I2C HDC1080 SENSOR DRIVERS

Dan Blanchette

February 20, 2023

1 Source Code

1.1 hdc1080drivers.h

```c
/**
 * @file hdc1080Drivers.h
 * @author Dan Blanchette
 * @brief HDC1080 Header File
 * @version 0.1
 * @date 2023-02-20
 *
 * @copyright Copyright (c) 2023
 *
 */

#ifndef HDC1080_DRIVERS
#define HDC1080_DRIVERS

#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/binary_info.h"
#include "hardware/i2c.h"

// I2C reserves some addresses for special purposes. We exclude these from the scan.
// These are any addresses of the form 000 0xxx or 111 1xxx

#define HDC1080ADDRESS 0x40 // In the Data Sheet this address identifies the device
#define HDC1080_MANF_DEVICE_ID_REG 0xFE
#define TEMPERATURE 0x00
#define HUMIDITY 0x00
#define CONFIG 0x02
#define SERIAL_FIRST 0xFB
#define SERIAL_MID 0xFC
#define SERIAL_LAST 0xFD
#define DEV_ID 0xFF

#define I2C_PORT i2c1



```

```c
/* ENUMERATED DATA TYPES*/
enum Temp_Type
{
   CEL = 0,
   FAHR = 1
};
enum HDC_Read_Measurements
{
   TEMP_C,
   TEMP_F,
   PERCENT_HUM
};
enum Resolution_Type
{
   HIGH = 14,
   MED = 11,
   LOW = 8
};
enum HDC_Config_Reg
{
   T_OR_H_14R = 0x00, // read the temperature and humidity at 14 bit resolution
   TEMP_11R = 0x004,  // read the temperature at 11 bit resolution
   HUMID_11R = 0x01,  // read the humidity at 11 bit resolution
   HUMID_8R = 0x02,   // read the humidity at 8 bit resolution
   BOTH_14 = 0x10,    // read both the temperature and humdity at 14 bit resolution
   BOTH_11 = 0x15,    // read both the temperature and humdity at 11 bit resolution
   RESET_VAL = 0x10,  // reset the configuration register - this cannot be read
   HEATER_1 = 0x20,   // turn heater on
   HEATER_0 = 0x10    // turn off the heater which is achieved by resetting the config register
};

/*FUNCTIONS*/

// calculate the temperature from the sensor
float temperature(enum Temp_Type, enum Resolution_Type);
// read the unique serial ID
int readSerial1(void);
int readSerial2(void);
int readSerial3(void);
// end serial ID functions


int readDeviceID(void);
void setConfig(enum HDC_Config_Reg);

int readConfig(void); // reads the bits of the config register

//read temps at 14 bit Res
float tempFahr(void);
float tempCels(void);
float calc_humidity(enum Resolution_Type);
#endif
```

