Wake-Up Harvester Design for Batteryless IoT System

FINAL REPORT

sdmay21-14

Henry Duwe

Jacob Bernardi, Edmund Duan, Douglas Zuercher, Kwanghum Park, Bryce Staver, Zacharias Komodromos

sdmay21-14@iastate.edu

https://sdmay21-14.sd.ece.iastate.edu

Table of Contents

Table of Figures

1 Introduction

1.1 Acknowledgements

The senior design team would like to formally acknowledge and thank Henry Duwe, Assistant Professor at Iowa State University, for his technical advice on wake-up radios and low-power IoT devices as well as the use of his lab space and components. His experience and knowledge has helped guide the team and provide insight on potential issues which may be faced.

Additionally, we would like to thank Nathan Neihart, Assistant Professor at Iowa State University for his time and help on debugging as well as the use of his lab space and equipment, which was invaluable for debugging transmission and reception issues.

Finally, the team would also like to thank Vishak Narayanan, one of Professor Duwe's graduate students, for his assistance in testing and providing access to prior research. His prior research and experience with the MCU used by the team dramatically decreased the learning curve and allowed us to obtain second-opinions on suspected issues.

1.2 Project Statement Summary

Many Internet of Things devices are designed to be batteryless. In order for them to operate, some make use of an energy harvesting system. Once power is harvested, it is stored in a capacitor, which is then used to power the device with values neither constant nor of high magnitude. The result is that it is highly likely, if not guaranteed, that there will be times when the system is not powered when dealing with a batteryless device. Our purpose is to provide such a device for Dr. Duwe and his team so that the device can enter a low power state, and upon an RF trigger to wake up and complete some task.

The solution presented by our team to this problem is the creation of a device which utilizes a wake-up system to wait for a wake-up signal to be received. Upon reception, the wake-up system will interrupt the microcontroller (MCU) to wake it up from a low power mode to normal operation. In doing so, we end up with a system where a user is able to instruct the MCU to follow a process, and it will enter a sleep mode when done. If it needs to activate at a certain time, it can do so by itself, but if the user wants an external stimulus, they can use the wake up trigger to accomplish this. This will create a more power-efficient system that wakes up the IoT device. It allows for greater ability for IoT devices to be synchronized by the trigger signal, that is repeated by a device when received, and to perform tasks in unison if they have enough charge to be powered on and are in sleep-mode.

1.3 Previous Work And Literature

Although we are the first senior design team to implement a Wake-Up harvester, the concept of Wake-Up radios has existed before, despite being relatively recent [1]. Wake-Up radios are described as devices that are active only when they receive a specific signal or trigger and many have been made in the past few years [1]. In our system, we are using a premade component for our wake-up radio in addition to premade components for our power harvester, MCU, and DC-DC regulator.

1.4 Design Revisions

Since the previous semester, the test team has made several changes to the originally proposed design. At the request of the client, the team has added the capability for the MCU located on the provided board to transmit at RF frequencies without the aid of a transceiver. While this method of transmission is not recommended for low-power purposes, it will provide an additional means for the client to perform research. Additionally, due to the silicon shortage, the team was not able to obtain some components. The result of this is that the current working design does not use an additional boost converter to power the transceiver, and so it is also powered off the harvester output.

While modifications have been necessary due to supply chain shortages, the team has worked to minimize the impact of these shortages on our client. Switches have been added so that, when components can be ordered, the current product will still be able to function as initially intended.

1.5 Operational Environment

The end-product is expected to be operated in indoor commercial environments, such as office buildings and lab spaces, where electro-magnetic noise is not of significant concern and where harvestable RF signals are prominent. The environment is expected to be clean and climate-controlled such that dust and humidity are unlikely to interfere with electronics.

1.6 Requirements

The functional requirements for this system are as follows:

- The system should reject incorrect trigger signals, activating only when a valid trigger has been received
- The system should repeat the trigger signal to other nodes once received
- When a trigger is received, the device should enter normal operation
- The maximum average off time of a node should be no greater than one minute if a source is within 1m
- The device is required to be easily interfaceable with user-provided sensors

The nonfunctional requirements for this system are as follows:

- The size of any node in the complete system should be no greater than 4” x 4”
- Each node should be batteryless, instead storing collected energy on a capacitor
- Nodes must be able to send and receive a trigger signal a distance of at least two meters
- Nodes should operate correctly in temperatures above 0°C and below 50°C.

1.7 Standards

In terms of standards implemented in this project, we took the following into consideration:

ITU Radio Regulations for ISM bands

- License-free bands where we can transmit with limited regulations
- Transceiver transmits at 433 MHz (acceptable range: 433.05 - 434.79 MHz)
- RF energy harvested at 915 MHz (acceptable range: 902 - 928 MHz)

IPC 7351A & B

- Includes standards on the design of surface mount land patterns
- Includes adjustments for lead-free reflow cycles and newer land patterns, such as QFN devices
- IPC standards required by Oshpark in order to manufacture PCBs

IPC-A-610

- Guidelines for assembly of PCB, including hardware installation, soldering criteria, and various connection requirements

1.8 Engineering Constraints

For this project, we had to consider these two types of limitations.

Limitations of the environment

- Nodes will be in an extremely noisy environment
- High levels of humidity, dust, temperatures
- Availability of RF energy to harvest

Limited efficiency of RF harvesting technology

- Nodes are only able to harvest so much energy over a given period of time
- MCU will consume a relatively large amount of power when in an on-state
- Using lower voltage leads to less efficiency which causes wasted power

1.9 Security Concerns and Countermeasures

1.9.1 Physical Security

The trigger is not intended to be secure from external sources; meaning that if someone knows the frequency pattern being used, they will be able to trigger the IoT device if it has enough charge to support the device being active.

Potential countermeasures could include creating a unique preamble that would need to be present in the transmission to allow wake up of the MCU. This would add a level of security to the wake up functionality that has been added by the team and is not known to the public. This countermeasure was not implemented in this project as the main concern was functionality, not a high level of security, and can easily be set by users later on.

1.9.2 Cybersecurity

This device is not connected to the internet or any other sort of local network so there is no security concern of someone gaining access to the system from a remote location. A potential "attacker" would only be able to disrupt the system if they were physically next to a node, and this is discussed above.

2 Implementation

2.1 Overall Design

This project is implemented on a printed circuit board or PCB as a one-piece system. Mounting holes have been added so that individual nodes can be mounted in locations defined by the user. The three main components, being the harvester, MCU, and wake-up radio (WuR) have been tested individually as well as together. Our final tested design, which as mentioned in Section 1.2 is different from our proposed and intended design, is depicted in the image below. The harvester will be able to provide the system with power, while the MCU is able to complete its tasks and communicate using RF through the WuR component. While the MCU is not in use, it can set itself into a sleep mode for low power consumption. If the WuR receives a wake-up signal, it will be able to interrupt the MCU to exit its sleep mode.

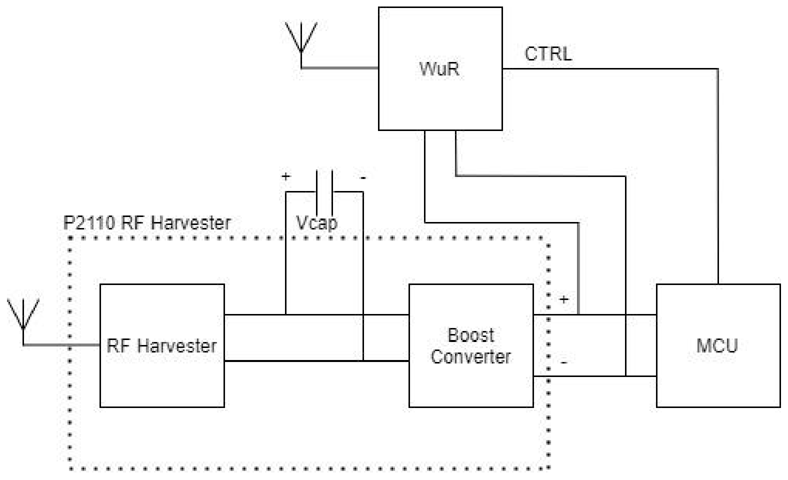

Figure 1: Current Working Design

2.2 PCB Implementation

To create the PCB, we used Altium PCB designer. From the left to right side of the board as seen in Figure 2, we started with inputs and organized components by function before reaching and working on the outputs. All necessary information for making the RF traces came from the PCB manufacturer's, Oshpark's, two-layer prototype section, which included dielectric constant and manufacturing capabilities among other aspects. Capabilities were an important factor to consider since it could decide whether the board was manufacturable or not. On the left side of the PCB, the harvester SMA connector is connected to the harvester using a grounded coplanar waveguide. To avoid ESD issues, the team also created an RF ground for the SMA connectors by adding a gap between the grounds. This was done because the team wanted a ground pour on the top of the PCB with vias stitching the top and bottom ground layers together to make a lower inductance ground.

