Embedded Networking

Dongbing Gu

School of Computer Science and Electronic Engineering
University of Essex
UK

Spring 2018

Outline

Introduction

- In the non-embedded world, TCP/IP over Ethernet, WiFi, etc. dominates.
- No single embedded network or network protocol dominates.
- Many TCP/IP features unnecessary or undesirable in embedded networks.
- Reliability of individual packets is important as opposed to building reliability with retransmission.

OSI model

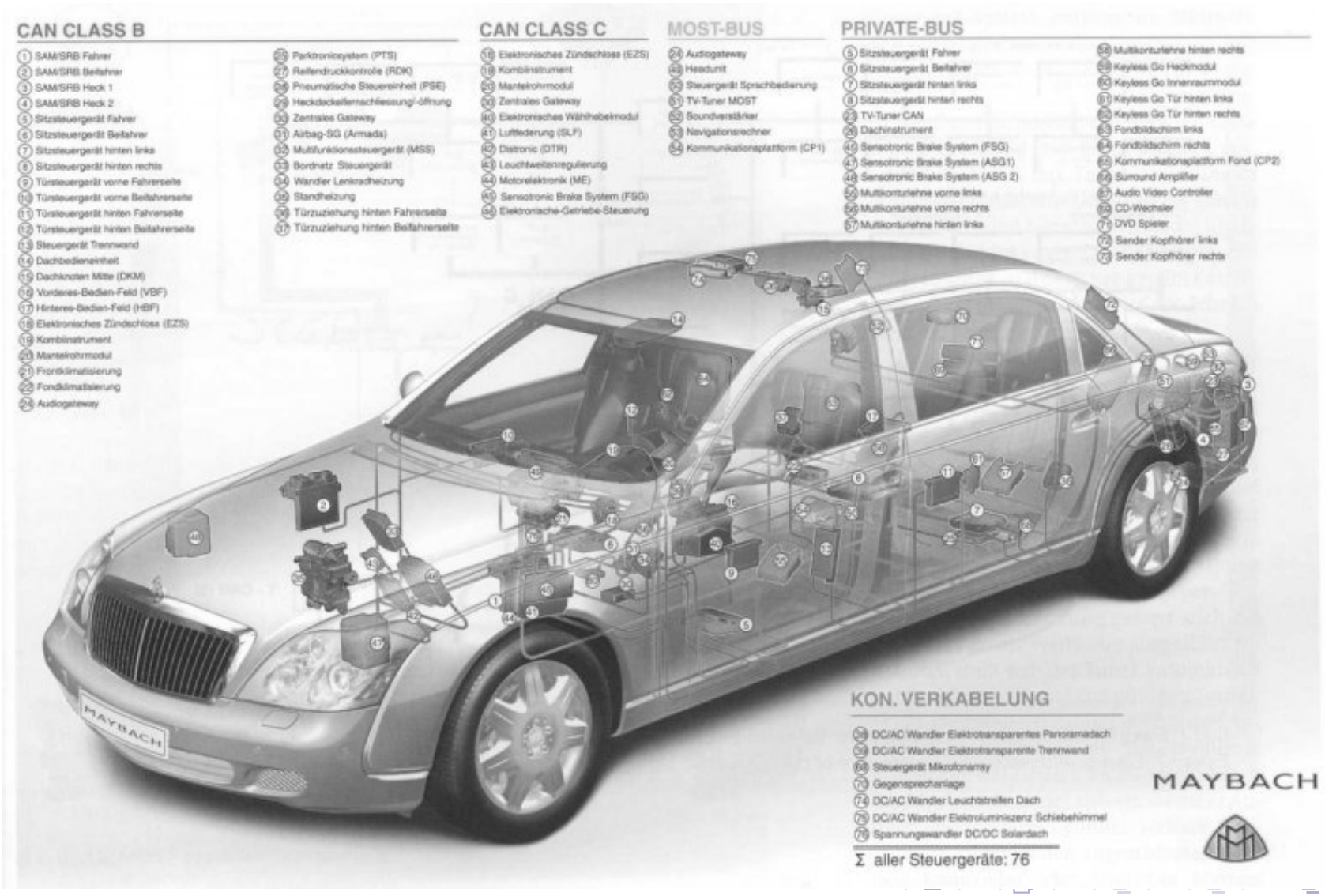
7. Application	Data	High-level APIs, including resource sharing, remote file access
6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
4. Transport	Segment (TCP) / Datagram (UDP)	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
3. Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
2. Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
1. Physical	Bit	Transmission and reception of raw bit streams over a physical medium

