# RTOS
## RF and Temperature driver implementation

Nick Tompkins
Matthew Davidson
Jacob Hochstetler

**CSCE5620 - RTOS Design**
Fall 2016
Instructor Dr. Song Fu

# Introduction

Our project goal to use KubOS, to implement temperature monitoring as well as an RF driver.  KubOS is software that accesses functions from the real-time operating system FreeRTOS (Kubos, 2016). FreeRTOS is stored in a container in Docker. By using KubOS to implement FreeRTOS, we are able to leverage the KubOS hardware abstraction layer (HAL) to enable wider deployment on different chips/boards.. This project would involve intimate knowledge of the build system within KubOS as well as the interface information for each of the chips we are writing drivers for.

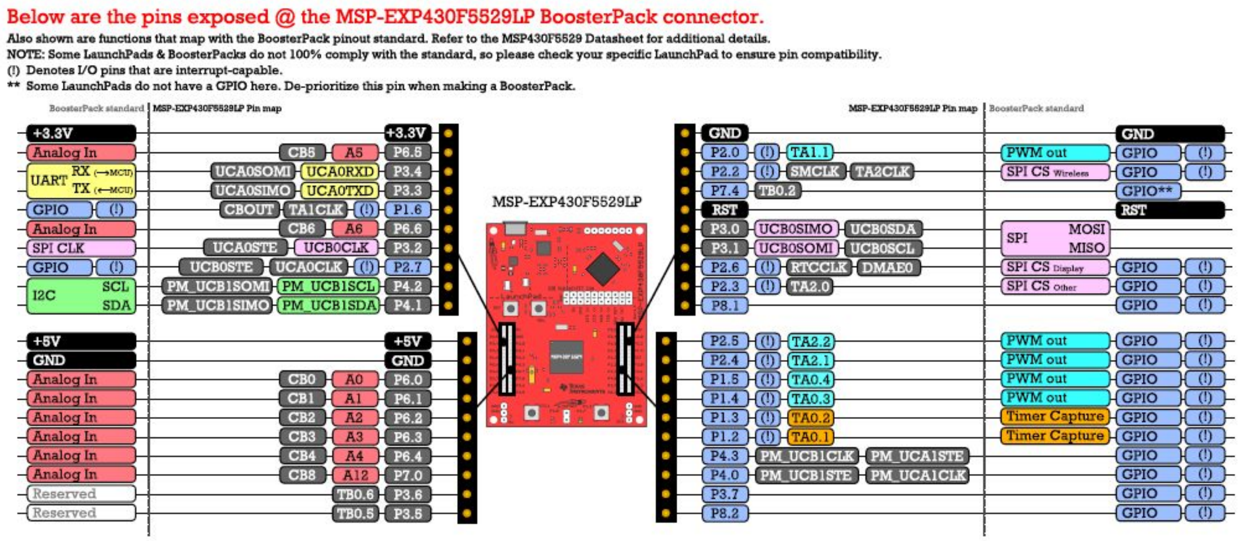
KubOS is middleware that is used to communicate with FreeRTOS. The purpose of KubOS is to make programming using FreeRTOS easier. KubOS is supported by Unix systems, such as Linux and Mac. Using a HAL, KubOS communicates with embedded systems and microcontrollers. The current microcontrollers supported by KubOS is the MSP430 and the STM32F4. KubOS's middleware SDK enables programmers to access functions from FreeRTOS and implement tasks on the OS (Kubos, 2016). The FreeRTOS SDK is stored in a container in Docker and  the container contains all the tools to compiles the libraries needed for FreeRTOS. KubOS wrote a command-line wrapper around both compiling code inside the Docker container, and flashing the embedded RTOS board connected to the USB port. The wrapper is implemented in Python so everything is cross-platform compatible.

The two boards used in this project were the MSP430 and the STM32F4. The reason why we are using this boards is because they are currently supported by KubOS. Both of the boards support I2C, SPI, and UART communication. One major difference between the boards is that the STM32F4 requires an external FTDI adapter to use its UART. The MSP430 immediately communicates once it is plugged in by USB.

The paper is laid out as follows: the next two sections cover both the embedded system boards we tested our code with. Then we explain the hardware for the drivers we wrote. After that, we examine some example code we wrote for both the drivers and tasks in KubOS. Next we discuss the Docker build environment, and finally the conclusion.

## MSP430

The MSP430 is a 16-Bit Ultra-Low-Power Microcontroller with 128KB Flash, 8KB RAM, USB Support, 12-bit ADC, and 2 SPI/I2C Interfaces (Texas Instruments, Inc., 2016a). This is a unique addition to KubOS due to the fact that it is a 16-bit processor and is not an ARM core.



Below are the pins exposed @ the MSP-EXP430F5529LP BoosterPack connector.

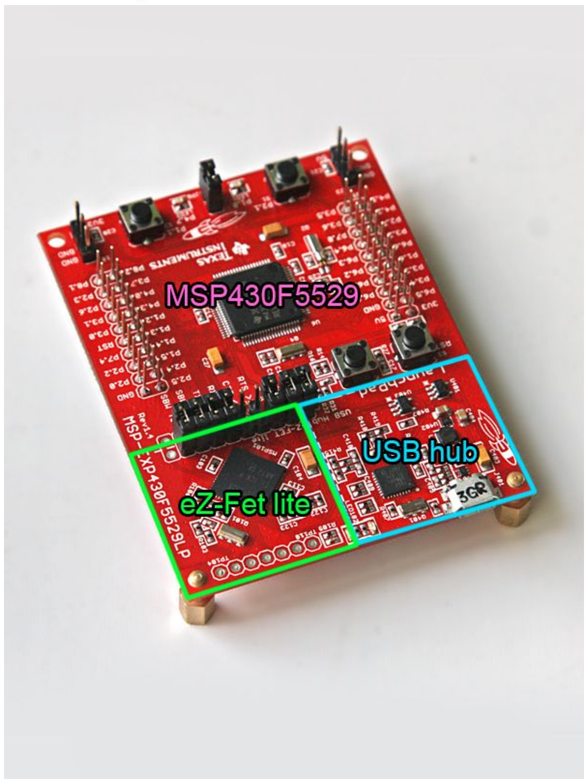
Also shown are functions that map with the BoosterPack pinout standard. Refer to the MSP430F5529 Datasheet for additional details.

