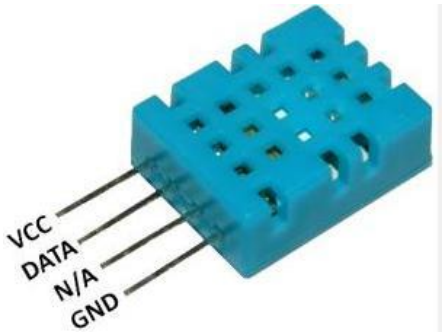
Температурний датчик DH11

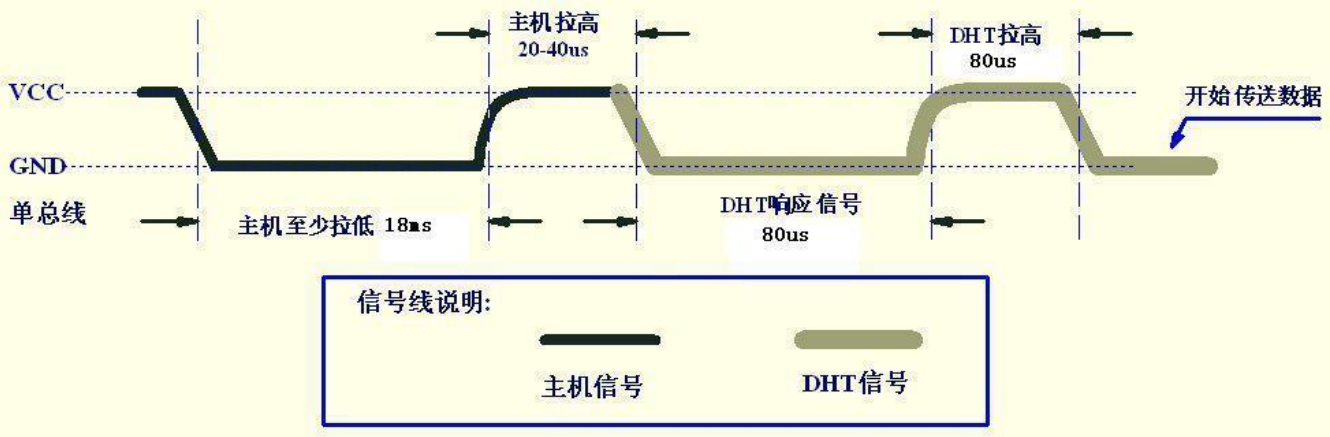
DHT11 - це композитний датчик температури і вологості з відкаліброваним цифровим вихідним сигналом, в якому використовується спеціальна технологія збору даних з цифровим модулем і технологія вимірювання температури і вологості, що забезпечує надзвичайно високу надійність і чудову довгострокову стабільність продукту. Датчик включає в себе резистивний чутливий елемент вологості і елемент вимірювання температури NTC і з'єднаний з високопродуктивним 8-бітовим однокристальним мікрокомп'ютером. Отже, продукт має відмінну якість, надшвидкий відгук, здібних до перешкод Сильні, економічні та інші переваги. Надзвичайно компактний розмір і низьке енергоспоживання роблять його кращим вибором для такого типу додатків і навіть для самих вимогливих додатків. Продукт являє собою 4-контактний однорядний штировий корпус для легкого підключення.



Пін	**Ім’я**	**Коментар**
1	VDD	Джерело живлення 3-5,5 В постійного струму
2	DATA	Серійні дані
3	NC	Не використовується, але потрібна підтяжка до живлення
4	GND	Земля

Дані передаються в пакеті 40 біт даних за один раз = 8 біт цілочисельних даних вологості + 8 біт десяткових даних вологості + 8 біт цілочисельних даних температури + 8 біт десяткових даних температури + 8 біт контрольної суми. Коли дані передаються правильно, дані контрольної суми рівні останнім 8 бітам результату «8-бітові цілочисельні дані про вологість + 8-ми десяткові дані про вологість + 8-бітові цілочисельні дані про температуру + 8-бітові десяткові дані про температуру».

