IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Wake-Up Harvester Design for Batteryless IoT System

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

Team Leader Email:
esduan@iastate.edu

Advisor & Client:
Prof. Henry Duwe

Team sdmay21-14:
Edmund Duan, Jacob Bernardi, Douglas Zuercher
Kwanghum (Ted) Park, Bryce Staver, Zacharias (Zack) Komodromos

Project Need and Goal

Motivation for project:

- It is useful for IoT devices to synchronize between themselves for communication of data, distribution of sensed values, and low-cost decision making.
- Batteryless systems are intermittent (not always powered, dependant on harvestable energy)
- Makes it unlikely that two IoT devices are regularly on at the same time
- Conserve power to increase the likelihood two nodes can be active at the same time.

Requirements

Functional Design Requirements

- Wake up only on valid triggers
- Repeat trigger once received
- Enter normal operation once triggered
- Maximum off time no greater than 1 minute

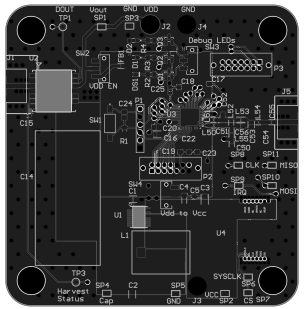
Non-Functional Design Requirements

- Small package size for device (4” x 4”)
- Batteryless device
- Triggerable within 1 meters

Resource Requirements:

- Limitations of the environment
  - Nodes may be in an environment with RF noise which can interfere with trigger signal
  - 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

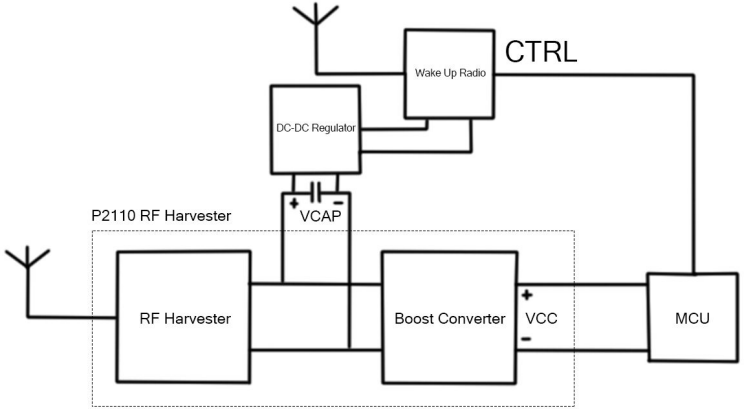


Risk Identification & Mitigation

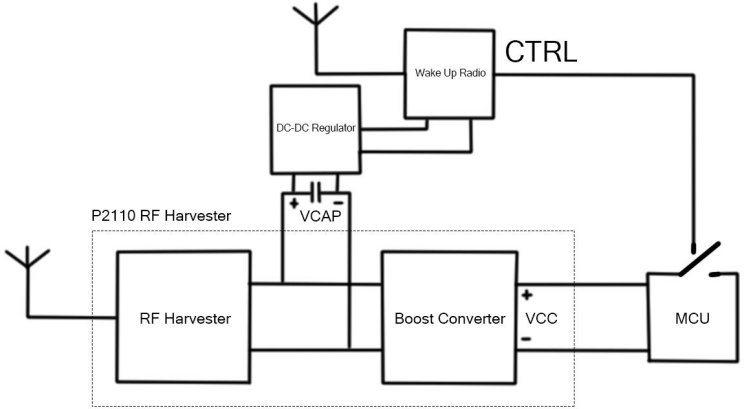
- Not connected to the internet or any other sort of local network, cybersecurity concerns unlikely
- If the trigger preamble is correctly guessed, anyone can trigger the device
- To restrict access, a unique preamble could be created, only known to team