NOTE: Some LaunchPads & BoosterPacks do not 100% comply with the standard, so please check your specific LaunchPad to ensure pin compatibility.

(!) Denotes I/O pins that are interrupt-capable.

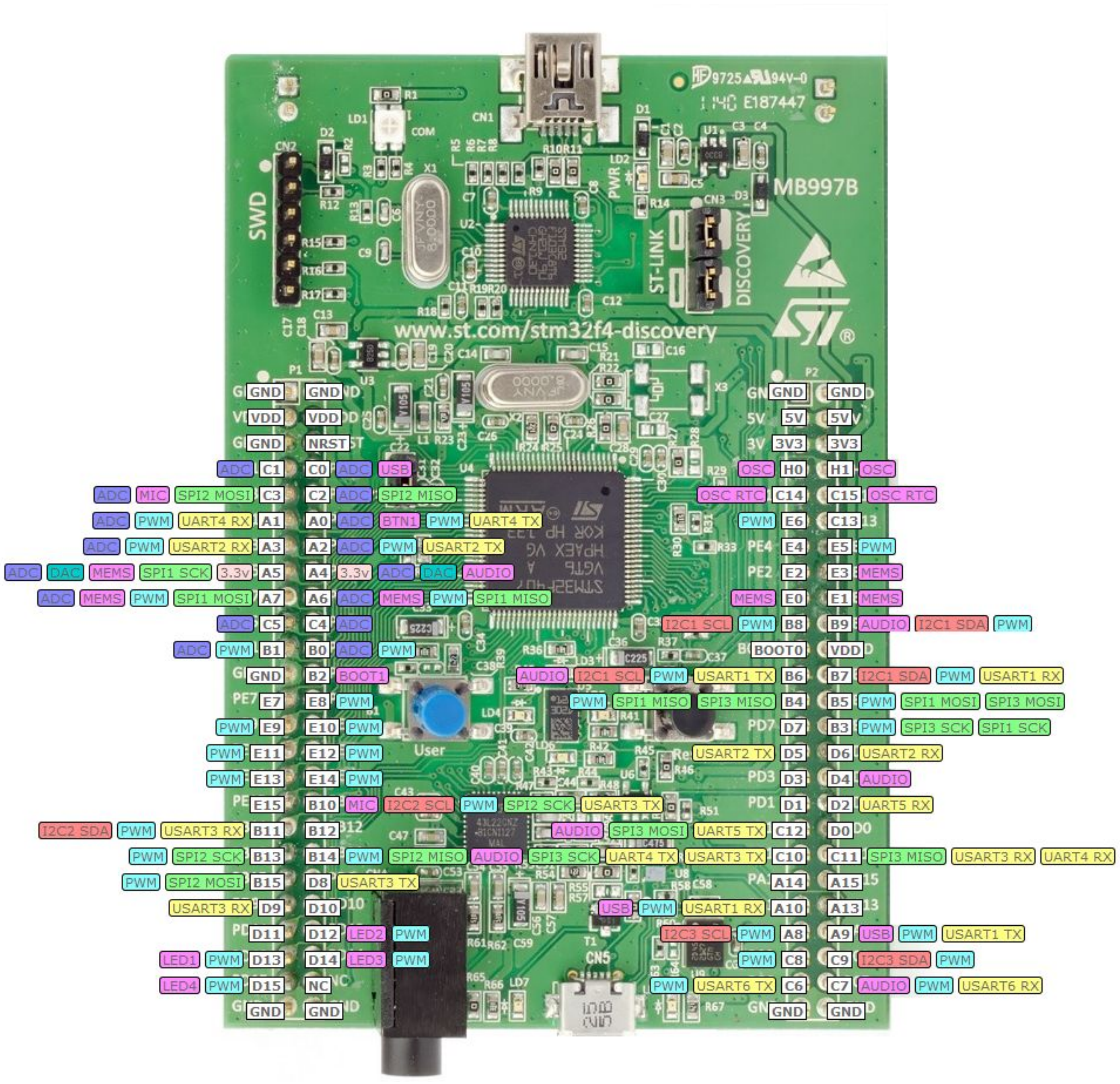
** Some LaunchPads do not have a GPIO here. De-prioritize this pin when making a BoosterPack.

