Thales Mendes Sampaio

Radio Frequency Power Detector

A Handheld Power and Signal Source Detector

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Thesis

4.1.2018

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Thales Mendes Sampaio Radio Frequency Power Detector 24 pages + 16 appendices 4 January 2018
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	
Instructor(s)	Matti Fischer, Principal Lecturer

The purpose of this project was to design and implement a portable device capable of measuring the power level of specific radio transmissions, and identifying if the signal comes from the desired source. The results are displayed to the user via an Android application and embedded LEDs on the device.

To achieve the desired results, a detector diode circuit and a microcontroller were embedded into a printed circuit board, along with other components responsible for increasing signal-to-noise ratio. Principles of Radio Engineering Design had to be studied for the correct construction of the board. The firmware, written in C language, performed all the component's control and data analysis.

Field tests with the final product showed that all features worked reliably, displaying correct input power level and signal source detection.

Keywords	Radio, Power, Frequency, Detector, Sensor, PSoC.

Contents

List of Abbreviations

AC – Alternate current;
ADC – Analogue to digital converter;
BLE – Bluetooth low energy;
DC – Direct current;
FR – Flame retardant;
IDE – Integrated development environment;
LED – Light-emitting diode;
LiPo – Lithium-ion polymer;
LNA – Low noise amplifier;
OPAMP – Operational amplifier;
PCB – Printed circuit board;
PROG – Programming;
PSoC – Programmable system on chip;
RF – Radio frequency;
RFID – Radio frequency identification;
TP – Test point;
UART – Universal asynchronous receiver-transmitter; and
XOR – Exclusive OR.

1 Introduction

Radio waves are a type of electromagnetic radiation on the frequency spectrum between 3 kHz and 300 GHz. [1] It can be challenging to detect and measure such radiation due to its relatively high frequency. Furthermore, most measurement radio equipment have a high cost and often are not considered portable devices.

The purpose of this work is to go through detailed information about the design and implementation of a radio frequency power detector. This device can detect specific radio transmissions, measure its power and display on a smartphone, via Bluetooth. Principles of Radio Engineering and diode detectors were used to build the hardware of the device. Special attention was given to the firmware, since it is responsible for controlling most components and performing data analysis of the signal.

Applications involving radio frequencies can sometimes have unexpected results. There are several variables that could cause interference on the results. For that reason, it was important to perform multiple tests during development phase to detect and correct these interferences.

2 Power Detection and Diodes

One way of measuring the power level of a radio signal is using a Schottky diode as a detector, exploiting its non-linear V-I relationship and its low *pn* junction capacitance. Detection means the diode will perform a demodulation of an amplitude-modulated signal, as Figure 1 illustrates.

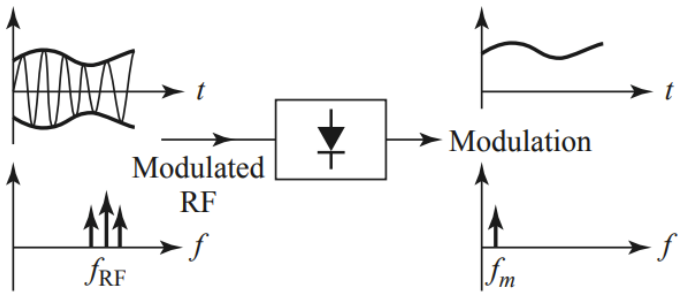


Figure 1. Frequency conversion operation of detection by a diode detector. [2]

The small-signal V-I relationship of a Schottky diode can be expressed as

$$I(V) = I_0 + i = I_0 + vG_d + \frac{v^2}{2}G'_d, \quad (1)$$

where I_0 is the DC bias current, i is the small AC signal current, v is the small AC signal voltage, and G_d is the dynamic conductance of the diode. [2] In this project, the signal to be detected has a RF carrier frequency $\omega_0 = 866\,MHz$, and modulation frequency $\omega_m = 2\,kHz$. The diode voltage can be expressed as

$$v(t) = v_0(1 + m\cos\omega_m t)\cos\ \omega_0 t, \quad (2)$$

where $\omega_0 \gg \omega_m$ and m is defined as the *modulation index* ($0 \le m \le 1$). [2]

The Equations (1) and (2) can be used to determine the diode current i:

$$\begin{aligned} i(t) &= v_0 G_d(1 + m\cos\omega_m t)\cos\ \omega_0 t + \frac{{v_0}^2}{2}G'_d\,(1 + m\cos\omega_m t)^2\cos^2\ \omega_0 t \\ &= v_0 G_d\left[\cos\ \omega_0 t + \frac{m}{2}\cos\ (\omega_0 + \omega_m)t + \frac{m}{2}\cos\ (\omega_0 - \omega_m)t\right] \\ &+ \frac{{v_0}^2}{4}G'_d\left[\begin{array}{c} 1 + \frac{m^2}{2} + 2m\cos\omega_m t + \frac{m^2}{2}\cos 2\omega_m t + \cos\ 2\omega_0 t \\ + m\cos\ (2\omega_0 + \omega_m)t + m\cos\ (2\omega_0 - \omega_m)t\ + \frac{m^2}{2}\cos\ 2\omega_0 t \\ + \frac{m^2}{4}\cos\ (2\omega_0 + \omega_m)t + \frac{m^2}{4}\cos\ (2\omega_0 - \omega_m)t \end{array}\right]. \quad (3) \end{aligned}$$

The frequency spectrum of the diode's output current can be plot from Equation (3), as in Figure 2. The graph exemplifies how the desired demodulated frequency ω_m can be obtained with a simple low-pass filter. [2]

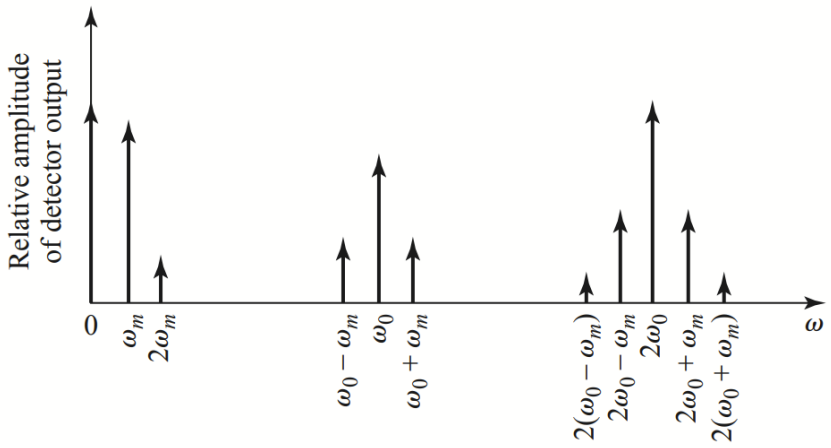


Figure 2. Frequency spectrum of a detected AM modulated signal. [2]

It is worth mentioning the proportion between output and input power is not constant. It varies according to the graph in Figure 3. In this project, the input power reaches the start of saturation region, but it is not an issue, since a calibration lookup table is going to be used to determine the correct input power.

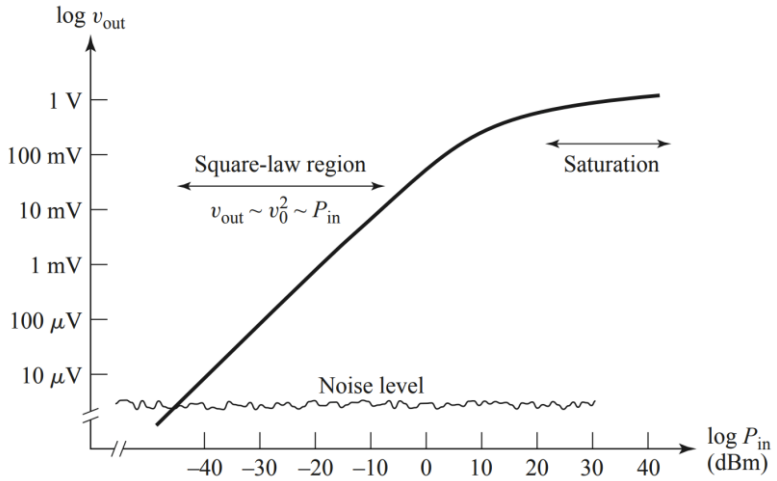


Figure 3. Output voltage by input power level for a typical diode detector. [2]

An example of a detector circuit can be seen on Figure 4. The inductors are used for input matching, the diode rectifies the signal, and the capacitor filters out the RF frequencies.

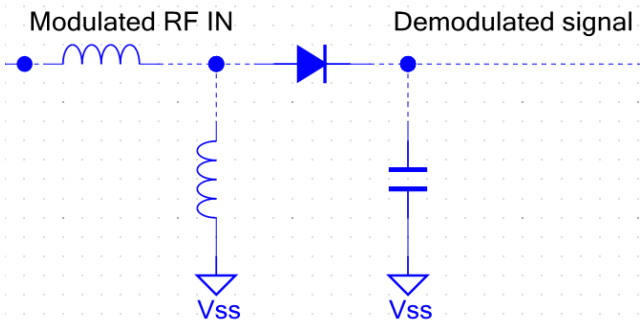


Figure 4. Example of detector diode circuit.

