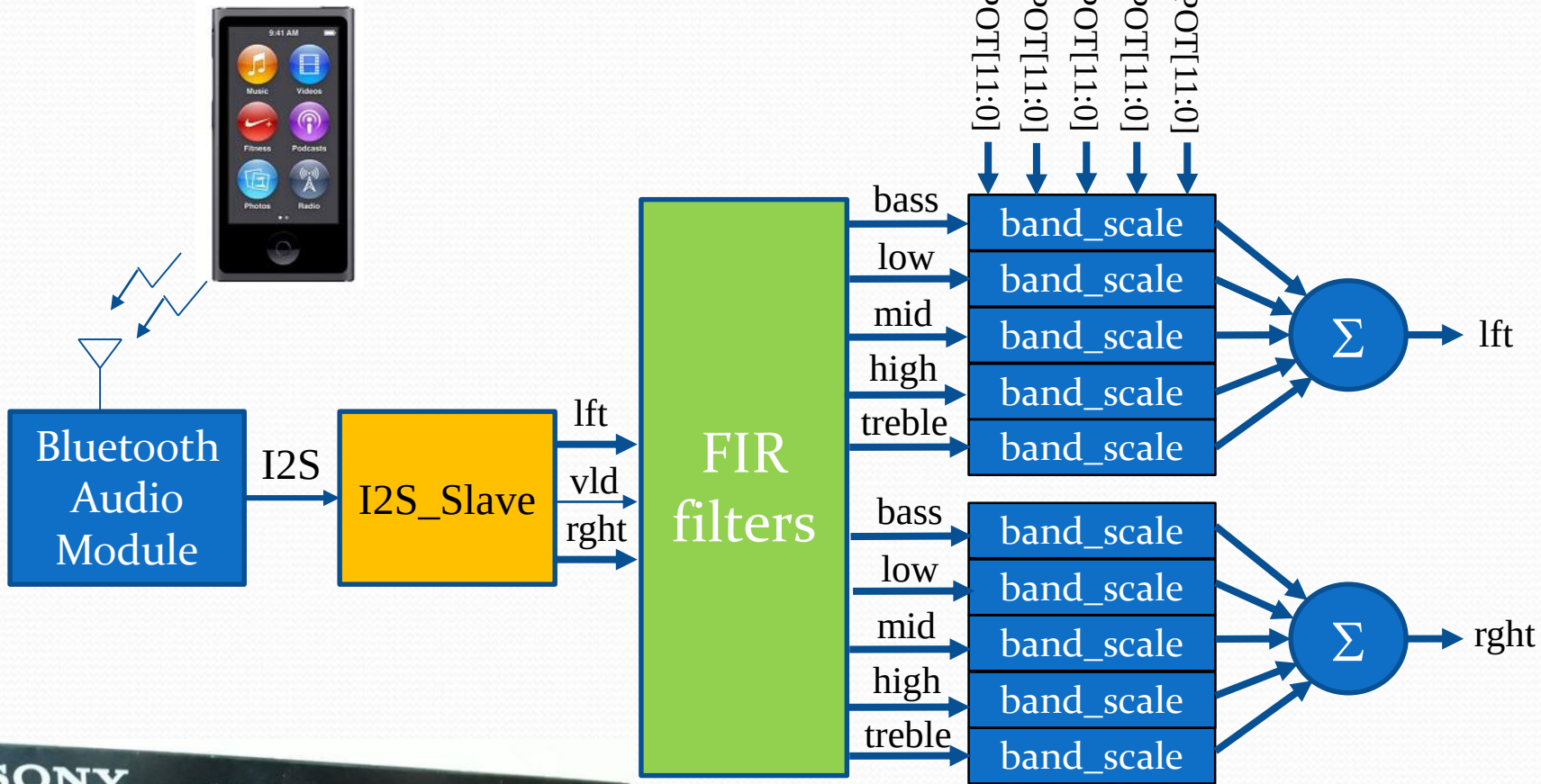
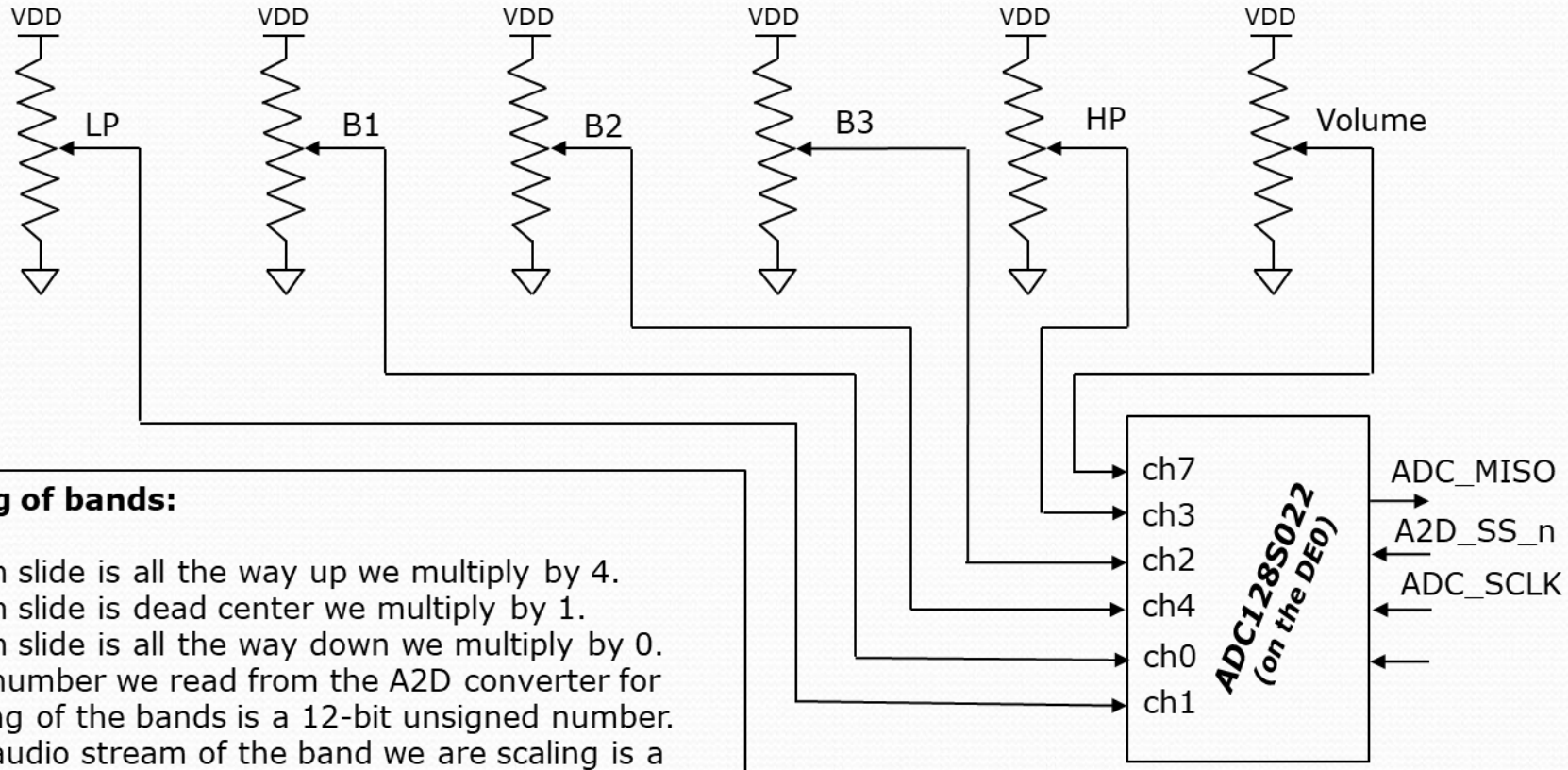
Exercise 6 (Background on Project):



