Báo cáo — Giải thích từng dòng code (chi tiết) – AHT10

File AHT10.h

```
#ifndef __AHT10_H
#define __AHT10_H

#include <stdint.h>

void AHT10_Init(void);
uint8_t AHT10_Read(float *temp, float *hum);

#endif
```

Giải thích từng dòng

1. `#ifndef __AHT10_H`
   — Khai báo bắt đầu include-guard: nếu `__AHT10_H` chưa được định nghĩa thì tiếp tục. Ngăn chặn việc include trùng lặp file header.
2. `#define __AHT10_H`
   — Đánh dấu rằng file đã được include (để include-guard hoạt động).
3. `#include <stdint.h>`
   — Kéo các kiểu nguyên thủy cố định kích thước (`uint8_t`, `uint32_t`, ...). Quan trọng để code dùng các kiểu này compile đúng.
4. `void AHT10_Init(void);`
   — Prototype (khai báo) hàm khởi tạo AHT10 (cấu hình GPIO & gửi lệnh init).
5. `uint8_t AHT10_Read(float *temp, float *hum);`
   — Prototype hàm đọc dữ liệu: trả về `1` nếu đọc thành công, `0` nếu thất bại; kết quả trả về qua con trỏ `temp`, `hum`.
6. `#endif`
   — Kết thúc include-guard.

File AHT10.c

```
#include "stm32f10x.h"
#include "aht10.h"

// Chân I2C bit-bang (PB6=SCL, PB7=SDA)
#define SCL_H() GPIOB->BSRR = GPIO_BSRR_BS6
#define SCL_L() GPIOB->BSRR = GPIO_BSRR_BR6
#define SDA_H() GPIOB->BSRR = GPIO_BSRR_BS7
#define SDA_L() GPIOB->BSRR = GPIO_BSRR_BR7
#define SDA_IN()  {GPIOB->CRL &= ~(GPIO_CRL_MODE7); GPIOB->CRL |= GPIO_CRL_CNF7_0;}   // Input floating
#define SDA_OUT() {GPIOB->CRL &= ~(GPIO_CRL_CNF7);  GPIOB->CRL |= GPIO_CRL_MODE7_0;} // Output 10MHz

#define AHT10_ADDR 0x70   // 0x38 << 1

static void delay_us(uint32_t t) {
    for(uint32_t i=0;i<t*8;i++) __NOP();   // chỉnh cho F_CPU=72MHz
}

// I2C bit-banging
void I2C_Delay(void){ delay_us(5); }

void I2C_Start(void){
    SDA_OUT(); SDA_H(); SCL_H(); I2C_Delay();
    SDA_L(); I2C_Delay();
    SCL_L();
}
```

```
void I2C_Stop(void){
    SDA_OUT();
    SCL_L(); SDA_L(); I2C_Delay();
    SCL_H(); I2C_Delay();
    SDA_H(); I2C_Delay();
}
uint8_t I2C_WriteByte(uint8_t data){
    SDA_OUT();
    for(uint8_t i=0;i<8;i++){
        if(data & 0x80) SDA_H(); else SDA_L();
        I2C_Delay();
        SCL_H(); I2C_Delay();
        SCL_L();
        data <<= 1;
    }
    SDA_IN(); I2C_Delay();
    SCL_H(); I2C_Delay();
    uint8_t ack = (GPIOB->IDR & GPIO_IDR_IDR7)?0:1;
    SCL_L();
    SDA_OUT();
    return ack;
}
uint8_t I2C_ReadByte(uint8_t ack){
    uint8_t data=0;
    SDA_IN();
    for(uint8_t i=0;i<8;i++){
```

```
        data <<=1;
        SCL_H(); I2C_Delay();
        if(GPIOB->IDR & GPIO_IDR_IDR7) data |= 0x01;
        SCL_L(); I2C_Delay();
    }
    SDA_OUT();
    if(ack) SDA_L(); else SDA_H();
    SCL_H(); I2C_Delay();
    SCL_L();
    SDA_H();
    return data;
}

// ============ AHT10 DRIVER ============
void AHT10_Init(void){
    // enable clock GPIOB
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
    // PB6, PB7 output open-drain
    GPIOB->CRL &= ~(GPIO_CRL_MODE6 | GPIO_CRL_CNF6 | GPIO_CRL_MODE7 | GPIO_CRL_CNF7);
    GPIOB->CRL |= (GPIO_CRL_MODE6_0 | GPIO_CRL_CNF6_0 | GPIO_CRL_MODE7_0 | GPIO_CRL_CNF7_0);

    // send init
    I2C_Start();
    I2C_WriteByte(AHT10_ADDR);
    I2C_WriteByte(0xE1);
    I2C_WriteByte(0x08);
```

```
    I2C_WriteByte(0x00);
    I2C_Stop();
}

uint8_t AHT10_Read(float *temp, float *hum){
    uint8_t raw[6];
    uint32_t hum_raw, temp_raw;

    // trigger measure
    I2C_Start();
    I2C_WriteByte(AHT10_ADDR);
    I2C_WriteByte(0xAC);
    I2C_WriteByte(0x33);
    I2C_WriteByte(0x00);
    I2C_Stop();

    for(volatile int i=0;i<720000;i++); // ~80ms delay

    // read 6 bytes
    I2C_Start();
    I2C_WriteByte(AHT10_ADDR | 1);
    for(int i=0;i<5;i++) raw[i]=I2C_ReadByte(1);
    raw[5]=I2C_ReadByte(0);
    I2C_Stop();

    if(raw[0] & 0x80) return 0;
```

```
    hum_raw = ((uint32_t)raw[1]<<12)|((uint32_t)raw[2]<<4)|((raw[3]&0xF0)>>4);
    temp_raw= ((uint32_t)(raw[3]&0x0F)<<16)|((uint32_t)raw[4]<<8)|raw[5];

    *hum = hum_raw*100.0/1048576.0;
    *temp= temp_raw*200.0/1048576.0 - 50.0;
    return 1;
}
```