A Few Embedded Networks

- Low-end: SPI, I2C, RS-232, RS485
- Medium-end: CAN, Modbus, Profibus.
- High-end: Ethernet, Profinet.

Network from a High End Car

- Some new cars contain more than 3 miles of wire



Section 1

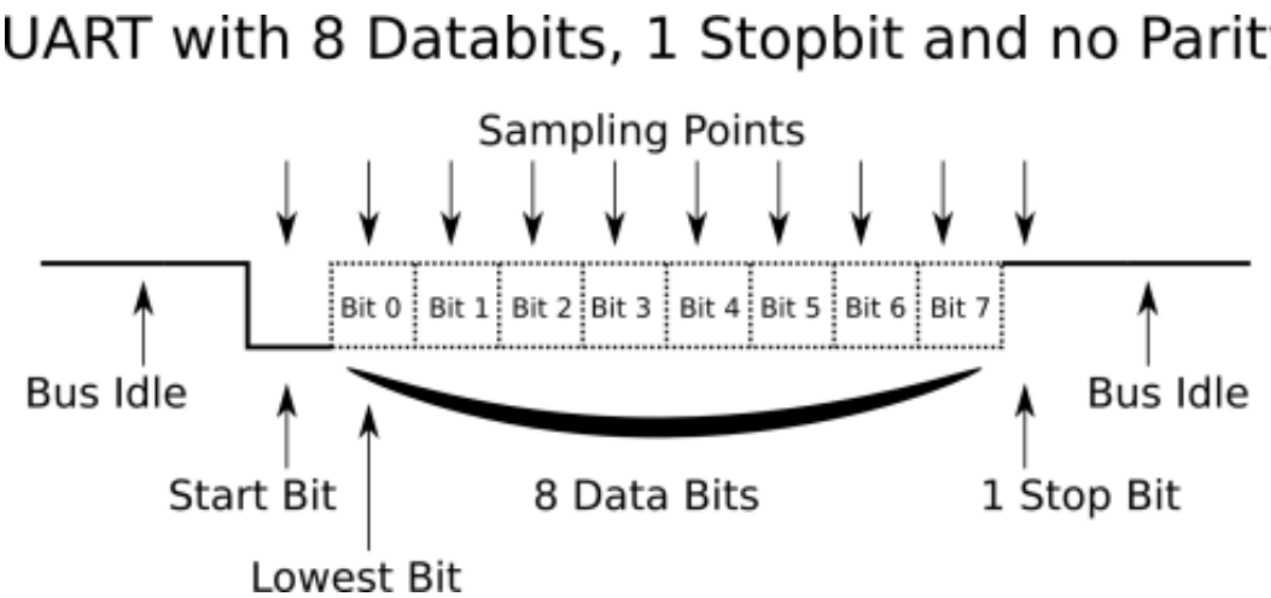
UART Serial Port

UART Serial Port

- Universal asynchronous receiver transmitter (UART) : provides serial communication.
- Allows many communication parameters to be programmed (using the UART's Control register).
- UARTs are a standard port in microcontrollers.
- In the RS-232 standard's simplest implementation, only three wires are used in the cable.
- One wire is used to transmit data (TD), one to receive data (RD) and one to connect the signal ground (GND).

