Remote Sensor Node
Firmware & Hardware Tutorial

Created & Written By
Sargis S Yonan
University of California, Santa Cruz
Jack Baskin School of Engineering

January 2016

Preface: My goal was to create the firmware and hardware for a modular smart household passive and actuating sensing unit. The device, depending on the sensor attached, will push, to a database, to be viewed by home dweller on the internet. The user can also set the node to turn on and off a device.The following instructions will instruct you on how to wire, setup, install, and use a smart sensing node from scratch. The code is open source, and the project will continue with a proprietary PCB controller. **Note:** as of today, only one sensor per node is supported.
Last Revised On: 22 March 2016

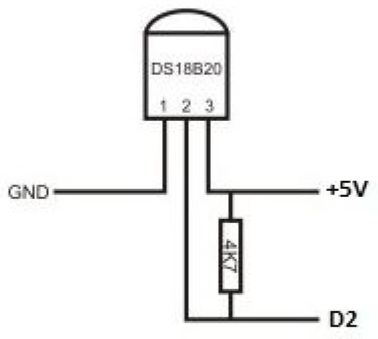
Table of Contents

Setup

DS18B20 One-Wire Temperature Sensor

Connect the red wire of the DS18B20 temperature sensor to +5Volts and a 4.7KΩ resistor in a pull-up resistor configuration bridging the white "data" wire to the +5Volt wire. Then connect the black wire of the sensor to common or ground. Connect the DS18B20 One Wire Sensor's white data wire to PORT C, or PIN 0 on an AVR Microcontroller, which is located:

- D2 → Pin 37 on an Arduino MEGA
- D2 → Pin 23 on an AVR Atmega 328P or an AVR Atmega 88P



```
# only fully supported for the Arduino Mega 2560
# environment variables must be adjusted in the Makefile
$ make temp
```

DHT One-Wire Temperature Sensor

Connect the middle pin of the DHT humidity/temperature sensor to +5Volts and a 4.7KΩ resistor in a pull-up resistor configuration bridging the middle "data" pin to the +5Volt left pin. Then connect the right-most pin of the sensor to common or ground. Connect the DHT One Wire Sensor's data pin to PORT C, or PIN 0 on an AVR Microcontroller, which is located:

- D2 → Pin 37 on an Arduino MEGA
- D2 → Pin 23 on an AVR Atmega 328P or an AVR Atmega 88P

```
# no C implementation is working as of today
# check for Arduino sketch
# only fully supported for the Arduino Mega 2560
# environment variables must be adjusted in the Makefile
$ make dht
```

GY-30 I2C Digital Light Sensor
Check both the datasheet and the avr chip pinout for your specific GY-30 case.

```
# no C implementation is working as of today
# check for Arduino sketch
# only fully supported for the Arduino Mega 2560
# environment variables must be adjusted in the Makefile
$ make light
```

XBee Configuration and Setup - Configure and connect an XBee (IEEE 802.15.4) radio to the USART0 of the AVR microcontroller above.

If an XBIB-U development board is being used to configure the XBee, make sure the XBIB-U USB drivers are properly configured.

Then, using DIGI's XCTU software, make sure that the XBee is configured with:

- Baudrate = 19200
- PAN ID: 0xBEEF
- API MODE = DISABLED (TRANSPARENT MODE)
- ADDRESS = 0x0001 (16-Bit Address), count up by one for multiple nodes

Then Connect the RX/TX pins to the proper USART0 pins on the microcontroller

- Arduino MEGA: DOUT → TX0 (Pin 1), DIN → RX0 (Pin 0)
- AVR Atmega 328P/88P: DOUT → TXD (PCINT16/PD0) Pin 3, DIN → RXD (PCINT17/PD1) Pin 4

Then connect VCC to +3.3Volts (Atmega 328P/88P must use a +5V→ +3.3V voltage regulator, mostly likely with a proprietary power supply if current to the radio is below the datasheet thresholds)