Giải thích từng dòng / từng khối

1. `#include "stm32f10x.h"`
   — Kéo định nghĩa register, macro, cấu trúc cho STM32F1 (CMSIS/StdPeriph header). Cung cấp `GPIOB`, `RCC`, `GPIO_CRL_*`, `GPIO_BSRR_*`, `GPIO_IDR_*` v.v.
2. `#include "aht10.h"`
   — Kéo prototype hàm để đảm bảo tương thích với header.

Khai báo chân/I2C macros

3. `// Chân I2C bit-bang (PB6=SCL, PB7=SDA)`
   — Ghi chú: dùng PB6 làm SCL, PB7 làm SDA (phù hợp với I2C1 default pins).
4. `#define SCL_H() GPIOB->BSRR = GPIO_BSRR_BS6`
   — Khi gọi `SCL_H()` sẽ viết vào thanh ghi `BSRR` để **set** bit 6 (PB6) — làm SCL = 1. Dùng BSRR vì atomic, nhanh.
5. `#define SCL_L() GPIOB->BSRR = GPIO_BSRR_BR6`
   — Viết vào BSRR để **reset** (clear) PB6 → SCL = 0.
6. `#define SDA_H() GPIOB->BSRR = GPIO_BSRR_BS7`
   — Set PB7 (SDA high).
7. `#define SDA_L() GPIOB->BSRR = GPIO_BSRR_BR7`
   — Reset PB7 (SDA low).
8. `#define SDA_IN() {GPIOB->CRL &= ~(GPIO_CRL_MODE7); GPIOB->CRL |= GPIO_CRL_CNF7_0;}`
   — Chuyển PB7 thành **input floating**:
   - `GPIO_CRL_MODE7` clear → MODE = `00` (input).
   - `GPIO_CRL_CNF7_0` set bit CNF0 = `1` → CNF = `01` (floating input).
     (Mục tiêu: để slave có thể kéo SDA xuống khi trả ACK / khi slave muốn truyền.)
9. `#define SDA_OUT() {GPIOB->CRL &= ~(GPIO_CRL_CNF7); GPIOB->CRL |= GPIO_CRL_MODE7_0;}`
   — Chuyển PB7 thành **output** (OUTPUT 10MHz) và CNF=00 (General purpose output

push-pull) — nhưng **IDEALLY** ta muốn open-drain; trong code `AHT10_Init` sau đó cấu hình CNF6_0 / CNF7_0 để tạo open-drain. Ý chính: cho phép ghi SDA khi cần.

