ARL-TN-0920 ● OCT 2018

US Army Research Laboratory

Experimentation with Inexpensive Internet of Things (IoT) Modules: A Thermometer Using LoRaWAN

by Timothy C Gregory

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0920 ● OCT 2018

US Army Research Laboratory

Experimentation with Inexpensive Internet of Things (IoT) Modules: A Thermometer Using LoRaWAN

by Timothy C Gregory
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
October 2018	Technical Note	1 January 2018–31 August 2018

4. TITLE AND SUBTITLE
Experimentation with Inexpensive Internet of Things (IoT) Modules: A Thermometer Using LoRaWAN

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Timothy C Gregory

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
Computational and Information Sciences Directorate (ATTN: RDRL-CII-B)
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0920

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The US Army Research Laboratory's Battlefield Information Processing Branch is experimenting with inexpensive Internet of Things (IoT) modules. This technical note describes the implementation of a wireless thermometer using such modules.

15. SUBJECT TERMS
Internet of Things, IoT, sensors, thermometer, LoRa, LoRaWAN, Pycom

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy C Gregory
Unclassified	Unclassified	Unclassified	UU	22	19b. TELEPHONE NUMBER (Include area code) (301) 394-5604

Contents

List of Figures

1. Introduction

The availability of commercial Internet of Things (IoT) modules has enabled the development of IoT for the battlefield.

This document describes a simple electronic thermometer that reports through the LoRaWAN infrastructure. The device will send temperature information and geographic position to the LoRaWAN gateway, which will distribute it via an Internet port.

2. Hardware

There are many development boards available for experimenting with IoT. The Pycom LoPy was chosen because of availability and low price.

2.1 Pycom LoPy

The Pycom LoPy module was selected as the foundation for the project. The LoPy is a MicroPython-enabled development board. The LoPy provides connectivity through WiFi, LoRa, Bluetooth, and Sigfox.

The specifications of the LoPy are shown in Fig. 1.

CPU:
- Xtensa® dual–core 32–bit LX6 microprocessor(s), up to 600 DMIPS
- Hardware floating point acceleration
- Python multi–threading
- An extra ULP–coprocessor that can monitor GPIOs, the ADC channels and control most of the internal peripherals during deep–sleep mode while only consuming ~25uA

Memory:
- RAM: 520KB + 4MB
- External flash: 8MB

WiFi
- 802.11b/g/n 16mbps
- 3.4 Bluetooth
- Low energy and classic

LoRa
- LoRaWAN 1.0.2 stack - Class A and C devices
- Node range: Up to 40km
- Nano-gateway: Up to 22km (Capacity up to 100 nodes)

Fig. 1 LoPy specifications

2.2 Temperature Sensor

The temperature data are provided by a prewired and waterproofed version of the DS18B20 temperature sensor. The DS18B20 interfaces with the LoPy board through a 1-Wire bus. The 1-Wire bus uses only one wire for data. There are MicroPython libraries available that provide support for the DS18B20.

2.3 Location Sensor

The Pycom Pytrack expansion board is used to provide the GPS data. The Pytrack board also provides a header for accessing some of the general purpose input/output pins on the Pycom board.

3. Programming Environment

3.1 MicroPython

The LoPy is programmed using the MicroPython programming language. MicroPython is an implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers and in constrained environments.

3.2 Libraries

There are MicroPython drivers and libraries to support various devices. They libraries are available through GitHub at https://github.com/micropython/micropython.

For this application, we use the OneWire class to enable using the 1-wire interface on the LoPy. We also use the DS18X20 class to operate the DS18X20 thermometer.

The source code for the OneWire and DS18X20 libraries are shown in Appendix A.

3.3 Development Environment

Running MicroPython code on the Pycom board can be done through the Read Evaluate Print Loop (REPL) interactive terminal. You can connect to the REPL terminal through a USB serial port. Once you have tested and debugged your code, you can install it in a file named “main.py” on the Pycom board. “main.py” is run each time the board boots up. Further information on connecting to and programming the module is located on the Pycom website (https://docs.pycom.io/gettingstarted/programming).

3.4 LoPy File System

The file system of the LoPy contains two MicroPython scripts: “boot.py” and “main.py”. These are executed when power is applied to the device or the device is rebooted. “main.py” is run at startup after “boot.py”. See the Pycom website (https://docs.pycom.io/product-info/development/lopy) for more information on the boot order and file system.

4. Connecting the Temperature Sensor

The connections between the DS18B20 temperature sensor and the Pytrack board are shown in Fig. 2. The wire colors may be different depending upon the source of the prewired DS18B20. Figures 3 and 4 show the relative positions of the header pins. Figure 5 details the functions of the header pins.