1.2 hdc1080.c

```c
/**
 * @file hdc1080.c
 * @author Dan Blanchette
 * @brief Device drivers for the HDC1080 Temperature and Humidity Sensor
 * @version 0.1
 * @date 2023-02-19
 *
 * @copyright Copyright (c) 2023
 *
 * CREDITS: James Lasso and Garett Wells for help with this project.
 *
 */

#include "hdc1080Drivers.h"

int main()
{
  // Enable UART so we can print status output
  stdio_init_all(); // Initialize STD I/O for printing over serial
  // while (!tud_cdc_connected()) { sleep_ms(100);  }
  printf("HDC1080 connected()\n");

  printf("Test Print\n");

  // This example will use I2C1 on the default SDA and SCL pins
  // Parameter 1 specifies the port address for i2c device, this value is measured in HZ and is initilaized to 100,000 or 100Khz
  // Max pico speed is 1Mhz
  i2c_init(I2C_PORT, 100 * 1000);
  gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
  gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
  gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
  gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);
  // Make the I2C pins available to picotool
  bi_decl(bi_2pins_with_func(PICO_DEFAULT_I2C_SDA_PIN, PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C));
  sleep_ms(1000);

  while (1)
  {
    int deviceID, serialID1, serialID2, serialID3, config;
    float tempF, tempC, perH;
    deviceID = readDeviceID();
    serialID1 = readSerial1();
    serialID2 = readSerial2();
    serialID3 = readSerial3();
    config = readConfig();
    tempC = tempCels();
    tempF = tempFahr();
    perH = calc_humidity(HIGH);
    printf("HDC1080_device: ID=0x%X\n", deviceID);
    printf("Unique_serial:  ID_1=0x%X, ID_2=0x%X, ID_3=0x%X\n", serialID1, serialID2, serialID3);
    printf("Config Register: 0x%X\n\n", config);
    printf("Temp Farhenheit: %f\n", tempF);
    printf("Temp Celsius: %f\n\n", tempC);
    printf("Percent Humidity: %f\n\n", perH);
    sleep_ms(1000);
    // RTOS Scheduler() Here
  }

  return 0;
}

/*FUNCTION DEFNITIONS*/

/**
 * @brief
 *
 * @return float
 */
float temperature(enum Temp_Type degrees, enum Resolution_Type resolution)
{
  // points to address 0x00
  const uint8_t TEMP_REGISTER = TEMPERATURE;
  // byte array
  uint8_t data[2];

  // read just the temperature
  if (resolution == HIGH)
  {
    // for high resolution reading 14 bits
    setConfig(T_OR_H_14R);
  }
  else
  {
    // for med resolution reading 11 bits
    setConfig(TEMP_11R);
  }

  // get the reading from the temperature sensor
  int ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &TEMP_REGISTER, 1, false);

  if (resolution == HIGH)
  {
    sleep_ms(9);
  }
  else if (resolution == MED)
  {
    sleep_ms(5);
  }

  ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, data, 2, false);

  int16_t bit_temp_val = data[0] << 8 | data[1];
  double hex_conv = ((double)bit_temp_val) / ((double)65536);
  float final_temp = (hex_conv * 165) - 40;

  if (degrees == CEL)
  {
    return final_temp; // degrees C
  }

  return (final_temp * 1.8) + 32; // degrees F conversion
}

/**
 * @brief
 *
 * @return int
 */
int readDeviceID()
{
  // unsigned integer array that holds the device ID's
  uint8_t deviceID[2];
  int ret;

  /*Assign Register*/
  uint8_t manReg = HDC1080_MANF_DEVICE_ID_REG;

  ret = i2c_write_blocking(I2C_PORT,       // type of port
                   HDC1080ADDRESS, // device address
                   &manReg,     // device's register address to read
                   1,         // expected data size to receive in bytes
                   false        // bool value to tell the I2C controller to: True = use and hold onto the bus, False = release the bus
  );

  ret = i2c_read_blocking(I2C_PORT,    // type of port
                  HDC1080ADDRESS, // device address
                  deviceID,    // pass the uint_8 array to receive the data from the register
                  2,           // expected data size to receive in bytes
                  false        // bool value to tell the I2C controller to: True = use the bus, False = release the bus
  );

  int returnValue = deviceID[0] << 8 | deviceID[1];

  return returnValue;
}

/**
 * @brief This Block of Functions reads and returns the unique serial
 * number from the HDC1080 Device
 *
 * @return int
 */
int readSerial1()
```

```c
{
  uint8_t deviceID[2];
  uint8_t serial1 = SERIAL_FIRST;

  int ret1;
  ret1 = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial1, 1, false);

  ret1 = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2, false);

  int returnVal1 = deviceID[0] << 8 | deviceID[1];

  return returnVal1;
}

int readSerial2()
{
  uint8_t deviceID[2];
  uint8_t serial2 = SERIAL_MID;

  int ret;

  ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial2, 1, false);

  ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2, false);

  int returnVal = deviceID[0] << 8 | deviceID[1];

  return returnVal;
}

int readSerial3()
{
  uint8_t deviceID[2];
  uint8_t serial3 = SERIAL_LAST;

  int ret;
  ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial3, 1, false);

  ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2, false);

  int returnVal = deviceID[0] << 8 | deviceID[1];

  return returnVal;
}

/**
 * @brief Set the configuration for the HDC1080's output for temperature and humidity
 *
 * @param conf_val
 */
```

```c
void setConfig(enum HDC_Config_Reg conf_val)
{
  // CONFIG = 0x02
  const uint8_t configReg = CONFIG;
  uint8_t set[] = {configReg, conf_val, 0x00};

  // write 3 bytes at a time
  int value = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &set[0], 3, false);
}

/**
 * @brief
 *
 * @return int
 */
int readConfig()
{
  int ret;
  uint8_t configOut[2];
  uint8_t config = CONFIG;

  ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &config, 1, false);

  ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, configOut, 2, false);

  int returnVal = configOut[0];

  return returnVal;
}

/**
 * @brief
 *
 * @return float
 */
float tempFahr(void)
{
  return temperature(FAHR, HIGH);
}

/**
 * @brief
 *
 * @return float
 */
float tempCels(void)
{
  return temperature(CEL, HIGH);
}

float calc_humidity(enum Resolution_Type resolution)
{
  // address 0x01
  const uint8_t HUMIDITY_REG = HUMIDITY;
  uint8_t data[2];

  // set config to read humidity
  if (resolution == HIGH)
  {
    setConfig(T_OR_H_14R);
  }
  else if (resolution == MED)
  {
    setConfig(HUMID_11R);
  }
  else
  {
    setConfig(HUMID_8R);
  }

  // write humidity reading to register
  int hum_val = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &HUMIDITY_REG, 1, false);

  if(resolution == HIGH)
  {
    sleep_ms(9);
  }
  else if (resolution == MED)
  {
    sleep_ms(7);
  }
  else
  {
    sleep_ms(5);
  }

  hum_val = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, data, 2, false);

  int16_t bit_val = data[0] << 8 | data[1];
  double hex_conv = ((double) bit_val) / ((double)65536);
  float tot_hum = hex_conv * 100;
  return tot_hum;
}
```