BoosterPack standard
MSP-EXP430F5529LP Pin map
MSP-EXP430F5529LP
+3.3V | +3.3V
Analog In | CB5 | A5 | P6.5
UART RX (→MCU) | UCA0SOMI | UCA0RXD | P3.4
UART TX (←MCU) | UCA0SIMO | UCA0TXD | P3.3
GPIO (!) | CBOUT | TA1CLK | (!) | P1.6
Analog In | CB6 | A6 | P6.6
SPI CLK | UCA0STE | UCB0CLK | P3.2
GPIO (!) | UCB0STE | UCA0CLK | (!) | P2.7
I2C SCL | PM_UCB1SOMI | PM_UCB1SCL | P4.2
I2C SDA | PM_UCB1SIMO | PM_UCB1SDA | P4.1
+5V | +5V
GND | GND
Analog In | CB0 | A0 | P6.0
Analog In | CB1 | A1 | P6.1
Analog In | CB2 | A2 | P6.2
Analog In | CB3 | A3 | P6.3
Analog In | CB4 | A4 | P6.4
Analog In | CB8 | A12 | P7.0
Reserved | TB0.6 | P3.6
Reserved | TB0.5 | P3.5
GND | GND
P2.0 | (!) | TA1.1 | PWM out | GPIO | (!)
P2.2 | (!) | SMCLK | TA2CLK | SPI CS Wireless | GPIO | (!)
P7.4 | TB0.2 | GPIO**
RST | RST
P3.0 | UCB0SIMO | UCB0SDA | SPI MOSI
P3.1 | UCB0SOMI | UCB0SCL | SPI MISO
P2.6 | (!) | RTCCLK | DMAE0 | SPI CS Display | GPIO | (!)
P2.3 | (!) | TA2.0 | SPI CS Other | GPIO | (!)
P8.1 | GPIO | (!)
P2.5 | (!) | TA2.2 | PWM out | GPIO | (!)
P2.4 | (!) | TA2.1 | PWM out | GPIO | (!)
P1.5 | (!) | TA0.4 | PWM out | GPIO | (!)
P1.4 | (!) | TA0.3 | PWM out | GPIO | (!)
P1.3 | (!) | TA0.2 | Timer Capture | GPIO | (!)
P1.2 | (!) | TA0.1 | Timer Capture | GPIO | (!)
P4.3 | PM_UCB1CLK | PM_UCA1STE | GPIO | (!)
P4.0 | PM_UCB1STE | PM_UCA1CLK | GPIO | (!)
P3.7 | GPIO | (!)
P8.2 | GPIO | (!)


MSP430F5529
USB hub
eZ-Fet lite

In our testing the MSP430 work much quicker for debugging as we did not need to have to find another FTDI adapter to enable the UART to USB. The debugger has this support natively.

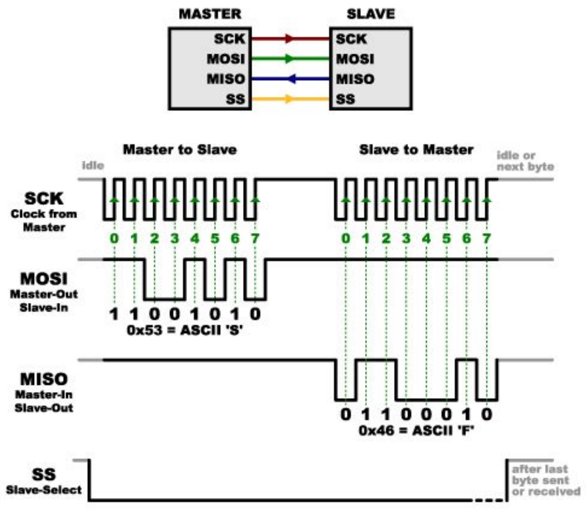
## STM32F4

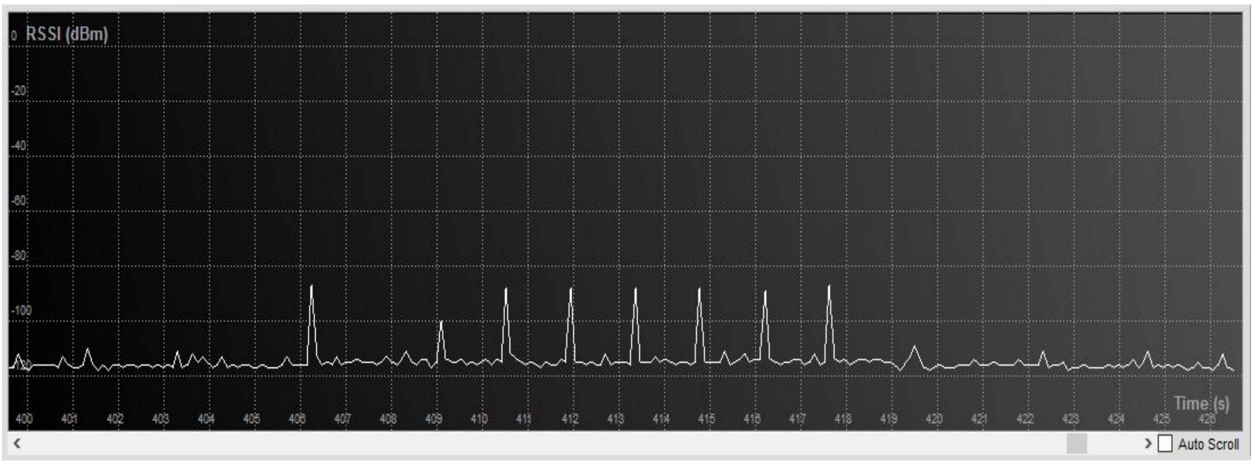

GND GND
VDD VDD
GND NRST
ADC C1 C0 ADC USB
ADC MIC SPI2 MOSI C3 C2 ADC SPI2 MISO
ADC PWM UART4 RX A1 A0 ADC BTN1 PWM UART4 TX
ADC PWM USART2 RX A3 A2 ADC PWM USART2 TX
ADC DAC MEMS SPI1 SCK 3.3v A5 A4 3.3v ADC DAC AUDIO
ADC MEMS PWM SPI1 MOSI A7 A6 ADC MEMS PWM SPI1 MISO
ADC C5 C4 ADC
ADC PWM B1 B0 ADC PWM
GND B2 BOOT1
E7 E8 PWM
PWM E9 E10 PWM
PWM E11 E12 PWM
PWM E13 E14 PWM
E15 B10 MIC I2C2 SCL PWM SPI2 SCK USART3 TX
I2C2 SDA PWM USART3 RX B11 B12
PWM SPI2 SCK B13 B14 PWM SPI2 MISO
PWM SPI2 MOSI B15 D8 USART3 TX
USART3 RX D9 D10
D11 D12 LED2 PWM
LED1 PWM D13 D14 LED3 PWM
LED4 PWM D15 NC
GND GND
GND GND
5V 5V
3V3 3V3
OSC H0 H1 OSC
OSC RTC C14 C15 OSC RTC
PWM E6 C13
E4 E5 PWM
E2 E3 MEMS
MEMS E0 E1 MEMS
I2C1 SCL PWM B8 B9 AUDIO I2C1 SDA PWM
BOOT0 VDD
AUDIO I2C1 SCL PWM USART1 TX B6 B7 I2C1 SDA PWM USART1 RX
PWM SPI1 MISO SPI3 MISO B4 B5 PWM SPI1 MOSI SPI3 MOSI
D7 B3 PWM SPI3 SCK SPI1 SCK
USART2 TX D5 D6 USART2 RX
D3 D4 AUDIO
D1 D2 UART5 RX
AUDIO SPI3 MOSI UART5 TX C12 D0
AUDIO SPI3 SCK UART4 TX USART3 TX C10 C11 SPI3 MISO USART3 RX UART4 RX
A14 A15
USB PWM USART1 RX A10 A13
I2C3 SCL PWM A8 A9 USB PWM USART1 TX
PWM C8 C9 I2C3 SDA PWM
PWM USART6 TX C6 C7 AUDIO PWM USART6 RX
GND GND
SWD
ST-LINK
DISCOVERY
MB997B
www.st.com/stm32f4-discovery
User