The waveguide was created to make the width of the trace or microstrip as close as possible to the width of the connector and harvester input. The waveguide gap is tightly coupled at 6 mils to shrink the microstrip and maintain a 50 Ω impedance match. Lastly for the RF input, to avoid excessive attenuation, the line was made as short as possible.

After the harvester, we branch to a banana connector and slide switch. The slide switch enables power for the MCU and the banana connector is for testing the MCU without needing to depend on the harvester's erratic supply. The harvester also has a large capacitor for storing energy for the device to use. We provide a connection to a secondary boost converter as per our original design to increase efficiency with the lower loads for the transceiver which is on the lower right side of the board. In order to account for the boost converter not being present on the board due to it being out of stock, a switch was added to bypass the connection.

The MCU which is located above the previously-mentioned section has a header pin for programming and supporting circuitry including debugging LEDs and connectors for the user. This is surrounded by a copper pour for powering the MCU. This copper pour for power was made to reduce noise by not creating a random distribution of power, a similar reason being applied for the ground pours on the top and bottom of the board. As this board does not carry any high power sections, most traces were kept at the Altium default of 10 mil which is more than enough of a width for this application. Moving on to the test points, they were kept on the edges of the board to ensure easy access for the user. Lastly is an RF balun circuit for the MCU to have Rx and Tx capabilities. This section follows the same guidelines as the harvester input in order to avoid the above mentioned issues.

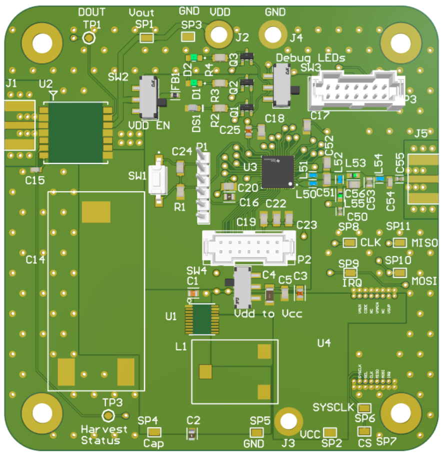

Figure 2: PCB layout

2.3 MCU Software Implementation

To take advantage of pre-existing libraries and MCU functionality, as well as satisfy client requests, the MCU was first loaded with TI-RTOS, allowing for the use of a set of libraries provided by Texas Instruments for their devices. Using the TI-RTOS provided dramatically more functionality and a better user experience since it decreases the amount of low-level code necessary. The different additional tools used include the SPI client, a timer client and its GPIO pins for LEDs and interrupts.

Once TI-RTOS was implemented, the team proceeded by generating a set of standard SPI methods which can be called from anywhere in the program, and then began writing standard methods for communicating with the transceiver. The goal of these methods is to make it possible to operate the transceiver easily from anywhere in the program, and includes transmit, receive, Tx/Rx configuration, and PLL ranging methods.

With standard methods written, the team proceeded by testing the methods with the transceiver and then creating an example program which implements the base-functionality requested by the client. This program allows for a node to be designated as either a controller or an agent. If designated to be a controller, the MCU will enter an infinite loop where the MCU first instructs the transceiver to send a packet, and then to listen for a packet to be sent back. The MCU will wait for the packet to be returned for some period of time, which can be user defined. If the packet has not been returned in the specified time period, the MCU assumes neighboring agents missed the packet and restarts the loop. If the packet is received in the specified period of time, the MCU increments a count and restarts the loop.

If a node is instead designed to be an agent, the MCU will enter an infinite loop where the MCU first instructs the transceiver to listen for a packet. The transceiver will listen for a packet indefinitely, without timeout, until a packet is received. Once received, the MCU will then re-transmit the packet using the transceiver before returning to a listen-only mode.

By configuring one device as a controller and one device as an agent, the ability for a node to receive, and then re-transmit, some wake-up signal can be verified. If additional nodes are manufactured and added to the system, they will all wake up. This achieves the project's base functionality.

2.4 Wake-Up Radio Implementation

The wake-up radio (WuR) component, which is an AX5043 transceiver, requires external circuitry to function appropriately. This circuitry is predominantly for the transceiver's RF input and output, although some is also necessary for the transceiver's reference clock. Because it was important for the team to obtain a usable transceiver as soon as possible for software purposes, the team opted to purchase an evaluation-board with the transceiver and supporting circuitry already implemented instead of designing our own board. After using the evaluation-board, the team decided to stick with the board and implement the WuR on the final product using two headers which the evaluation board can plug in to.

Because the evaluation board is intended to be used in a larger evaluation system sold by ON Semiconductor, several signals on the evaluation board are not used in our design. One component on the evaluation board, the TCXO, does not have a part number provided. To confirm the TCXO operates at the expected frequency and can be powered by a 3.3V connection, the team contacted ON Semiconductor. ON Semiconductor provided the TCXO part number (NT2016SA from NDK), and from that the test team was able to confirm that the TCXO can be powered by a 3.3V connection and does output a 48MHz signal.

In terms of software, the transceiver requires an MCU to function. The WuR communicates with the MCU through an SPI bus. The transceiver interprets the SPI select line being low as an "active" signal, meaning it will only interpret data on the MOSI line when select is low. The SPI clock positive-edge triggered, where the positive-edge is expected to be located in the middle of any MOSI/MISO data packet. Through this bus, the CC1352R MCU is able to read to and write from the AX5043's registers and the transceiver can be configured. This also allows for the transceiver's operation mode to be set, along with the data which will be sent.

In addition to the SPI bus, the AX5043's IRQ pin is also connected to the MCU to implement the wake-up component of the project. In the default configurations offered by the senior design team, the transceiver is configured to set the IRQ pin high when it receives wireless data. This is used by the MCU to determine when a wake-up signal has been received, at which point user code is run.

The AX5043 transceiver is extremely versatile and has many registers which can be set to achieve a variety of different operating modes, power levels, and so on. The test team used ON

Semiconductor's "AXRadioLab" software to calculate the register values for the options needed. Details on how to use the MCU to configure and use the AX5043 are provided in the operation manual, which can be found in the report Appendix.

3 Testing

The main parts of the project that were tested in isolation were the RF harvester's power harvesting capabilities, the ability for the MCU to communicate and program the WuR, and the ability for a WuR to interrupt and wake up an MCU. All these were tested in controlled environments.

3.1 Harvester Testing

For the harvester, the capacitor can ideally store 0.0391 J of energy and has a value of 50 mF. In terms of testing, we set the Powercast harvester at varying distances from the transmitter and with five loads/resistors at each distance. The distance was measured using a yardstick on a lab bench and the resistors were measured using a digital multimeter before use. We used the oscilloscope to probe the harvester output voltage and measured the on and off times for each test set, adjusting when necessary. A note on testing energy harvesting devices such as this, is that any slight changes can have noticeable effects on the data collected. This includes changes in weather and the testing environment. Something we noticed over a few weeks of testing is that the results varied when in a larger room compared to a smaller lab, and the temperature outside seemed to affect results slightly. The results we collected, as mentioned, are only over the five resistance values. Using this data one can calculate the current available by the harvester for different loads as well as the duration it can provide it for. Below are plots of our results, with the table of the results in the appendix under data:

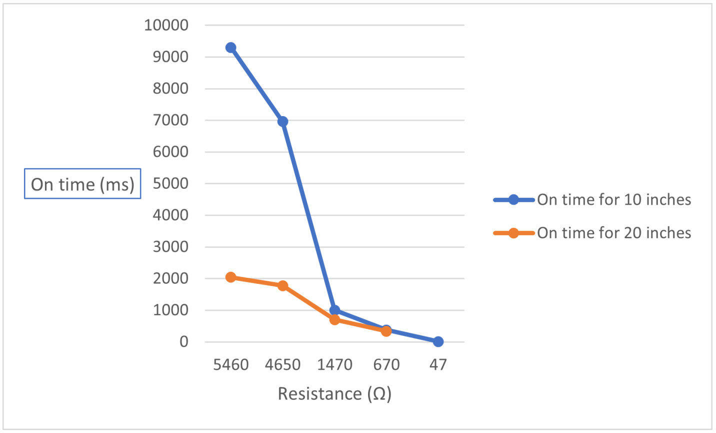

Figure 3: Powercast Harvester On Time Graph

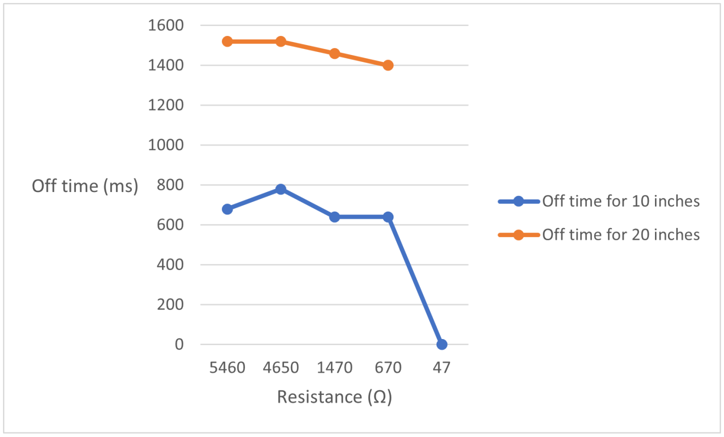

Figure 4: Powercast Harvester Off Time Graph

3.2 MCU to WuR Testing

To test the MCU's ability to program and communicate with the AX5043 we used a LaunchXL-CC1352R, an evaluation board offered by Texas Instruments that includes a CC1352R and appropriate debugging circuitry on. This allowed the team to easily debug and program the MCU using a micro-USB to USB connection with a computer running Code Composer Studio. The launchpad's MCU pins were connected using jumpers to a breadboard that housed the AX5043 which was on its own testboard. Once all the necessary pins were connected, the team used TI's functions to open an SPI connection at 4MHz with a clock phase such that the rising edge is centered to the data being communicated. Using a test register on the AX5043 called "SCRATCH", the team was able to get the MCU to write a value to that register and read it back. By checking that the read-back data matched with the written data, the team confirmed the ability for the MCU and WuR to communicate with each other. The SPI communication was further confirmed by using a logic analyzer to monitor the SPI line and decode data while example data was sent to/from the WuR. Sample results from the logic analyzer are shown below.

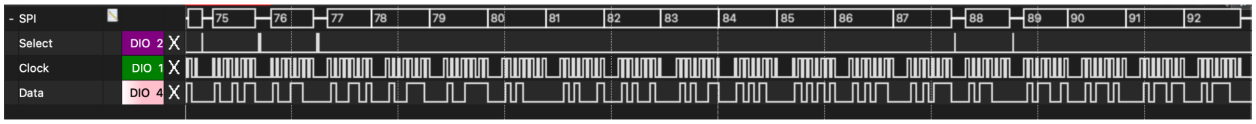

Figure 5: Oscilloscope Reading of SPI

3.3 WuR to Interrupt MCU Testing

To test the wake up capability of the WuR, the team configured one MCU to be a "controller" and another MCU to be an "agent". Each MCU had a WuR device connected to it. The controller was configured to create and transmit a packet at the desired carrier frequency repeatedly without changing power modes. The agent was then configured into "continuous receive" mode, where the transceiver constantly waits for a signal to be received. The agent was configured to use its IRQ pin such that, when a valid packet was received, it would interrupt the connected MCU and inform it that a message had been received. The MCU then read the received data and instructed the transceiver to re-transmit it before returning to a continue receive mode.

Using this setup, the team was able to not only confirm the ability of the WuR to interrupt the MCU, but also to observe basic information on how transmission range varies with transmission power. To observe this information, the team configured the controller to use three different transmission power values and moved the agent away slowly, tracking the number of successful “pings” at each distance. This was done using low transmission power values since this optimizes for power efficiency.

A graph of the number of times the device was able to receive a packet and repeat it in one second, for 3 different transmit power values versus distance, is provided below.

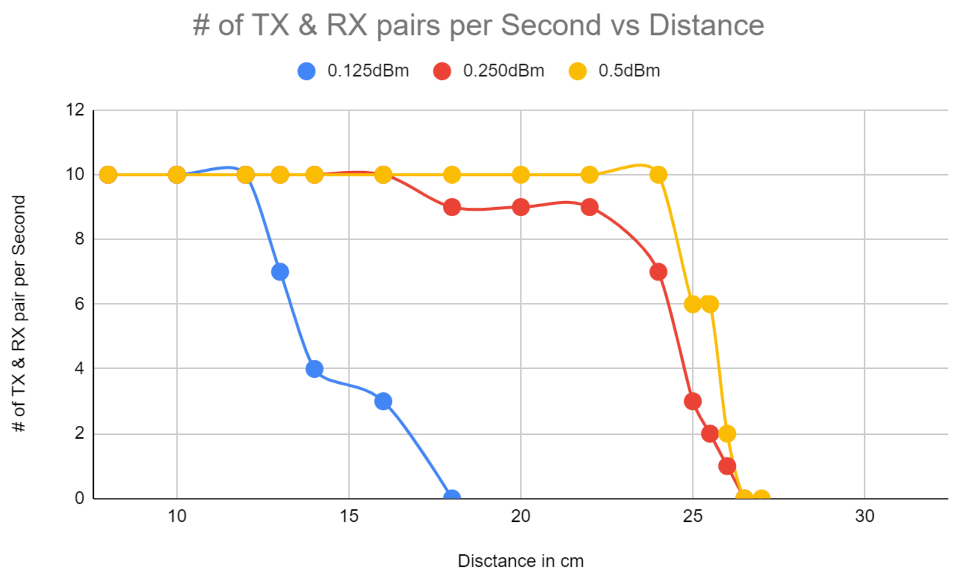

Figure 6: # of Tx and Rx pairs per Second vs Distance

It is observed that the maximum transmission distance is severely limited, however we note that this is achieved at a very low transmission power value. Additional testing at larger transmission power values shows the ability to transmit over several meters, however the power consumption in this scenario increases dramatically and further testing to determine the validity of using these power values is necessary. Further, note that the data was collected while the transmitter was sending at 866MHz and the attached antennas were centered at 433MHz, meaning the reported distances are lower than the distances expected to be observed during normal operation of the final product.

Because the controller device was instructed to repeatedly transmit data, it consumed significantly more power than is expected to be observed during normal use of the provided product. The current consumption observed by the team was between 13.08mA to 14.45mA at 3.3V when transmitting at 0.125dBm, 13.57mA to 14.98mA at 3.3V when transmitting at 0.25dBm, 14.82mA to 16.35mA at 0.5dBm. These values were found by measuring the peak current consumption in a 5 second window, long enough for approximately 324 transmissions to occur.

Because the agent device was instructed to operate in continuous receiver mode instead of wake-on radio mode, the agent power consumption was also larger than expected to be observed during normal operation of the final product. The average current consumption on the agent observed by the team was between 7.62mA to 8.4mA at 3.3V at 0.125dBm, 7.7151mA to 8.52mA at 3.3V when transmitting at 0.25dBm, and 7.875mA to 8.695mA at 0.5dBm. Note that this current was measured while the agent was switching between standby, receiving, and transmitting modes. Further, a small increase in current is observed when the transmission power is increased, however the increase is small since the transmission time is relatively short compared to the receiver time.

4 Appendices

4.1 Operation manual

4.1.1 Hardware Overview

At a high level, the hardware is composed of three major components, with the remainder of the hardware consisting of supporting circuitry. These major components are the RF harvester, the MCU, and the transceiver. The MCU and RF harvester are soldered directly to the PCB, eliminating the need for complicated external RF wiring to other boards. The transceiver is an external evaluation-board offered by ON Semiconductor which connects to the PCB through a set of male headers (item 1 in the below figure). The antennas used by the RF harvester and MCU (for operational MCU-based transmit/receive) connect to the PCB through two SMA connections (item 2 and 3, respectively). The antenna used by the transceiver connects to the SMA connector located on the evaluation board labeled “RX/(TX)”.