Exercise 6 (band_scale.v for Project):



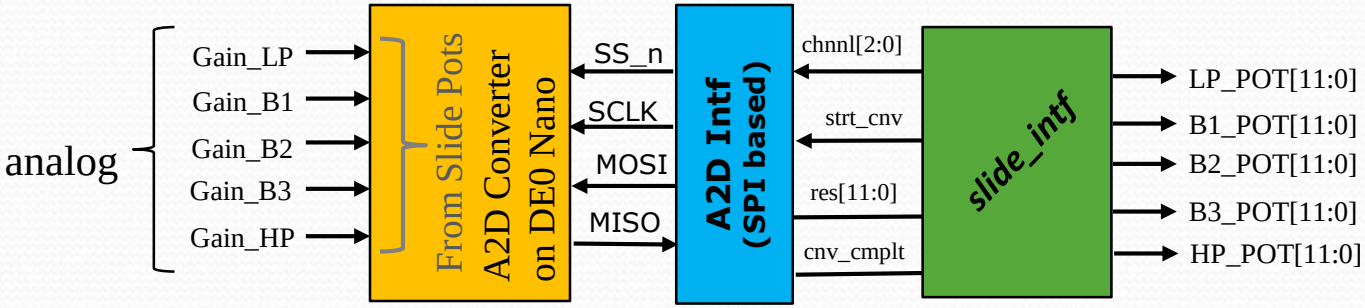
Scaling of bands:

- When slide is all the way up we multiply by 4.
- When slide is dead center we multiply by 1.
- When slide is all the way down we multiply by 0.
- The number we read from the A2D converter for scaling of the bands is a 12-bit unsigned number.
- The audio stream of the band we are scaling is a 16-bit **signed** number.
- Think about this…this is a little tricky.