Lưu ý: Các macro CRL sử dụng bit-field CMSIS (`GPIO_CRL_MODE7`, `GPIO_CRL_CNF7_0`...). Chúng phụ thuộc header `stm32f10x.h`. Mục đích macro trên: chuyển đổi nhanh SDA giữa chế độ output (khi truyền) và input (khi đọc ACK/bit).

10. `#define AHT10_ADDR 0x70 // 0x38 << 1`
    — Địa chỉ 8-bit dùng khi ghi vào bus: AHT10 7-bit address = 0x38; left-shift 1 → 0x70 (LSB dùng cho R/W).

Delay microsecond

11. `static void delay_us(uint32_t t) { for(uint32_t i=0;i<t*8;i++) __NOP(); }`
    — Delay bằng vòng `__NOP()` nhiều lần; vòng nhân `t*8` là một hiệu chỉnh thô cho MCU chạy ~72MHz (NOP ~1 cycle). **Quan trọng:** cần tinh chỉnh hệ số `8` theo clock hệ thống (HCLK). Đây là delay thô, dùng cho timing bit-bang (không chính xác nhưng đủ).

Hàm hỗ trợ I2C bit-bang

12. `void I2C_Delay(void){ delay_us(5); }`
    — Delay cố định nhỏ dùng giữa các cạnh SCL/SDA để tạo xung I2C.
13. `void I2C_Start(void){ ... }`
    — Tạo **START condition** theo I2C: SDA high & SCL high → SDA kéo thấp khi SCL high → rồi kéo SCL low. Chi tiết:
    - `SDA_OUT()` chuyển SDA thành output, `SDA_H()` + `SCL_H()` đảm bảo bus released level trước; `I2C_Delay()` chờ; `SDA_L()` kéo SDA low → START; `SCL_L()` kéo SCL low để bắt đầu truyền bit.
14. `void I2C_Stop(void){ ... }`
    — Tạo **STOP condition**: kéo SDA low khi SCL low, nâng SCL lên rồi nâng SDA → STOP (SDA rising while SCL high).
15. `uint8_t I2C_WriteByte(uint8_t data){ ... }`
    — Ghi 1 byte qua bit-bang:
    - `SDA_OUT()` đảm bảo ta điều khiển SDA.
    - Lặp 8 lần: kiểm tra MSB (`data & 0x80`) để set/clear SDA, chờ, đưa SCL lên để slave sample bit, hạ SCL, shift dữ liệu left 1.
    - Sau 8 bit: chuyển SDA thành input (`SDA_IN()`) để **nhận ACK** từ slave (slave sẽ kéo SDA xuống nếu ACK).
    - Khi SCL high, đọc `GPIOB->IDR` bit 7 (SDA): nếu `IDR7==0` là ACK (slave kéo xuống). Code: `uint8_t ack = (GPIOB->IDR & GPIO_IDR_IDR7)?0:1;` → trả về `1` khi có ACK (SDA==0), `0` khi NACK (SDA==1).

- Hạ SCL & chuyển SDA thành output trở lại.

16. `uint8_t I2C_ReadByte(uint8_t ack){ ... }`
— Đọc 1 byte:
    - `SDA_IN()` để release SDA (slave sẽ drive SDA).
    - Lặp 8 lần: đưa SCL high, đọc bit SDA, hạ SCL. Kết quả ghép dần vào `data`.
    - Sau khi đọc 8 bit: chuyển SDA thành output để gửi ACK/NACK về slave: nếu `ack`=1 → kéo SDA low (ACK), else để SDA high (NACK). Tạo xung SCL để truyền ACK/NACK. Cuối cùng release SDA bằng `SDA_H()`.