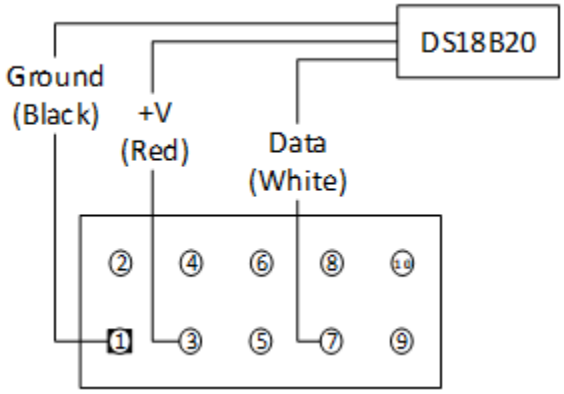

Fig. 2 Temperature sensor wiring

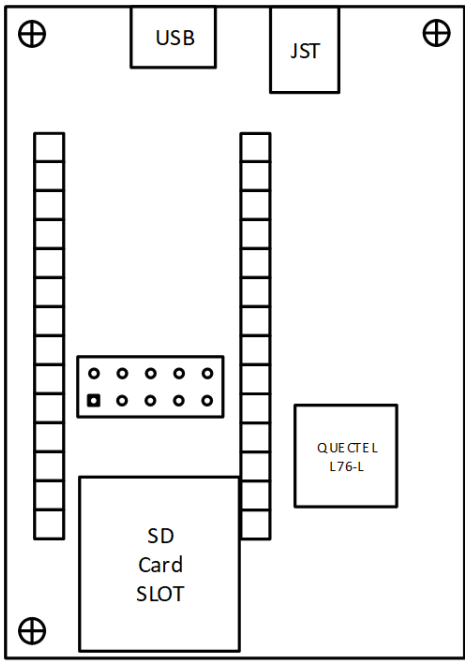

Fig. 3 Pytrack board layout

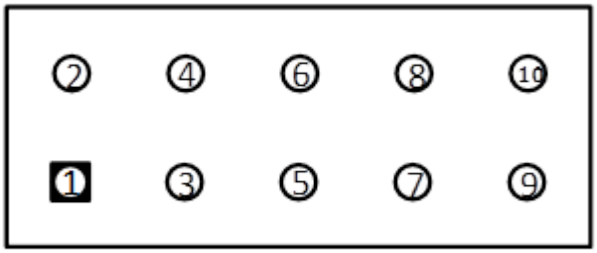

Fig. 4 Pytrack header details

1. GND
2. Power Enable
3. 3.3 Volts sensors
4. 3.3 Volts PyCom module
5. EXTIO_0
6. EXTIO_1 (P9)
7. EXTIO_2 (P10)
8. EXTIO_3 (P11)

Fig. 5 Pytrack header terminal assignments

5. MicroPython Source Code

Appendix B details the MicroPython source code that reads sensor data and sends it to the LoRaWAN gateway.

6. Summary

The availability of inexpensive IoT modules has facilitated experimentation with the IoT technology. This experimentation should enable the evaluation of different ideas and components to determine their application to the Army.