Design Thought Process

- Planned to use third-party harvester, use lower capacitor values
- Access to DC-DC converter to trigger harvester was not available
- Trigger won't change operation if MCU is active, switch doesn't work as intended
- No DC-DC regulator due to the silicon shortage



First Design

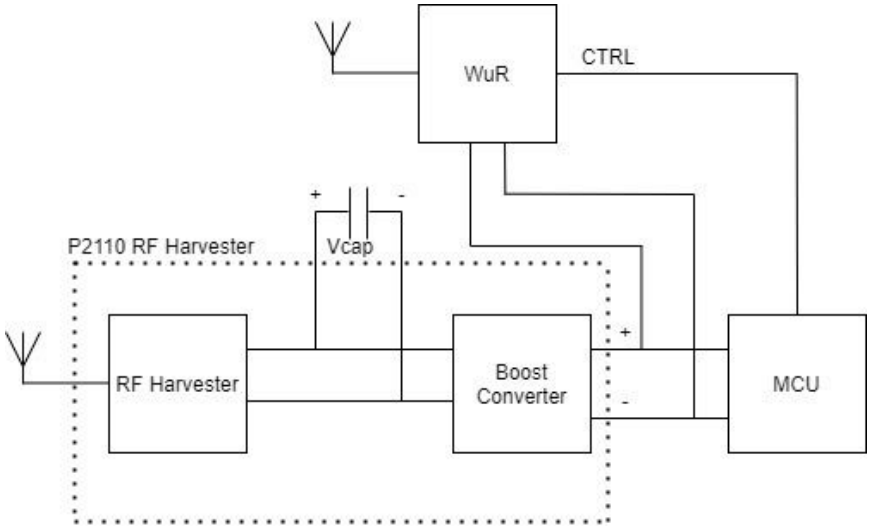


Second Design

System Analysis

Functions of system

- Supply system with power from RF harvester
- When wake up signal is received, send wake up signal to next node
- Meant to run MCU code between 1.02-1.25 V
- System should consume as little power as possible

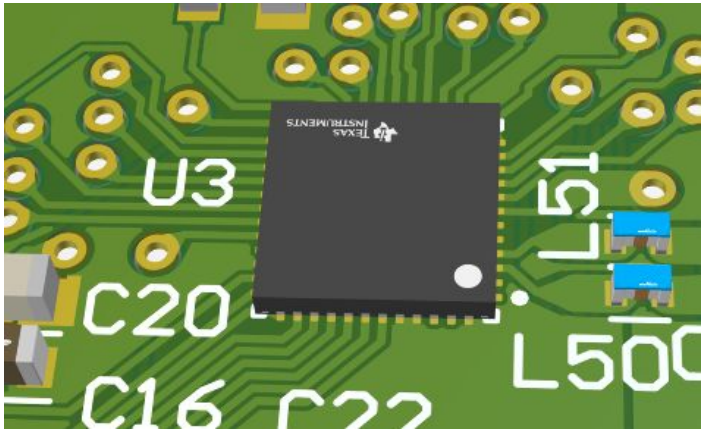


Hardware Implementation (Main Components)

RF Harvester
- P2110
- Harvests power efficiently in 915 MHz ISM band
- Given by client

MCU
- TI CC1352
- Target of project, meant to turn on at specific parameters
- Recommended by client
- Primarily used in IoT applications

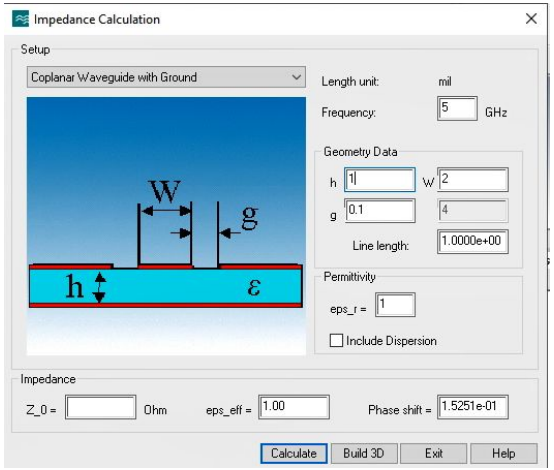
Wake Up Radio/Transceiver
- AX-5301 evaluation board
- Found by team
- Sub-GHz, very low power transceiver