3 Initial Approach: Line-up Analysis

To define the reading range of the device, a few considerations had to be taken. It was known the transmitter used on the tests could provide power levels between -70 dBm and -10 dBm, depending on the distance from the receiver. To achieve such vast range of readings, it was necessary to implement a couple of amplifiers and a gain switch. The first set-up to gather data consisted of a low noise amplifier (LNA), a power detector circuit, matched for 50 Ω, and a microcontroller CYBLE-014008, which contained an amplifier and an analogue to digital converter (ADC). These three components are shown respectively on Figures 5, 6 and 7.

Figure 5. Low Noise Amplifier with 30 dB gain.

Figure 6. Detector diode circuit with two inductors for input matching.

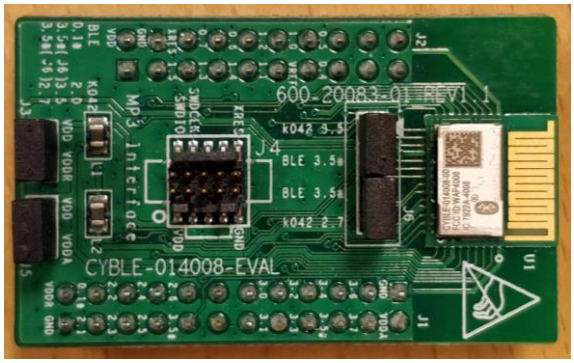


Figure 7. Evaluation board of System of Chip CYBLE-014008.

The test set-up was arranged as Figure 8 illustrates.

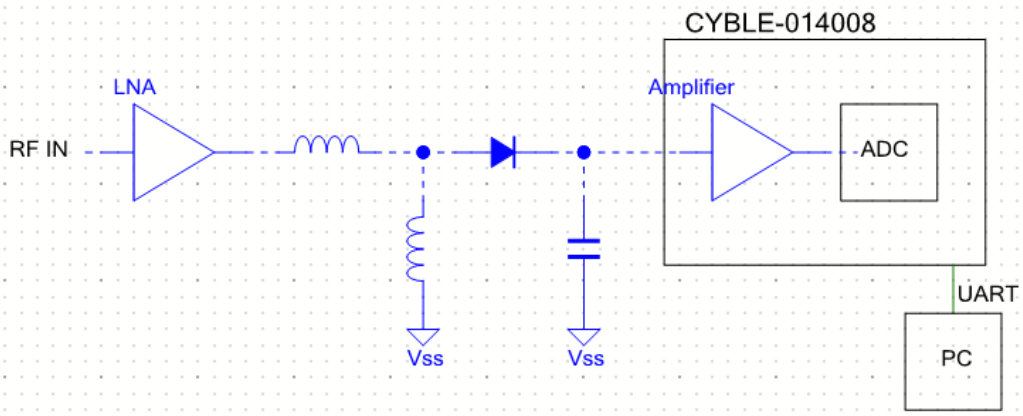


Figure 8. Test set-up with LNA, detector diode and microcontroller.

The purpose of this test was to create a model with variable LNA and Amplifier gains, so those could be adjusted to achieve the desired range of input/output. The ADC converts the analogue data and send the value on millivolts to the computer's console via UART. The gain of the LNA was 30 dB and the gain of the microcontroller's amplifier was 21 dB. A multimeter was used to measure the voltages on the input of the OPAMP and ADC. All the values were stored as table 1 summarizes.

Table 1. Results indicating input values of each component and output value of the test system.

MEASURED VALUES						
Input Values						
Input Power (dBm)	LNA (dBm)	Detector (dBm)	OPAMP (mV)	ADC (mV)	Output (mV)	
-85	-85	-55	0.01			
-80	-80	-50	0.01			
-75	-75	-45	0.03			
-70	-70	-40	0.90	11.00	11	
-65	-65	-35	2.80	32.00	32	
-60	-60	-30	8.60	97.00	96	
-55	-55	-25	24.00	275.00	274	
-50	-50	-20	63.00	708.00	706	
-45	-45	-15	145.00	1613.00	1610	
-40	-40	-10	303.00	3340.00	2048	ADC Clamped
-35	-35	-5	590.00			
-30	-30	0	1088.00			
-25	-25	5	1850.00			

These values were used to obtain two 5th-order-polynomial-regression functions of the OPAMP input with the input power. For more accurate results, Function (4) was the approximation for low power levels (-70 dBm to -41 dBm) and Function (5) the approximation for high power levels (-40 dBm to -10 dBm).

$$f(x) = 8{,}9202 \cdot 10^{-6} \cdot x^5 + 3{,}3014 \cdot 10^{-3} \cdot x^4 + 0{,}4882 \cdot x^3 + 36{,}0777 \cdot x^2 + 1332{,}9235 \cdot x + 19710{,}2892 \quad (4)$$

$$y(x) = -6{,}5333 \cdot 10^{-6} \cdot x^5 - 1{,}2352 \cdot 10^{-3} \cdot x^4 - 0{,}83274 \cdot x^3 - 20{,}5042 \cdot x^2 + 73{,}18961173 \cdot x + 7670{,}1551 \quad (5)$$

With these functions, table 2 was created to simulate what the values of each component's input would be, based on the gain of the amplifiers.

Table 2. Line-up analysis for the detector test circuit.

			CALCULATED VALUES			
	Gain (dB)		Input			
LNA	15		LNA (dBm)	Detector (dBm)	OPAMP (mV)	ADC (mV)
OPAMP	16		-70	-55	0.01	0.04
			-65	-50	0.01	0.07
			-60	-45	0.03	0.19
			-59	-44	0.14	0.90
			-58	-43	0.29	1.81
			-57	-42	0.46	2.92
			-56	-41	0.67	4.21
			-55	-40	0.90	5.68
			-54	-39	1.17	7.36
			-53	-38	1.47	9.29
			-52	-37	1.83	11.56
			-51	-36	2.27	14.30
			-50	-35	2.80	17.67
			-49	-34	3.47	21.90
			-48	-33	4.32	27.29
			-47	-32	5.42	34.18
			-46	-31	6.81	43.00
			-45	-30	8.60	54.26
			-44	-29	10.87	68.57
			-43	-28	13.73	86.62

			-42	-27	17.31	109.19
			-41	-26	21.74	137.20
			-40	-25	24.07	151.89
			-39	-24	27.65	174.47
			-38	-23	33.59	211.93
			-37	-22	41.53	262.02
			-36	-21	51.23	323.23
			-35	-20	62.56	394.76
			-34	-19	75.51	476.44
			-33	-18	90.14	568.73
			-32	-17	106.61	672.63
			-31	-16	125.15	789.65
			-30	-15	146.09	921.75
			-29	-14	169.79	1071.29
			-28	-13	196.69	1241.00
			-27	-12	227.26	1433.91
			-26	-11	262.03	1653.29
			-25	-10	301.55	1902.65
			-24	-9	346.40	2185.63
			-23	-8	397.17	2505.99
			-22	-7	454.48	2867.54
			-21	-6	518.91	3274.12
			-20	-5	591.09	3729.51
			-19	-4	671.58	4237.41
			-18	-3	760.96	4801.36
			-17	-2	859.77	5424.75
			-16	-1	968.48	6110.71
			-15	0	1087.56	6862.07
			-14	1	1217.41	7681.35
			-13	2	1358.36	8570.66
			-12	3	1510.67	9531.69
			-11	4	1674.54	10565.64
			-10	5	1850.07	11673.17

These results helped to have a starting point for which components to use on the prototype design to achieve the desired range of operation. As is can be noticed on Figure 9, a strong amplification is necessary on low power levels, whereas for high power levels, the voltage level is already sufficient to be read by the ADC.

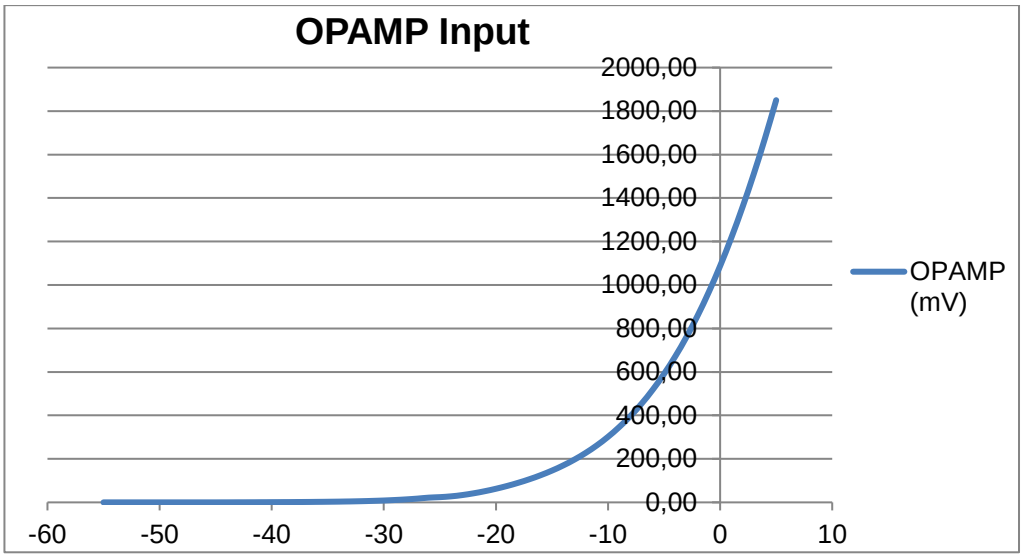


Figure 9. Voltage level on the input of the OPAMP based on input power.