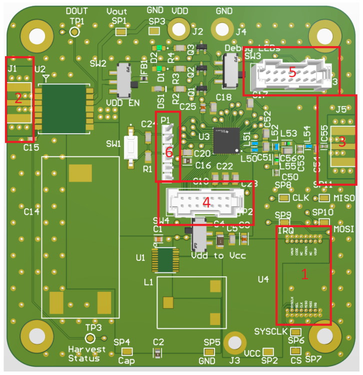

Figure 7: PCB with major components highlighted

To interact with the MCU externally, two 7x2 male connectors are provided (P2 and P3, or items 4 and 5 in the above figure). The connectors are designed to interface with either the female version complimentary to this or female jumper cables depending on the application and permanence thereof. Each connector has the following pin number scheme:

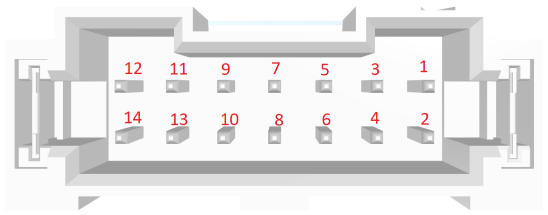

Figure 8: MCU port on PCB

Together, these connectors allow for the following signals to be externally accessed and used.

Signal Name	Connector	Connector Pin
MCU Pin 12 (DIO_7)	P2	14
MCU Pin 14 (DIO_8) / Debug LED	P3	1
MCU Pin 15 (DIO_9) / Debug LED	P3	2
MCU Pin 16 (DIO_10) / Debug LED	P3	3
MCU Pin 17 (DIO_11)	P3	4
MCU Pin 18 (DIO_12)	P3	5
MCU Pin 19 (DIO_13)	P3	6
MCU Pin 20 (DIO_14)	P3	7
MCU Pin 21 (DIO_15)	P3	8
MCU Pin 26 (DIO_16)	P3	9
MCU Pin 27 (DIO_17)	P3	10
MCU Pin 28 (DIO_18)	P3	11
MCU Pin 29 (DIO_19)	P3	13
MCU Pin 30 (DIO_20)	P3	12
MCU Pin 31 (DIO_21)	P3	14
MCU Pin 32 (DIO_22)	P2	1

MCU Pin 36 (DIO_23)	P2	2
MCU Pin 37 (DIO_24)	P2	3
MCU Pin 38 (DIO_25)	P2	4
MCU Pin 39 (DIO_26)	P2	5
MCU Pin 40 (DIO_27)	P2	6
MCU Pin 41 (DIO_28)	P2	7
MCU Pin 42 (DIO_29)	P2	8
MCU Pin 49 (GND)	P2	11, 13
MCU Pins 13, 22, 34, 44, 45, 48 (VDD)	P2	10, 12
Transceiver IRQ	P2	9

For programming purposes, a JTAG interface is also provided. The interface utilizes a 6-pin male header (P1, or item 6 in the above figure) to break out the necessary JTAG signals for external programming.

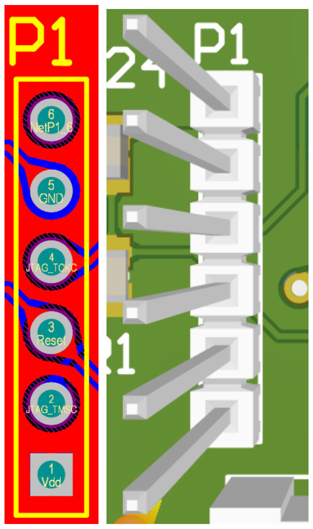

Figure 9: PCB JTAG Interface

Signal Name	Connector	Connector Pin
No Connection	P1	1
GND	P1	2
JTAG_TCKC	P1	3
Reset	P1	4
JTAG_TSMC	P1	5
Vdd	P1	6

4.1.2 Hardware Configuration

For research, testing, and debugging purposes, the hardware can be configured in a number of ways. Configuration changes are made possible by three switches located on the board, as shown below.

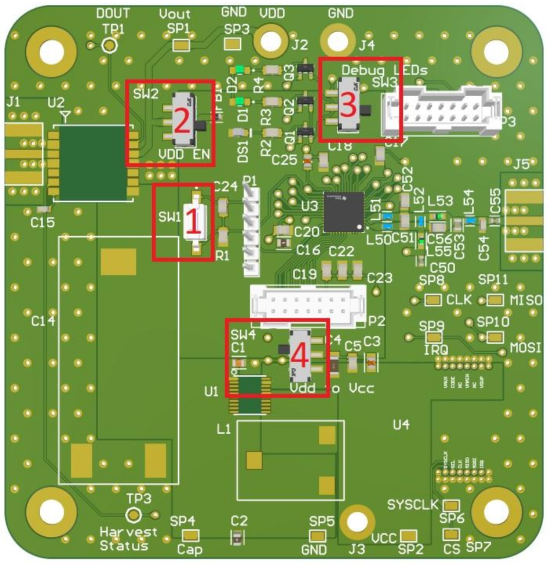

Figure 10: PCB with configurable parts

Switch 1 is no more than a push-button which resets the MCU. When pressed, all running code in the MCU is terminated, the MCU reboots, and code execution begins from the beginning.

Switch 2 is a VDD enable switch that connects or disconnects the harvester's Vout to the rest of the components. Allowing the MCU to be disconnected if so desired.

Switch 3 is used to choose whether the Debug LED's on board should be used or not. This is because LEDs use a lot of power and these pins are also wired to the connector for use by the user.

Switch 4 allows the transceiver to be powered by the harvesters boost converter through the Vdd copper pour instead of powering off the secondary boost converter. This is because the boost converter is out of stock.

4.1.3 Software Overview & Configuration

To aid in research activities, the software which accompanies the hardware contains not only an existing program which implements base project functionality, but also exposes a large set of methods which allow for custom communication between the MCU and the low-power transceiver. The existing program is located in the example_program.c file. By default, this template will configure a device as a slave in the IoT network. In this state, the MCU will sleep until a wake-up signal is received, at which point it will wake up, retransmit the wake-up signal to neighboring slave devices, and call the usercode() method. This method is left blank and is where user code is intended to be added. Once execution of the usercode() method has been completed, the MCU will return to sleep until the next wake-up signal.

The custom communication methods provided for research purposes are located in the trns.c file. Several key methods are discussed below, but additional detailed documentation is provided through in-code documentation.

- uint8_t trns_testSPICommunication()
  Communication between the MCU and transceiver is established using SPI. To confirm if the MCU and transceiver are able to communicate with each other over SPI, this method may be used. The method writes and then reads a sequence of bytes to the transceiver's SCRATCH register. If the written data matches the received data, SPI communication is assumed to be good, and the method returns 1. If communication fails, the method returns 0.

- uint8_t trns_testInterrupt()
  The transceiver is able to inform the MCU of a received message through an interrupt. To test that the MCU and transceiver's interrupt capabilities are working correctly with each other, this method may be used. The method instructs the MCU to wait until it receives an interrupt, and then tells the transceiver to set its interrupt in high. If successful, the method will return a 1. If the interrupt is not received or sent, the method will continue looping.

- void tnrs_setPwrMode(uint8_t mode)

To aid in changing the power mode of the transceiver, this method is provided. The method expects a valid power mode (refer to the transceiver programming manual for valid power modes) as input and instructs the transceiver to enter the power mode. Some power modes cannot be switched to immediately, and so the transceiver programming manual recommends reading transceiver registers to see when the power mode change has been completed. This method does not implement these checks.

- trns_setupTx()
  A large number of registers must be set to properly configure the transceiver for transmission. This method writes to the target registers and provides a default transmission configuration. The list of registers to be written to should always stay the same. You can change some configuration settings (carrier frequency, modulation scheme, frequency deviation, bit rate etc) by obtaining those values from AXRadioLab. See below for information on AXRadiolab.

- trns_setupRx()
  A large number of registers must be set to properly configure the transceiver for receiving. This method writes to the target registers and provides a default receiving configuration.The list of registers to be written to should always stay the same. You can change some configuration settings (carrier frequency, modulation scheme, frequency deviation, bit rate, max offsets etc) by obtaining those values from AXRadioLab. See below for information on AXRadiolab.

- trns_transmit(uint8_t *pkt, uint16_t pktlen)

  This method allows for a custom packet to be transmitted wirelessly from the transceiver to a neighboring device. The method expects an address to an unsigned 8-bit integer array to be passed in, along with the length of the packet. The packet is placed directly into the transceiver FIFO, so it should contain any chunk header or command information needed by the transceiver. Refer to the transceiver’s programming manual. The method does not automatically configure the transceiver for transmit mode or change its power mode; this must be done before the transmit method is called.