Scaling of volume:

- The volume of the audio stream (coming from the RN-52 via I2S) can be scaled using the source device.
- The intent of the volume slider is to have a quick easy way to reduce volume. When the slider is all the way up we should be multiplying by unity. When it is all the way down we multiply by zero.

Scaling of Bands



Given you have a block (***slide_intf***) that performs A2D conversions on the slide potentiometers continuously. You now have a 12-bit unsigned number that represent the amount to scale each band. Now how to make use of those scaling numbers?

For Band scaling we need to use the square of the potentiometer reading as our scaling factor. This gives us the 4X gain when the pot is at full, 1X gain when it is centered, and zero when it is all the way low. (Human perception of sound is to the square root of amplitude, so we have to use square of the pot value to make the effect seem linear)

Starting with a 12-bit unsigned potentiometer reading we square it to get a 24-bit unsigned product. One can next just use bits [23:12] of this product as the 12-bit number to scale the respective FIR output by. Next we want to extend this 12-bit unsigned number to a 13-bit signed number. This is simply done by appending a 1'b0 as the MSB. This **signed** 13-bit factor will be multiplied by the 16-bit **signed** result of the respective FIR filter output to produce a 29-bit **signed** product. We will use bits [25:10] as the result (feel free to check my math…by using bits [25:10] a full POT reading would give 4X gain for that channel).

However, what about the data in bits [28:26] that we are discarding? Was there overflow? Was the full result more positive or more negative than we can represent with bits [25:10]? You need to check for overflow and saturate if necessary.

Exercise 6 (band_scale.v a.k.a HW2 Prob4):

- Create Verilog for a module called **band_scale.v**. This module should implement the functionality shown on the slide above. It should have the following interface:

Signal:	Direction:	Description:
POT[11:0]	In	Represents A2D reading from the slide potentiometer used to set the gain for the frequency band of the equalizer. This should be regarded as an **unsigned** number.
audio[15:0]	In	Represents the **signed** 16-bit audio signal coming from a FIR filter for the particular band. This number is scaled by the potentiometer reading squared to produce the scaled result.
scaled[15:0]	Out	**Signed** 16-bit output that is the scaled result of the input audio by the slide potentiometer reading.

- Create a testbench and check against some corner cases. Is your saturation working?
- Turn in whatever you have done to the dropbox by end of Monday's class.

Some things to note:

- Pay attention to what operands are considered signed and what ones are not.
- To perform a signed multiply the source and destination signals need to be defined as ***signed***. (i.e. wire signed [14:0] op1,op2;)
- The saturation detection using bits [28:25] is important.

Hints on Saturating:

- Lets take a look at some examples saturating an 8-bit signed number to a 5-bit signed number.

  Lets say you had the 8-bit number: **01010110**
  You know it is positive because the MSB is 0
  What is the greatest positive number we can represent in 5-bits? … **01111**
  Is **01010110** greater than **01111**? Yes. How do we know? Because it is positive and it has bit(s) in the [6:4] range set. So we saturate to **01111**

  Lets say you had the 8-bit number: **11010110**
  You know it is negative because the MSB is 1
  What is the greatest negative number we can represent in 5-bits? … **10000**
  Is **11010110** more negative than **10000**? Yes. How do we know? Because it is negative and it has bit(s) in the [6:4] range clear. So we saturate to **10000**

  Lets say you had the 8-bit number: **11110110**
  You know it is negative because the MSB is 1
  What is the greatest negative number we can represent in 5-bits? … **10000**
  Is **11110110** more negative than **10000**? No. How do we know? Because it is negative but all the bits in the [6:4] are also set. So we simply keep bits [4:0] as our 5-bit result.

(more on next slide)

Hints on Saturating:

- So..from the previous slide you can derive the following pseudo code:

```
if (number is negative) {
  if (any upper bits (in certain range) are zero) {
    we set flag to know to saturate negative
  }
}

if (number is positive) {
  if (any upper bits (in certain range) are one) {
    we set flag to know to saturate positive
  }
}

if (sat negative flag set)
  saturate result to most negative number
else if (sat positive flag set)
  saturate result to most positive number
else
  just copy over bits in range of interest
```

Of course this is **dataflow** Verilog, so we are NOT using **if**. Instead use tertiary assign statements.

What is an effective way to tell if any bits in a range are 1 or 0? Think about the reduction operator.