4 Designing the Device

The device should be able to perform four tasks: calculate power level, display it on the smartphone, indicate if the power is high enough for its application, and detect if the signal comes from the desired transmitter. These will be discussed in more detail in the software section. To obtain a good performance the hardware has to provide a reliable signal acquisition, and that was ensured with a carefully designed prototype. Prototype schematics and layout was designed using the software PADS. Refer to the schematics on Appendix 1 to follow this chapter.

The first operational amplifier is used to increase the signal on the low end of the range of detection. The input signal on this region is not high enough to be detected by the ADC, therefore, it needs to be amplified. Also, it was important to consider the maximum value the ADC could detect, in this case, using an internal reference voltage, the ADC could convert voltages only up to 2,048 V. The line-up analysis indicated that, with a fixed gain on the amplifier, the high end of the device's range would output voltages up to 11 V. For this reason, the gain on the input had to be manipulated to prevent the input of the ADC to be no more than 2,048 V. The simple solution found was to implement an analogue multiplexer, which would switch the input signal between the amplifier and a direct connection to the ADC, depending on the input power. In the case of direct connection, the final voltage value is digitally multiplied by an approximate value of the amplifier gain, 11 in this case.

This method was chosen because, the transfer function had a more linear behaviour on the high end of the range, making it simpler to determine the real input signal by a simple multiplication, once the amplifier is disabled. Before the ADC input, an active low pass filter with unity gain and 3 dB bandwidth of 10.6 kHz was connected to reduce noise.

4.1 Main Components

The most important components to be implemented were the LNA, a voltage regulator, a band-pass filter, the detection diode, and the microcontroller. These components, when placed on a correct signal chain, provide the device a better signal-to-noise ratio, and consequently, a more reliable data analysis.

4.1.1 Low Noise Amplifier (MGA-68563)

To be able to detect radio signals with power as low as -70 dBm, an LNA was placed after the device's antenna to provide the microcontroller a high enough input level.

4.1.2 Voltage Regulator (TPS79901DDC)

The voltage regulator was used to feed a known and stable DC voltage to the LNA. A change on the bias voltage of the LNA would cause inconsistency on the amplification, and since the voltage of a battery varies with charge, this component was essential. The battery used was a lithium-ion polymer (LiPo) battery with 3.7 V output and the regulator was levelling this to a stable 3 V.

4.1.3 Band-Pass Filter (B3717)

Placed after amplification, the band-pass filter attenuates unwanted frequencies, letting signals pass on the range from 860 MHz to 873 MHz, with centre frequency of 866.5 MHz. The transfer function of the filter is visualized in Figure 10.

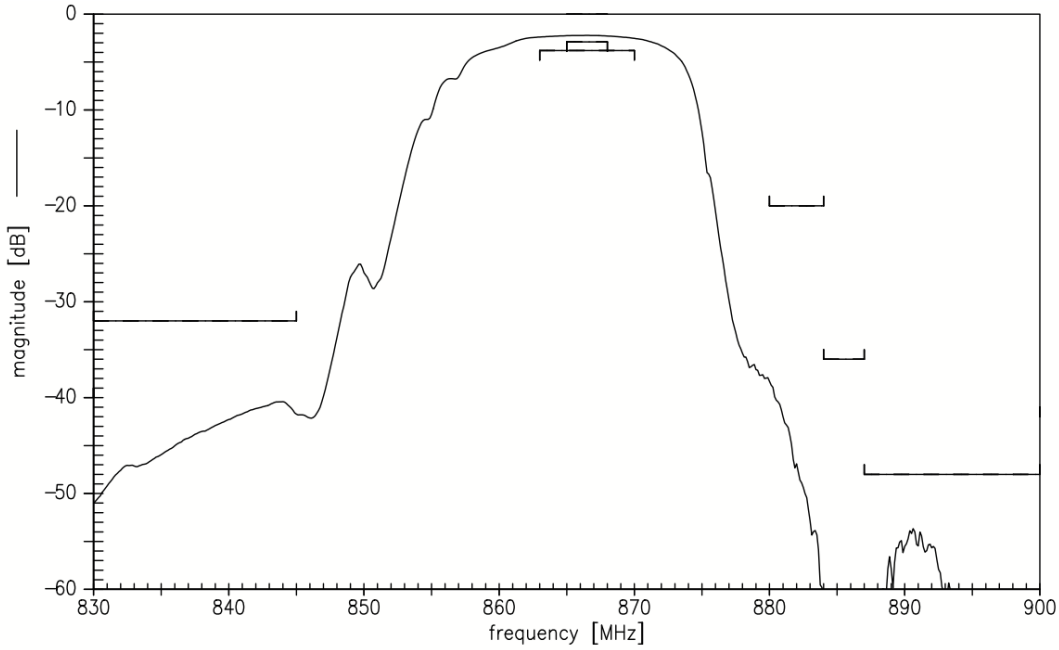


Figure 10. Transfer function of the band-pass-filter.

4.1.4 Detector Diode Circuit

Detector diode converts the output of the filter and sends to the microcontroller a signal that can be detected by the ADC. A matching circuit has to be put before the diode for it to be matched to the output of the filter, which is 50 Ω.

4.1.5 Microcontroller (CYBLE-014008)

The microcontroller is the most important component, since it will be performing functions such as amplification, filtering, conversion, analysis, and transmission of the acquired signal.

The chip has embedded four operational amplifiers, of which two will be used in this application. The first as an amplifier for the detector diode output, and the second as an active low-pass filter. The analogue data is converted to digital so it can be analysed and transmitted. The signal chain inside the microprocessor is visualized in Figure 11. The functionality of each element will be discussed in the software section. The components

indicated on blue, are external of the microcontroller and had to be placed on the printed circuit board.

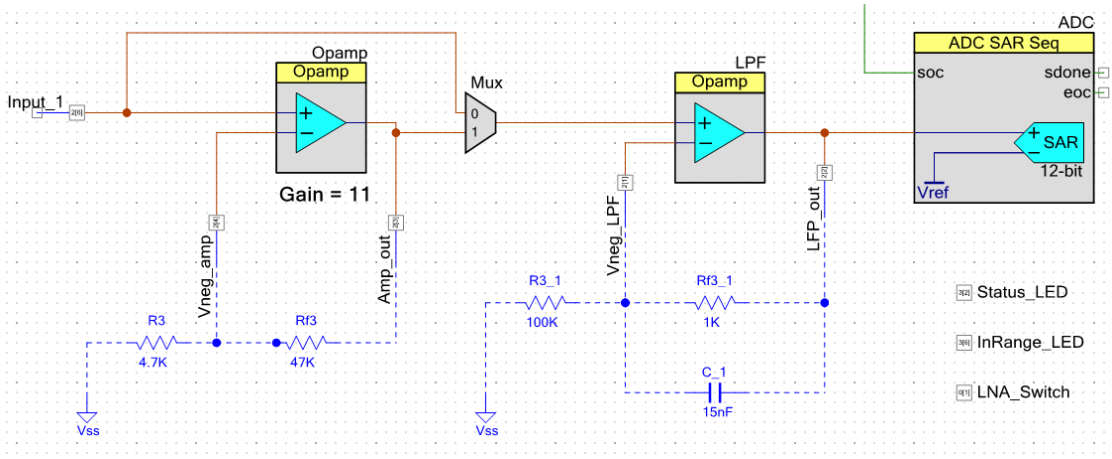


Figure 11. Block diagram on the microcontroller's IDE.

4.2 Schematics

The complete schematics of the device can be found in Appendix 1. It was designed using PADS Logic. Each component mentioned under the Section 4.1 was connected, making sure to have a correct impedance matching between RF components. A test connector was put in the start of the signal chain to help determining the impedance of the antenna and allow the calculation of the correct LNA matching. Finally, the microcontroller block was created and connected according to its ports usage. Pins named "PROG" were used to program the microcontroller. Pins labelled "UART" were used in the software developing phase for debugging. "TP" pins are simply test points for the ports which were not in use.

In addition, DC block capacitors were added after the antenna and in between main components on the RF area of the design.

4.3 PCB Layout

The layout of the PCB was designed using PADS Layout. It consists of 4 layers: Top, where the microprocessor and most components are; Bottom, where the battery, connectors and LEDs can be found; and two inner layers: VDD and Ground. Top and bottom layers are shown in Figures 12 and 13, respectively.

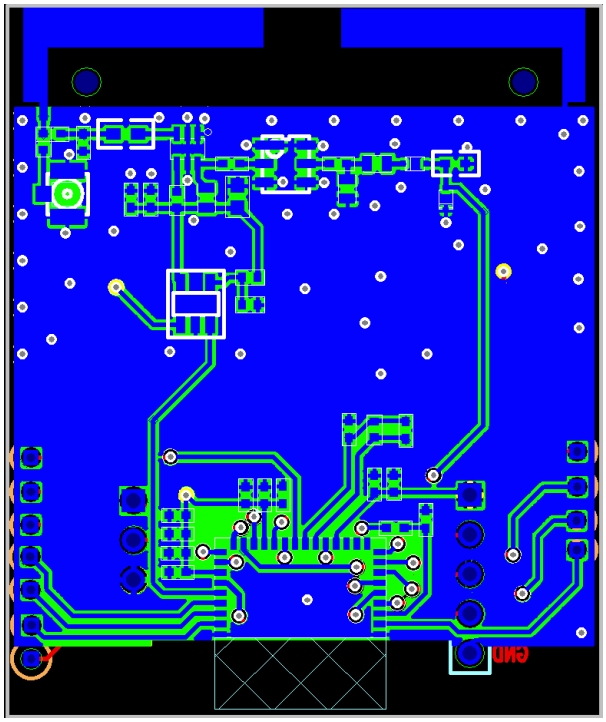


Figure 12. Top layer of the PCB layout.