- trns_receive_isr(uint8_t *ax5043_rxbuffer,uint8_t *ax5043_pktlen,int32_t

  *ax5043_offset,int16_t *ax5043_rssi, int16_t *ax5043_offset_hz)

  This method should be called once the AX5043 throws an interrupt indicating a packet was received. The method expects an unsigned 8-bit integer array, ax5043_rxbuffer, that is at least the expected packet size length (to mitigate this the array can be set to have a size of 256 which is the max packet the AX5043 can receive). The method will also return the packet length, offset, RSSI, and frequency offset and save those values using pointers passed in as arguments. The rxbuffer will contain any chunk header or command information from the packet and the packet length will reflect this as well. The method keeps the device in the same mode it was while receiving the packet.

Due to the complexity of the transceiver and the number of registers that need to be configured for proper use, along with the lack of documentation of some registers, it is not recommended to calculate all register values by hand when changing transmit and receive configurations. ON Semiconductor provides software, called "AXRadioLab", which provides a GUI and is capable of generating register values given some desired Tx/Rx configuration. ON Semiconductor also recommends in their documentation using this tool for determining register values. The tool can be found here. Use of this tool is not necessary for the example project as all register values have already been determined.

4.1.4 Example Code Instructions

The below instructions assume that two copies of this project are being used, one for the slave device and one for the master. If desired, a different transmitter can be used to generate the wake-up signal that the slave node expects.

For hardware, connect a CC1352 MCU to an AX5043 transceiver evaluation board. This may be done either using the provided hardware or by breadboard. You may power the system off either the RF harvester or a constant voltage supply, although operating off the RF harvester may be inconvenient for first-time use. Do the same with another CC1352 MCU and AX5043, although ensure that this pair is powered off a constant voltage supply. The first pair will serve as a slave node in our IoT network, and the second will serve as a master node. You will need to connect an XDS110 probe to interface Code Composer Studio (CCS) running on a computer with the MCU to program it. For software, you will need the provided trns.c, trns.h, spi.c, spi.h, trns_registers.h, and example_program.c files. These should already be located in the provided project. You will also need the TI-RTOS project in the workspace, which can be downloaded from the Resources Explorer in CCS.

Once all files are installed, connect the computer to the slave MCU through JTAG and open the example_program.c file. The file contains several existing methods which can be left as-is. In the masterThread method, ensure the MCU will be configured as a slave node by ensuring the trnsSlave() method is being called, not the trnsMaster() method. The file will also contain a usercode() method which is blank. Add any code that you want executed when a wake-up signal is received on the slave node; for example, if operating off a constant power supply instead of the RF harvester, you may wish to toggle the debug LEDs on and then off. If operating off the RF harvester, note that the code being executed should not leave the MCU on long enough to drain the harvester capacitor or present a large load.

Flash the code to the MCU. The slave node will initially remain inactive because no wake-up signal is being transmitted. Repeat the programming process with the master MCU. Ensure the masterThread() method is calling trnsMaster() instead of trnsSlave(). Do not worry about the usercode() method since this method is not used by the master node, only slave nodes. Do not worry about packet contents, length, or the preamble, as this is handled already.

Once flashed, the master node will begin transmitting the default wake-up signal to all surrounding slave nodes. If using debug LEDs in the usercode() method, you will be able to see the slave node LEDs toggle on and off as the device receives wake-up signals. You may also debug the master node to observe console outputs indicating when a packet has been sent by the master and received by a slave.

4.1.2 Additional Resources

Additional resources are not necessary to use the example project, which is a demonstration of the base product requested of the senior design team. However, the end goal of this project is to serve as a research platform for wireless IoT communication, and so it is anticipated that additional work which builds upon the provided project may be performed. In this case, the following resources (focused on the CC1352 MCU and AX5043 transceiver) may be beneficial.

1. CC1352 Datasheet
2. CC1352 Technical Reference Manual
3. AX5043 Datasheet
4. AX5043 Programming Manual
5. AX5043 Evaluation Board Schematic
6. Online Reference Source for AX5043

4.2 Alternative versions

Each iteration of the design process had an issue that significantly changed the overall design. Starting from the first design, we assumed there was a way to bypass the harvester module's hysteresis loop to force it on during its off cycle. While it looks like a good plan in theory, after looking into the datasheet and trying to get in touch with the manufacturer, we concluded that it was not possible to have access to the on board DC-DC converter to trigger it on. Instead of making our own harvester in which we could control the DC-DC converter powering the rest of the devices, we opted to stay with the third-party harvester since our knowledge is limited and implementing a harvester would significantly add complexity to the project that was unnecessary.

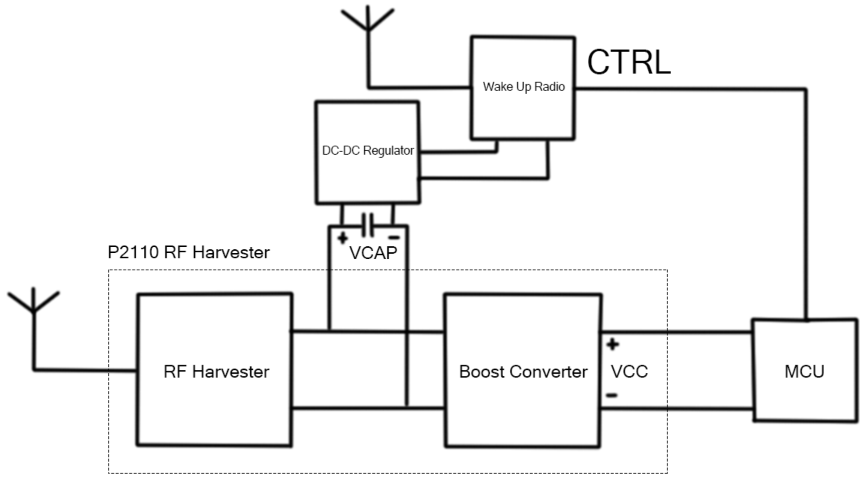

Figure 11: Proposed Design

For the second design idea, its simplicity is what was most attractive. The biggest issue with this design is that the MCU would not be off when the trigger is received. Specifically, since the onboard harvester's hysteresis loop is not something we have access to, the trigger won't change the operation if the MCU is active. On the other hand, if the MCU is off, it is because the Powercast DC-DC converter is off, meaning that it is in the charging state. If the trigger connected the MCU to the Powercast output, it would still not turn on. Thus this design was redundant and didn't meet the goals and requirements of the final device.

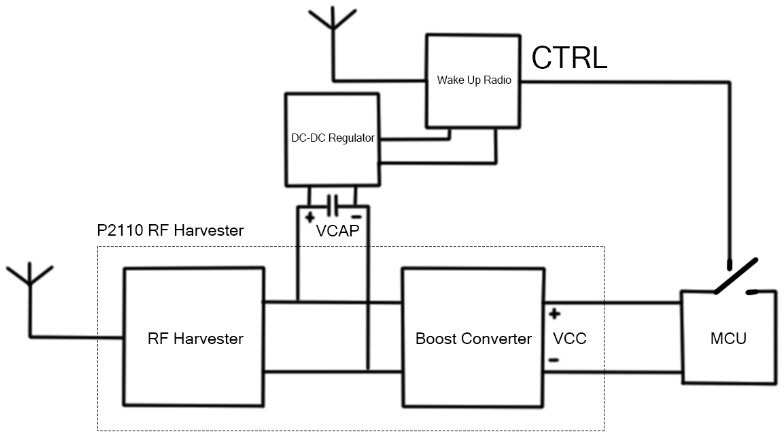

Figure 12: Design Option 2 - Rejected

Our accepted design is depicted in the figure below. The reason we could not end up implementing this design fully is mentioned in section 1.4 on design revisions. Due to the silicon shortage, we were not able to procure the DC-DC regulator or a CC1352R MCU surface mount component to use on our PCB. Since we were not able to test the DC-DC regulator, we opted to change our design so it is not required. Luckily in terms of the MCU not being available, we were able to use CC1352R MCUs on a Launchpad and also in its SensorTag package and connect to the other components through jumpers and breadboards.