The STM32F407VGT6 is a microcontroller featuring 32-bit ARM® Cortex® -M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM (STMicroelectronics, 2016). This processor is much faster than the MSP430 shown before. The available I/O is also much better as well. The reason we have decided to use two different boards is to test our driver in Kubos to ensure that we have compatibility between the platforms.

# CC1120 SPI Radio

The first chip that we wrote drivers for is the CC1120 (Texas Instruments Inc., 2016). This chip is an RF transceiver meaning that if you matched an antenna to one side of the chip and understood its state machine variables via the control and communication interface you could transmit packets of data to some other transceiver using a common protocol. The communication interface to the IC is known as SPI or serial peripheral interface. The interface uses a common clock and TX and RX lines but uses a hardware enable line to slide information in both the RX and TX registers. This action is similar to a shift register.





*Here is an output seen on a matching radio graphing RSSI values.*

We have fallen short on ensuring that we had reliable 2 way packet transfer yet we are able to see that the radio is transmitting via our software layer in terms of controlling the radio. Given another month I am certain we would see the errors perhaps in our setup registers. Below is a state diagram for the CC112X family showing the various states and transitions.

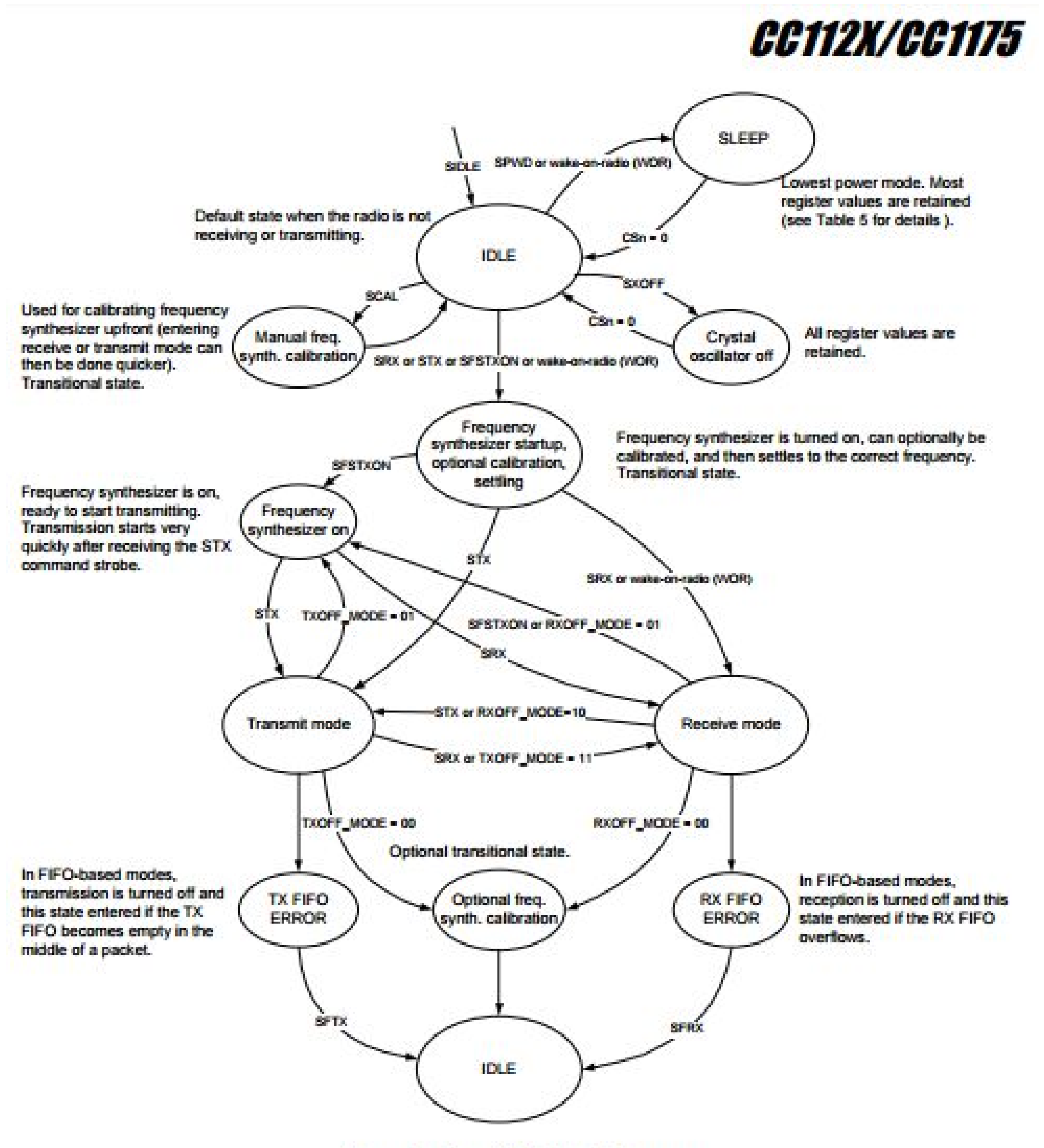
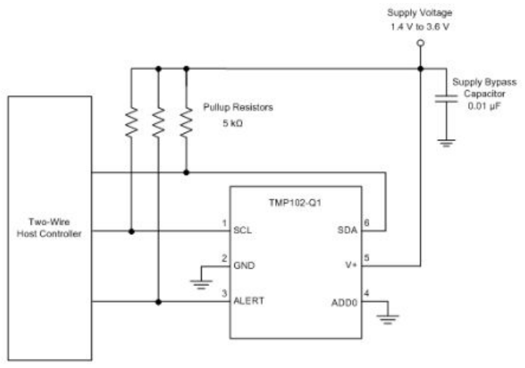

