = '1' then  
    output_signal_int <= not output_signal_int;  
end if;
```

LED controller

The LED controller has just to select the correct LED color based on two signals: ***mute_enabled*** (whether the audio is muted or not) and ***filter_enable*** (whether the moving average filter is enabled or not).

	filter_enable = 0	filter_enable = 1
mute_enable = 0	GREEN	BLUE
mute_enable = 1	RED	RED

LED controller implementation

This can be done with simple combinatorial logic, both in dataflow

```
led_r    <= x"ff" when mute_enable = '1' else x"00" ;  
led_g    <= x"ff" when mute_enable = '0' and filter_enable = '0' else x"00" ;  
led_b    <= x"ff" when mute_enable = '0' and filter_enable = '1' else x"00" ;
```

or in behavioral

```
...  
if mute_enable = '1' then  
    led_r    <= (others => '1');  
elsif filter_enable = '1' then  
    led_b    <= (others => '1');  
else  
    led_g    <= (others => '1');  
end if;
```



Filters

On the audio path, transferred through an AXI4-Stream interface, we have four “filters”:

- Mute controller
- Moving average filter
- Balance controller
- Volume controller



They are connected in daisy chain and their order is not particularly important.



Mute controller

To mute the audio stream it is sufficient to remove its data component (by zeroing it or by keeping it constant to the last value) for both audio channels.

It is important to remember that the data samples must always be transferred at the 44.1 kHz rate, it is not possible to decimate them (for example, by keeping tvalid low).

Mute controller implementation

To do so, one solution is to introduce a simple combinatorial circuit which zeros the data channel when ***mute_enable*** is high.

Control signals pass unmodified

```
m_axis_tvalid    <= s_axis_tvalid;  
m_axis_tlast     <= s_axis_tlast;  
s_axis_tready    <= m_axis_tready;  
  
m_axis_tdata     <= s_axis_tdata when mute_enable = '0' else (others => '0');
```

tdata is zeroed when mute_enable is high



Moving average filter

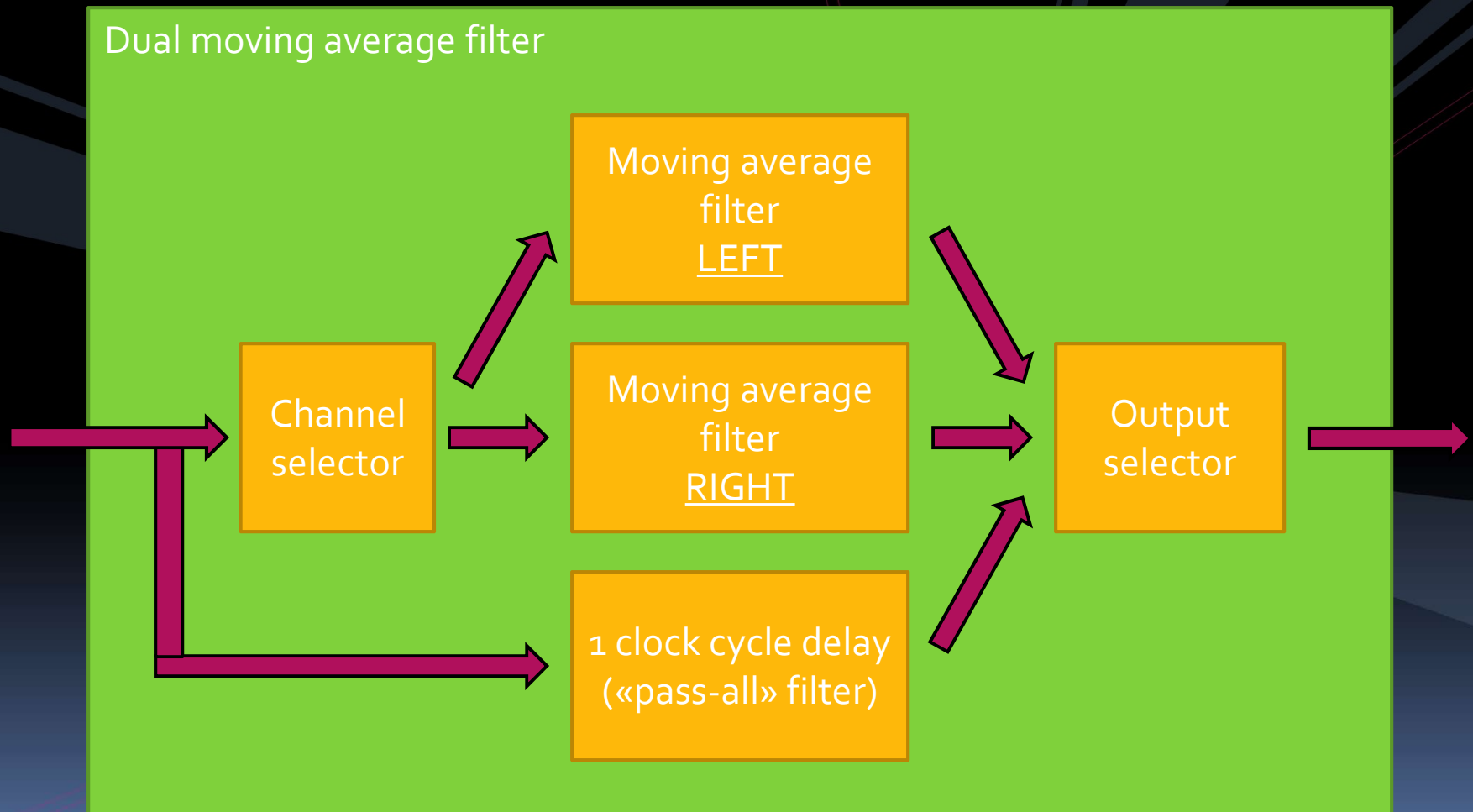
The two audio channels are independent, so they should be filtered independently.

The easiest way to do so is to design a single-channel moving average filter, instantiate it twice and describe the logic to route the data samples to the correct filter.



Additionally, data is transferred unmodified if *filter_enable* is low.

Moving average overview



Moving average filter notes

Note that the filters always process the input data, regardless of the value of *filter_enable*.


In this way, at every moment, we have both the filtered and unfiltered streams and we can freely choose one or another based on the value of *filter_enable*, without having to wait for the shift registers of the moving average filters to fill up.



Balance controller

The scale of the balance requires a multiplication by 2^{**n} , which translates into bit shifts.

The balance controller is split in two parts:

1. Compute the number of positions to shift the data, based on the value of the joystick axis.
 2. Shift the correct channel by the required amount
- 

Balance controller implementation

Step 1.

Move the 0 to 1023 scale
to a -512 to 511 one

Center the steps