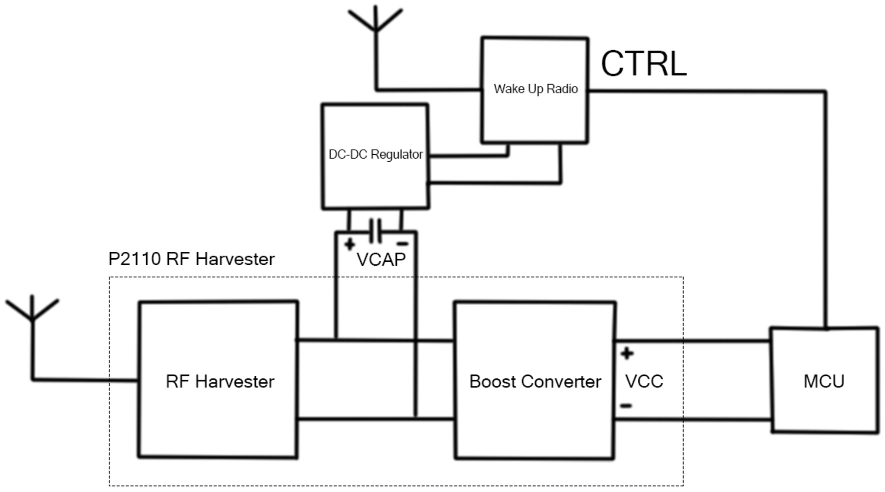

Figure 13: Intended Design

4.3 Considerations

During the testing phase where we were trying to get the AX5043 transceivers to communicate, we had multiple failures. Our different testing methods are listed below.

Setup:

We had a CC1352 MCU and AX5043 transceiver wired together so that the MCU could communicate with the transceiver through SPI. The MCU set register values to configure the transceiver.

For all tests, we started by confirming that we could actually program the transceiver with SPI (achieved by writing and reading from a test register) and also tested the interrupt function by telling the transceiver to trip its interrupt pin and waiting on the MCU. This allowed us to confirm that SPI and interrupts were not causing us to fail tests. Furthermore, when we wrote to the transceiver over SPI, we read the register written to immediately after to confirm a miscommunication did not occur.

What we had tried:

- Configuring both AX5043 transceivers for FSK modulation with 9600 baud rate and frequency deviation of 3kHz. One was set to keep transmitting and the other was set to throw an interrupt when it receives any data (the internal FIFO is not empty). Our transmitter sends a preamble of 0x55 for 20 bytes, followed by a data set. We had CRC on for this set of tests.
    1) We initially tried sending a 6 byte message

        2) We also tried filling up the transmitter’s data FIFO until it was full. We checked a flag to confirm if the FIFO was full, committed the data, and then instructed the transmitter to send the data. Once transmission was complete, we checked the flag register again to confirm that the FIFO was completely empty.
            a) Seeing the FIFO go from full to empty is one reason we believe we are able to transmit data, and that our problems are on the receiver.
        3) Reading the FIFO on the receiver yielded zeros, and reading the number of bytes received also yielded zero.

- Configuring both AX5043 transceivers for FSK modulation with 9600 baud rate and frequency deviation of 9kHz. We had CRC off for this set of tests, and the receiver was just saving the raw received bits.
    - We increased the frequency deviation because we thought 3kHz was too constrained.

- Configuring a CC1352 MCU LaunchPad to act as an FSK receiver with a 9600 baud rate and frequency deviation of 9kHz. We confirmed the LaunchPad’s receiving functionality by configuring another LaunchPad as a transmitter and testing communication. Then, we used the receiver to try to receive data from an AX5043 configured as a transmitter.
    - The goal here was to prove that we actually were transmitting data. However, the API used by the CC1352 MCU includes additional features (such as sync word) that we are not sure we correctly implemented. So, this test has not yielded meaningful results. The problem could either be our AX5043 isn’t transmitting or our Tx/Rx have different configurations.

- We changed the setup so that both are in receive mode and are able to transmit by pressing a button. This was mostly a convenience factor, but also allowed us to confirm that our process of going from one state (Tx/Rx) to the other (Rx/Tx) is correct.

- With the help of Dr. Neihart and his Spectrum Analyzer, we tested whether we could actually transmit. The result was a notable signal in the 200MHz range with a large BW that would disappear when our device was powered down. After some time we confirmed this was a result of the external oscillator on board since cutting power to that while powering everything else made the signal disappear. This confirmed we were not, in fact, transmitting.

Components:
2 x CC1352R MCUs
2 x AX5043 Transceivers set for 433MHz RF communication

4.4 Data

Distance (in)	R(Ω)	Ton (ms)	Toff (ms)
5	5460	on	N/A
	4650	on	N/A
	1470	on	N/A
	670	570	380
	47	on	N/A
10	5460	9300	680
	4650	6960	780
	1470	1000	640
	670	380	640
	47	10.6	N/A
15	5460	2220	1040
	4650	1880	1100
	1470	680	980
	670	320	960
	47	9	N/A
20	5460	2040	1520
	4650	1780	1520
	1470	700	1460
	670	340	1400
25	5460	2080	2360
	4650	1860	2140
	1470	700	2060
	670	340	2100
30	5460	2160	7280

	4650	1920	5220
	1470	780	4680
	670	340	4360
35	5460	2120	11900
	4650	1920	7140
	1470	780	8060
	670	340	9460
40	5460	2120	10900
	4650	1880	11680
	1470	720	11440
	670	324	12940

Figure 14: Powercast Harvester Test Results (complete)

Pout (dBm)	Ping per 5 second	Avg. Ping per 1 sec	Distance (cm)
0.1252441406	52	10	8
0.1252441406	51	10	10
0.1252441406	48	10	12
0.1252441406	33	7	12.5
0.1252441406	20	4	13
0.1252441406	15	3	13.5
0.1252441406	0	0	14
0.2502441406	52	10	8
0.2502441406	52	10	10
0.2502441406	51	10	12
0.2502441406	51	10	13
0.2502441406	51	10	14
0.2502441406	51	10	14.5
0.2502441406	47	9	15
0.2502441406	45	9	15.5

0.2502441406	43	9	16
0.2502441406	33	7	16.5
0.2502441406	13	3	17
0.2502441406	10	2	17.5
0.2502441406	3	1	18
0.2502441406	0	0	18.5
0.5002441406	51	10	8
0.5002441406	51	10	10
0.5002441406	51	10	12
0.5002441406	51	10	13
0.5002441406	51	10	14
0.5002441406	51	10	16
0.5002441406	51	10	18
0.5002441406	51	10	20
0.5002441406	50	10	22
0.5002441406	49	10	24
0.5002441406	31	6	25
0.5002441406	30	6	25.5
0.5002441406	12	2	26
0.5002441406	0	0	26.5
0.5002441406	0	0	27
0.5002441406	49		30
0.5002441406			32

Figure 15: AX5043 example Agent-Controller packet results

Master TXing		
TXPWRCOEFFB0	Lower bound on Maximum Peak Current Consumption that could last for average of 15.4ms	Upper bound on Maximum Peak Current Consumption that could last for average of 15.4ms
0x20	0.01445205047	0.01308942857
0x40	0.01498138801	0.01356885714
0x80	0.01635772871	0.01481542857
Slave on continuous Rx and Tx (using example program)		
TXPWRCOEFFB0	Average current consumption of example program w/o user code	Average current consumption of example program w/o user code
0x20	0.008413249211	0.00762
0x40	0.00851829653	0.007715142857
0x80	0.008694952681	0.007875142857