It can be noticed that on the area close to the antenna there are many vias. The reason for that is to increase the ground efficiency on the RF area. [3] When stripes of conductor are left without vias on RF areas, they can act as antennas for unwanted frequencies, therefore increasing the noise level. The vias connect the ground planes on every layer creating a stronger ground around it.

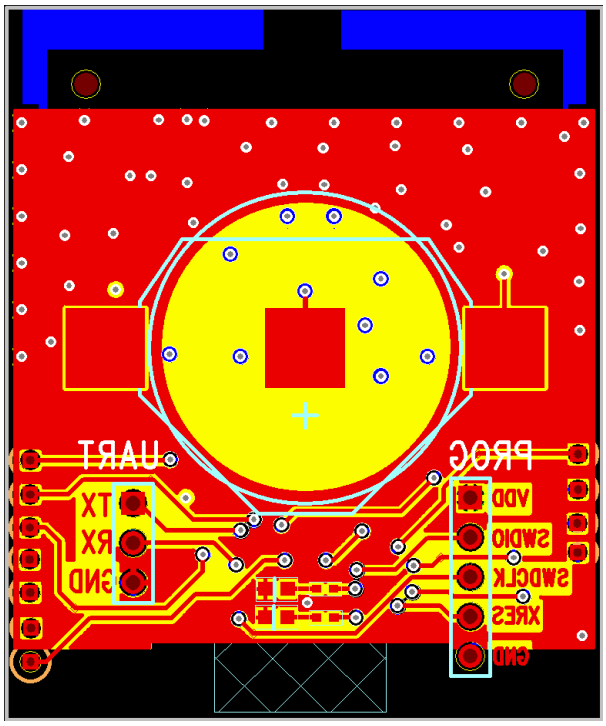


Figure 13. Bottom layer of the PCB layout.

The routing on the RF area (everything before the detector diode) had to be done carefully to ensure the traces have a 50 Ω impedance to avoid reflections. The width of the stripline was obtained using Equation (6). The material used as the core of the board is FR-4, with dielectric constant permittivity of 4.34 at 1 GHz. [4] Dielectric thickness is 0.8 mm. The width of the stripline can be calculated by

$$W = b\left(\frac{30\pi}{\sqrt{\epsilon_r} Z_0} - 0{,}441\right), \tag{6}$$

where b is the thickness of the dielectric, ϵ_r is the dielectric permittivity, and Z_0 is the characteristic impedance. (2) Following the calculations, the result for the width was 0.371 mm.

Two copper pads were drawn on top of the board to solder the antenna.

5 Prototype and First Tests

With the assembly of the power detector prototype completed, some initial tests and adjustments were required. The theoretical calculations and results obtained were a good starting point for the calibration of the device, but due to the fact its measurements involve radio frequencies, the results on the first prototype were expected to be slightly different.

An early version of software was programmed into the microcontroller to enable some data acquisition. The device would be connected to the smartphone, via Bluetooth, and display the power level being detected. A calibrations table was created using the initial components on the board. This table provided a good starting point for the setup of the device, but later it had to be updated to allow the device to display correct values of input power.

5.1 Matching Circuit

Since the input involves high frequency signals, a study for matching the components was required. The components on the radio section of the board had to be connected with 50Ω strip lines, since they all have inputs and outputs matched for 50Ω. The challenge begins when trying to match the input of the LNA to the antenna.

The impedance of the antenna was unknown, since it is basically a wire loop, so it could not be connected straight to the LNA input. To determine the antenna impedance, its output was connected to a network analyser via the U.FL connector soldered onto the antenna pad. With this setup, the input impedance matching for the LNA was calculated based on Equation (7):

$$Z_{in} = Z^*_{antenna}. \text{ [2]} \quad (7)$$

The antenna impedance is visualized in Figure 14. To obtain a matched connection, the input of the LNA should have an impedance $Z_{in} = 21{,}2 + j23{,}1\ [\Omega]$, which is the x-axis mirror of the antenna impedance. That impedance was reached with the addition of a serial inductor with a value of 27 nH on the input of the LNA.

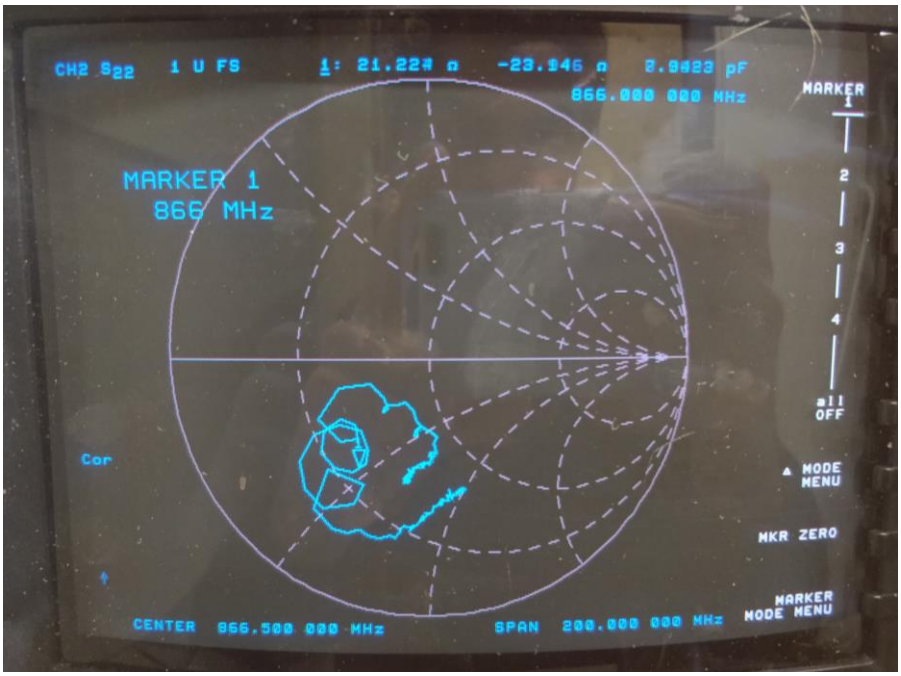


Figure 14. Smith Chart from a network analyser displaying the antenna impedance.

The conjugate matching applies in theory, but as mentioned before, dealing with radio frequencies can be unpredictable, and in this case, the theoretical matching was not optimal. A 50Ω antenna with circular polarization and 8 dB antenna gain was connected to the LNA to get a reference value for a fixed power transmission. This value, -25 dBm, was compared to the device's antenna with the calculated matching, -39 dBm. These results implied some tweaking on the matching circuit was necessary.

First, a parallel capacitor was added and its value was changed until the power reached its maximum at -31 dBm, with 1.5 pF capacitance. Then the value of the serial inductor was changed on the same way, until reaching a maximum power output at -29 dBm, with a 20 nH inductor. These values were used for the final antenna matching, with a reasonable antenna gain of 1dB, which can be calculated by Equation (8):

$$\begin{aligned} Antenna\ Gain &= Max\ Antenna\ Value \\ &\quad - (Max\ Reference\ Value - Reference\ Antenna\ Gain + 3\ dB) \\ &= -29 - (-25 - 8 + 3) \\ &= 1\ dB. \end{aligned} \tag{8}$$

5.2 Calibration

Calibration was necessary to assure the right value being input on the device was being displayed on the output. For that, a signal generator was used to send a known high

frequency signal at a vast range of power. The output of the generator was connected straight to the input of the LNA, via the U.FL connector, simulating a signal detected by the antenna.

To determine the power loss on the cable used to transfer the signal, a power meter was connected to the end of the cable. With that it was concluded that the cable had a 2.6 dBm power loss. This information was then used to offset the input of the generator to achieve the desired range of detection.

A table was created with the signal generator outputting -67.4 dBm with increments of 1 dBm, up to -7.4 dBm. In this case the input of the LNA was ranging from -70 dBm to -10 dBm, which is the desired power range of the device. The output voltage for each power level was recorded, so the calibration table could be updated on the software, enabling the correct display of input power on the application. The graph of the calibration table is visualized in Figure 15.

