From API to Electrons

A journey on Tessel from JS to the metal...

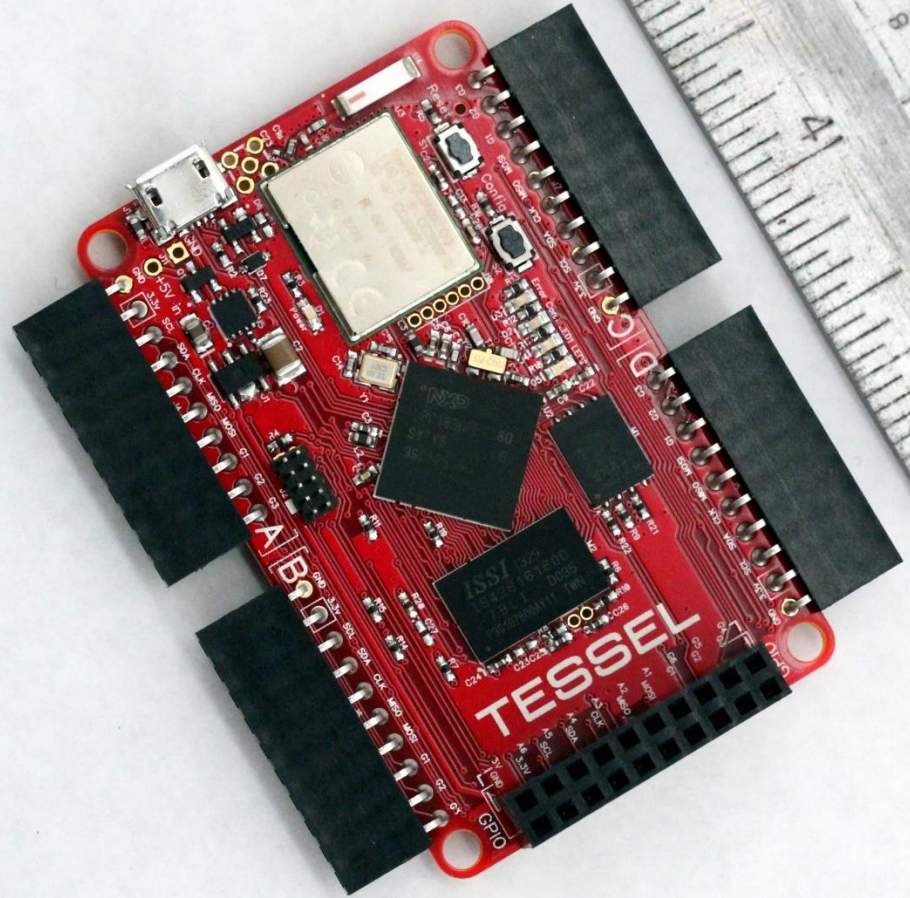
...and back

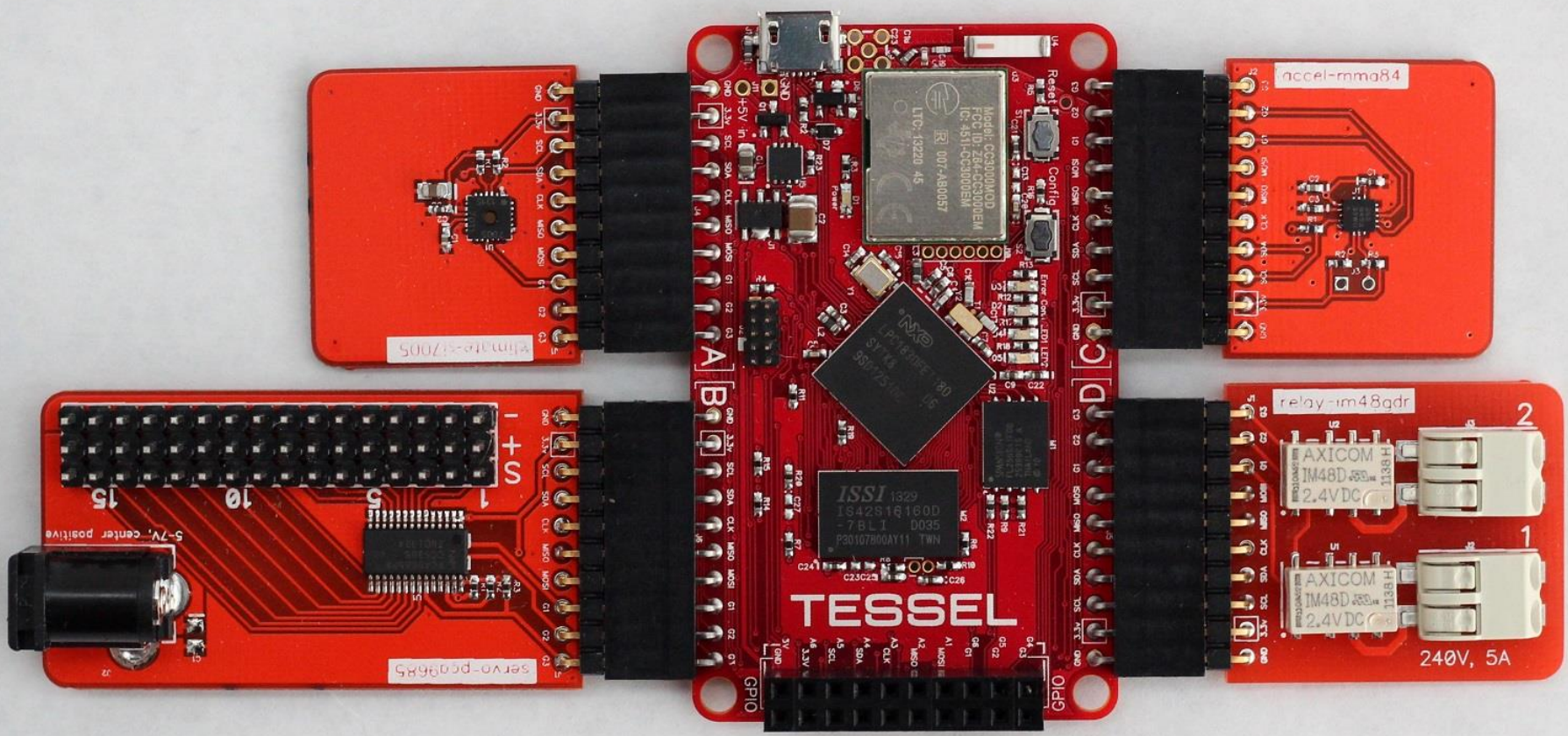
Eric Kolker Technical Machine FIT 2014

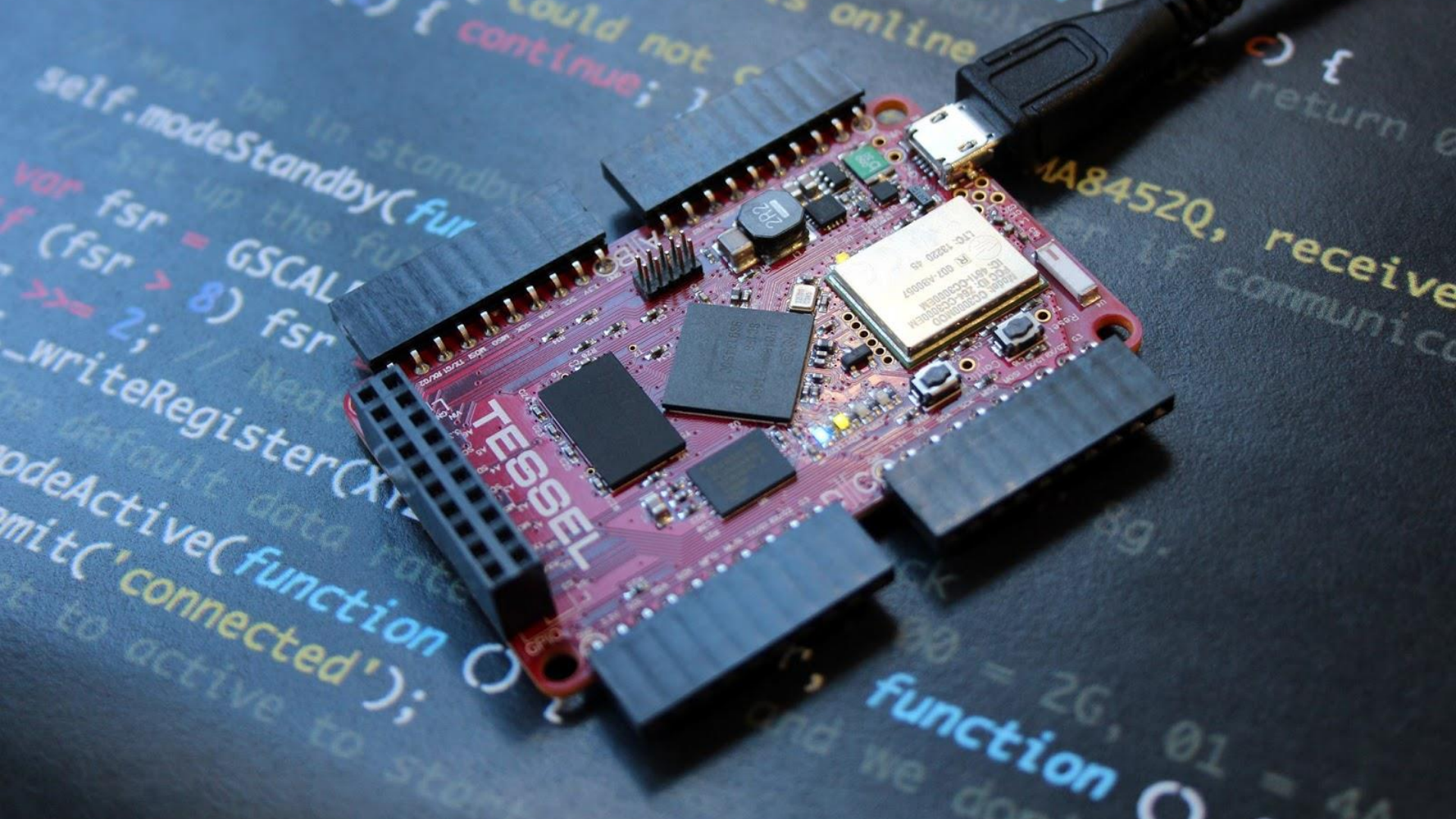