DC-DC Regulator
- TPS61100
- Found by team
- High efficiency to supplement harvester converter at low power consumption

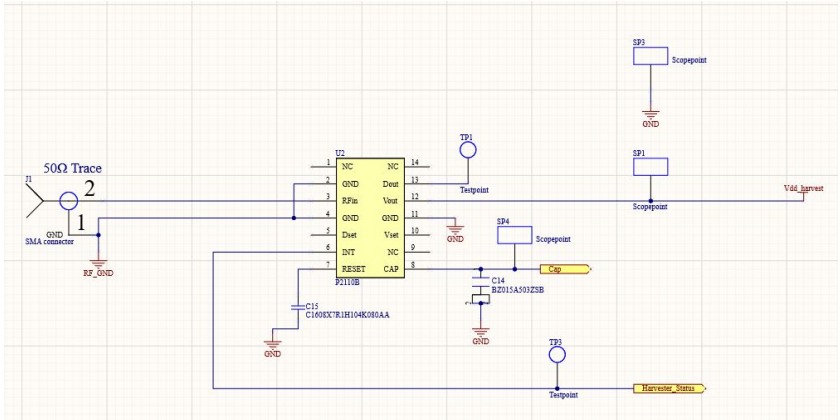
Hardware Implementation (Tools)

- Learned how to use Altium Designer for the schematic and PCB
  - Organizing schematics in a readable fashion
  - Component placement on layout
  - Routing of PCB
- CST Studio Suite
  - Used the macros in CST to construct grounded coplanar waveguide
- Microsoft Excel
  - Organized BOM (Bill of Materials) to order parts for PCB
  - Graphed and organized data from testing



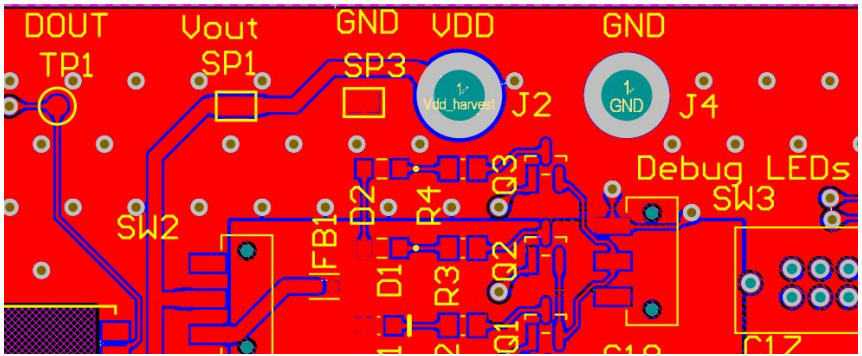
Hardware Implementation (Design Process)

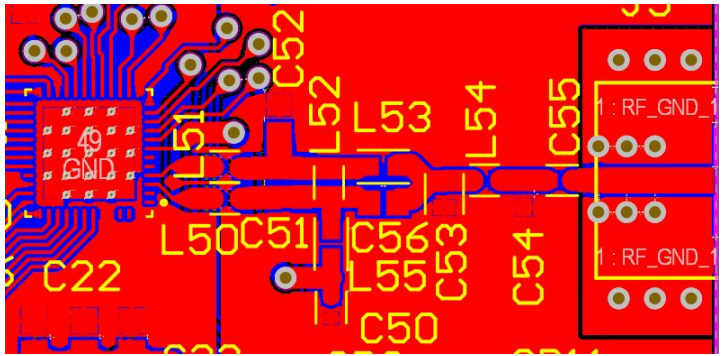
- Parts Selection
  - Searching Digi-Key for available parts
  - Finding parts with footprint already in Altium library
- Schematic Organization
  - Schematic pages for major components
- Footprint and symbol creation
  - Made custom footprints for components that were not already in library
  - Used datasheets to find physical dimensions