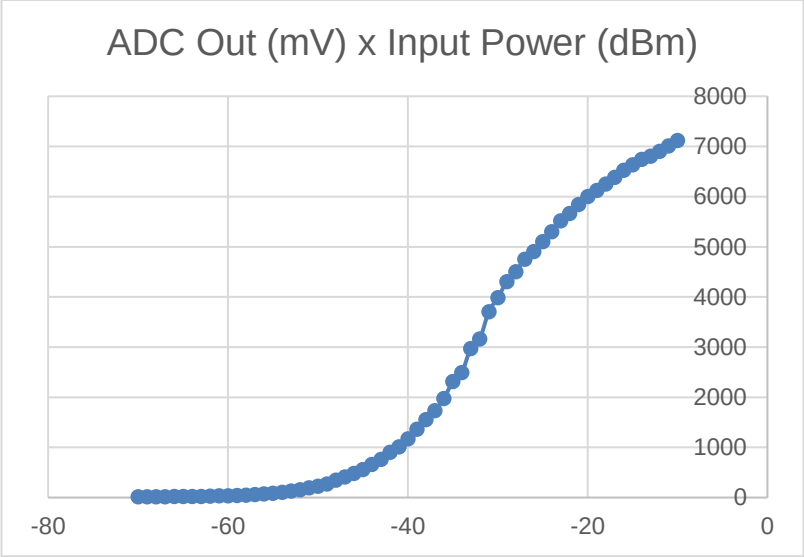


Figure 15. Plot of calibration table.

5.3 Power Consumption

An analysis of the power consumption of the device was required to determine whether the original design with a coin-cell battery was going to provide enough energy for a reasonable amount of time.

The initial test results indicated the drawn current on advertisement and connection state were, respectively, 27.8 mA and 29.2 mA. Such high values were causing the battery to be drained in just a few minutes. That lead to the creation of a power optimization routine on the software to increase battery life.

The most important components to be taken into consideration were the Bluetooth module, LNA and microcontroller's processor. These components' power consumption were studied individually to help improve overall battery life.

5.3.1 Bluetooth Module

There are two states for the Bluetooth module on this application: advertising, when the server device is looking for a client device to establish connection; and connected, when the devices are paired and ready to exchange data.

For both states, the module utilizes a high frequency oscillator to generate its clock, therefore, this is one parameter that can be manipulated to reduce power consumption.

During advertisement, an interval of one second was set between transmissions, reducing the amount of time the transmitter is turned on. When connection is made, a function is responsible for setting the device to alternate between deep sleep and normal operation, significantly reducing the average clock speed without compromising performance.

5.3.2 Low Noise Amplifier

A substantial portion of the current drainage on the device is used to power up the LNA. With a maximum bias current of 12 mA, a method to reduce average power consumption needed to be implemented. The way that was done was inserting a switching mechanism to turn the LNA on and off according to the state of the device. During advertisement, no signal is being acquired and all the data processing is on stand-by. On this case, the LNA is completely turned off. When connection is stablished and data starts to be received and processed, the LNA must be turned on. During this state, the amplifier does not necessarily need to be on at all times. Therefore, it is kept off when data is not being received, turned on for a short period of time, while the signal is being acquired, then

turned off again, repeating this cycle. With this implementation a significant reduction of power consumption was achieved.

The type of battery used was changed after LNA studies. It was opted to use a 250 mAh LiPo battery for a more reliable current delivery and longer lifetime.

5.3.3 Microcontroller's Processor

The microcontroller has multiple clocks that run continuously on normal conditions. These include an external crystal oscillator (ECO) at 24 MHz, an internal main oscillator (IMO) at 48 MHz, an internal low-speed oscillator (ILO) at 32 kHz, and a watch crystal oscillator (WCO) at 32.768 kHz. Not every oscillator needs to be working at all times. The power optimization was done by managing when to use each clock.

The processor and each individual clock can be put into sleep mode when not in use. A function was implemented on the main loop of the code to put the processor in deep sleep when it is not being used. When in deep sleep, interrupts can be used to wake up the processor. This way, whenever other peripheral is processing or acquiring data, the Bluetooth and main processor can be put into sleep mode without affecting performance. Table 3, containing the current consumption of both phases of the device's operation, was created to compare each optimization step. Advertisement is the state where the device is looking for an Android device to stablish connection. Connected state is when the data is being collected, processed, and transferred to the smartphone.

Table 3. Current consumption on advertisement and connected states for each optimization.

	Base Values (mA)	Opt. 1 (mA)	Opt. 2 (mA)	Opt. 3 (mA)
Advertisement	27.8	15.8	0.47	0.47
Connected	29.2	29.2	22.5	4.5

- Optimization 1 is simply done by turning off the LNA while the device is looking for connection.
- Optimization 2 is done by disabling all unnecessary oscillators and reducing the advertisement frequency of the Bluetooth module. Optimization 1 was kept.

- Optimization 3 achieves much lower current consumption on connected state by lowering some clock speeds and switching the LNA on and off as data is acquired. Optimization 2 was kept.

6 Software

Software development was divided into two parts, for microcontroller and android application. The android application was kept as simple as possible, only providing Bluetooth connection with the device and displaying the power value. Programming the microcontroller was a more complex task.

6.1 Android

The smartphone application was simply responsible for stablish connection with the device and receive data to show the power value on the interface. The source code of the application can be found on appendix 3. Figure 16 shows how the user interface looks on the smartphone.

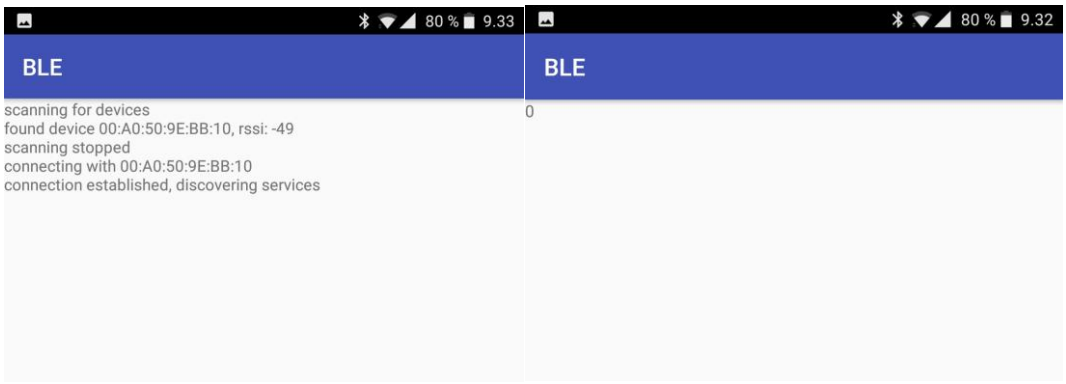


Figure 16. Android application user interface.

6.2 Microcontroller

The software environment used for Cypress microcontrollers is called PSoC Creator. It is divided into two development sections: a block diagram page, as in Figure 17, where the components being used and its connections are set, and a firmware page, where the

C code is written. The complete code of the microcontroller and android device can be found on appendixes 2 and 3.

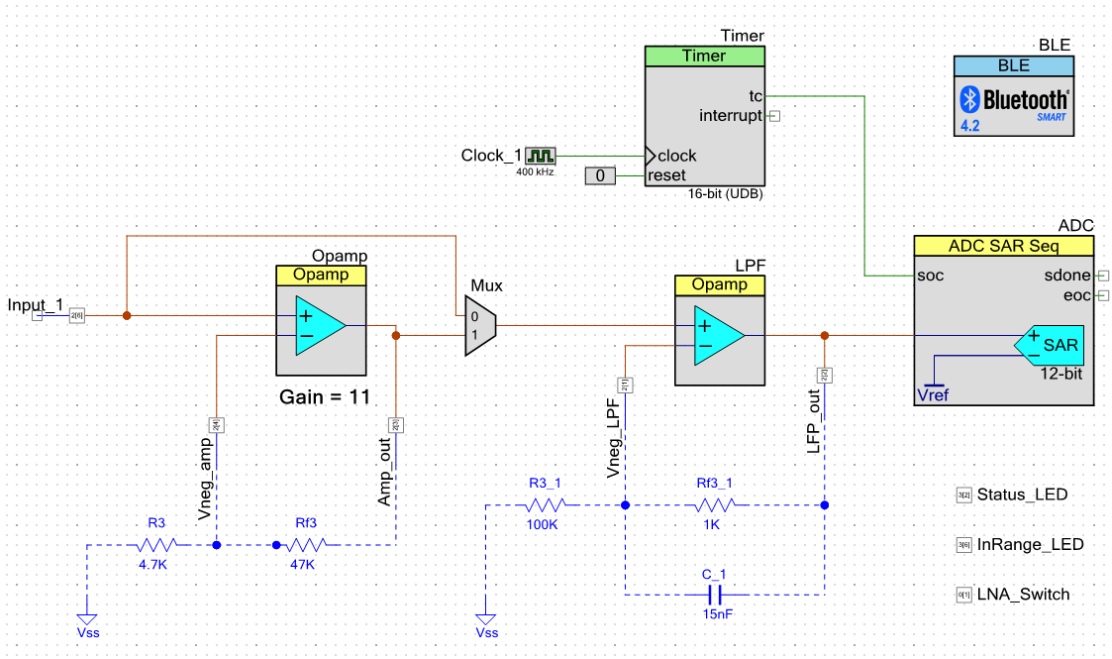


Figure 17. Block diagram of the internal components on the microcontroller's IDE.

The components seen on the block diagram are the microcontroller's hardware parts that are being enabled. *Input_1* is the signal coming from the detector diode. This signal is amplified or not by *Opamp*, depending on its value. The gain of the non-inverting amplifier is defined by the resistors R3 and Rf3 by the formula

$$Gain = 1 + \frac{Rf3}{R3} = 1 + \frac{47000}{4700} = 11. [5] \quad (9)$$

The analogue multiplexer *Mux* is responsible for this switching, and its behaviour is controlled by the firmware.