Intro

- I'm Eric Kolker
- Electrical engineer, @technicalhumans
- Our adventure today:

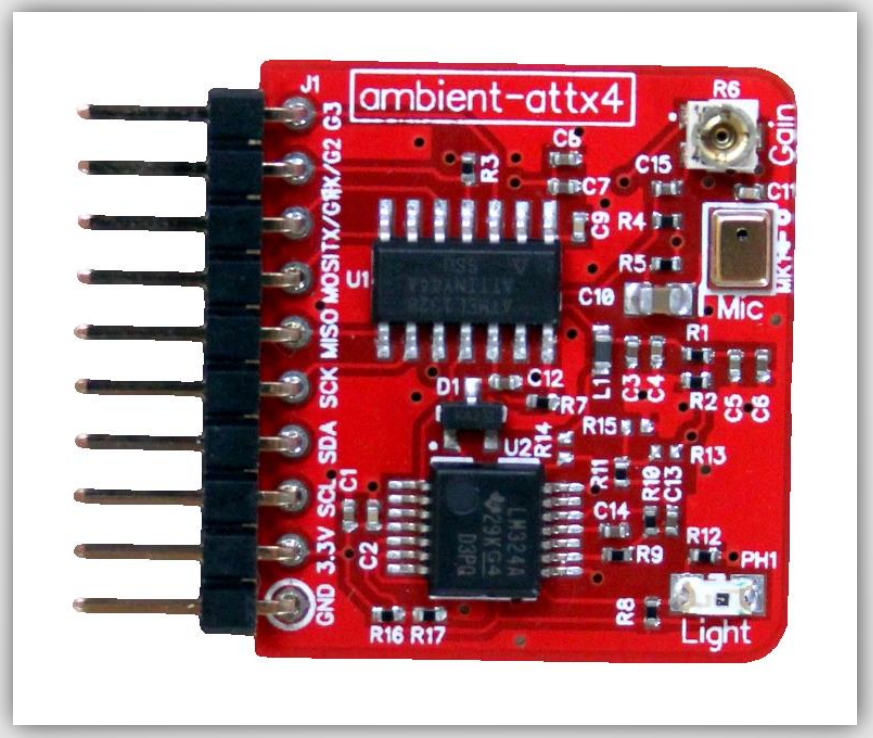
What happens when you set a noise trigger with Tessel and the Ambient module?