Hardware Implementation (Design Process)

- PCB Organization
  - Logical placement
  - Routing of signals
  - Priority of signals
- RF Routing
  - PCB materials and characteristics
  - Implementing waveguides





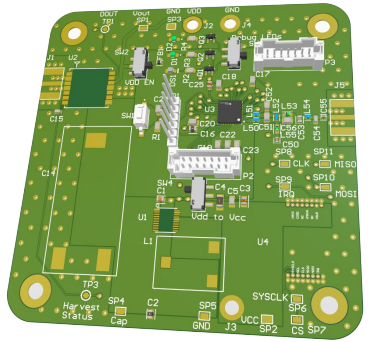
Hardware Implementation (Considerations)

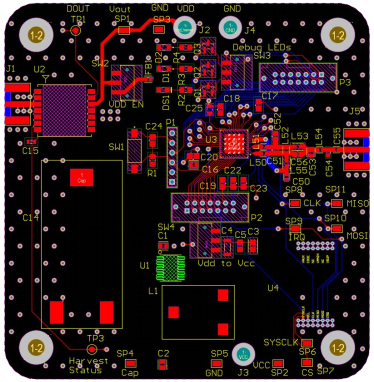
Manufacturing and Assembly
- Rules set up with capabilities of board house
- Components sized to be hand soldered when possible

RF Portions
- Traces kept short and straight with linear tapers
- Grounded coplanar waveguide
- Direct connections over of thermal reliefs for lower inductance connections

Power
- Wider traces and copper pours to lower resistance
- Pour around MCU for power connection

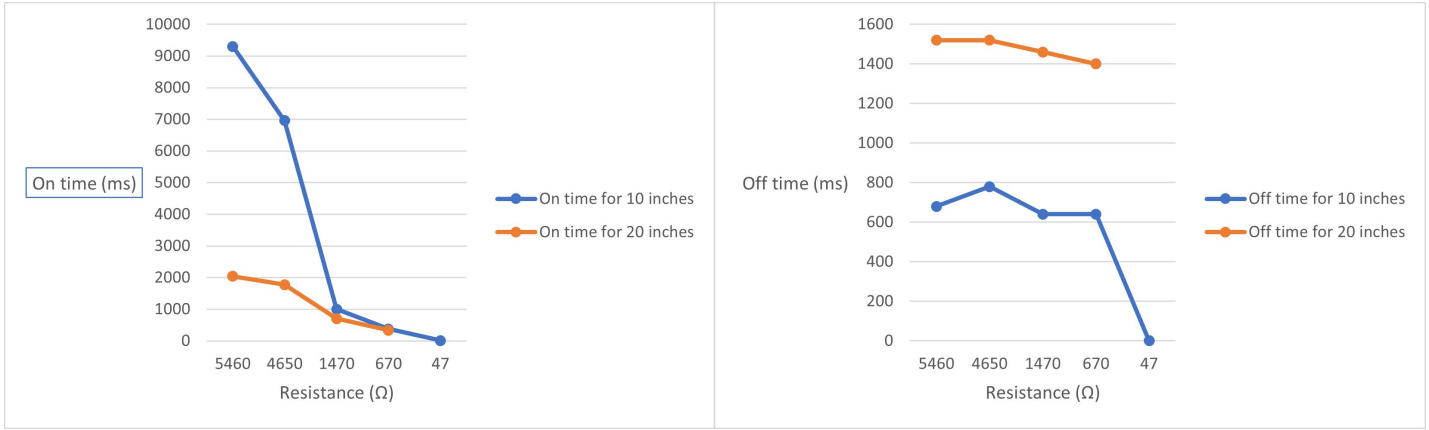
Ease of Use
- Board flows from left to right, or input to output
- Banana connections for testing with power supply
- Oscilloscope probe hooks for testing
- Test points for voltage measurements
- Three debug LEDs for the MCU
- Slide switches for control of power and LEDs