Serial Communication

- Bits are transmitted separately, one at a time, as a sequence of bits.
- For example, an n-bit character would be sent as a sequence of bits.
- An extra Parity Bit may be added to the end of the sequence as a check for transmission errors.



Serial Communication Parameters

- The parameters are set initially, before the peripheral interface is used, by setting appropriate bit patterns in one or more control registers associated with the serial port.
  - Baud (bit) rate.
  - Number of bits per character: Usually 8 data bits, although 5, 6 and 7 are allowed.
  - The START bit is always low and forces a transition from line idle to indicate a new data byte.
  - Parity/no parity: optional, indicates ODD or EVEN bit parity in the data byte
  - Length of stop bit (1, 1.5, 2 bits):The STOP bit is always high and forces the line idle state at the end of the transmission
- Today, data rate can range as high as 115.2 kbits/s. Typical data rates are 1200, 2400, 4800, 9600, 19,200, 38,400, and 115,200 bits/s.

Serial Communication Parameters

- Besides the synchronization provided by the use of start and stop bits, a parity bit may optionally be transmitted along with the data.
- It helps detect data corruption that might occur during transmission.
- When even or odd parity is being used, the number of marks (logical 1 bits) in each data byte are counted, and a single bit is transmitted following the data bits to indicate whether the number of 1 bits just sent is even or odd.
- For example, when even parity is chosen, the parity bit is transmitted with a value of 0 if the number of preceding marks is an even number, or with a value 1 if the number of preceding marks is an odd number.
- Parity error checking is very rudimentary. While it will tell you if there is a single bit error in the character, it doesn't show which bit was received in error. Also, if an even number of bits are in error then the parity bit would not reflect any error at all.

UART on the mbed

```
#include "mbed.h"
Serial pc(USBTX, USBRX);
int main() {
  pc.baud(19200);
  pc.printf("Hello World!\n");
  while(1) {
     pc.putc(pc.getc() + 1);
  }
}
```

```
#include "mbed.h"
DigitalOut led1(LED1);
DigitalOut led2(LED2);
Serial pc(USBTX, USBRX);
void callback() {
  printf("%c\n", pc.getc());
  led2 = !led2;
}
int main() {
  pc.attach(&callback);
  while (1) {
     led1 = !led1;
     wait(0.5);
  }
}
```

RS-485

- The RS-485 standard specifies differential signalling on two lines rather than single-ended with a voltage referenced to ground. A logic 1 is a level smaller than -200 mV, and a logic 0 is a level greater than +200 mV.
- Typical line voltage levels from the line drivers are a minimum of $\pm$ 1.5 V to a maximum of about $\pm$ 6 V.
- The differential format produces effective common-mode noise cancellation
- The standard transmission medium is twisted-pair wire.
- Data rates up to 10 Mbit/s and distances up to 1,200 m.
- RS-485 is not a protocol, it's simply an electrical interface.
- TX+/RX+ or D+ as alternative for B (high for MARK i.e. idle). TX-/RX- or D- as alternative for A (low for MARK i.e. idle).
- SC is the common signal reference ground.

RS-485

- The standard has not defined a specific communication protocol. The standard UART protocol is sometimes used. Most applications define a unique protocol.
- The standard does not define specific connectors. Various connection methods have been used, including the RS-232 DE-9 connector.
- It is widely used in industrial applications where higher speeds and longer distances are needed. It is used in the same type of equipment as defined for the RS-232 interface. Networks defined by field buses like Profibus and Modbus use it as well.
- RS-232 for low-speed over short-distance. RS-485 for higher speeds over longer ranges with duplex networking capability.