Для запуску сенсору необхідно перевести шину у високий стан. Послідовність запуску складається із підтягнення піна до землі на 18 мс, після чого підтягнення до живлення на час 20-40 мкс. Як результат запуску сенсор кидає старт тест з’єднання, що складається із підтягнення шини до замлі та до живлення на 80 мкс послідовно.



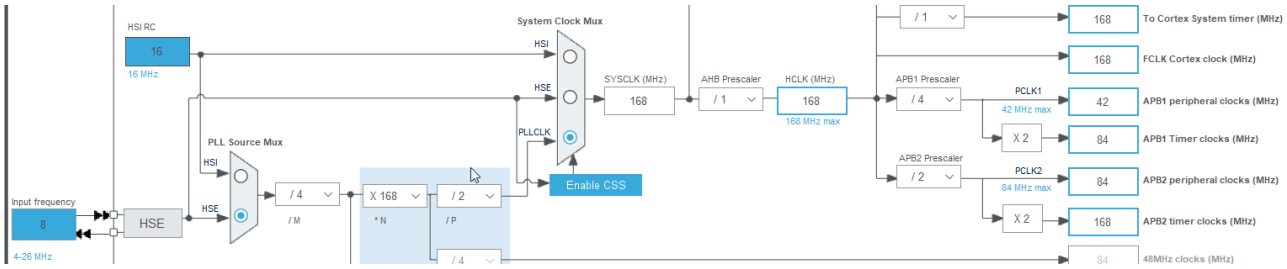
Після цього йдуть дані. Кожен біт даних починається із підтягнення шини до землі на 50 мкс, після чого йде підтягнення до живлення на 26-28 мкс для передачі 0, або на 80 мкс для передачі 1.

Після закінчення передачі датчик витримує низький рівень на протязі 50 мкс.

Програмування мікроконтролера

Як бачимо, в програмі необхідні очікування коротші за мікросекунди. Для очікувань необхідно налаштувати таймер. На контролер STM32F407 містить 14 таймерів, серед яких 2 загального використання. Таймери загального використання – це таймери без додаткового функціоналу, який містить лише базові функції таймерів. В мікроконтролері це TIM6 та TIM7.

Створимо базовий проект для мікроконтролера, налаштуємо тактування із зовнішнього кварца. Відповідно до таблиці шин, можна побачити, що таймери загального призначення знаходяться на шині APB1. Відповідно до схеми тактування отже частота роботи таймера можлива максимум 84 MHz.



Для налаштування таймера поставимо галочку “Activated” для TIM6. Таймери загального призначення мають наступні параметри:

- Prescaler – подільник системної частоти таймера.
- Counter Period – лічильник переповнення після якого спрацює переривання оновлення таймера.

Для старту функції використовують функцію `HAL_TIM_Base_Start_IT()` параметр якої передається об’єкт таймера. Для отримання поточної кількості тіків таймера необхідно використати функцію `__HAL_TIM_GetCounter()`

Напишемо модуль затримок програми, що базується на таймері:

```
#include "delay.h"

static TIM_HandleTypeDef* objSP_delay_timer;
```

```
void delay_init(TIM_HandleTypeDef* objLP_timer)
{
  objSP_delay_timer = objLP_timer;
  HAL_TIM_Base_Start(objSP_delay_timer);
}

void delay_ms(uint32_t u32L_time_ms)
{
  for (uint32_t u32L_i = 0; u32L_i < u32L_time_ms; ++u32L_i)
  {
    delay_us(1000);
  }
}

void delay_us(uint32_t u32L_time_us)
{
  uint32_t u32_crnt_time = __HAL_TIM_GetCounter(objSP_delay_timer);
  while (__HAL_TIM_GetCounter(objSP_delay_timer) - u32_crnt_time < u32L_time_us);
}
```

Розпочнемо реалізацію модуля роботи з Напишемо функцію для ініціалізації роботи з сенсором. За допомогою цих функцій пін налаштовується на вхід чи вихід.