TESSEL
A
B
C
D
GPIO

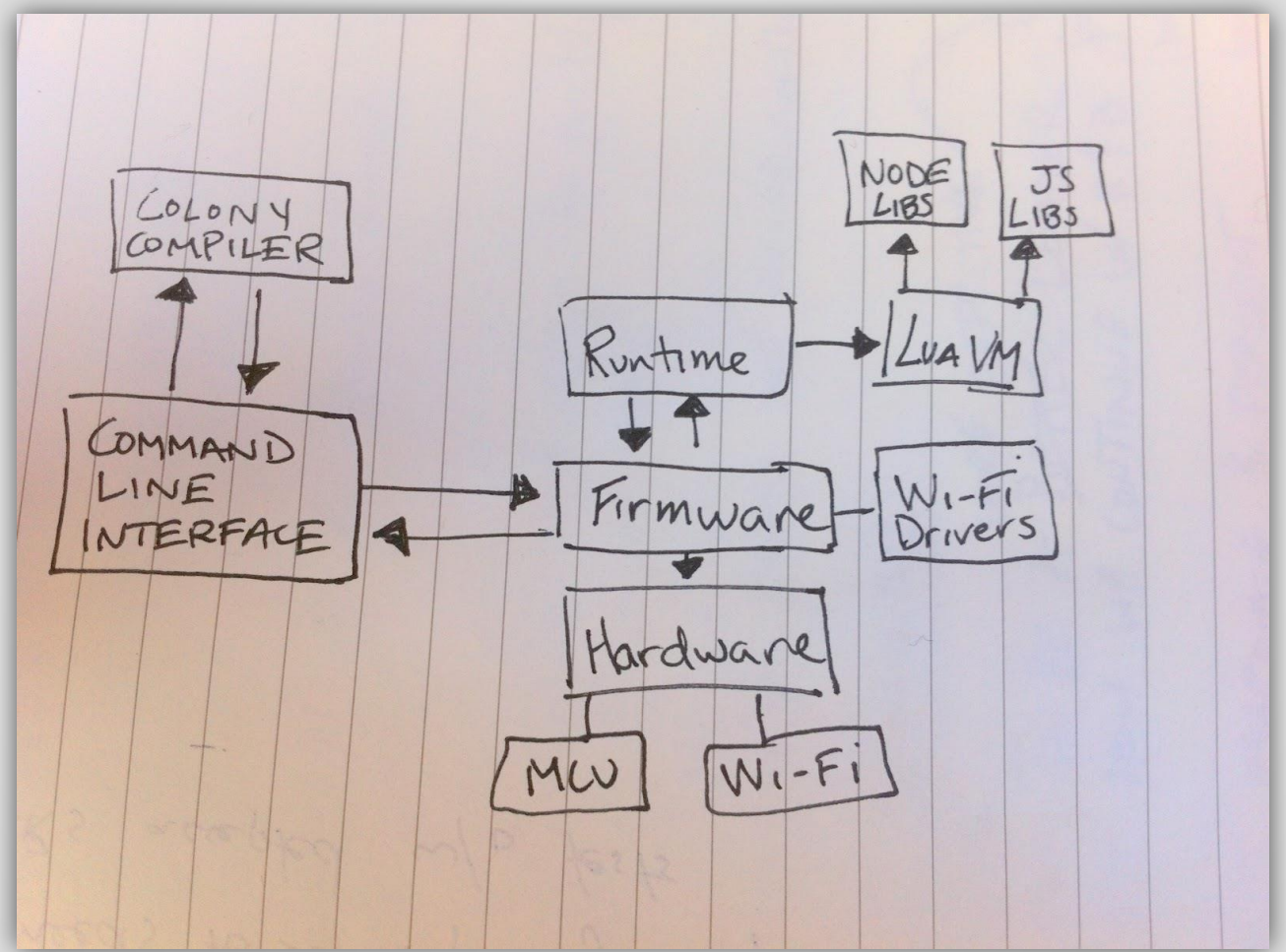

TESSEL
accel-mma84
relay-im48gdr
AXICOM
IM48D
2.4VDC
240V, 5A
servo-pca9685
5-7V_center positive
ISSI
IS42S16160D
-7BLI
GPIO
GPIO


TESSEL

Case study: the Ambient Module

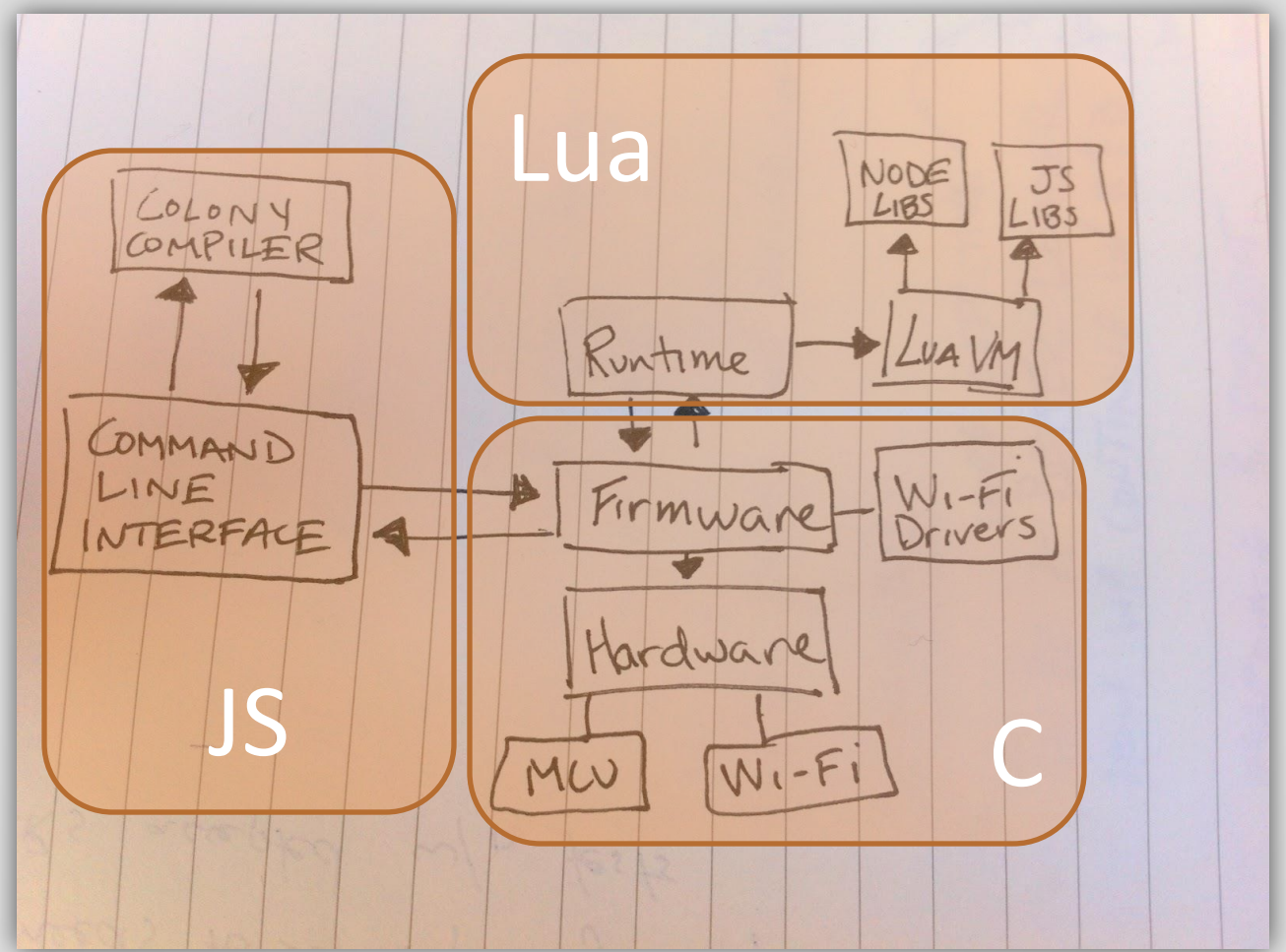


```
var tessel = require('tessel');
var ambientlib = require('ambient-attx4');
var ambient = ambientlib.use(tessel.port['A']);

ambient.on('ready', function () {
  // Set a sound level trigger
  ambient.setSoundTrigger(0.1);

  ambient.on('sound-trigger', function(data) {
    console.log("Something happened with sound: ", data);
    // Clear it
    ambient.clearSoundTrigger();
    // After 1.5 seconds reset sound trigger
    setTimeout(function () {
      ambient.setSoundTrigger(0.1);
    }, 1500);
  });
});

ambient.on('error', function (err) {
  console.log(err)
});
```


COLONY COMPILER
NODE LIBS
JS LIBS
Runtime
LUA VM
COMMAND LINE INTERFACE
Firmware
Wi-Fi Drivers
Hardware
MCU
Wi-Fi

`tessel run ambient.js`

- JS transpiled to Lua via Colony
  - Embeddable, low memory footprint, semantically similar to JS
- Dependencies bundled
- Bundle stored in RAM, executed
  - **`tessel push`** = saved in flash


Lua
NODE LIBS
JS LIBS
COLONY COMPILER
Runtime
LUA VM
COMMAND LINE INTERFACE
Firmware
Wi-Fi Drivers
Hardware
JS
MCU
Wi-Fi
C

```
var tessel = require('tessel');
tessel.leds[0].rawWrite(1);
```

```
Pin.prototype.rawWrite = function rawWrite(value) {
  hw.digital_write(this.pin, value ? hw.HIGH : hw.LOW)
  return this;
};
```

```
LUALIB_API int luaopen_hw(lua_State* L)
{
  luaL_reg regs[] = {

    { "digital_write", l_hw_digital_write },
```

```
var tessel = require('tessel');
tessel.leds[0].rawWrite(1);
```

```
Pin.prototype.rawWrite = function rawWrite(value) {
  hw.digital_write(this.pin, value ? hw.HIGH : hw.LOW)
  return this;
};
```

```
LUALIB_API int luaopen_hw(lua_State* L)
{
  luaL_reg regs[] = {

    { "digital_write", l_hw_digital_write },
```

```
static int l_hw_digital_write(lua_State* L)
{
```

```
var tessel = require('tessel');
tessel.leds[0].rawWrite(1);
```

```
Pin.prototype.rawWrite = function rawWrite(value) {
  hw.digital_write(this.pin, value ? hw.HIGH : hw.LOW)
  return this;
};
```

```
LUALIB_API int luaopen_hw(lua_State* L)
{
  luaL_reg regs[] = {

    { "digital_write", l_hw_digital_write },
```

```
static int l_hw_digital_write(lua_State* L)
{
  uint32_t pin = (uint32_t)lua_tonumber(L, ARG1);
  uint32_t level = (uint32_t)lua_tonumber(L, ARG1 + 1);

  hw_digital_write(pin, level);

  return 0;
```

```
LUALIB_API int luaopen_hw(lua_State* L)
{
    luaL_reg regs[] = {

        { "digital_write", l_hw_digital_write },
```