Section 2

SPI Bus

SPI Characteristics

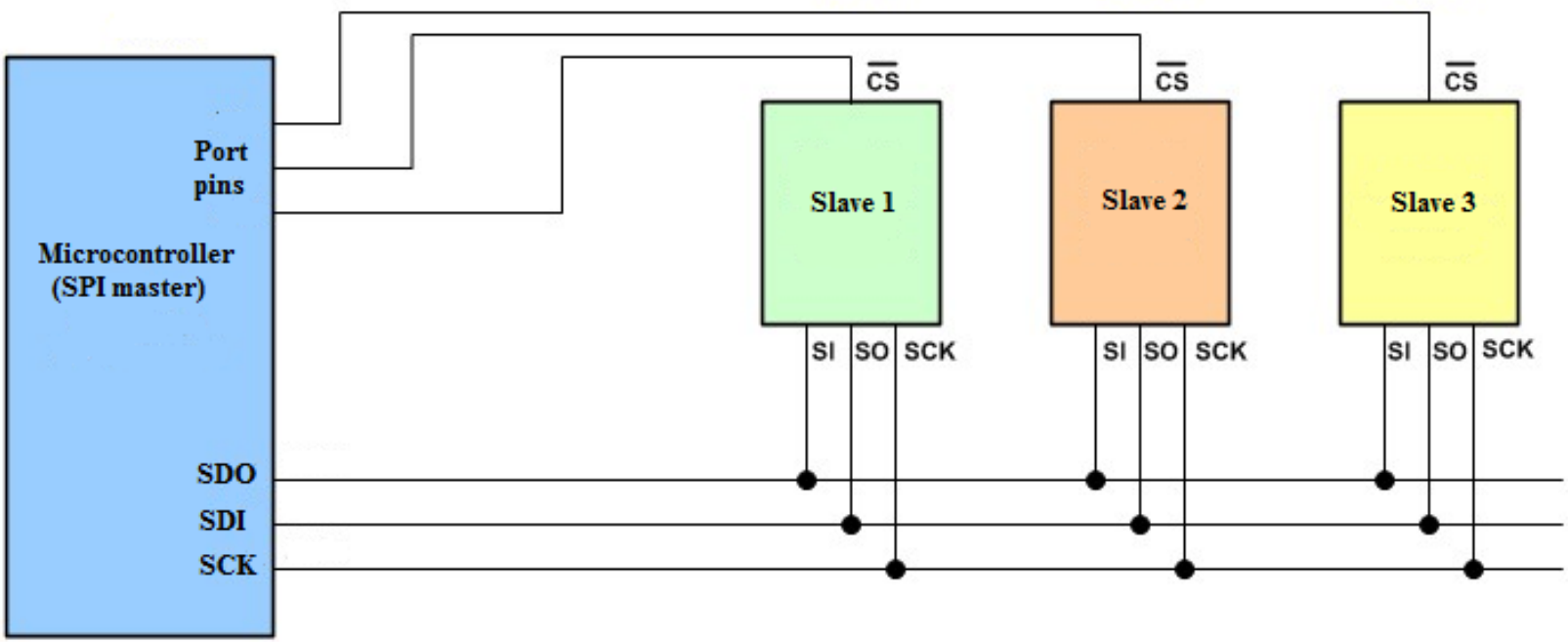
- Very local area - designed for communicating with other chips on the same PCB, such as DAC, flash memory, etc.
- Full-duplex.
- Low / medium bandwidth.
- Master / slave.

SPI Connection

- SPI has the following four signal lines: Serial Clock (SCKL), Chip Enable or Select (CS), Serial Data Input (SDI), Serial Data Output (SDO).
- The microprocessor drives the CS and SCKL signal lines. SPI slave devices get their clock and chip select input from microprocessor.
- Whenever an SPI device is not selected, its SDO output line is tri-stated.
- SPI interface hardware contains shift registers. One shift register is used to send out data and another shift register is used to receive data.
- The clocks are all synchronous and they use SCKL.