CC112X/CC1175
SLEEP
SIDLE
SPWD or wake-on-radio (WOR)
Lowest power mode. Most register values are retained (see Table 5 for details ).
Default state when the radio is not receiving or transmitting.
CSn = 0
IDLE
SXOFF
SCAL
CSn = 0
Used for calibrating frequency synthesizer upfront (entering receive or transmit mode can then be done quicker). Transitional state.
Manual freq. synth. calibration
Crystal oscillator off
All register values are retained.
SRX or STX or SFSTXON or wake-on-radio (WOR)
Frequency synthesizer startup, optional calibration, settling
Frequency synthesizer is turned on, can optionally be calibrated, and then settles to the correct frequency. Transitional state.
SFSTXON
Frequency synthesizer is on, ready to start transmitting. Transmission starts very quickly after receiving the STX command strobe.
Frequency synthesizer on
STX
SRX or wake-on-radio (WOR)
STX
TXOFF_MODE = 01
SFSTXON or RXOFF_MODE = 01
SRX
Transmit mode
STX or RXOFF_MODE=10
Receive mode
SRX or TXOFF_MODE = 11
TXOFF_MODE = 00
RXOFF_MODE = 00
Optional transitional state.
In FIFO-based modes, transmission is turned off and this state entered if the TX FIFO becomes empty in the middle of a packet.
TX FIFO ERROR
Optional freq. synth. calibration
RX FIFO ERROR
In FIFO-based modes, reception is turned off and this state entered if the RX FIFO overflows.
SFTX
SFRX
IDLE

**Figure 2: Simplified State Diagram**

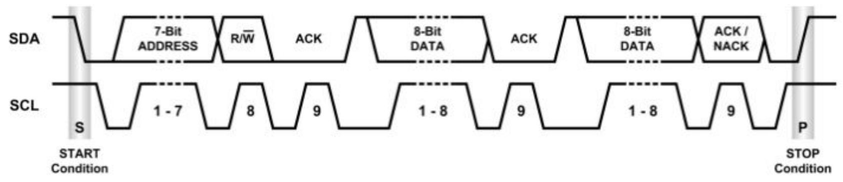
## TMP102 I2C Temperature Sensor

On the other hand we are able to claim success on a driver for the TMP102 temperature sensor developed by TI (Texas Instruments, Inc., 2016b). This sensor stands as a replacement of needing to have an analog interface that might be subject to noise when looking at precision temperatures. Per TI:

*The TMP102 device is a digital temperature sensor ideal for NTC/PTC thermistor replacement where high accuracy is required. The device offers an accuracy of ±0.5°C without requiring calibration or external component signal conditioning. IC temperature sensors are highly linear and do not require complex calculations or lookup tables to derive the temperature. The on-chip 12-bit ADC offers resolutions down to 0.0625°C. -Texas Instruments.*