Hardware Implementation (Component Testing)

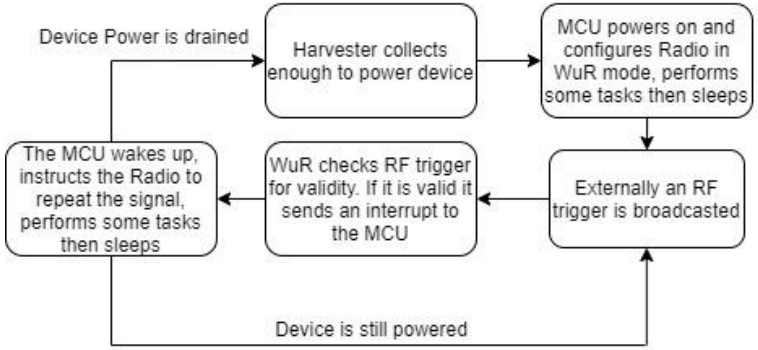
- Done by measuring on and off times with oscilloscopes with varying loads and distances from transmitter
- Results were expected
- Learned how to use the harvester in a lab setting



Software Implementation (MCU)

The process of tackling software was based on our device's state diagram. The first step when powered on is configuring the AX5043. We opted to use TI-RTOS for built in functionalities like:

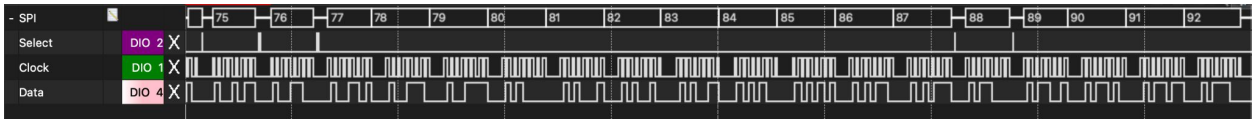
- SPI Handler
- Timer Handler
- Interrupt Handler/GPIO Functionality



Once TI-RTOS was imported, we set up custom functions to send the address and value over SPI to the AX5043 for read/write.

We learned about:

- Using TI-RTOS functions by reading through its documentation
- Basics of SPI and different standards (polarity, phase etc)
- Integrating 2 different companies' devices to interface



Software Implementation (Transceiver)

With the MCU capable of programing the AX5043, we could now work on completing the first part of the device cycle. For testing, we wrote functions to confirm basic functionality:

- testing SPI: Write to SCRATCH register and read from it to see if the value changed
- testing Interrupt: Instruct AX5043 to throw an interrupt and wait on the MCU to see if it triggers

Next we needed to lay the groundwork for the functions by writing custom write and read functions, and use them for common operations:

- trns_write(address,value)
- trns_read(address,value)

Software Implementation (Transceiver)

Using those base functions, we could create a library to be used to configure the AX5043:
tnrs_setPwrMode(uint8_t mode)

- Sets power mode using input (TX,RX,WuR,idle etc)

trns_setupTx()

- Writes to multiple registers configuring the AX5043's transmit details

trns_setupRx()

- Writes to multiple registers configuring the AX5043's transmit details

trns_transmit(uint8_t *pkt, uint16_t pktlen)

- Writes packet with predefined preamble to FIFO register, commits and sends

trns_receive_isr(..)