The non-inverting active low-pass filter *LPF* is responsible for cutting off unwanted high frequencies to increase the signal to noise ratio on the ADC. Its gain A and cut-off frequency f_c can be calculated as following:

$$A = 1 + \frac{Rf3\_1}{R3\_1} = 1 + \frac{1000}{100000} \approx 1, [6] \quad (10)$$

$$f_c = \frac{1}{2\pi \times Rf3\_1 \times C\_1} = \frac{1}{2\pi \times 10^3 \times 15 \times 10^{-9}} \approx 10.6\ kHz, [6] \quad (11)$$

After filtering, the ADC samples the analogue signal at a frequency of 16 kHz, or 16000 samples per second. The sampling is being triggered by the block named *Timer,* which is responsible for under-sampling the acquisition to 4 kHz to regulate the amount of data

being stored on the buffer. The buffer will later be used to detect the source of the signal, and if the sampling rate is too high, the data length will not be long enough for detection. A 12-bit conversion and 4 kHz sampling frequency provide the microcontroller the correct amount of data to allow the firmware to calculate the input power and perform the signal source detection.

The Bluetooth block *BLE* is responsible for connection and exchange of data with the Android device. Its configurations can be changed via a user interface on PSoC Creator and most of the functions and macros are well documented on the datasheet of the component.

6.2.1 Power Calculation

An exponential moving average filter is used to stabilise the input signal, since it fluctuates due to the modulation frequency. It works recursively, and it can be determined by the formula

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}, \tag{12}$$

where α is a constant for smoothing factor between 0 and 1, Y_t is the current value at a time t and S_t is the average at time t. [7] In this project α was kept close to zero to provide smoother transitions.

A calibration table is then used to compare the averaged value obtained with the calibrations results. The corresponding power value is stored and displayed. The value is compared also with a user defined threshold that is used to indicate if the power value exceeds it.

6.2.2 Signal Source Detection

The transmission to be detected uses On-Off Keying modulation to encode the data. This data includes a Barker code and a Preamble to identify the transmitter. The idea of the signal detection is to implement an algorithm to extract these codes and compare with a known value to indicate if the signal comes from the correct source. The purpose of this was to distinguish unwanted signals, i.e. RFID, from the signal on focus to be analysed.

A signal sample of the diode's output was acquired with an oscilloscope, illustrated in Figure 18. As each sample is converted by the ADC, a function assigns it one bit, zero or one, and stores it in a 64-bit buffer. The decision if the sample is going to be high or low is based on a threshold equal to the average ADC input voltage level divided by 1.5 (this value was found experimentally and it showed reliable results). This creates a vector of 64 elements of zeros and ones, which is constantly being shifted as data comes in.

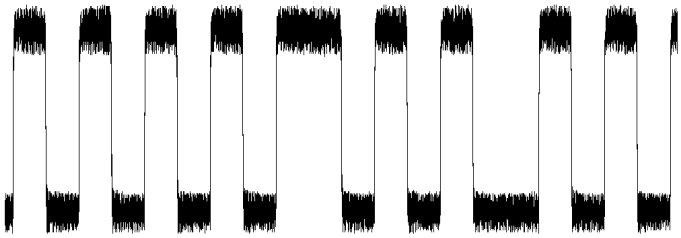


Figure 18. Oscilloscope sample of the diode's output during signal reception.

A 64-bit constant containing the correct value of the encoding message is compared with the buffer by a XOR function, bit by bit. A complete match would give a 64-bit vector of zeros, otherwise each mismatch stores a one on the variable.

To finalize the procedure, a Hamming Weight function is applied on the resulting vector to count the number of ones. The total number of bits, 64 in this case, is subtracted by the count number, and the result is used to determine if there is a match on the decoding. a value higher than 57 is enough to guarantee the signal comes from the transmitter.

7 Conclusion

On this project, multiple fields of electronics were explored to help developing a portable, efficient, and reliable device to measure the power of radio signals. The feature of detecting the signal source is very important for this application, since it allows the user to be sure the correct signal is being detected, and identify if there are unwanted transmissions on the area.

Tests showed the basic principles of the devices were functioning. The power levels were being displayed correctly, the signal source detection was also working, although

at high power levels it had a chance of not detecting the source as expected. The reasons behind that remained unknown. Tweaks on the ADC acquisition clock and correlation threshold made the detection work in an acceptable level.

It was noticed, with the signal source detection functionality, the more aggressive power saving implementation was causing anomalies on data sampling rate. The reduction of clock speeds and time breaks between acquisitions were the main cause of conflict. The solution found was to go back to the Optimization 2, not implementing the LNA switching during connected state.

Connection with the smartphone was stable and reliable, and the application worked as expected. As for working range, the device was kept more than 30 meters away from the smartphone without losing connection, as expected on a Bluetooth 4.2 connectivity. Considering the reduction of power saving methods, the lifetime of the device was still considered long, at an average of 11 hours at connected state or 22 days at advertisement.

The biggest challenges and the points that could use improvements are the antenna impedance matching, due to the uncertainty of the way it interacts with other components; and the time management of the signal chain on the microprocessor, meaning sampling frequencies and processing delays. Those are possibly the main cause of errors on the measurements.

References

1 Lehto A, Räisänen A. RF – ja mikro-aaltotekniikka. Otatieto; 2001. Finnish.

2 Pozar DM. Microwave engineering. 4th ed. John Wiley & Sons, Inc.; 2012.

3 Hall SW, Hall GW, McCall JA, et al. High-Speed Digital Systems Design, A Handbook of Interconnect Theory and Design Practices. New York: John Wiley & Sons, Inc.; 2000.

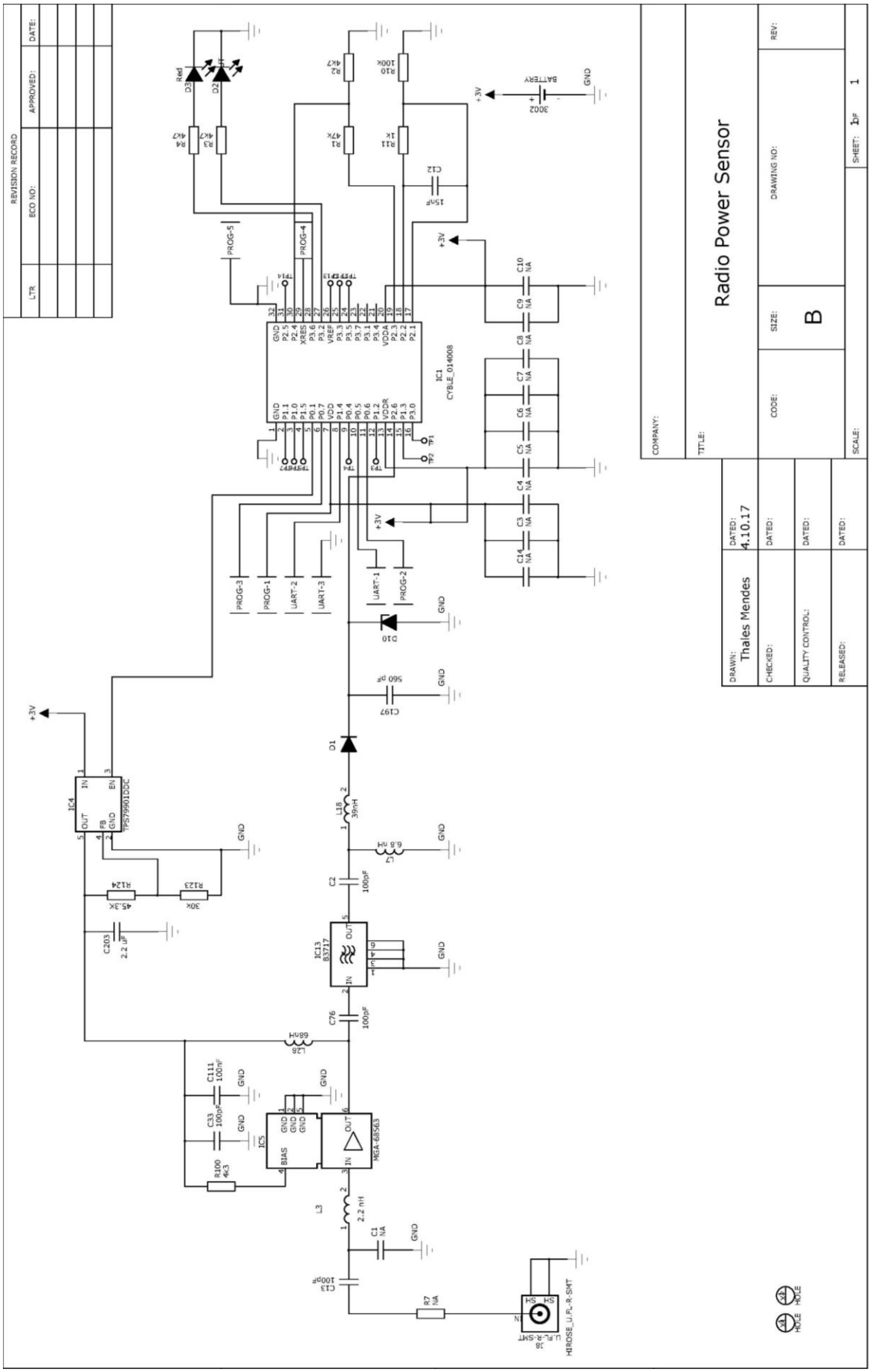
4 Sarvar F, Poole NJ. "PCB glass-fibre laminates: Thermal conductivity measurements and their effect on simulation" Journal of Electronic Materials [Internet]. 1990 [cited 4 Jan 2017]; 19 (12): 1345–1350, Available from: doi:10.1007/bf02662823.