Ghi chú: ACK handling: `ack==1` nghĩa là ta gửi ACK cho slave, thông thường khi đọc nhiều byte bạn ACK tất cả byte trừ byte cuối cùng (gửi NACK). Code gọi `I2C_ReadByte(1)` cho 5 lần, rồi `I2C_ReadByte(0)` cho byte cuối.

Hàm khởi tạo AHT10

17. `void AHT10_Init(void){` — bắt đầu hàm init.
18. `RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;`
— Bật clock cho Port B (cần để truy cập GPIOB registers). Nếu không bật, ghi vào GPIOB vô nghĩa.
19. `GPIOB->CRL &= ~(GPIO_CRL_MODE6 | GPIO_CRL_CNF6 | GPIO_CRL_MODE7 | GPIO_CRL_CNF7);`
— Clear các bit cấu hình CRL cho chân PB6, PB7 (reset cấu hình cũ).
20. `GPIOB->CRL |= (GPIO_CRL_MODE6_0 | GPIO_CRL_CNF6_0 | GPIO_CRL_MODE7_0 | GPIO_CRL_CNF7_0);`
— Set các bit MODEx_0 và CNFx_0 cho PB6/PB7: mục tiêu là **MODE = 01 (Output 10MHz)** và **CNF = 01 (Output open-drain)** cho cả SCL/SDA. Như vậy chân hoạt động ở output open-drain (phù hợp I2C).
— **Chú ý:** các macro `_0` thiết lập bit thấp của field; tác hợp với CMSIS để tạo kiểu cấu hình mong muốn.
21. `I2C_Start(); ... I2C_Stop();` (gửi 0xE1 0x08 0x00)
— Gửi lệnh khởi tạo AHT10 theo datasheet: `0xE1 0x08 0x00`. Lệnh này dùng để thiết lập chế độ hoạt động. Sau lệnh init, cảm biến cần vài chục ms để sẵn sàng.

Hàm đọc AHT10

22. `uint8_t AHT10_Read(float *temp, float *hum){`
— Hàm đọc sensor, lưu kết quả qua con trỏ; trả về `1` khi OK, `0` khi thất bại.
23. `uint8_t raw[6]; uint32_t hum_raw, temp_raw;`
— Mảng `raw[6]` chứa 6 byte trả về từ cảm biến; `hum_raw` và `temp_raw` để ghép bit.
24. `I2C_Start(); I2C_WriteByte(AHT10_ADDR); I2C_WriteByte(0xAC); I2C_WriteByte(0x33); I2C_WriteByte(0x00); I2C_Stop();`

— Gửi lệnh đo `0xAC 0x33 0x00` theo datasheet (số bit trigger). Lệnh này bắt cảm biến bắt đầu đo.