Connect VSS to common or ground

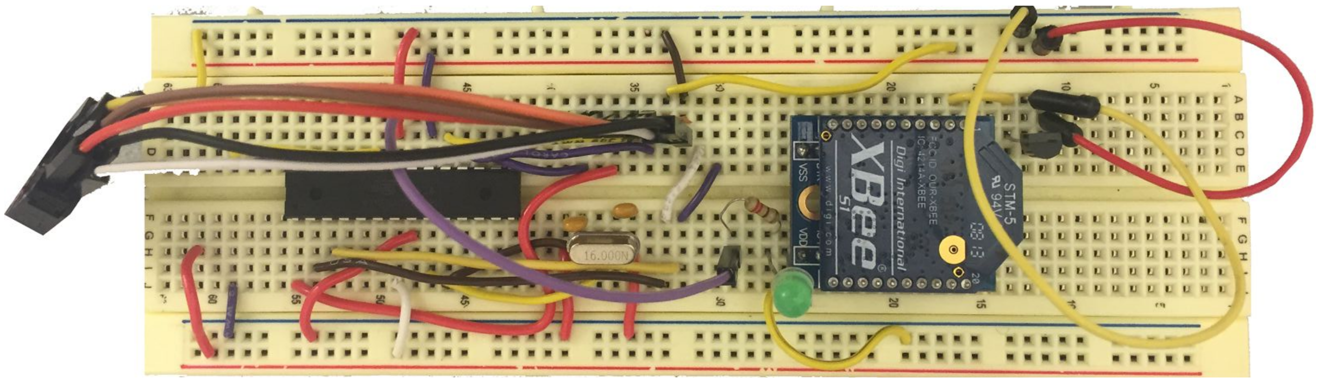
Connect VREF to +3.3Volts

3. **Actuator Setup** - Connect a relay with a ~ >+3.3Volts triggering voltage to PORTB, PIN 5 on the microcontroller.
   - Arduino MEGA: PWM Pin 13
   - AVR Atmega 328P/88P: PB5 - Pin 19

4. **Microcontroller Setup (Optional) ONLY FOR 328P/88P** - wire the Inline Serial Programmer to the microcontroller
   a. connect the ISP's:
      i. MOSI → Pin 17

ii. MISO → Pin 18
iii. /Reset (Active Low Reset Pin) → Pin 1
iv. SCK → Pin 19
v. +5Volts → VCC (ISP) → Pins: 7, 20, and 21
vi. GND → GND (ISP) → Pins 8 and 22
vii. Connect a 16MHz crystal oscillator to XTAL 1 & 2 via Pins 5 and 6 with a 22pF capacitors from each of the crystal's pins to ground or common
viii. In the Makefile:
    1. use AVR-GCC to compile for the given system
    2. use AVR-OBJCPY for ELF→ HEX conversion
    3. use AVR-DUDE to burn program through the ISP → chip
    4. set F_CPU = 16000000
    5. select the proper -p, -P, and -c flags for AVR-DUDE for the given microcontroller and ISP and the proper -mmcu flags for AVR-GCC

Or use the given Makefile in the top level of /src/firmware. The Makefile uses an AVR USBASP ISP, but you can just change the -c flag for AVR-DUDE (**RECOMMENDED)**

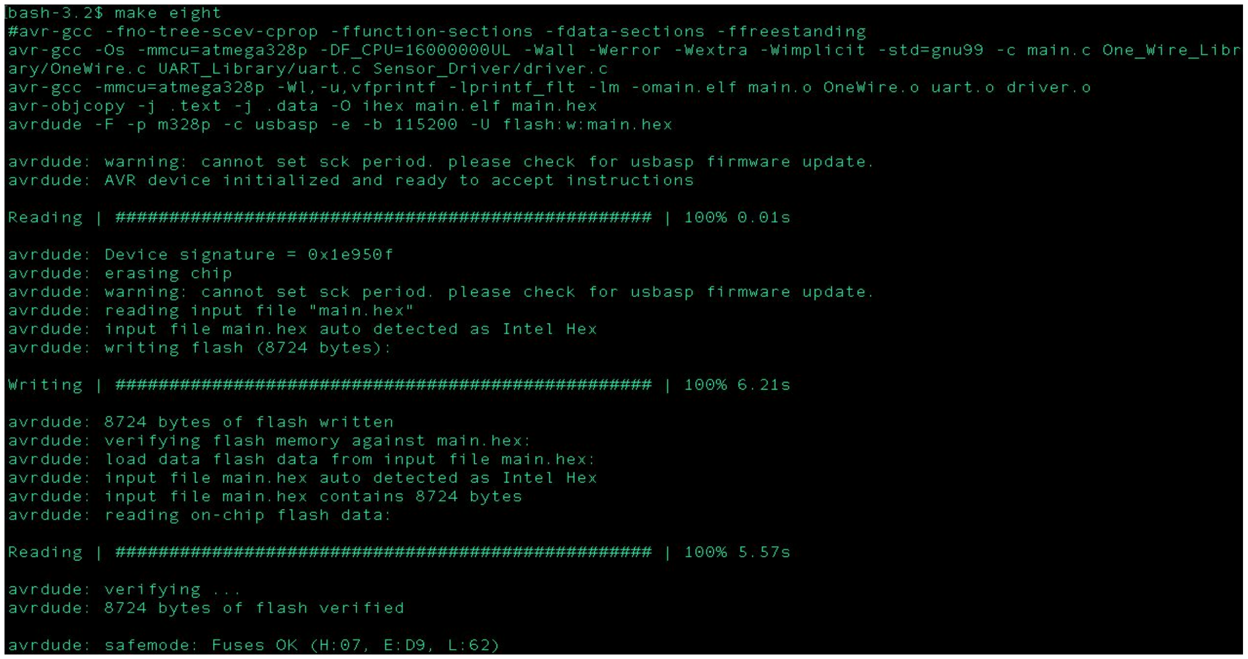


328P Setup Figure: *excuse the ratsnest…*

5. **Downloading The Firmware** - Clone the current repository into a working directory:
```
$ git clone https://github.com/SargisYonan/SensorNode.git
$ cd SensorNode/src/firmware
```