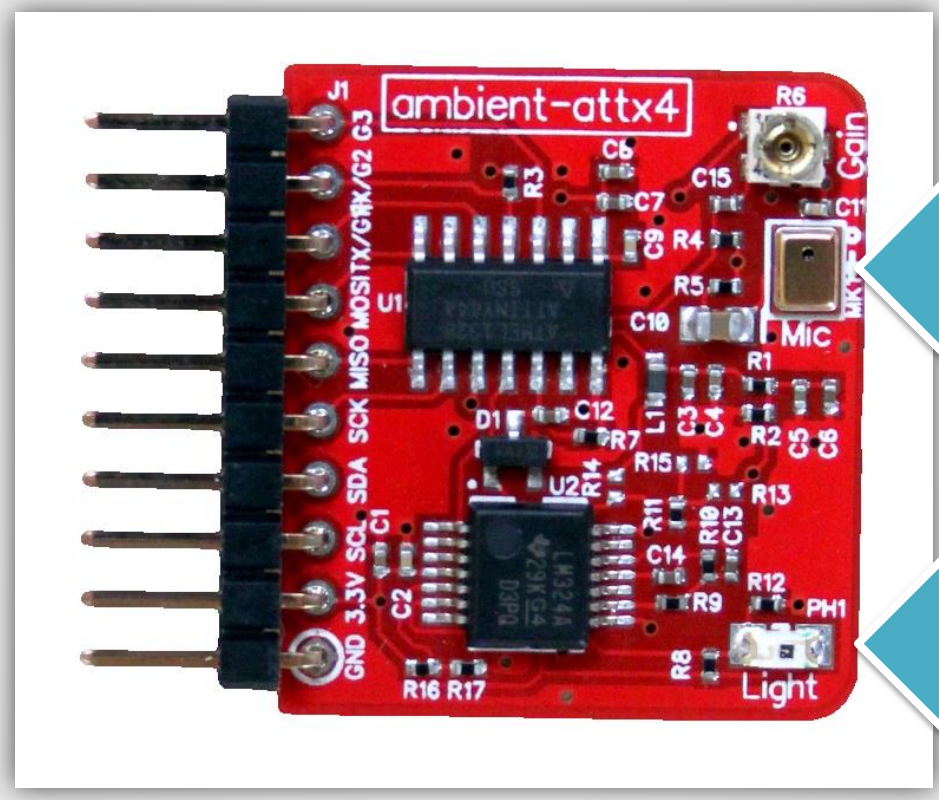
```
static int l_hw_digital_write(lua_State* L)
{
    uint32_t pin = (uint32_t)lua_tonumber(L, ARG1);
    uint32_t level = (uint32_t)lua_tonumber(L, ARG1 + 1);

    hw_digital_write(pin, level);

    return 0;
}
```

```
void hw_digital_write (size_t ulPin, uint8_t ulVal)
{
    if (ulVal != HW_LOW) {
        GPIO_SetValue(g_APinDescription[ulPin].portNum, 1 << g_APinDescription[ulPin].bitNum);
    } else {
        GPIO_ClearValue(g_APinDescription[ulPin].portNum, 1 << (g_APinDescription[ulPin].bitNum)
    }
```

```
{
    uint32_t pin = (uint32_t)lua_tonumber(L, ARG1);
    uint32_t level = (uint32_t)lua_tonumber(L, ARG1 + 1);

    hw_digital_write(pin, level);

    return 0;
}
```

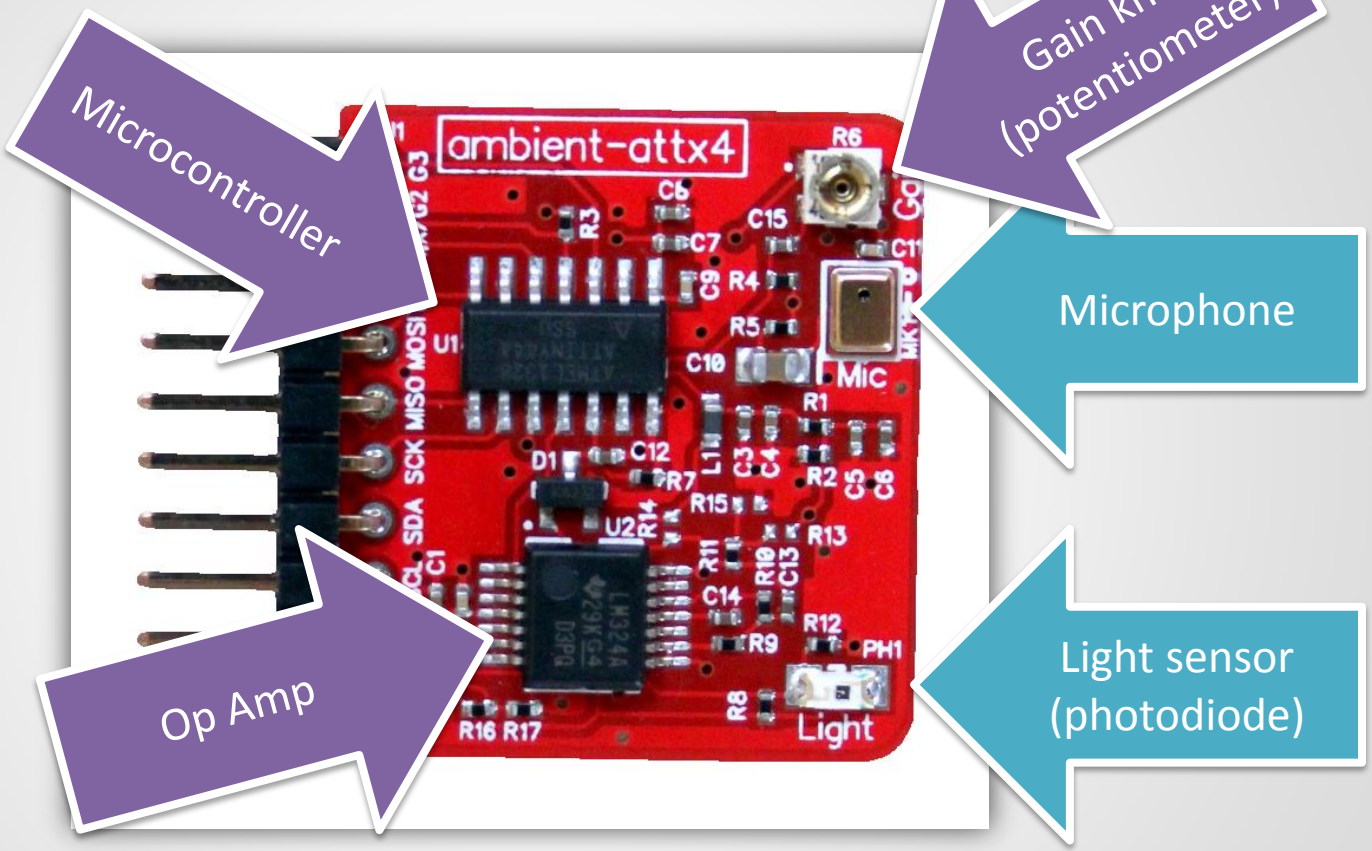
```
void hw_digital_write (size_t ulPin, uint8_t ulVal)
{
    if (ulVal != HW_LOW) {
        GPIO_SetValue(g_APinDescription[ulPin].portNum, 1 << g_APinDescription[ulPin].bitNum);
    } else {
        GPIO_ClearValue(g_APinDescription[ulPin].portNum, 1 << (g_APinDescription[ulPin].bitNum)
    }
}
```

```
var tessel = require('tessel');
var ambientlib = require('ambient-attx4');
var ambient = ambientlib.use(tessel.port['A']);

ambient.on('ready', function () {
  // Set a sound level trigger
  ambient.setSoundTrigger(0.1);

  ambient.on('sound-trigger', function(data) {
    console.log("Something happened with sound: ", data);
    // Clear it
    ambient.clearSoundTrigger();
    // After 1.5 seconds reset sound trigger
    setTimeout(function () {
      ambient.setSoundTrigger(0.1);
    }, 1500);
  });
});

ambient.on('error', function (err) {
  console.log(err)
});
```



Microphone

Light sensor (photodiode)


Microcontroller
Gain knob (potentiometer)
Microphone
Op Amp
Light sensor (photodiode)
ambient-attx4
Mic
Light

To SublimeText!