5 Carter B, Brown TR. “Handbook of operational amplifier applications” Texas Instruments. [Internet]. 2001 [cited 4 Jan 2018] 53-54, Available from: http://www.ti.com/lit/an/sboa092b/sboa092b.pdf.

6 Sedra AS, Smith KC. Microelectronic Circuits, 3rd ed. Saunders College Publishing; 1991.

7 NIST/SEMATECH “e-Handbook of Statistical Methods” [Internet]. National Institute of Standards and Technology; 6.3.2.4. EWMA Control Charts [updated 2013 Oct 30; cited 2018 Jan 4]. Available from: http://www.itl.nist.gov/div898/handbook/pmc/section3/pmc324.htm.

Power Detector Schematics



Source Code for Microcontroller

main.c

```
//  Power Detector
//  Thales Mendes

#include "project.h"
#include <main.h>

/* Calibration table */
static conv_t table[TABLE_SIZE] =
{
    {-70, 11},
    {-69, 12},
    {-68, 13},
    {-67, 14},
    {-66, 16},
    {-65, 17},
    {-64, 19},
    {-63, 21},
    {-62, 25},
    {-61, 29},
    {-60, 33},
    {-59, 39},
    {-58, 47},
    {-57, 57},
    {-56, 70},
    {-55, 85},
    {-54, 101},
    {-53, 129},
    {-52, 150},
    {-51, 193},
    {-50, 222},
    {-49, 270},
    {-48, 342},
    {-47, 407},
    {-46, 480},
    {-45, 555},
    {-44, 660},
    {-43, 760},
    {-42, 900},
    {-41, 1010},
    {-40, 1166},
    {-39, 1360},
    {-38, 1551},
    {-37, 1727},
    {-36, 1969},
    {-35, 2310},
    {-34, 2490},
    {-33, 2979},
    {-32, 3160},
    {-31, 3700},
    {-30, 3980},
    {-29, 4300},
    {-28, 4500},
```

```
    {-27, 4750},
    {-26, 4900},
    {-25, 5100},
    {-24, 5300},
    {-23, 5511},
    {-22, 5660},
    {-21, 5840},
    {-20, 6000},
    {-19, 6120},
    {-18, 6250},
    {-17, 6380},
    {-16, 6520},
    {-15, 6633},
    {-14, 6740},
    {-13, 6800},
    {-12, 6900},
    {-11, 7007},
    {-10, 7117},
};

/*'deviceConnected' flag is used by application to know whether device
* has been connected.*/
extern uint8 deviceConnected;

/*'startNotification' flag is set when the central device writes to CCC (
Client
* Characteristic Configuration) to enable notifications */
extern uint8 startNotification;

/* 'restartAdvertisement' flag is used to restart advertisement */
extern uint8 restartAdvertisement;

volatile uint32 windowFlag = 0u;
volatile uint8  dataReady  = 0u;
volatile uint8  switchFlag = 0u;
volatile int adcbufIndex = 0u;
volatile uint16 adcbuf[ADCBUF_SIZE] = {0};
volatile int i = 0;
volatile int j = 0; //Counter for LED flag
uint16 din = 0;
float voltbitConv = 0;

uint16 sample = 0;
uint64_t sig = 0;
//Mirror of the manchester binary for source detection:
1010101010101010101010101010101010101010101010010111010011001110
uint64_t pat =
0b0110011001011010010101010101010101010101010101010101010101010101 ;
uint8 corr = 0;

CY_ISR_PROTO(ADC_ISR_Handler);
```

```
int main(void)
{
    /* This function will initialize the system resources*/
    InitializeSystem();

    for(;;)
    {
        /*Process Event callback to handle BLE events. The events generated and
        * used for this application are inside the 'CustomEventHandler' routine*/
        CyBle_ProcessEvents();

        /* To achieve low power in the device, disabled if source detection is active*/
//        LowPowerImplementation();

        /* Function to handle LED status*/
        HandleStatusLED();

        /* Handle proximity data and CCCD value update only if BLE device is connected */
        if(TRUE == deviceConnected)
        {
            /* When the Client Characteristic Configuration descriptor (CCCD) is written
            * by Central device for enabling/disabling notifications, then the same
            * descriptor value has to be explicitly updated in application so that
            * it reflects the correct value when the descriptor is read */
            UpdateNotificationCCCD();

            /* Enables the LNA BIAS voltage */
            LNA_Switch_Write(1u);

            /*Delay to charge LNA bias capacitors.*/
//            CyDelay( 10 );

            /* Send data when notifications are enabled */
            if(startNotification & CCCD_NTF_BIT_MASK)
            {
                /*Check for data change and report to BLE device
                */
                HandlePowerValue();

                /*Delay to reduce power consumption. Disabled when source detection is active*/
//                  LNA_Switch_Write(0u);
//                  CyDelay( 20 );
            }
        }
```

```
        if(restartAdvertisement)
        {
            LNA_Switch_Write(0u);
            InRange_LED_Write(0u);

            /* Reset 'restartAdvertisement' flag*/
            restartAdvertisement = FALSE;

            /* Start Advertisement and enter Discoverable mode*/
            CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
        }
        j++;
    }
}


void InitializeSystem(void)
{
    ADC_Start();
    Opamp_Start();
    LPF_Start();
    Mux_Start();
    ADC_IRQ_StartEx(ADC_ISR_Handler);


    /* Enable global interrupt mask */
    CyGlobalIntEnable;

    Timer_Start();
    Mux_Select(0);
    Status_LED_Write(0u);

    /* Start BLE component and register the CustomEventHandler function. This
    * function exposes the events from BLE component for application use */
    CyBle_Start(CustomEventHandler);

    /* Set drive mode of the status LED pin to Strong*/
    Status_LED_SetDriveMode(Status_LED_DM_STRONG);
}


void HandlePowerValue(void)
{
    /* Static variable used as counter for reading the new value */
    static uint16 powerCounter = TRUE;
    static uint8 mux_position = 0;
    static uint16 gain = 1;

    /* 'result' stores the voltage level read from ADC */
    uint16 result = 0;

    /* Check if powerCounter has reached its counting value */
```

```
    if(FALSE == (--powerCounter))
    {
        /* Set 'powerCounter' to the POWER_COUNTER_VALUE. This counter prevents sending
           of large number of power data to BLE Central device */
        powerCounter = POWER_COUNTER_VALUE;

        /* Wait for ADC conversion */
        while(dataReady == 0u)
        {
            ;
        }

        result = ADC_CountsTo_mVolts(0u, filter(din));

        /* Mux Routine - Switching of the amplifier*/
        if(result > 1098)
        {
            mux_position = 0;
            gain = 11;
            Mux_FastSelect(mux_position);
        }
        if(mux_position == 0 && result <= 98)
        {
            mux_position = 1;
            gain = 1;
            Mux_FastSelect(mux_position);
        }

        //Generate threshold for 1-bit conversion of the signal
        BitConvThreshold(gain*result);

        //LED indication if the correlation is big enough
        CorrPass(Correlation(gain*ADC_CountsTo_mVolts(0u, din)));

        /* Send power data to BLE device by notifications*/
        SendDataOverNotification(PowerConversion(table, gain*result));

        //LED to indicate if power is over the threshold
        if( PowerConversion(table, gain*result) <= POWER_THRESHOLD && PowerConversion(table, gain*result) !=0) InRange_LED_Write(1u);
        else InRange_LED_Write(0u);

        dataReady = 0u;
    }
}

/*Uses data from the Line-up analysis to convert the voltage value to its corresponding power value*/
int PowerConversion(conv_t* table, uint16 result)
{
    for( int i = 0; i < TABLE_SIZE - 1 ; i++ )
```

```
    {
        if( table[i].volts < result && result <= table[i+1].volts)
        {
            result = -table[i+1].decibels;
            return result;
            break;
        }
    }
    return 0;
}

/* Low power implementation to reduce CPU usage*/
void LowPowerImplementation(void)
{
    CYBLE_LP_MODE_T bleMode;
    uint8 interruptStatus;

    /* For advertising and connected states, implement deep sleep   *
     * functionality to achieve low power in the system.            */
    if((CyBle_GetState() == CYBLE_STATE_ADVERTISING) ||
       (CyBle_GetState() == CYBLE_STATE_CONNECTED))
    {
        /* Request BLE subsystem to enter into Deep-Sleep mode between
connection and advertising intervals */
        bleMode = CyBle_EnterLPM(CYBLE_BLESS_DEEPSLEEP);
        /* Disable global interrupts */
        interruptStatus = CyEnterCriticalSection();
        /* When BLE subsystem has been put into Deep-Sleep mode */
        if(bleMode == CYBLE_BLESS_DEEPSLEEP)
        {
            /* And it is still there or ECO is on */
            if((CyBle_GetBleSsState() == CYBLE_BLESS_STATE_ECO_ON) ||
               (CyBle_GetBleSsState() == CYBLE_BLESS_STATE_DEEPSLEEP))
            {
                CySysPmDeepSleep();
            }
        }
        else /* When BLE subsystem has been put into Sleep mode or is active */
        {
            /* And hardware doesn't finish Tx/Rx opeation - put the CPU into
Sleep mode */
            if(CyBle_GetBleSsState() != CYBLE_BLESS_STATE_EVENT_CLOSE)
            {
                CySysPmSleep();
            }
        }
        /* Enable global interrupt */
        CyExitCriticalSection(interruptStatus);
    }
}

//Counts the number of 1 bits on the argument
```

```
int popcount64(uint64_t x)
{
    const uint64_t m1  = 0x5555555555555555; //binary: 0101...
    const uint64_t m2  = 0x3333333333333333; //binary: 00110011..
    const uint64_t m4  = 0x0f0f0f0f0f0f0f0f; //binary:  4 zeros,  4 ones ...
    const uint64_t h01 = 0x0101010101010101; //the sum of 256 to the power of 0,1,2,3...

    x -= (x >> 1) & m1;             //put count of each 2 bits into those 2 bits
    x = (x & m2) + ((x >> 2) & m2); //put count of each 4 bits into those 4 bits
    x = (x + (x >> 4)) & m4;        //put count of each 8 bits into those 8 bits
    return (x * h01) >> 56;  //returns left 8 bits of x + (x<<8) + (x<<16) + (x<<24) + ...
}

//LED to indicate if the source detection triggered
void CorrPass(uint8 counts)
{
    if( counts >= MIN_CORR)
    {
        Status_LED_Write(1u);
    }
    if(  j > LED_ON_COUNTER )
    {
        Status_LED_Write(0u);
        j = 0;
    }
}

uint8 Correlation(uint16 sample)
{
    //Convert sample to 1 bit based on a threshold
    if( sample >= voltbitConv ) sample = 1;
    else sample = 0;

    //sample bit is added to the ringbuffer sig
    sig = (sig << 1) | sample;

    //Compare the 64-bit signal to a pattern of 64 bits and calculate the correlation of the signal
    corr = 64-popcount64(sig ^ pat); //sig XOR pat
    return corr;
}

// Exponential Moving Average filter
int16_t filter(int16_t value) {
    static int64_t avg = 0;
    const int64_t gain = 1;
```

```
    int64_t val = ((int64_t)value) << 11;
    avg = (gain*val + (2048-gain)*avg) >> 11;
    return (int16_t)(avg >> 11);
}

// Threshold for convertion of samples to 1-bit
void BitConvThreshold(uint16 input)
{
    voltbitConv = input/1.5;
}

// Interrupt handler for the ADC
CY_ISR(ADC_ISR_Handler)
{
    uint32 intr_status;

    /* Read interrupt status registers */
    intr_status = ADC_SAR_INTR_MASKED_REG;
    /* Check for End of Scan interrupt */
    if((intr_status & ADC_EOS_MASK) != 0u)
    {
        /* Read range interrupt status and raise the flag */
        windowFlag = ADC_SAR_RANGE_INTR_MASKED_REG;
        /* Clear range detect status */
        ADC_SAR_RANGE_INTR_REG = windowFlag;
        if (dataReady == 0)
        {
            din = ADC_GetResult16(0);
            dataReady = 1;
        }
    }
    /* Clear handled interrupt */
    ADC_SAR_INTR_REG = intr_status;
}
```