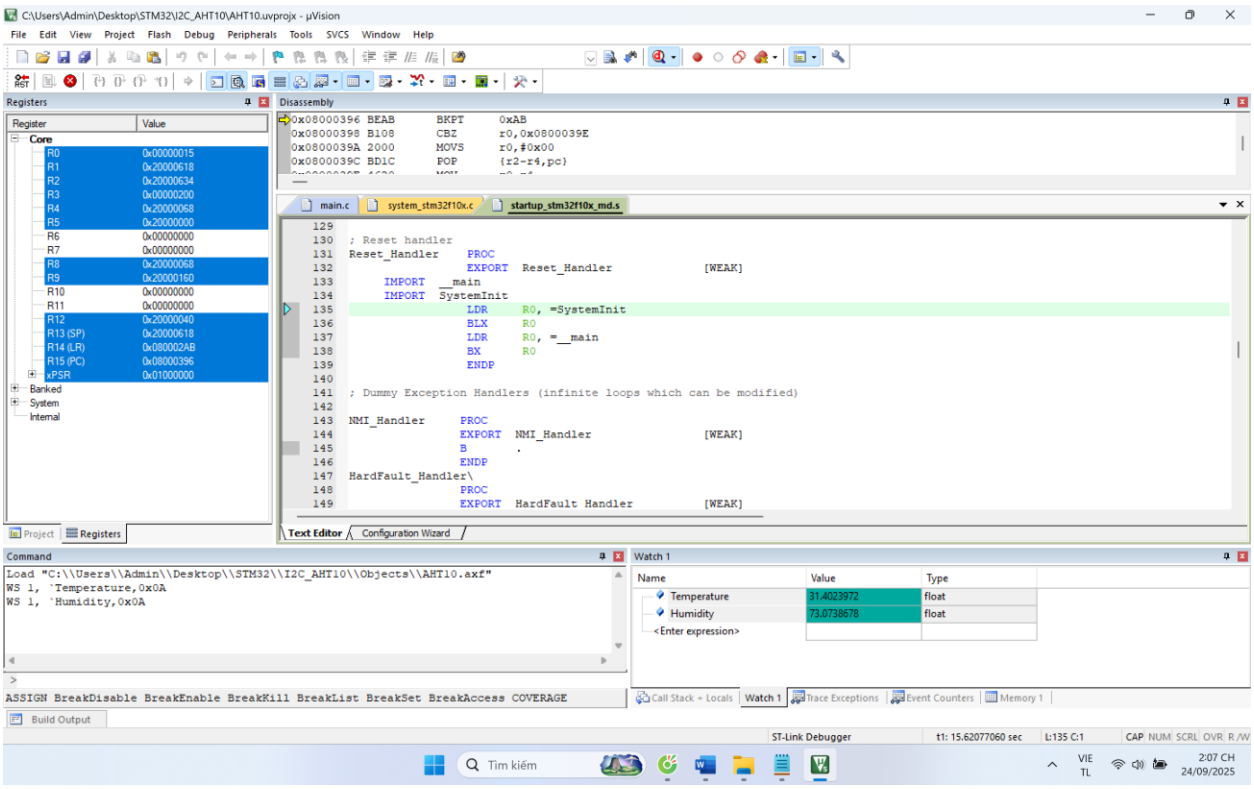
25. `for(volatile int i=0;i<720000;i++); // ~80ms delay`
    — Delay thô bằng vòng lặp để chờ cảm biến đo xong (~80ms). Bạn có thể thay bằng `delay_ms(80)` nếu có hàm SysTick. **Quan trọng:** thời gian này phải ≥ thời gian đo của AHT10.
26. `I2C_Start(); I2C_WriteByte(AHT10_ADDR | 1);`
    — Giáo tiếp đọc: gửi address với bit R/W = 1 (tức `AHT10_ADDR | 1` → đọc).
27. `for(int i=0;i<5;i++) raw[i]=I2C_ReadByte(1); raw[5]=I2C_ReadByte(0);`
    — Đọc 6 byte: gửi ACK sau 5 byte đầu để tiếp tục, NACK cho byte cuối để báo cho slave dừng.
28. `I2C_Stop();`
    — STOP để kết thúc giao tiếp.
29. `if(raw[0] & 0x80) return 0;`
    — Kiểm tra *busy bit* trong byte 0 (bit 7): nếu =1 nghĩa sensor vẫn busy → báo lỗi/không thành công (return 0).
30. `hum_raw = ((uint32_t)raw[1]<<12)|((uint32_t)raw[2]<<4)|((raw[3]&0xF0)>>4);`
    — Ghép 20-bit độ ẩm: raw[1] (MSB) <<12, raw[2]<<4, high nibble raw[3] >>4.
31. `temp_raw= ((uint32_t)(raw[3]&0x0F)<<16)|((uint32_t)raw[4]<<8)|raw[5];`
    — Ghép 20-bit nhiệt độ: low nibble raw[3] (4bit) <<16, raw[4]<<8, raw[5].
32. `*hum = hum_raw*100.0/1048576.0;`
    — Chuyển sang %RH theo datasheet: hum = hum_raw / 2^20 * 100.
33. `*temp= temp_raw*200.0/1048576.0 - 50.0;`
    — Chuyển sang °C: temp = temp_raw / 2^20 * 200 − 50.
34. `return 1;`
    — Trả về thành công.