- Tour of ambient JS driver (index.js)
- Tour of ambient firmware (ambient-attx4.c)

Firmware & driver recap

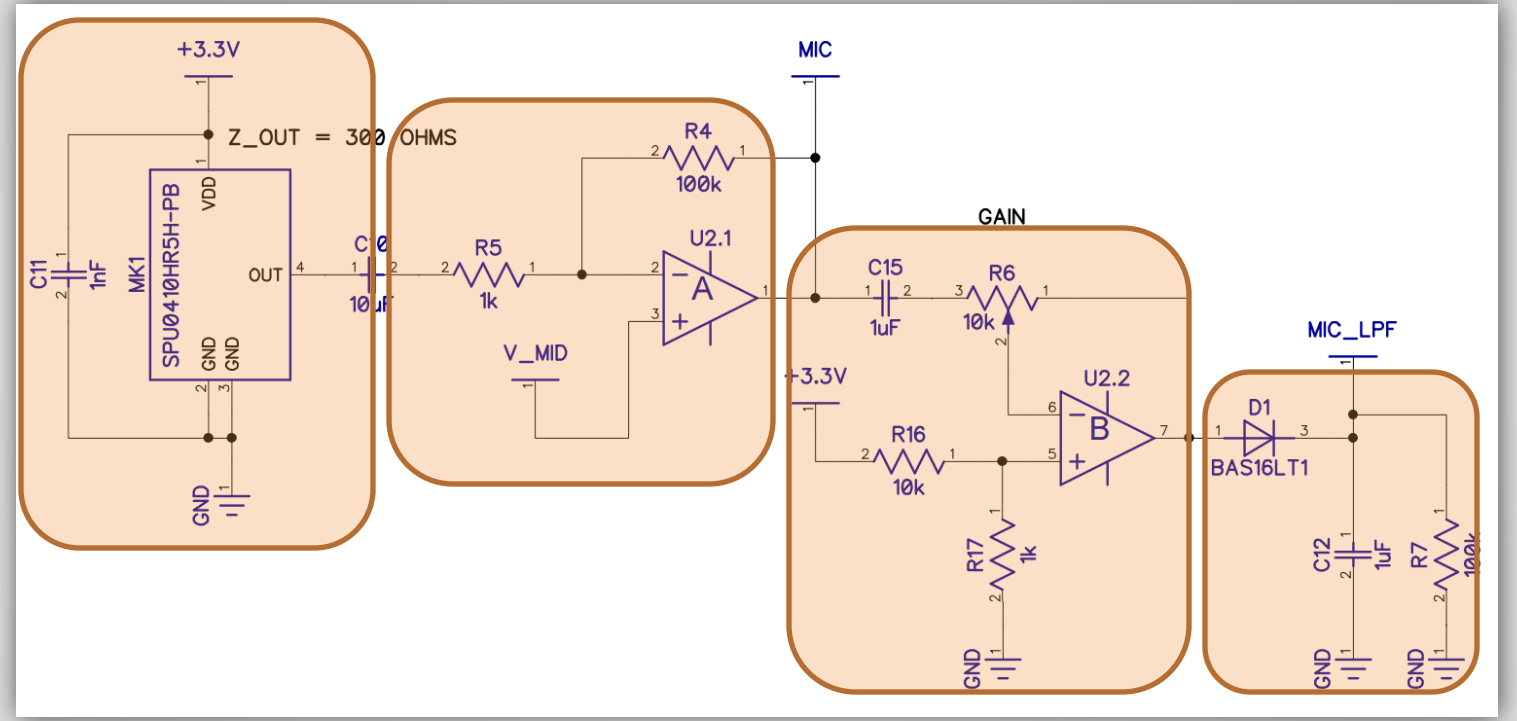
- Setup
- SPI communication
- Interrupt-driven
  - ADC read
  - Threshold detect

```
var tessel = require('tessel');
var ambientlib = require('ambient-attx4');
var ambient = ambientlib.use(tessel.port['A']);

ambient.on('ready', function () {
  // Set a sound level trigger
  ambient.setSoundTrigger(0.1);

  ambient.on('sound-trigger', function(data) {
    console.log("Something happened with sound: ", data);
    // Clear it
    ambient.clearSoundTrigger();
    // After 1.5 seconds reset sound trigger
    setTimeout(function () {
      ambient.setSoundTrigger(0.1);
    }, 1500);
  });
});

ambient.on('error', function (err) {
  console.log(err)
});
```

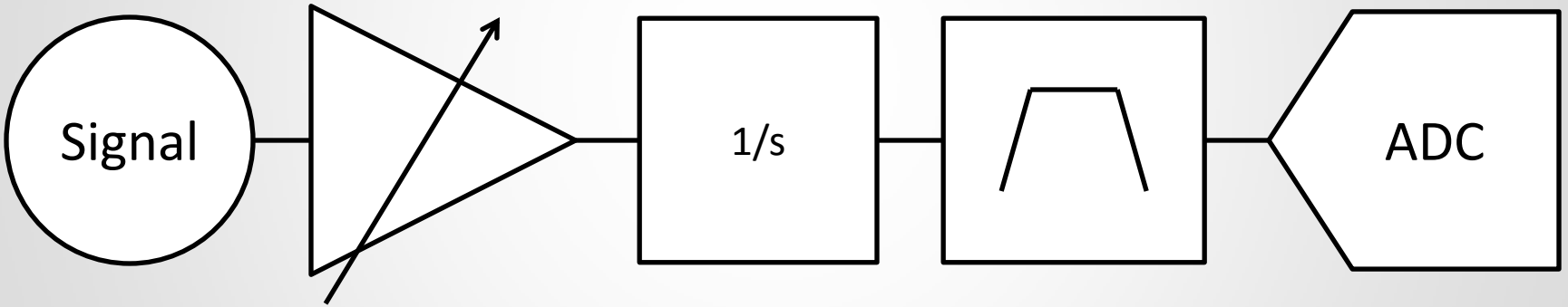
To the schematic!