Multiple SPI devices

- Typical SPI master with multiple SPI slaves connection



SPI Transfer

- Master selects a slave by configuring the SPI interface and writing a byte into the SPI data register.
- Transfer begins at the next clock edge
- Eight bits transferred in each direction
- Waiting for transfer to finish by checking the SPI flag
- Read SPI status register and data register
- Master deselects the slave

SPI Problems

- No sophisticated addressing for more slaves.
- No flow control
- No acknowledgements
- No error detection / correction

SPI on the mbed

- The SPI Interface can be used on pins p5/p6/p7 and p11/p12/p13.
- Default settings of the SPI interface on the mbed:
  - Default clock frequency of 1 MHz
  - Default data length of 8 bits
  - Default mode of 0 (see page 120 in the recommended book "Fast and efficient embedded systems design")

SPI on the mbed

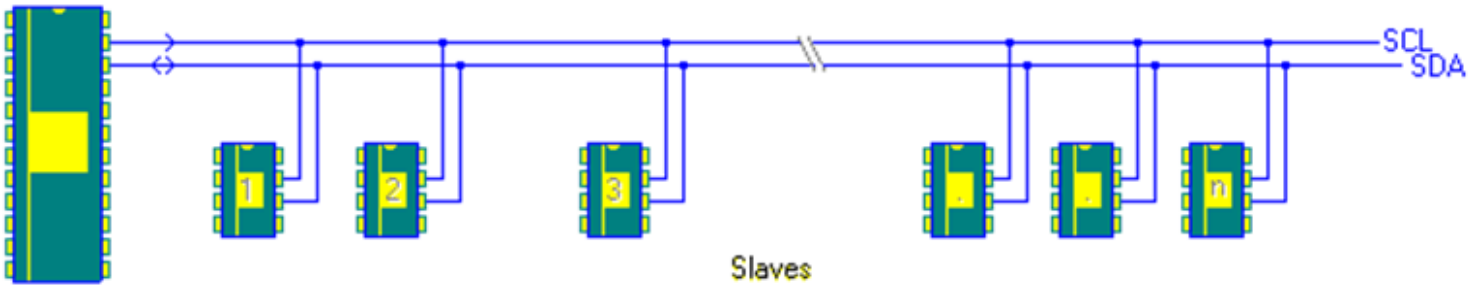
```
#include "mbed.h"
#include "TextLCD.h"
SPI sw(p5, p6, p7);
DigitalOut cs(p8);

int main() {
  cs=0;
  sw.format(16,0);
  sw.frequency(1000000);
  while (1) {
      sw.write(0x0000);
      cs = 1;       cs=0;
      wait(1);
      sw.write(0x5555);
      cs = 1;       cs=0;
      wait(1);
      sw.write(0xAAAA);
      cs = 1;       cs=0;
      wait(1);
      sw.write(0xFFFF);
      cs = 1;       cs=0;
      wait(1);
   }
}
```