main.h

```
/*
*********************************************************************************
*   Contains all macros and function declaration used in the main.c file
*********************************************************************************
*/
#if !defined(MAIN_H)
#define MAIN_H

#include <project.h>
#include <BLEApplications.h>

/*Calibration
:                                                                    */
/*Definition of conversion table from mV to dBm, according to measurement data
.*/
#define TABLE_SIZE                                  61
typedef struct { int decibels; double volts; } conv_t;

/*************************Function Declarations*****************************/
void InitializeSystem(void);
void HandlePowerValue(void);
void CustomEventHandler(uint32 event, void * eventParam);
int PowerConversion(conv_t* table, uint16 result);
void LowPowerImplementation(void);
uint8 Correlation(uint16 sample);
void CorrPass(uint8 counts);
int16_t filter(int16_t value);
void BitConvThreshold(uint16 input);

/*************************Macro Definitions*****************************/
#define TRUE                                    1
#define FALSE                                   0
#define ZERO                                    0

/*This counter prevents pushing
* large number of data to connected BLE Client. Modify
* this counter to change the rate of notification of data*/
#define POWER_COUNTER_VALUE                     1           //Default 5
#define LED_ON_COUNTER                          0xFFFF
#define ADCBUF_SIZE                             25          //Limited by RAM
overflow on ~2000
#define POWER_THRESHOLD                         52          //dBm
#define MIN_CORR                                57          // 0 - 64

#endif
```

Source Code for Android Application

```
package com.example.thalessampaio.ble;


import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ScrollView;
import android.widget.TextView;

import java.util.UUID;

public class MainActivity extends AppCompatActivity {


    private static final int REQUEST_ENABLE_BT = 1;
    private static final UUID thalesServiceUUID =
            UUID.fromString("0000CAB5-0000-1000-8000-00805F9B34FB");
    private static final UUID thalesCharacteristicUUID =
            UUID.fromString("0000CAA1-0000-1000-8000-00805F9B34FB");
    // Defined by the BLE standard
    private static final UUID clientCharacteristicConfigurationUUID =
            UUID.fromString("00002902-0000-1000-8000-00805F9B34FB");

    private BluetoothAdapter bluetoothAdapter;
```

```
private BluetoothGatt bleGatt;
private boolean scanning;
private TextView logView;
private ScrollView scrollView;

private final BluetoothGattCallback gattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(final BluetoothGatt gatt, int status, int newState) {
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    log("connection established, discovering services");
                    // Once we have connection start BLE service discovery
                    gatt.discoverServices();
                }
            });
        }
    }

    @Override
    public void onServicesDiscovered(final BluetoothGatt gatt, int status) {
        runOnUiThread(
                new Runnable() {
                    @Override
                    public void run() {
                        enableStreamingNotification(gatt);
                    }
                });
    }

    private void enableStreamingNotification(BluetoothGatt gatt) {
        log("services discovered, enabling notifications");
        // Get the MM service
        BluetoothGattService mmService = gatt.getService(thalesServiceUUID);
```

```
        // Get the streaming characteristic
        BluetoothGattCharacteristic streamingCharacteristic =
                mmService.getCharacteristic(thalesCharacteristicUUID);
        // Enable notifications on the streaming characteristic
        gatt.setCharacteristicNotification(streamingCharacteristic, true);
        // For some reason the above does not tell the ring that it should
        // start sending notifications, so we have to do it explicitly
        BluetoothGattDescriptor cccDescriptor =
                streamingCharacteristic.getDescriptor(clientCharacteristicConfigurationUUID);
        cccDescriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
        gatt.writeDescriptor(cccDescriptor);
    }

    @Override
    public void onCharacteristicChanged(
            BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
        final byte[] payload = characteristic.getValue();
        // Decode payload
        final int value = (payload[0] & 0xff) | ((payload[1] & 0xff) << 8);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                logView.setText(String.format("%d", value ));
            }
        });
    }
};

private final BluetoothAdapter.LeScanCallback scanCallback =
        new BluetoothAdapter.LeScanCallback() {
            @Override
            public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {
                log("found device %s, rssi: %d", device.getAddress(), rssi);
                stopScanning();
                runOnUiThread(
```

```
                        new Runnable() {
                            @Override
                            public void run() {
                                log("connecting with %s", device.getAddress());
                                // Establish a connection with the device.
                                bleGatt = device.connectGatt(MainActivity.this, false, gattCallback);
                            }
                        });
                }
            };

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    logView = (TextView) findViewById(R.id.logView);
    scrollView = (ScrollView) findViewById(R.id.scrollView);

    // Get the bluetooth manager from Android
    final BluetoothManager bluetoothManager =
            (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
    if (bluetoothManager == null) {
        log("bluetooth service not available");
        return;
    }
    // Get a reference to the bluetooth adapter of the device
    bluetoothAdapter = bluetoothManager.getAdapter();
    if (bluetoothAdapter == null) {
        log("bluetooth not supported");
        return;
    }
    // Check that bluetooth is enabled. If not, ask the user to enable it.
    if (!bluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
```

```
            return;
        }
        // If bluetooth was enabled start scanning for devices.
        startScanning();
    }

    // Called after user has allowed or rejected our request to enable bluetooth
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_ENABLE_BT) {
            if (resultCode == RESULT_OK) {
                startScanning();
            } else {
                log("bluetooth not enabled");
            }
        }
    }

    private void startScanning() {
        if (scanning) return;
        scanning = true;
        log("scanning for devices");
        // Start scanning for devices that support the Moodmetric service
        bluetoothAdapter.startLeScan(new UUID[]{thalesServiceUUID}, scanCallback);
    }

    private void stopScanning() {
        if (!scanning) return;
        scanning = false;
        log("scanning stopped");
        bluetoothAdapter.stopLeScan(scanCallback);
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        if (scanning) stopScanning();
        if (bleGatt != null) {
            bleGatt.close();
        }
    }

    private void log(final String fmt, final Object... args) {
        // Updates to the UI should be done from the UI thread
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                logView.append(String.format(fmt + "\n", args));
                scrollView.fullScroll(View.FOCUS_DOWN);
            }
        });
    }

}
```