TMP102 schematic

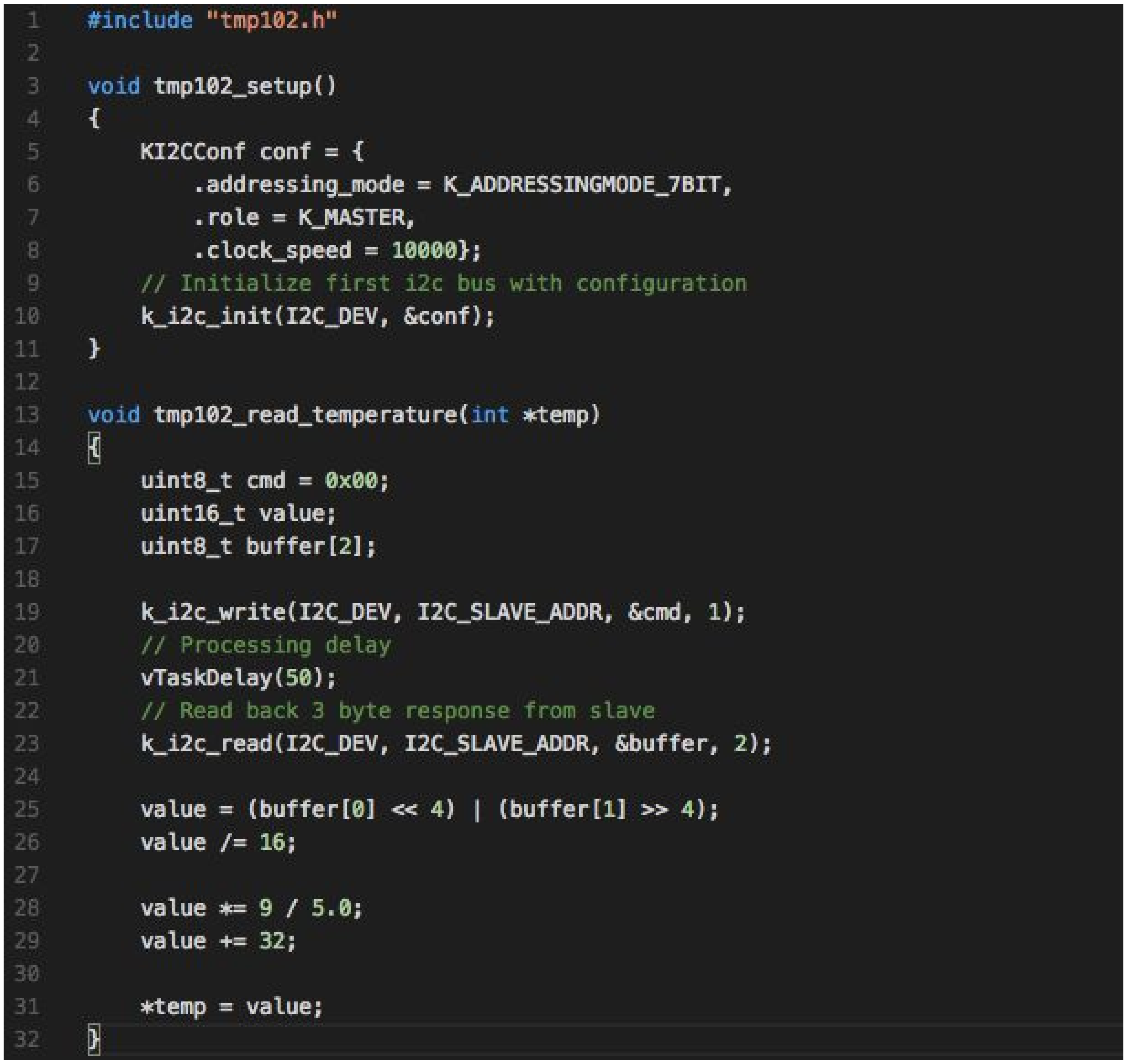


I2C signal diagram

This IC differs from the last IC in terms of the communication bus. As compared to SPI I2C aka (Inter-Integrated Circuit) uses a two tire bus as compared to the four wires required for the SPI chips. This is down by having assigned address in the hardware that we must address within our software to be able to communicate to the chip on that bus line. Beyond this simple bit math is applied to ensure that we are outputting the appropriate value for temperature.

# KubOS Driver and Task Examples (C code)

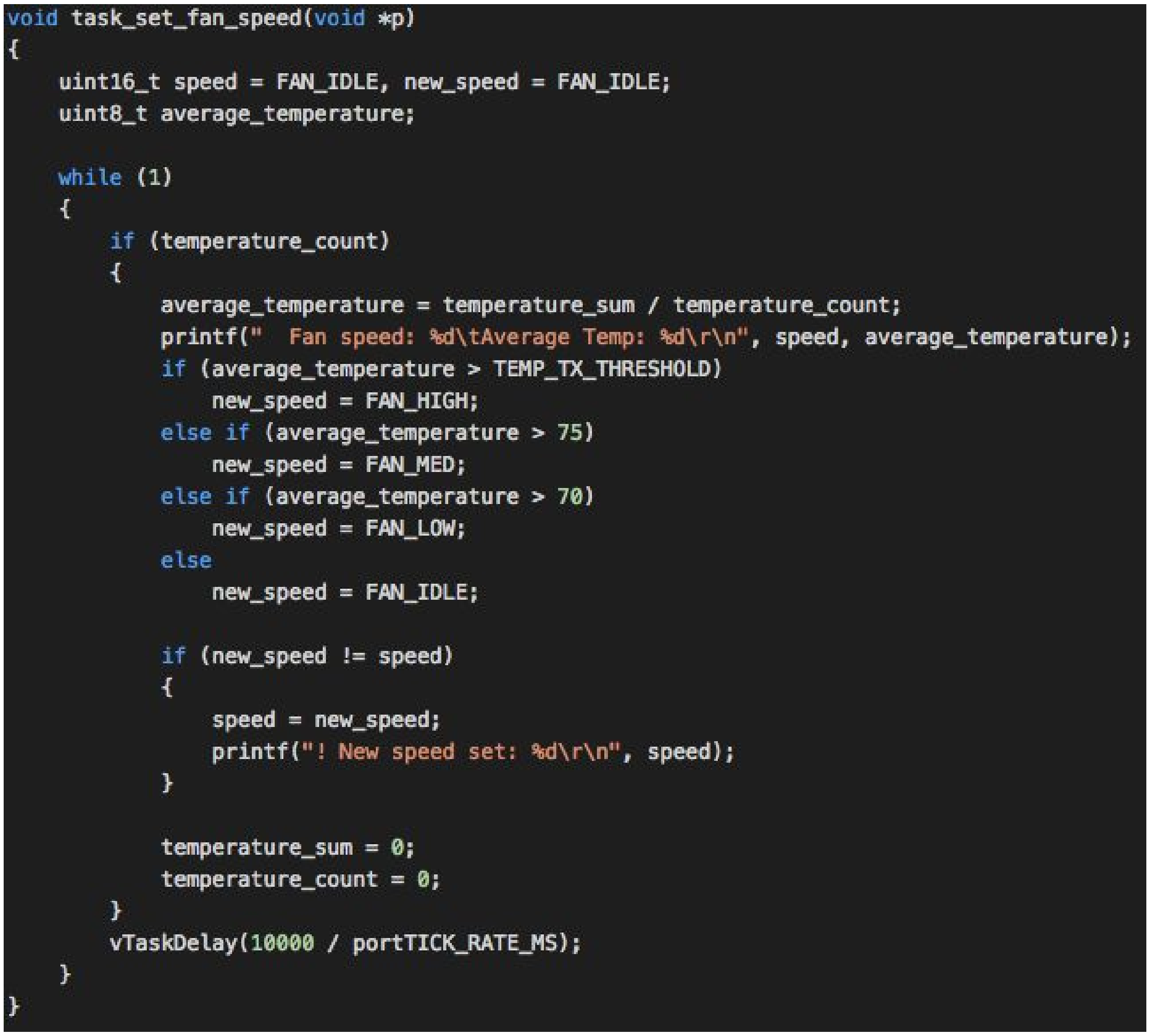
## TMP102 Driver



```c
#include "tmp102.h"

void tmp102_setup()
{
    KI2CConf conf = {
        .addressing_mode = K_ADDRESSINGMODE_7BIT,
        .role = K_MASTER,
        .clock_speed = 10000};
    // Initialize first i2c bus with configuration
    k_i2c_init(I2C_DEV, &conf);
}

void tmp102_read_temperature(int *temp)
{
    uint8_t cmd = 0x00;
    uint16_t value;
    uint8_t buffer[2];

    k_i2c_write(I2C_DEV, I2C_SLAVE_ADDR, &cmd, 1);
    // Processing delay
    vTaskDelay(50);
    // Read back 3 byte response from slave
    k_i2c_read(I2C_DEV, I2C_SLAVE_ADDR, &buffer, 2);

    value = (buffer[0] << 4) | (buffer[1] >> 4);
    value /= 16;

    value *= 9 / 5.0;
    value += 32;

    *temp = value;
}
```