- Ambient schematic
- All our design docs

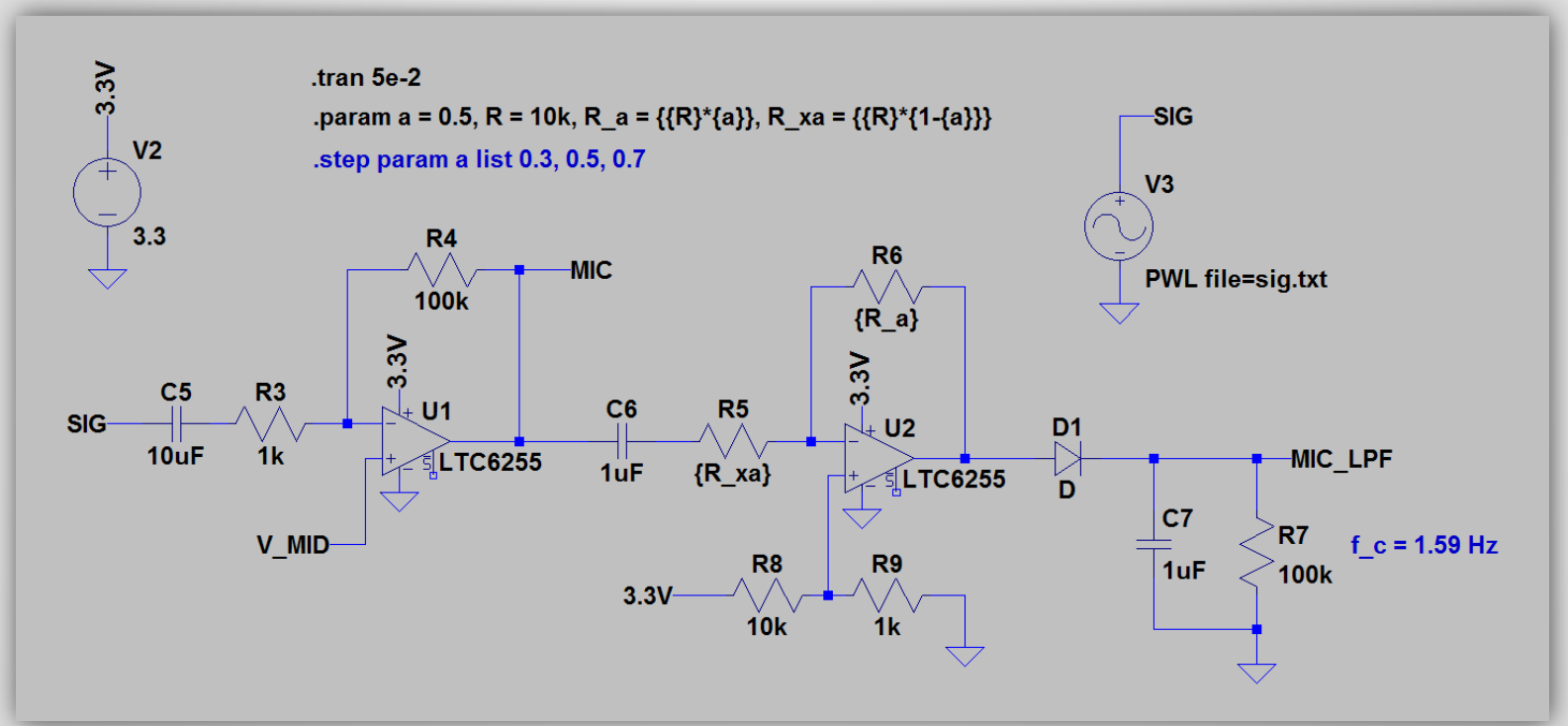
Functional blocks



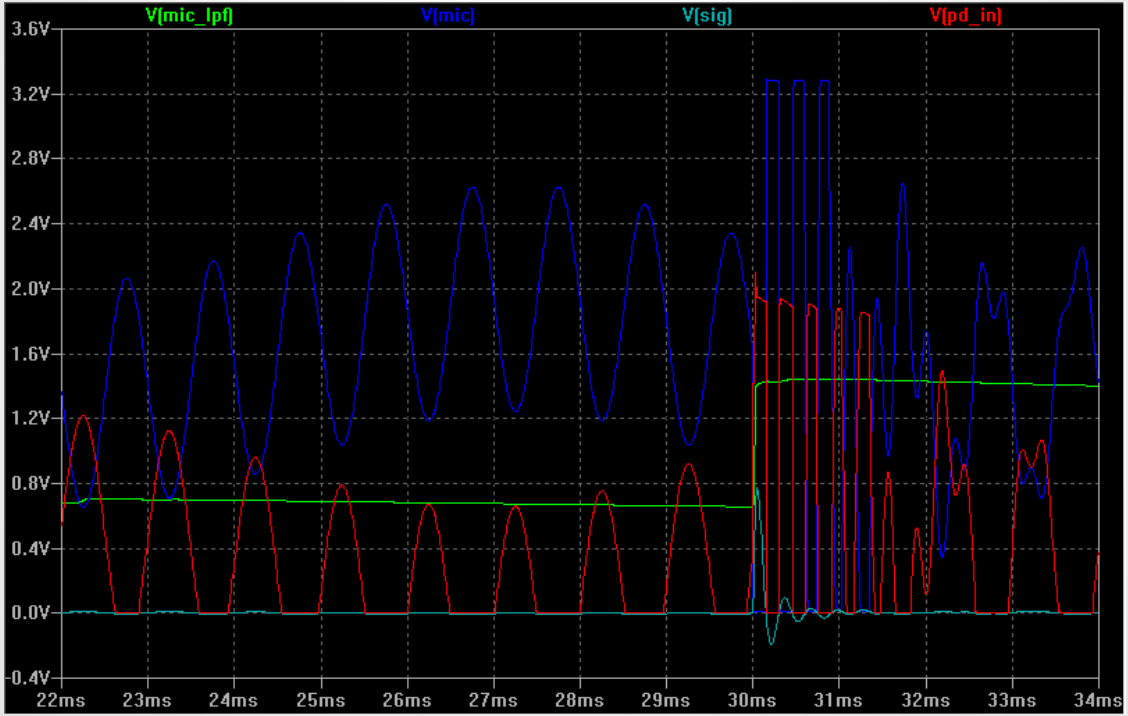
(Approximate) system block diagram



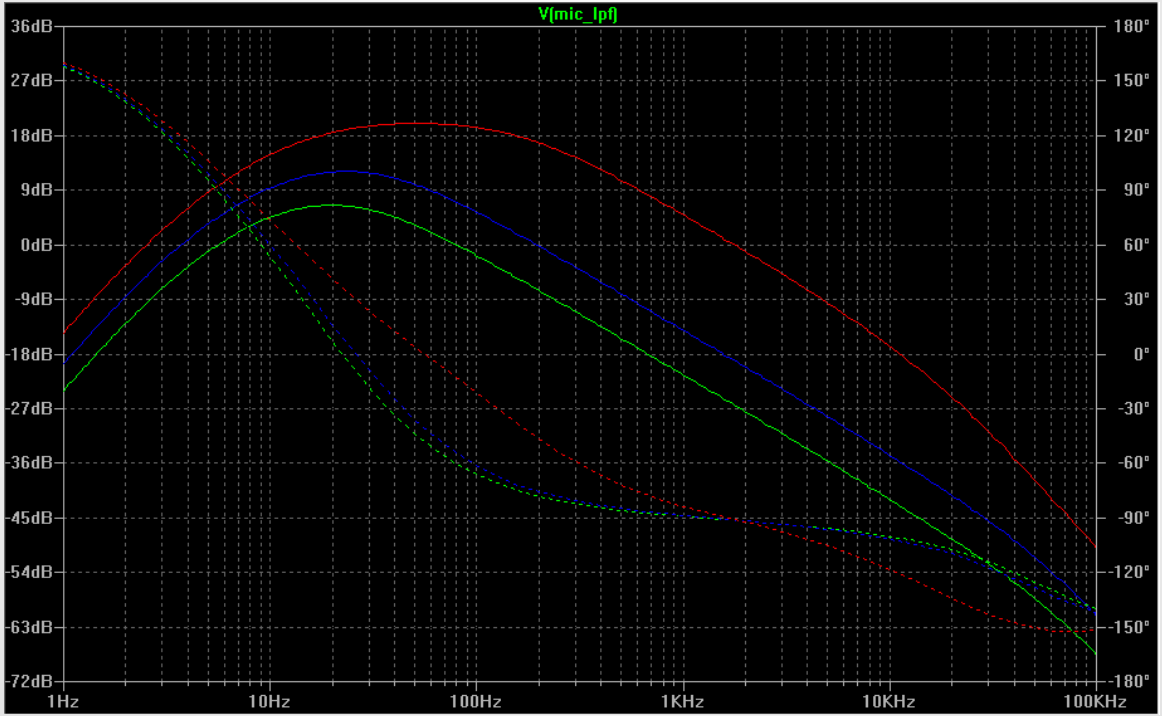
To SPICE!