Appendix A. OneWire and DS18X20 Library

This appendix appears in its original form, without editorial change.

```
#!/usr/bin/env python3


"""
OneWire library for MicroPython
"""

import time
import machine

class OneWire:
    CMD_SEARCHROM = const(0xf0)
    CMD_READROM = const(0x33)
    CMD_MATCHROM = const(0x55)
    CMD_SKIPROM = const(0xcc)

    def __init__(self, pin):
        self.pin = pin
        self.pin.init(pin.OPEN_DRAIN, pin.PULL_UP)

    def reset(self):
        """
        Perform the onewire reset function.
        Returns True if a device asserted a presence pulse, False
otherwise.
        """
        sleep_us = time.sleep_us
        disable_irq = machine.disable_irq
        enable_irq = machine.enable_irq
        pin = self.pin

        pin(0)
        sleep_us(480)
        i = disable_irq()
        pin(1)
        sleep_us(60)
        status = not pin()
        enable_irq(i)
        sleep_us(420)
        return status

    def read_bit(self):
        sleep_us = time.sleep_us
        enable_irq = machine.enable_irq
        pin = self.pin

        pin(1) # half of the devices don't match CRC without this line
        i = machine.disable_irq()
        pin(0)
        sleep_us(1)
        pin(1)
        sleep_us(1)
        value = pin()
        enable_irq(i)
        sleep_us(40)
        return value

    def read_byte(self):
        value = 0
        for i in range(8):
            value |= self.read_bit() << i
        return value
```

```
    def read_bytes(self, count):
        buf = bytearray(count)
        for i in range(count):
            buf[i] = self.read_byte()
        return buf

    def write_bit(self, value):
        sleep_us = time.sleep_us
        pin = self.pin

        i = machine.disable_irq()
        pin(0)
        sleep_us(1)
        pin(value)
        sleep_us(60)
        pin(1)
        sleep_us(1)
        machine.enable_irq(i)

    def write_byte(self, value):
        for i in range(8):
            self.write_bit(value & 1)
            value >>= 1

    def write_bytes(self, buf):
        for b in buf:
            self.write_byte(b)

    def select_rom(self, rom):
        """
        Select a specific device to talk to. Pass in rom as a bytearray
(8 bytes).
        """
        self.reset()
        self.write_byte(CMD_MATCHROM)
        self.write_bytes(rom)

    def crc8(self, data):
        """
        Compute CRC
        """
        crc = 0
        for i in range(len(data)):
            byte = data[i]
            for b in range(8):
                fb_bit = (crc ^ byte) & 0x01
                if fb_bit == 0x01:
                    crc = crc ^ 0x18
                crc = (crc >> 1) & 0x7f
                if fb_bit == 0x01:
                    crc = crc | 0x80
                byte = byte >> 1
        return crc

    def scan(self):
        """
        Return a list of ROMs for all attached devices.
        Each ROM is returned as a bytes object of 8 bytes.
        """
        devices = []
        diff = 65
        rom = False
        for i in range(0xff):
```

```
            rom, diff = self._search_rom(rom, diff)
            if rom:
                devices += [rom]
            if diff == 0:
                break
        return devices

    def _search_rom(self, l_rom, diff):
        if not self.reset():
            return None, 0
        self.write_byte(CMD_SEARCHROM)
        if not l_rom:
            l_rom = bytearray(8)
        rom = bytearray(8)
        next_diff = 0
        i = 64
        for byte in range(8):
            r_b = 0
            for bit in range(8):
                b = self.read_bit()
                if self.read_bit():
                    if b: # there are no devices or there is an error on
the bus
                        return None, 0
                else:
                    if not b: # collision, two devices with different bit
meaning
                        if diff > i or ((l_rom[byte] & (1 << bit)) and
diff != i):
                            b = 1
                            next_diff = i
                self.write_bit(b)
                if b:
                    r_b |= 1 << bit
                i -= 1
            rom[byte] = r_b
        return rom, next_diff

class DS18X20(object):
    def __init__(self, onewire):
        self.ow = onewire
        self.roms = \
             [rom for rom in self.ow.scan() if rom[0] == 0x10 or rom[0]
       == 0x28]
        self.fp = True
        try:
            1/1
        except TypeError:
            self.fp = False # floatingpoint not supported

    def isbusy(self):
        """
        Checks wether one of the DS18x20 devices on the bus is busy
        performing a temperature convertion
        """
        return not self.ow.read_bit()

    def start_conversion(self, rom=None):
        """
        Start the temp conversion on one DS18x20 device. Pass the 8-byte
bytes object
        with the ROM of the specific device you want to read.
        If only one DS18x20 device is attached to the bus you may
```

```
        omit the rom parameter.
        """
        if (rom==None) and (len(self.roms)>0):
            rom=self.roms[0]
        if rom!=None:
            rom = rom or self.roms[0]
            ow = self.ow
            ow.reset()
            ow.select_rom(rom)
            ow.write_byte(0x44)  # Convert Temp

    def read_temp_async(self, rom=None):
        """
        Read the temperature of one DS18x20 device if the conversion is
complete,
        otherwise return None.
        """
        if self.isbusy():
            return None
        if (rom==None) and (len(self.roms)>0):
            rom=self.roms[0]
        if rom==None:
            return None
        else:
            ow = self.ow
            ow.reset()
            ow.select_rom(rom)
            ow.write_byte(0xbe)  # Read scratch
            data = ow.read_bytes(9)
            return self.convert_temp(rom[0], data)

    def convert_temp(self, rom0, data):
        """
        Convert the raw temperature data into degrees celsius and return
as a fixed
        point with 2 decimal places.
        """
        temp_lsb = data[0]
        temp_msb = data[1]
        if rom0 == 0x10:
            if temp_msb != 0:
                # convert negative number
                temp_read = temp_lsb >> 1 | 0x80  # truncate bit 0 by
shifting, fill \
                                                  # high bit with 1.
                temp_read = -((~temp_read + 1) & 0xff) # now convert from
two's \
                                                       # complement
            else:
                temp_read = temp_lsb >> 1  # truncate bit 0 by shifting
            count_remain = data[6]
            count_per_c = data[7]
            if self.fp:
                return temp_read - 25 + (count_per_c - count_remain) /
count_per_c
            else:
                return 100 * temp_read - 25 +\
                 (count_per_c - count_remain) // count_per_c
        elif rom0 == 0x28:
            temp = None
            if self.fp:
                temp = (temp_msb << 8 | temp_lsb) / 16
            else:
```

```
            temp = (temp_msb << 8 | temp_lsb) * 100 // 16
        if (temp_msb & 0xf8) == 0xf8: # for negative temperature
            temp -= 0x1000
        return temp
    else:
        assert False
```