*TMP102 temperature sensor implementation file*

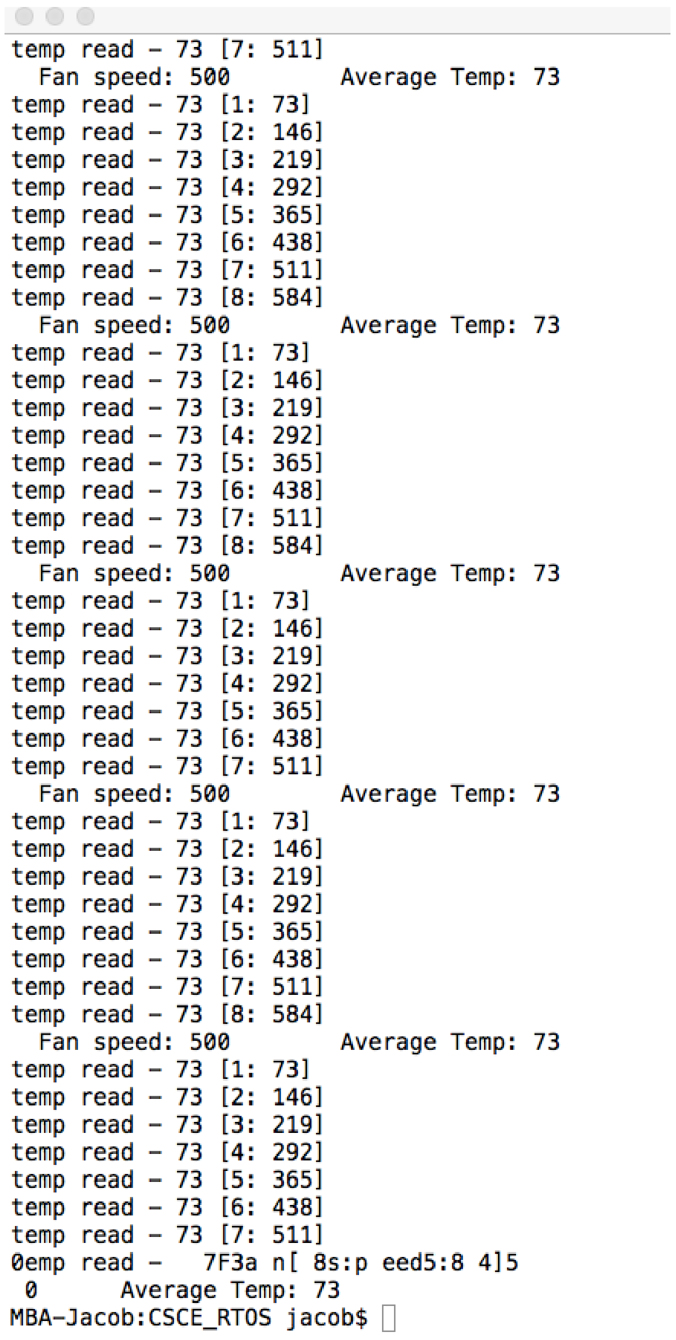
## Temperature and Fan Speed RTOS Task



```c
void task_set_fan_speed(void *p)
{
    uint16_t speed = FAN_IDLE, new_speed = FAN_IDLE;
    uint8_t average_temperature;

    while (1)
    {
        if (temperature_count)
        {
            average_temperature = temperature_sum / temperature_count;
            printf("  Fan speed: %d\tAverage Temp: %d\r\n", speed, average_temperature);
            if (average_temperature > TEMP_TX_THRESHOLD)
                new_speed = FAN_HIGH;
            else if (average_temperature > 75)
                new_speed = FAN_MED;
            else if (average_temperature > 70)
                new_speed = FAN_LOW;
            else
                new_speed = FAN_IDLE;

            if (new_speed != speed)
            {
                speed = new_speed;
                printf("! New speed set: %d\r\n", speed);
            }

            temperature_sum = 0;
            temperature_count = 0;
        }
        vTaskDelay(10000 / portTICK_RATE_MS);
    }
}
```