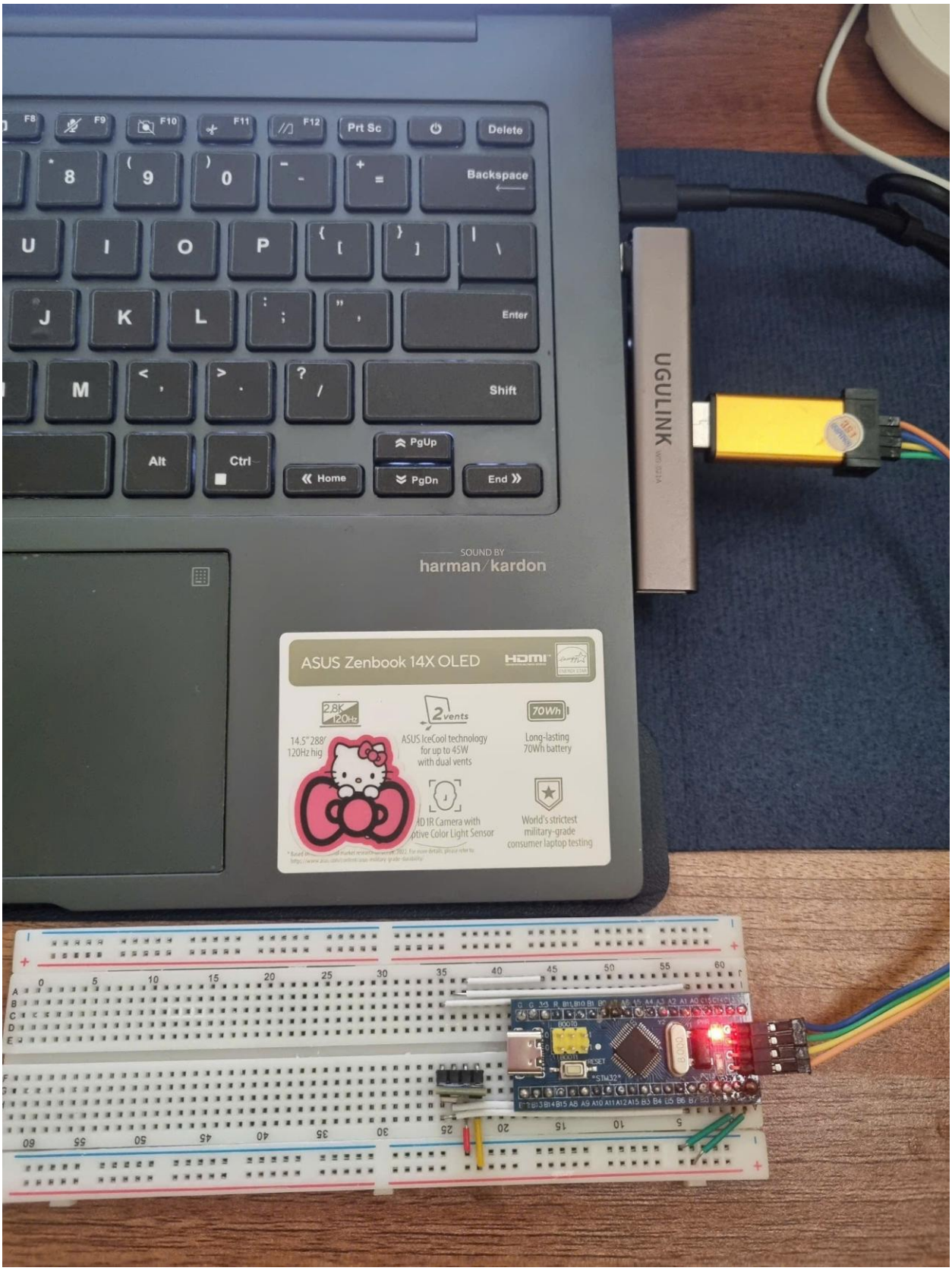
File : Main.c

```
#include "stm32f10x.h"

#include "aht10.h"



float Temperature, Humidity;



int main(void){

   AHT10_Init();

   while(1){

      if(AHT10_Read(&Temperature,&Humidity)){

         // xem Temperature, Humidity trong Watch Window

      }

      for(volatile int d=0; d<720000; d++); // delay ~100ms

   }

}
```