Appendix B. MicroPython Source Code

This appendix appears in its original form, without editorial change.

```
import time
import utils
import pycom
from machine import Timer
from  machine import Pin

import network
from network import WLAN
from network import LoRa
import socket

import binascii
import pycom
import struct

from onewire import DS18X20
from onewire import OneWire

print('GPS/DS18X20 Test')

ow = OneWire(Pin('P10'))
temp = DS18X20(ow)

# activate the Thread used by GPS  'M6N' or 'G76-L'
utils.GPSstart('G76-L')
####################

def select_subband(lora, subband):
    if (type(subband) is int):
        if ((subband<1) or (subband>8)):
            raise ValueError("subband out of range (1-8)")
    else:
        raise TypeError("subband must be 1-8")

    for channel in range(0, 72):
        lora.remove_channel(channel)

    for channel in range((subband-1)*8, ((subband-1)*8)+8):
        lora.add_channel(channel, frequency=902300000+channel*200000,
dr_min=0, dr_max=4)

# Initialize LoRa in LORAWAN mode.
lora = LoRa(mode=LoRa.LORAWAN)

# create an OTAA authentication parameters
# The values specified below for app_eui and app_key are placeholders.
# You will need to specify the correct “app_eui” and “app_key” for your
network.
# These can be found in the configuration of the your LoRa Network
Server.
# app_eui is the LoRa Network ID.
# app_key is the LoRa Network Key
app_eui = binascii.unhexlify('00:00:00:00:00:00:00:00'.replace(':',''))
app_key =
binascii.unhexlify('00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00'.repl
ace(':',''))
print("Network ID: ",binascii.hexlify(app_eui))
print("Network Key: ",binascii.hexlify(app_key))
print("Frequency: ",lora.frequency())

pycom.heartbeat(False)
# join the LoRa network using OTAA (Over the Air Activation)
for i in range(20):
```

```
   try:
    print('Connecting to LoRaWAN')
    stime = time.time()
    lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key),
timeout=20000)
    etime = time.time()
    print("Connection established in ", etime-stime," seconds")
    break
   except:
    pycom.rgbled(0x7f7f00)
    print("LoRaWAN connection timed out...retrying")

    i = 1
# wait until the module has joined the LoRa network
if lora.has_joined() != True:
    print("NO LORAWAN CONNECTION ESTABLISHED...EXITING!")
    pycom.rgbled(0x7f0000)
    sys.exit(-1)
pycom.rgbled(0x007f00)
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
s.setblocking(True)

GPS_Locked_Flag=False

while GPS_Locked_Flag==False:
    if utils.GPS_Is_Fixed==True :
            GPS_Locked_Flag=True
            print("GPS coordinates:")
            print(utils.GPSlat)
            print(utils.GPSlon)
    else:
        print ('No Sat')
    time.sleep(0.5)

msgCount=0
temp.start_conversion()
time.sleep(1)

while True:
    temperature = temp.read_temp_async()
    print("Temperature: "+str(temperature))
    if (temperature == None):
        temperature=0.0
    tempbytes = list(struct.pack("!f",temperature))
    print("Temperature: "+str(temperature))
    print("Latitude: " + str(utils.GPSlat))
    print("Longitude: "+str(utils.GPSlon))
    print("Time: "+str(utils.GPSDatetime))
    msgCount=msgCount+1
    msgcountBytes=msgCount.to_bytes(2,"little") #has to be fixed

    lat =  float(utils.GPSlat[0]+(utils.GPSlat[1]/60.0))
    if utils.GPSlat[2]=="S":
        lat=-lat
    latBytes=list(struct.pack("!f", lat))

    lon = float(utils.GPSlon[0]+(utils.GPSlon[1]/60.0))
    if utils.GPSlon[2]=="W":
        lon=-lon
    lonBytes=list(struct.pack("!f", lon))

    print("Lat:")
    print (lat)
```

```
print("Long:")
print (lon)

# build the array of data.
DataPack=bytes([33, msgcountBytes[1],
       msgcountBytes[0],
       latBytes[3],
       latBytes[2],
       latBytes[1],
       latBytes[0],
       lonBytes[3],
       lonBytes[2],
       lonBytes[1],
       lonBytes[0],
       tempbytes[3],
       tempbytes[2],
       tempbytes[1],
       tempbytes[0]]
       )
print("Sent: ",list(DataPack))
s.send(DataPack)
temp.start_conversion()
time.sleep(5)
```

List of Symbols, Abbreviations, and Acronyms

GPS	global positioning system
IoT	Internet of Things
REPL	Read Evaluate Print Loop
USB	Universal Serial Bus

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	ARL RDRL CII B T C GREGORY