.tran 5e-2
.param a = 0.5, R = 10k, R_a = {{R}*{a}}, R_xa = {{R}*{1-{a}}}
.step param a list 0.3, 0.5, 0.7
3.3V
V2
3.3
SIG
V3
PWL file=sig.txt
R4
100k
MIC
R6
{R_a}
3.3V
3.3V
C5
10uF
R3
1k
SIG
U1
LTC6255
C6
1uF
R5
{R_xa}
U2
LTC6255
D1
D
MIC_LPF
V_MID
C7
1uF
R7
100k
f_c = 1.59 Hz
R8
10k
R9
1k
3.3V

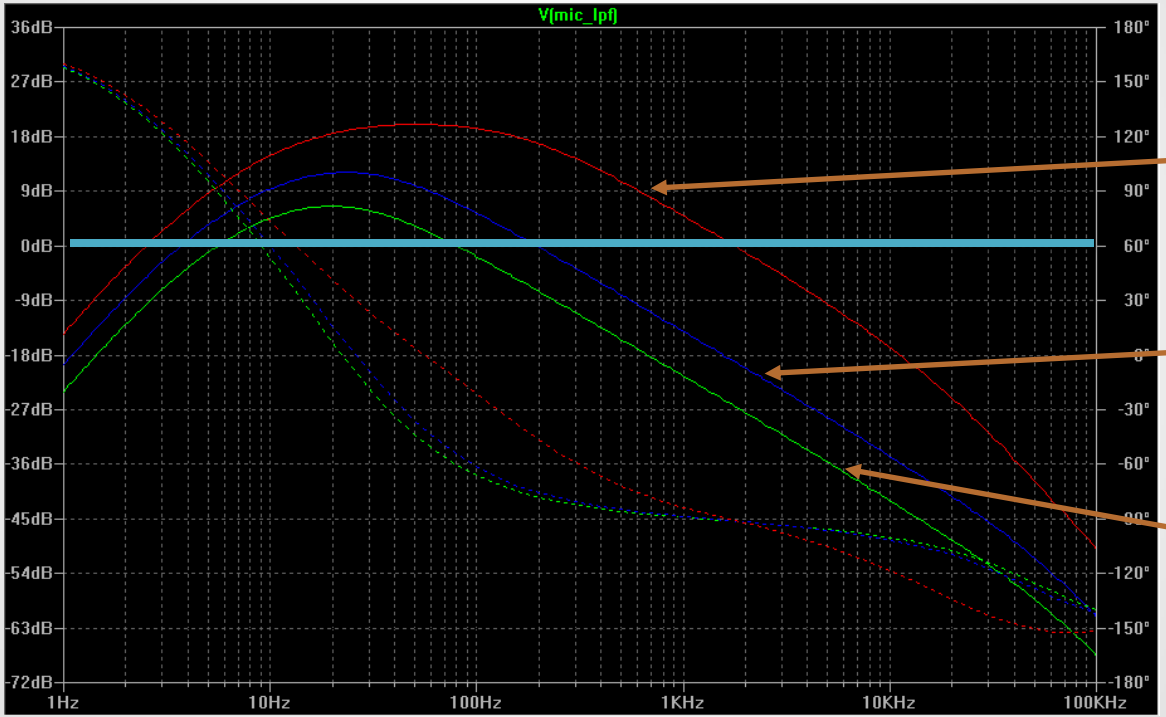
Simulation results: time domain



Simulation results: frequency domain



Simulation results: frequency domain

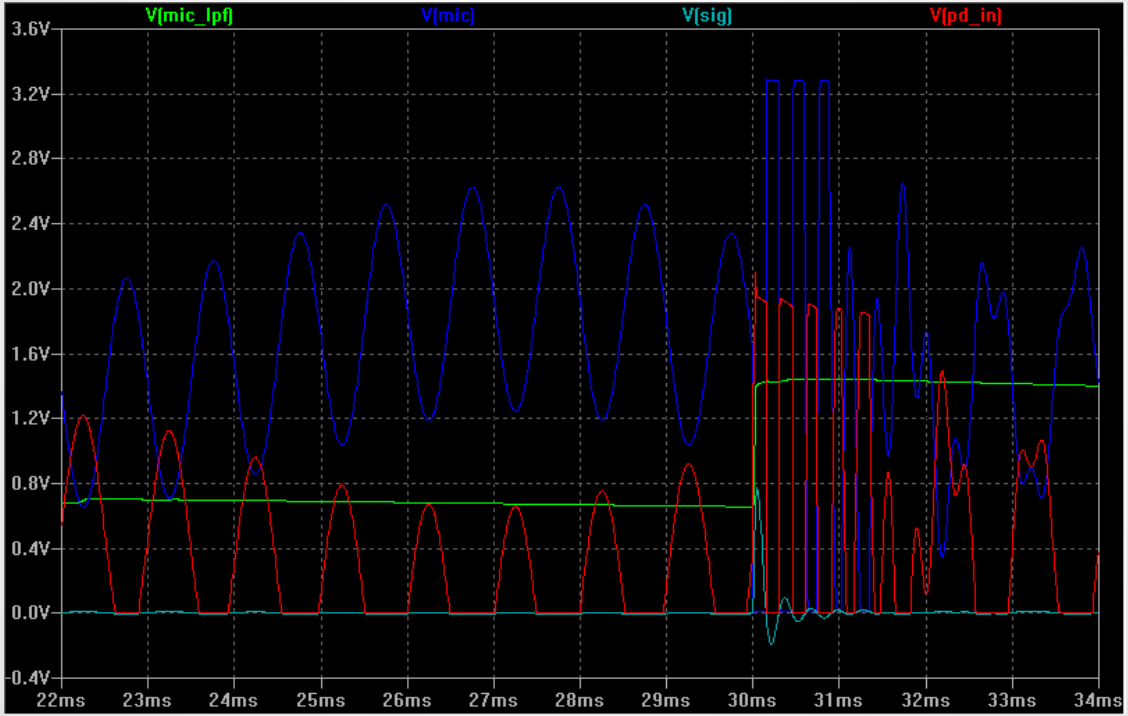


Welcome to the bottom

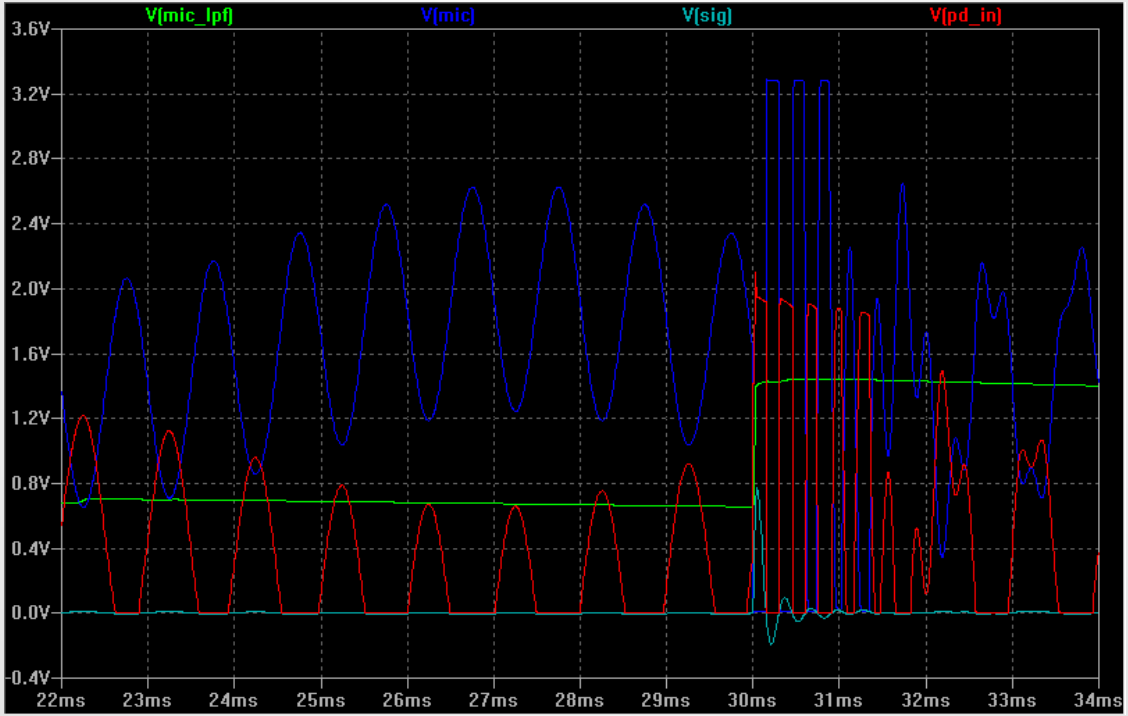
Time to head back to JS!

(This direction is a lot faster)

Simulation results: time domain

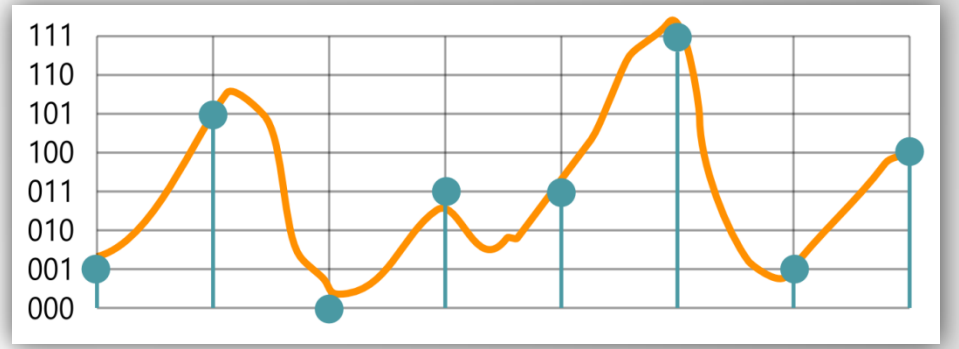


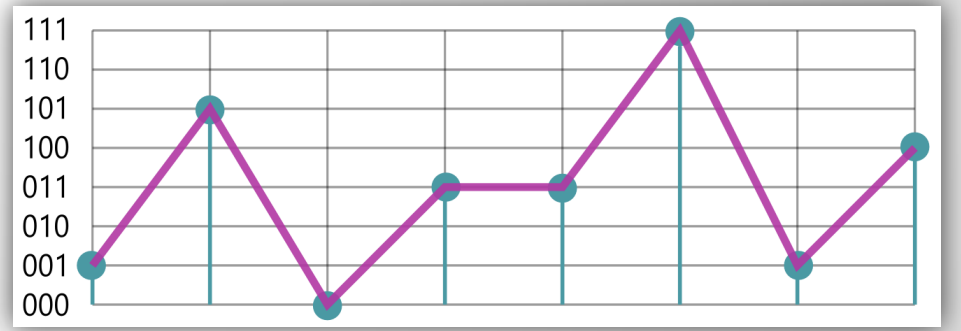
Simulation results: time domain



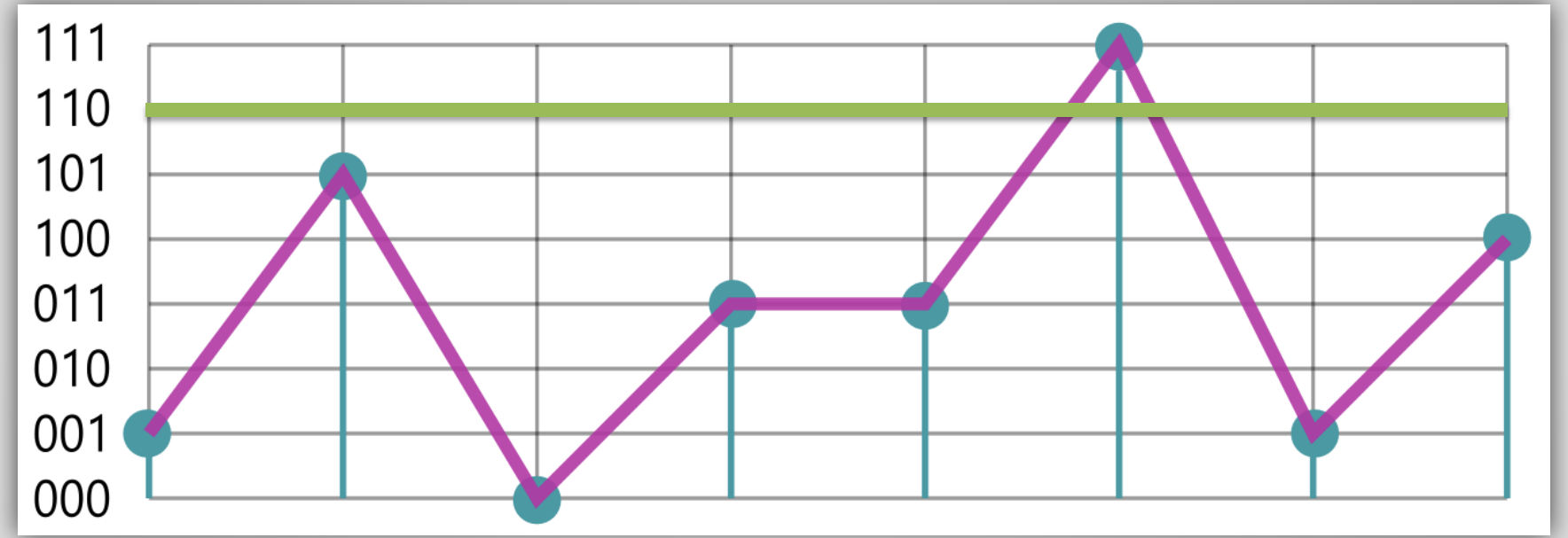
Everyone's take on the situation

- Analog vs. digital
  - Binning
- Sampling





Triggered!



Back to the firmware, JS

- ADC compare, module sets IRQ high
- Tessel handles SPI, saves data payload
- Callback pushed to top of queue
- Callback evaluated with given data

```
var tessel = require('tessel');
var ambientlib = require('ambient-attx4');
var ambient = ambientlib.use(tessel.port['A']);

ambient.on('ready', function () {
  // Set a sound level trigger
  ambient.setSoundTrigger(0.1);

  ambient.on('sound-trigger', function(data) {
    console.log("Something happened with sound: ", data);
    // Clear it
    ambient.clearSoundTrigger();
    // After 1.5 seconds reset sound trigger
    setTimeout(function () {
      ambient.setSoundTrigger(0.1);
    }, 1500);
  });
});

ambient.on('error', function (err) {
  console.log(err)
});
```

Recap

- User code
- JS binds to Lua, Lua VM in C binds to HW
- Communication over SPI
- Interrupt-driven ADC reads, compares
- Yay, op amps for signal processing! SPICE!
- Data sent to Tessel, event fires, callback runs

Thanks!

Questions?

e@technical.io
@twiddlee
ekolker

APPENDICES

Resources

https://github.com/ekolker/fit-2014