*RTOS task to average the input from the temperature sensor and set fan speed*

## Console Output



```
temp read - 73 [7: 511]
  Fan speed: 500        Average Temp: 73
temp read - 73 [1: 73]
temp read - 73 [2: 146]
temp read - 73 [3: 219]
temp read - 73 [4: 292]
temp read - 73 [5: 365]
temp read - 73 [6: 438]
temp read - 73 [7: 511]
temp read - 73 [8: 584]
  Fan speed: 500        Average Temp: 73
temp read - 73 [1: 73]
temp read - 73 [2: 146]
temp read - 73 [3: 219]
temp read - 73 [4: 292]
temp read - 73 [5: 365]
temp read - 73 [6: 438]
temp read - 73 [7: 511]
temp read - 73 [8: 584]
  Fan speed: 500        Average Temp: 73
temp read - 73 [1: 73]
temp read - 73 [2: 146]
temp read - 73 [3: 219]
temp read - 73 [4: 292]
temp read - 73 [5: 365]
temp read - 73 [6: 438]
temp read - 73 [7: 511]
  Fan speed: 500        Average Temp: 73
temp read - 73 [1: 73]
temp read - 73 [2: 146]
temp read - 73 [3: 219]
temp read - 73 [4: 292]
temp read - 73 [5: 365]
temp read - 73 [6: 438]
temp read - 73 [7: 511]
temp read - 73 [8: 584]
  Fan speed: 500        Average Temp: 73
temp read - 73 [1: 73]
temp read - 73 [2: 146]
temp read - 73 [3: 219]
temp read - 73 [4: 292]
temp read - 73 [5: 365]
temp read - 73 [6: 438]
temp read - 73 [7: 511]
0emp read -   7F3a n[ 8s:p eed5:8 4]5
 0      Average Temp: 73
MBA-Jacob:CSCE_RTOS jacob$ 
```

*Output from MSP board with 2 tasks:*

1. *Collect and then send temperature*
2. *Average the sum of collected temperatures and set fan speed accordingly*

# Build Environment

## Docker

Since building an RTOS, middleware and tasks require a significant amount of third-party libraries, specific tooling and versions, the KubOS team created a Docker image to ease development. The Docker image contains all the needed shared libs, headers, and the specific compiler needed to build FreeRTOS, and KubOS. This enabled development to take place on any system that can run Docker, namely Windows 7+, Linux 3.10+, and OSX 10.10+ (Docker Inc., 2016).

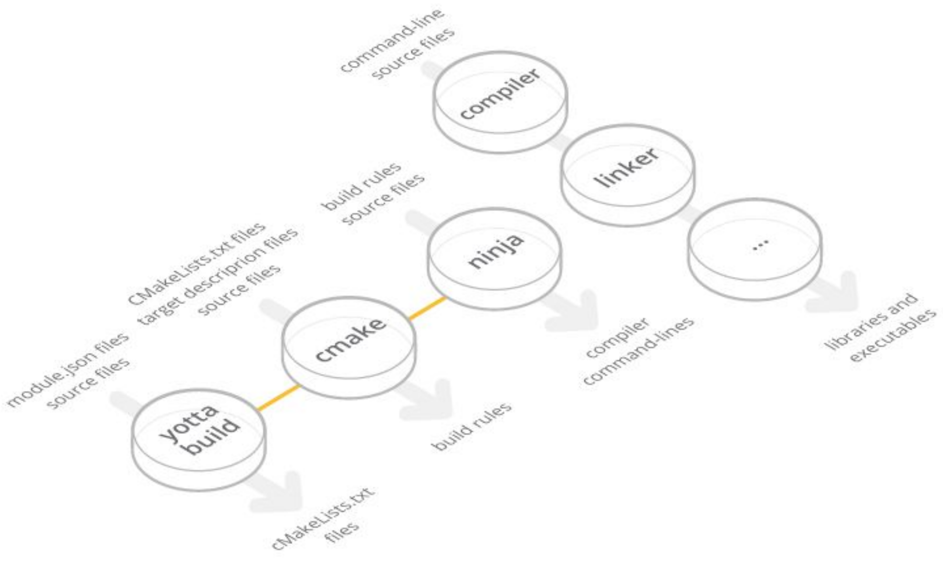
The build environment is wrapped with a simple command line app written in Python. This wrapper does three main things:

- Installs and updates the Docker build environment (Docker image/container)
- Builds and tests the environment (FreeRTOS/KubOS/userland code)
- Flashes embedded boards (through a connected port)

## Yotta

Yotta is a json-based module manager (ARM Ltd., 2016). Yotta simplifies reusing code among projects and enables different build targets for different embedded CPUs. Once a `module.json` file is created, yotta can download, build and inject C/C++ modules for easy reusability. It is based on the concepts of 'npm', which is the package manager for NodeJS.

With Yotta, drivers for different chips or sensors can be dynamically injected during the build process, shortening the overall process to increase development iteration. It also enables reuse among different manufacturers, since pinouts and lines can be set for each project without relying on static "defines" within each module.



## Conclusion

This project opened our eyes to the industry of RTOS middleware. Abstraction allows software developers in other domains to support and advance your project even when using an RTOS. Given the opportunity to further develop our RF driver would be fun as it was ambitious in our scope. We are grateful to be able to deliver to KubOS a working temperature driver and to provide feedback of their software environment. This was a fun course and an even more fun project. We look forward to applying our newly found skills in industry.

## References

ARM Ltd. (2016). yotta Documentation. Retrieved from http://yottadocs.mbed.com/

Docker Inc. (2016). Install Docker from binaries. Retrieved from https://docs.docker.com/engine/installation/binaries/

Kubos. (2016). KubOS. Retrieved from http://www.kubos.co/

STMicroelectronics. (2016). STM32F4DISCOVERY. Retrieved from http://www.st.com/en/evaluation-tools/stm32f4discovery.html

Texas Instruments Inc. (2016). CC1120. Retrieved from http://www.ti.com/product/CC1120

Texas Instruments, Inc. (2016a). MSP430F5529 Features. Retrieved from http://www.ti.com/product/MSP430F5529#features

Texas Instruments, Inc. (2016b). TMP102. Retrieved from http://www.ti.com/product/TMP102