Section 3

I2C Bus

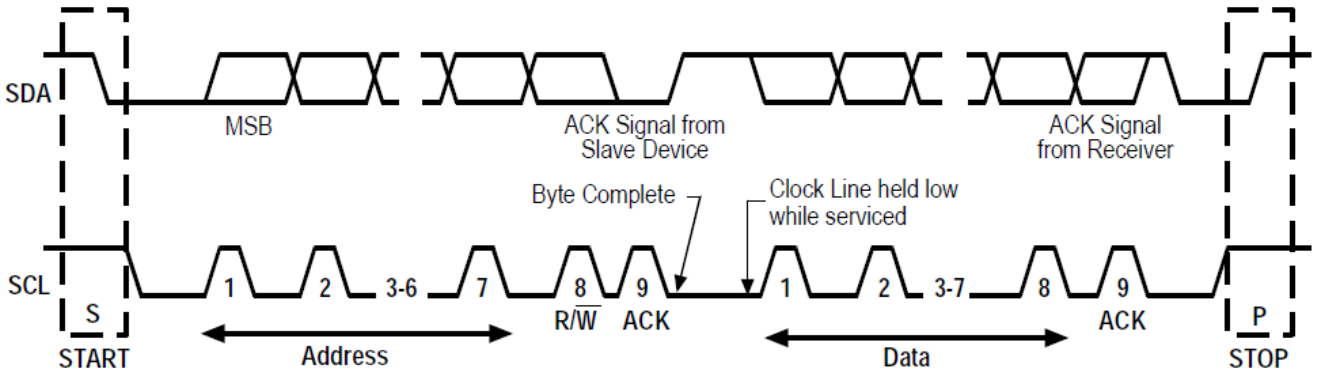
I2C Connection

- Designed for low-cost, medium data rate applications, generally limited to 400Kbps.
- An I2C bus has only two signal wires, SCL and SDA.
- SCL functions as a clock line and SDA can function as a 1-bit serial data line or as a 1-bit serial address line.
- A common ground is of course also required.



I2C Protocol

- I2C devices are either masters or slaves.
- Slaves respond to requests from the master.
- An I2C slave device is assigned a unique 7-bit I2C bus address.
- When a microprocessor (master) needs to communicate with an slave, it sends out a start sequence.
- To stop an I2C sequence, the microprocessor sends out a stop sequence.
- A start sequence is the only time SDA goes high to low while SCL is high and a stop sequence is the only time SDA goes low to high while SCL is high.



I2C Data Format

- Each address and data transfer contains a total of nine bits.
- 8-data bits with an acknowledge bit (ACK) used for handshaking.
- The bits are sent out in high to low order, one per clock.
- Seven address bits are used and the final eighth bit is a read/write (R/W) bit.
- The slave sends the last acknowledge (ACK) bit in each address and data transfer.
- ACK low indicates the slave is ready.

I2C Transaction

- Master issues a START condition (First pulls SDA low, then pulls SCL low)
- Master writes an address to the bus.
  - Plus a bit indicating whether it wants to read or write
  - Slaves that don't match address don't respond
  - A matching slave issues an ACK by pulling down SDA
- Either master or slave transmits one byte
  - Receiver issues an ACK
  - This step may repeat
- Master issues a STOP condition
  - First releases SCL, then releases SDA
  - At this point the bus is free for another transaction

SPI and I2C

- The SPI and I2C standards both provide excellent support for communication with low-speed devices on the same PCB.
- SPI is perhaps a better choice for applications that need to transfer higher bandwidth data streams that do not require address information, such as A/D and D/A converters.
- I2C requires addressing (2-wire bus), so the hardware is more complicated.
- SPI bus is a 3+n wire bus, n is the number of slave devices
- Some general purpose microcontrollers include I2C and SPI interfaces.
- Two or three bits on a GPIO port can be also used with the appropriate software driver to implement an SPI or I2C interface.

I2C on the mbed

- TMP102 temperature sensor TMP102 data sheet
- First send a data byte of 0x01 to specify that the Pointer Register is set to the Configuration Register.
- Send two bytes of data to perform the configuration, which is 0x60A0.
- Then send a data byte of 0x00 to specify that the Pointer Register is set to the Temperature Register.
- Read the 12-bit data in the temperature register.

I2C on the mbed

```
#include "mbed.h"
I2C tempsensor(p9, p10);
Serial pc(USBTX, USBRX);
const int addr = 0x90;
char config_t[3];
char temp_read[2];
float temp;

int main() {
  config_t[0] = 0x01;
  config_t[1] = 0x60;
  config_t[2] = 0xA0;
  tempsensor.write(addr, config_t, 3);
  wait(0.5);
  config_t[0] = 0x00;
  tempsensor.write(addr, config_t, 1);
  while(1) {
    wait(1);
    tempsensor.read(addr, temp_read, 2);
    temp = ((temp_read[0]<<8)+temp_read[1])/256.0;
    pc.printf("Temp = %.2f degC\n\r", temp);
  }
}
```