Giải thích từng dòng

1. `#include "stm32f10x.h"`
   — Như trên: register/CMSIS for STM32F1.
2. `#include "aht10.h"`
   — Lấy prototype hàm AHT10.
3. `float Temperature, Humidity;`
   — Biến toàn cục để lưu kết quả. **Quan trọng:** vì là global, Keil Watch Window có thể hiển thị chúng khi debug.
4. `int main(void){` — Hàm chính.
5. `AHT10_Init();`
   — Gọi hàm khởi tạo I2C bit-bang + gửi lệnh init tới cảm biến.
6. `while(1){` — Vòng lặp vô hạn.
7. `if(AHT10_Read(&Temperature,&Humidity)){ /*...*/ }`
   — Gọi hàm đọc; nếu trả về 1 (OK) thì `Temperature`/`Humidity` được cập nhật. Bạn có thể mở Watch Window trong Keil để quan sát 2 biến này thay đổi.
8. `for(volatile int d=0; d<720000; d++); // delay`
   — Delay thô giữa các lần đo (giảm tốc độ polling). Có thể thay bằng `delay_ms(1000)` chính xác hơn.
9. `}` — đóng vòng while.
10. `}` — đóng main


C:\Users\Admin\Desktop\STM32\I2C_AHT10\AHT10.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x00000015
R1 0x20000618
R2 0x20000634
R3 0x00000200
R4 0x20000068
R5 0x20000000
R6 0x00000000
R7 0x00000000
R8 0x20000068
R9 0x20000160
R10 0x00000000
R11 0x00000000
R12 0x20000040
R13 (SP) 0x20000618
R14 (LR) 0x080002AB
R15 (PC) 0x08000396
xPSR 0x01000000
Banked
System
Internal
Project Registers
Disassembly
0x08000396 BEAB BKPT 0xAB
0x08000398 B108 CBZ r0,0x0800039E
0x0800039A 2000 MOVS r0,#0x00
0x0800039C BD1C POP {r2-r4,pc}
main.c system_stm32f10x.c startup_stm32f10x_md.s
129
130 ; Reset handler
131 Reset_Handler PROC
132 EXPORT Reset_Handler [WEAK]
133 IMPORT __main
134 IMPORT SystemInit
135 LDR R0, =SystemInit
136 BLX R0
137 LDR R0, =__main
138 BX R0
139 ENDP
140
141 ; Dummy Exception Handlers (infinite loops which can be modified)
142
143 NMI_Handler PROC
144 EXPORT NMI_Handler [WEAK]
145 B .
146 ENDP
147 HardFault_Handler\
148 PROC
149 EXPORT HardFault Handler [WEAK]
Text Editor Configuration Wizard
Command
Load "C:\\Users\\Admin\\Desktop\\STM32\\I2C_AHT10\\Objects\\AHT10.axf"
WS 1, `Temperature,0x0A
WS 1, `Humidity,0x0A
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
Watch 1
Name Value Type
Temperature 31.4023972 float
Humidity 73.0738678 float
<Enter expression>
Call Stack + Locals Watch 1 Trace Exceptions Event Counters Memory 1
Build Output
ST-Link Debugger t1: 15.62077060 sec L:135 C:1 CAP NUM SCRL OVR R /W
Tìm kiếm
VIE TL
2:07 CH 24/09/2025


F8
F9
F10
F11
F12
Prt Sc
Delete
Backspace
Enter
Shift
Alt
Ctrl
PgUp
PgDn
Home
End
SOUND BY
harman/kardon
ASUS Zenbook 14X OLED
HDMI
2.8K 120Hz
14.5" 288
120Hz hig
2 vents
ASUS IceCool technology
for up to 45W
with dual vents
70Wh
Long-lasting
70Wh battery
IR Camera with
ptive Color Light Sensor
World's strictest
military-grade
consumer laptop testing
UGULINK
BOOT0
RESET
STM32