```
bal_s_f := signed(balance) - 2**(balance'length - 1) + 2**(BAL_STEP_SIZE - 1);  
bal_s <= bal_s_f(balance'high downto BAL_STEP_SIZE)
```

Ignore the last BAL_STEP_SIZE bits (to have steps of 64)

Balance controller implementation

Step 2 (part 1).

If we have to attenuate the left channel
AND we are holding a left sample

```
if balance_shifted > 0 and s_axis_tlast = '0' then  
    m_axis_tdata <= std_logic_vector(shift_right(signed(s_axis_tdata),  
        to_integer(balance_shifted)));
```

```
elsif balance_shifted < 0 and s_axis_tlast = '1' then  
    m_axis_tdata <= std_logic_vector(shift_right(signed(s_axis_tdata),  
        to_integer(-balance_shifted)));
```

...

If we have to attenuate the right channel
AND we are holding a right sample

Shift right by the
correct number of
positions

Shift right by the correct
number of positions (equal
to -balance_shifted)

Balance controller implementation

Step 2 (part 2).

If the previous conditions are not satisfied (we have to attenuate the left/right channel, but we are holding a right/left sample)

...

else

m_axis_tdata <= s_axis_tdata;


end if;

Do not modify the data



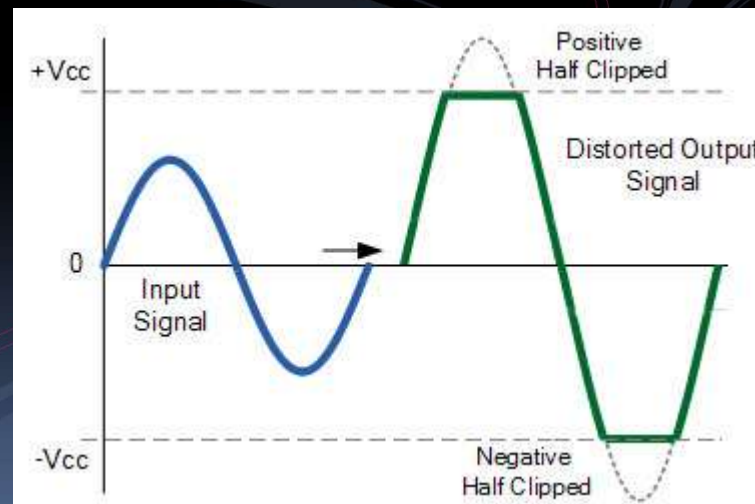
Volume controller

The volume controller is mostly identical to the balance one, with just two differences:

1. left/right channels are amplified/attenuated by the same value
 2. we can also amplify, so we have to make sure that the signal saturates, and not wraps around.
- 

Volume controller: clipping

When amplifying the signal, we should check whether the resulting value fits in the 24-bit signed integer; if not, we should saturate it to the maximum/minimum value.



Volume controller implementation

Part 1

If we have to amplify, data might overflow: use a larger variable to store the data.

```
scaled_data := resize(signed(s_axis_tdata), scaled_data'length);  
if volume_shifted >= 0 then  
    scaled_data := shift_left(scaled_data, to_integer(volume_shifted));  
...
```

If we have to amplify

Shift left by the required amount
(thanks to the temporary variable, this
will not overflow)

Volume controller implementation

Part 2

If the amplified value is over the maximum value

force it to the maximum value

...

```
if scaled_data > 2**s_axis_tdata'high - 1 then
```

```
    scaled_data := to_signed(2**s_axis_tdata'high - 1, scaled_data'length);
```

```
elsif scaled_data < -2**s_axis_tdata'high then
```

```
    scaled_data := to_signed(-2**s_axis_tdata'high, scaled_data'length);
```

```
end if;
```

...

If the amplified value is below the minimum value

force it to the minimum value

If the amplified value falls between the limits, just keep it

Volume controller implementation

Part 3

If instead we have to attenuate

Attenuate it, like with the balance controller (but without checking the channel)

...

else

scaled_data := shift_right(scaled_data, to_integer(-volume_shifted));

end if;

m_axis_tdata <= std_logic_vector(scaled_data(m_axis_tdata'range));

At the end, take just the 24 Least Significant bits