Figure 16: AX5043 example Agent-Controller power results

4.5 Hardware Schematics

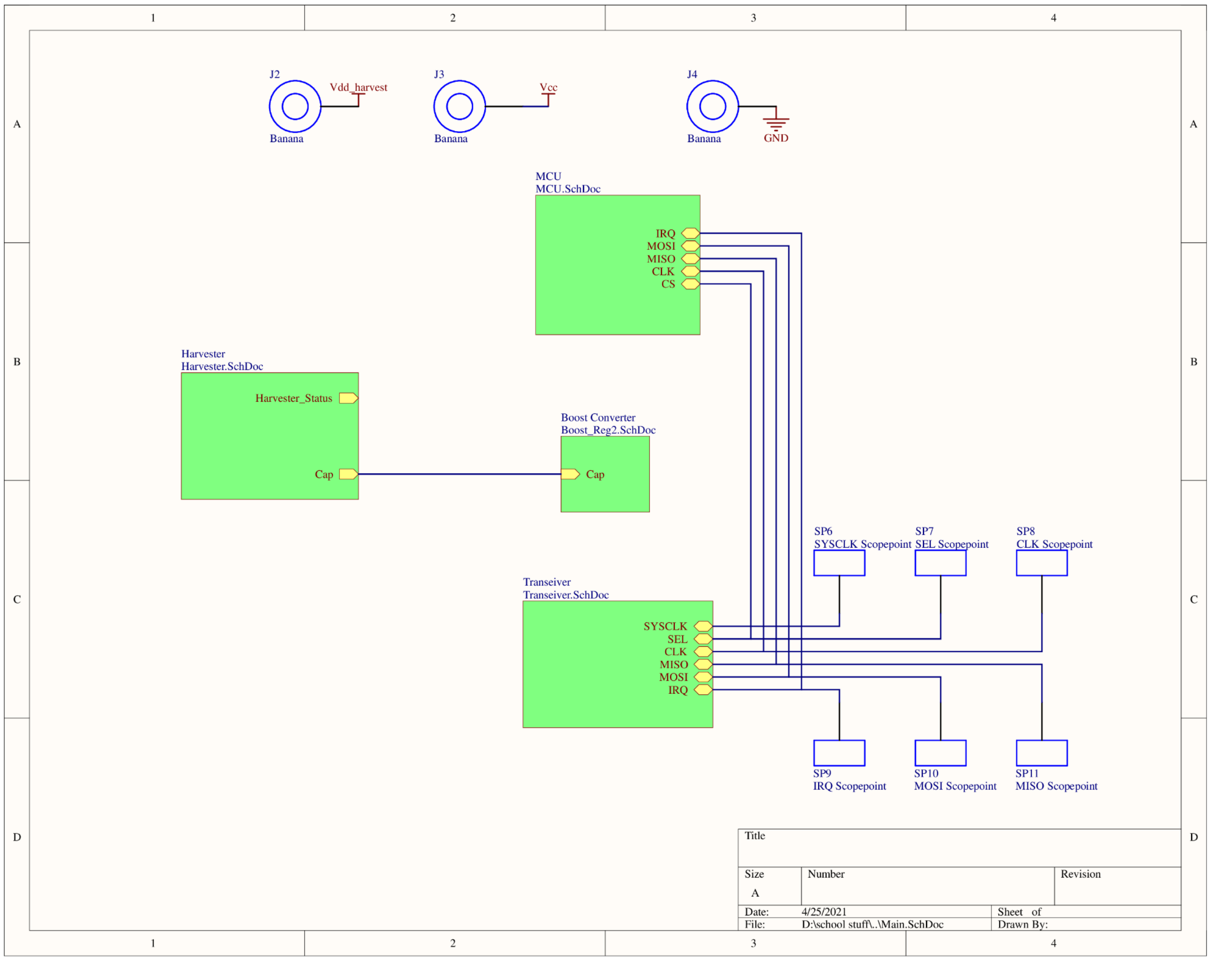

Figure 17: Main Schematic

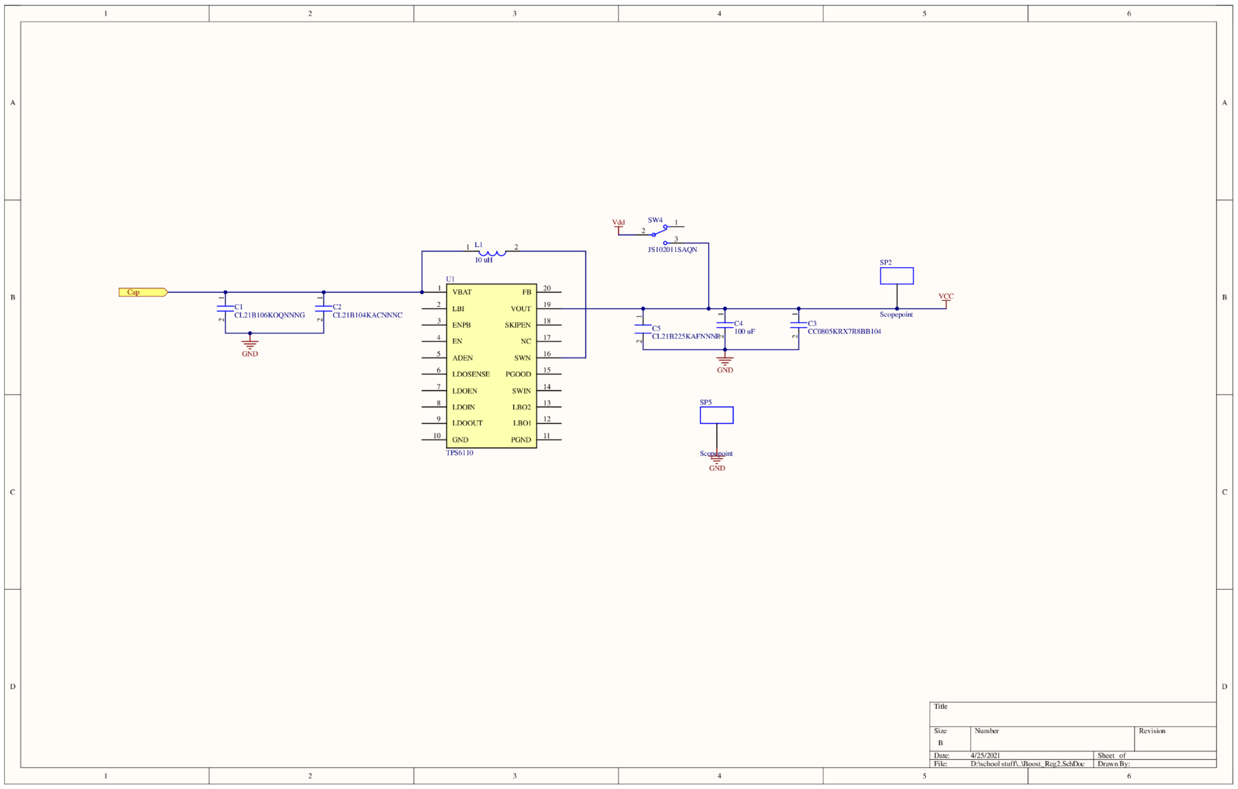

Figure 18: Boost Regulator Schematic

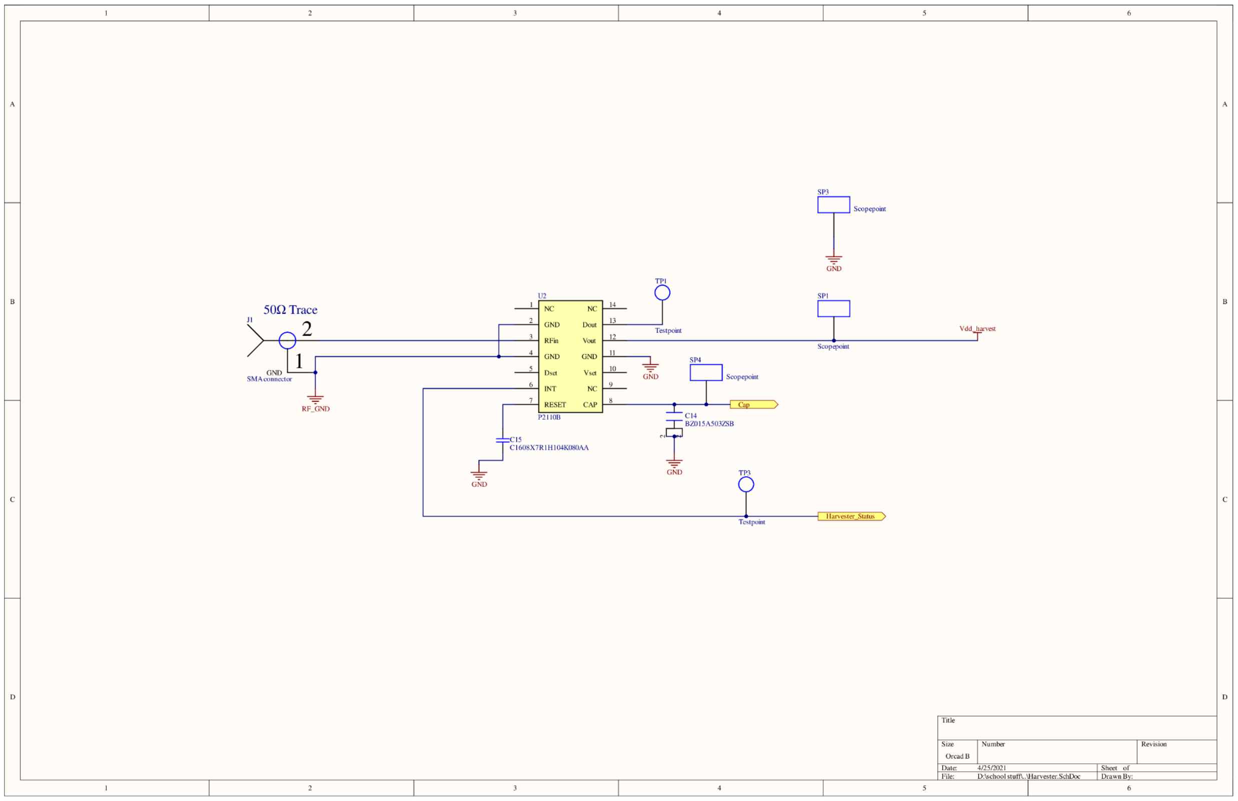

Figure 19: Harvester Schematic

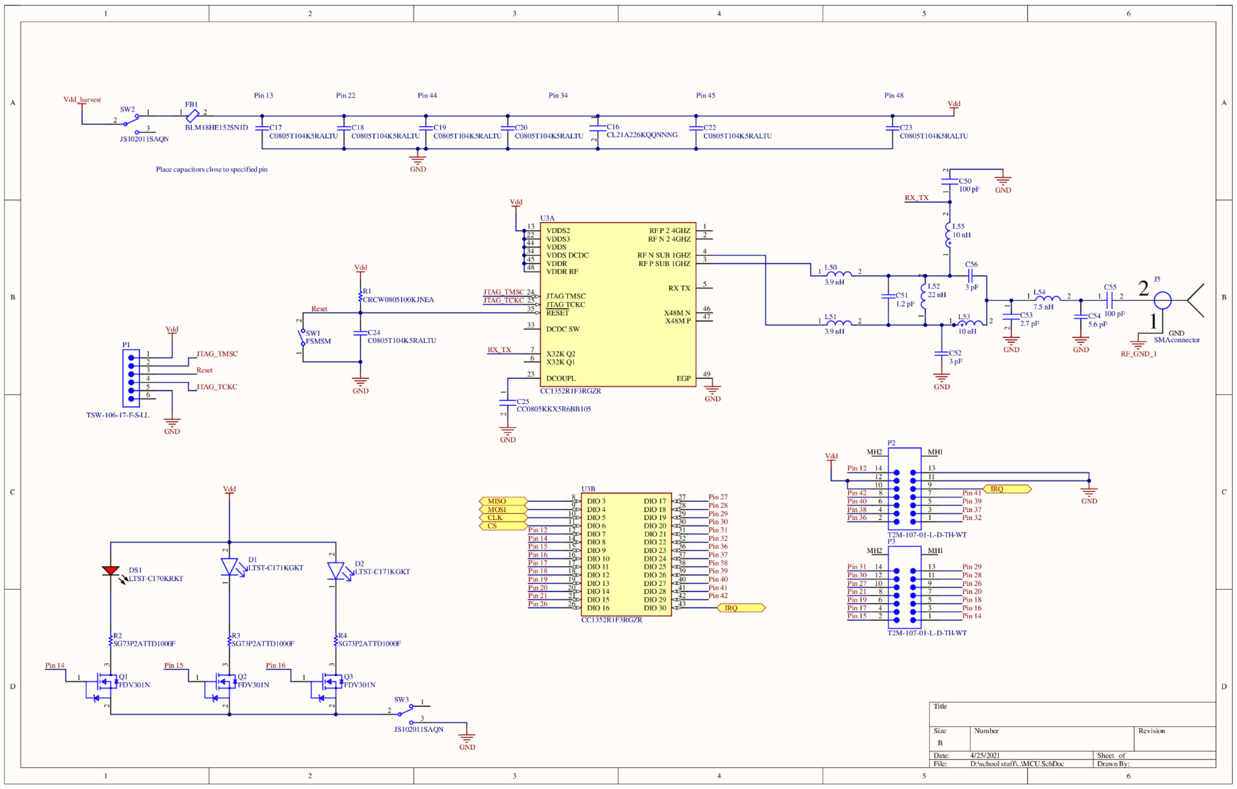

Figure 20: MCU Schematic

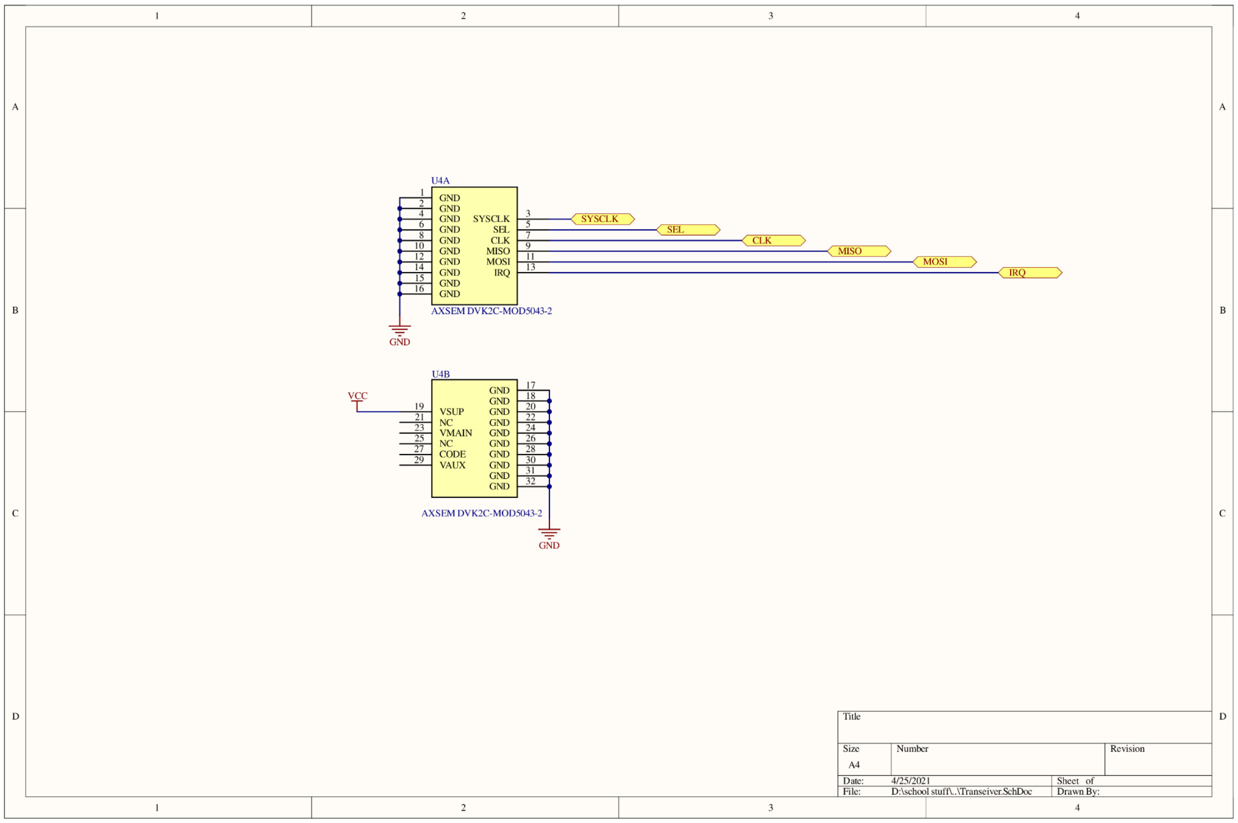

Figure 21: Transceiver Schematic

5 References

[1] “CC1352R,” Texas Instruments, May-2020. [Online]. Available: https://www.ti.com/product/CC1352R. [Accessed: 23-Oct-2020].

[2] L. Wilhelmsson and D. Sundman, “Wake-Up Radio – A key component of IoT?,” Ericsson.com, 18-Dec-2017. [Online]. Available: https://www.ericsson.com/en/blog/2017/12/wake-up-radio--a-key-component-of-iot. [Accessed: 23-Oct-2020].

[3] “Low Input Voltage Boost Converter,” Texas Instruments, Dec-2014. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps61202.pdf?HQS=TI-null-null-digikeymode-df-pf-null-wwe&amp;ts=1600912252796. [Accessed: 23-Oct-2020].

[4] “P2110B Datasheet,” Powercast, 2016. [Online]. Available: https://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf. [Accessed: 23-Oct-2020].

[5] “AX5043 Programming Manual,” ON Semiconductor, Feb-2017. [Online]. Available: https://bit.ly/3dO39A3 [Accessed: 24-Apr-2021].