6. **Flashing the Microcontroller** - Connect the ISP or MEGA to the build computer
    - **IMPORTANT**: disconnect the XBee temporarily while writing to the microcontroller. The microcontroller can not be programmed while its RX0/TX0 lines are active.
    Navigate to the project directory in your POSIX terminal, then /src/firmware and type:

- MEGA: change the `COMPILER_PATH` variable in the GNU Makefile to the proper port of the Arduino and type:
  $ `make`
- 328P/88P: **$** `make eight`
- (if using the Makefile in the git repository)



```
bash-3.2$ make eight
#avr-gcc -fno-tree-scev-cprop -ffunction-sections -fdata-sections -ffreestanding
avr-gcc -Os -mmcu=atmega328p -DF_CPU=16000000UL -Wall -Werror -Wextra -Wimplicit -std=gnu99 -c main.c One_Wire_Libr
ary/OneWire.c UART_Library/uart.c Sensor_Driver/driver.c
avr-gcc -mmcu=atmega328p -Wl,-u,vfprintf -lprintf_flt -lm -omain.elf main.o OneWire.o uart.o driver.o
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avrdude -F -p m328p -c usbasp -e -b 115200 -U flash:w:main.hex

avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f
avrdude: erasing chip
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: reading input file "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
avrdude: writing flash (8724 bytes):

Writing | ################################################## | 100% 6.21s

avrdude: 8724 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex auto detected as Intel Hex
avrdude: input file main.hex contains 8724 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 5.57s

avrdude: verifying ...
avrdude: 8724 bytes of flash verified

avrdude: safemode: Fuses OK (H:07, E:D9, L:62)
```

IMPORTANT: If using a microcontroller, 8KB or greater program memory is required, it is strongly suggested that you use the repository's Makefile due to the GCC optimizations necessary to have the .hex file fit into the microcontroller's text field for an 8KB microcontroller.

7. **Possible Commands** - Reconnect the XBee to RX0/TX0. The user should now have a system that behaves a smart sensor node that can be configured and manipulated wirelessly. The commands are sent wirelessly via the XBee radio using either a daemon or for testing purposes, DIGI's XCTU software, with a host XBEE with the same PAN ID as the one attached to the microcontroller, and API MODE enabled. The node can take the following commands each of which is a one byte hex value, followed by a one byte argument (0x00 if the command takes no arguments), followed by the delimiter currently equal to ASCII '-' or hex `0x2D`:

   a. **GET_SENSOR_STATUS (0x53) - ARGUMENT MUST BE 0x00**: microcontroller returns a system status string including the current device on/off status
   b. **ENABLE (0x45) - ARGUMENT MUST BE 0x00**:

enables a continuous stream of GET_SENSOR_VALUE commands in the format "/<SENSOR_READING>/"

c. **DISABLE (0x44) - ARGUMENT MUST BE 0x00**:
Disables the continuous flow of pushing data enabled by **ENABLE()**

d. **GET_SENSOR_STATUS (0x53)** - **ARGUMENT MUST BE 0x00**:
Sends a packet containing the current sensor reading in the format "/<SENSOR_READING>/"

e. **GET_SENSOR_TYPE (0x54)** - **ARGUMENT MUST BE 0x00**:
Returns a packet containing the sensor type in the format "/<SENSOR_TYPE>/"
RETURN VALUES:
- 0x01 - I2C_LIGHT_SENSOR
- 0x02 - ONE_WIRE_TEMP_SENSOR
- 0x03 - DHT_HUM/TEMP_SENSOR

Commands Listing With Returning Values (Acknowledgements)

COMMAND NAME	HEX VALUE	ARGUMENTS (ONE BYTE)	SUCCESS CODE	ERROR CODE
GET_STATUS	0x53	0x00 - const	N/A	0x00
ENABLE	0x45	0x00 - const	0xDA	0x00
DISABLE	0x44	0x00 - const	0xDB	0x00
GET_SENSOR_STATUS	0x53	0x00 - const	0x01	0xF2
GET_SENSOR_TYPE	0x54	0x00 - const	0x01	0xFA

RX_DELIMETER (`0x2D`) or ASCII '-' byte should be placed at the end of each command and argument as the third byte sent

INVALID_COMMAND_ERROR (`0xEF`) - returned in invalid command received

Default Error Code - **PROCESS_COMMAND_ERROR** (`0x00`)

System Initialized message: **SYSTEM_INITIALIZED** (`0x11`) - sent when initialization function completes successfully on startup

Use Case Examples:

Scenario 1: query sensor node for current sensor reading

1. Install and burn the firmware onto the system as described above (**Makefile and Installation on Microcontroller section**)
2. To send the command, send the following string:

```
>> 0x53 0x00 0x2D
OR the three bytes: S0-
```