- Used when AX5043 interrupts for receive
- Reads FIFO register for received data

pll_ranging()

- Performs PLL ranging that should be run before every tx and rx as per programming manual

Software Implementation (Transceiver)

Each function contains multiple writes and reads to the AX5043 to set options including:

- Carrier Frequency (27MHz-1050MHz)
- Modulation type (FSK,AFSK,GMSK,PSK...)
- Framing options (preamble size, length byte position, CRC)
- Bitrate

And each of these decisions lead to more registers needing values like:

- Decimation
- Frequency Deviation
- Filter BW
- RX thresholds

The programming manual for the AX5043 documents all essential registers and provides information on how to populate depending on these decisions. A clear process on what all needs changing or in what order is not present in the manual though. Initial testing resulted in a failure to transmit or a failure to receive or both.

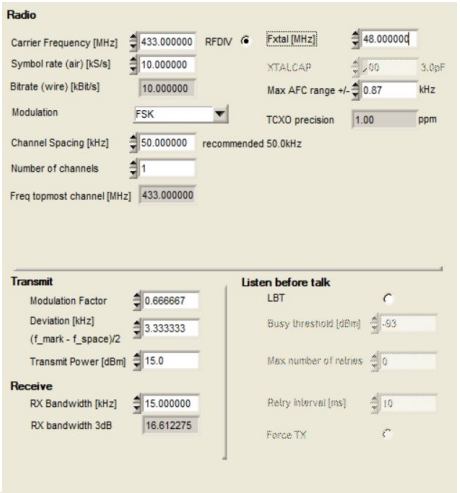
Software Implementation (Transceiver)

ON-Semi recommends AXRadioLab (source code generator for AX5043)

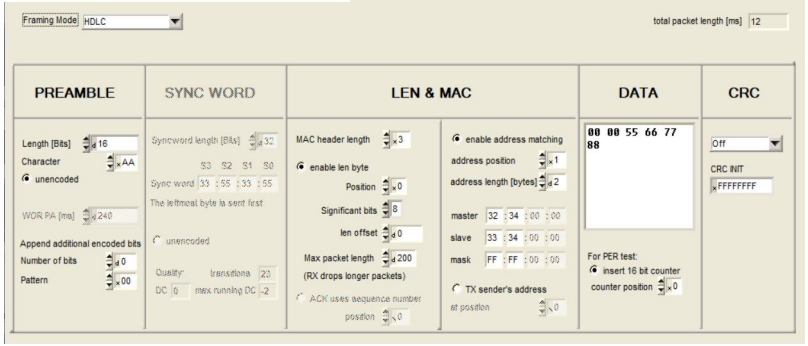
- Generates all necessary files for AX8052, ON-SEMI's MCU
- Provides Values for AX5043 registers based on user input configuration

Alternative is using the Programming manual provided calculations

- Not a clear list of registers to write to
- No way of knowing PERFTUNE registers since it says to only change a handful of values



After many trials and errors we managed to get a configuration to work. Small changes to the settings resulted in failure even with the values generated by AXRadioLab.



Software Implementation (Testing)

Once configured, need to confirm configuration works:

- Setup one device as “agent”
  - Waits to receive signal
  - Once signal is received, resend it
- Setup second device as “controller”
  - Send a signal
  - Wait until timeout for signal to be received

But, debugging both the Tx and Rx at the same time is hard. Some strategies we used:

- Monitoring power consumption
- Checking register outputs
- Configuring other MCUs as Tx/Rx devices
- Spectrum analyzer!

Software Implementation (Takeaways)

Learned:

- Difficult to debug communication since both Tx and Rx need to be working to confirm the other, Debugging a Tx/Rx pair without having an operating Tx/Rx to work with
- How the AX5043 handles data IO
- Some of the parameters which must be set for wireless communication (carrier freq, framing, encoding, power coeff, modulation etc)
- Using recommended resources as best as possible

Demo

Future Work

- Find methods and equipment to reliably test transceiver?
- Make PCB 4 layer and add transceiver on board
- Add shielding vias around RF traces
- Find another high efficiency DC-DC converter that is available
- Explore plausibility of using MCU to send/receive wake up signal

Thank You

- Questions?