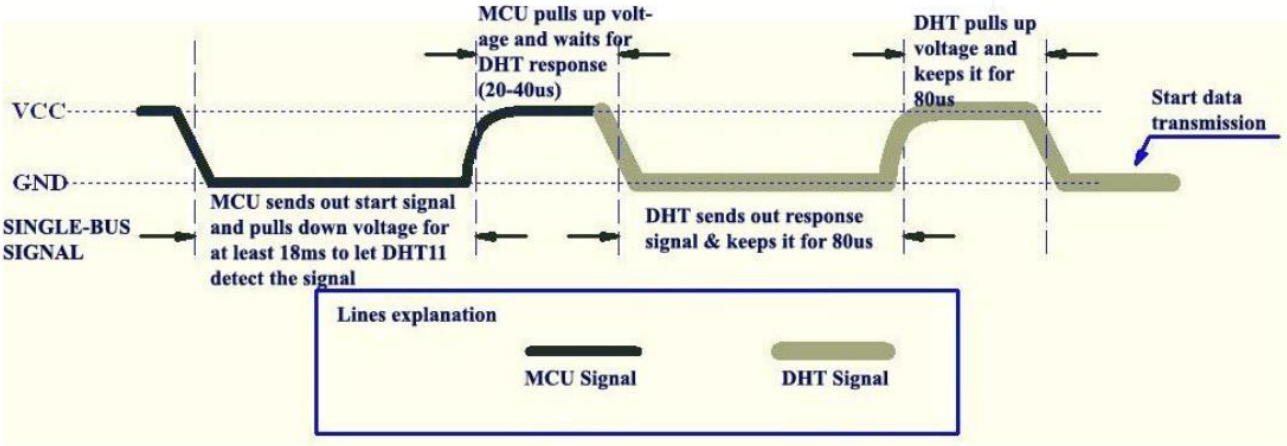
```
void DHT11_config_to_in(void)
{
  GPIO_InitTypeDef objL_init_struct;

  objL_init_struct.Pin  = u16L_sensor_pin;
  objL_init_struct.Mode = GPIO_MODE_INPUT;
  objL_init_struct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(objSP_sensor_port, &objL_init_struct);
}

void DHT11_config_to_out(void)
{
  GPIO_InitTypeDef objL_init_struct;

  HAL_GPIO_WritePin(objSP_sensor_port, u16L_sensor_pin, GPIO_PIN_SET);

  objL_init_struct.Pin   = u16L_sensor_pin;
  objL_init_struct.Mode  = GPIO_MODE_OUTPUT_PP;
  objL_init_struct.Pull  = GPIO_PULLUP;
  objL_init_struct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  HAL_GPIO_Init(objSP_sensor_port, &objL_init_struct);
}
```

Кожен модуль повинен мати деяку функцію ініціалізації, тож для модуля температурного датчика вона матиме наступний вигляд:

```
void DHT11_init(GPIO_TypeDef* objL_port, uint16_t u16L_pin)
{
  objSP_sensor_port = objL_port;
  u16L_sensor_pin   = u16L_pin;
  ФУНКЦІЯ ВВІМКНЕННЯ ТАКТУВАННЯ ПОРТА
}
```

Одним із ключових моментів не варто забувати про тактування. Без тактування робота з пінами не має сенсу. Наприклад, для тактування **portD** використовують функцію **__HAL_RCC_GPIOC_CLK_ENABLE();**

На старті роботи необхідно пін налаштувати на вихід, подати низький рівень, утримати 20 мілісекунд, підняти до живлення та утримати на протязі 30 мікросекунд. Ці процеси виконуються за допомогою наступної функції.



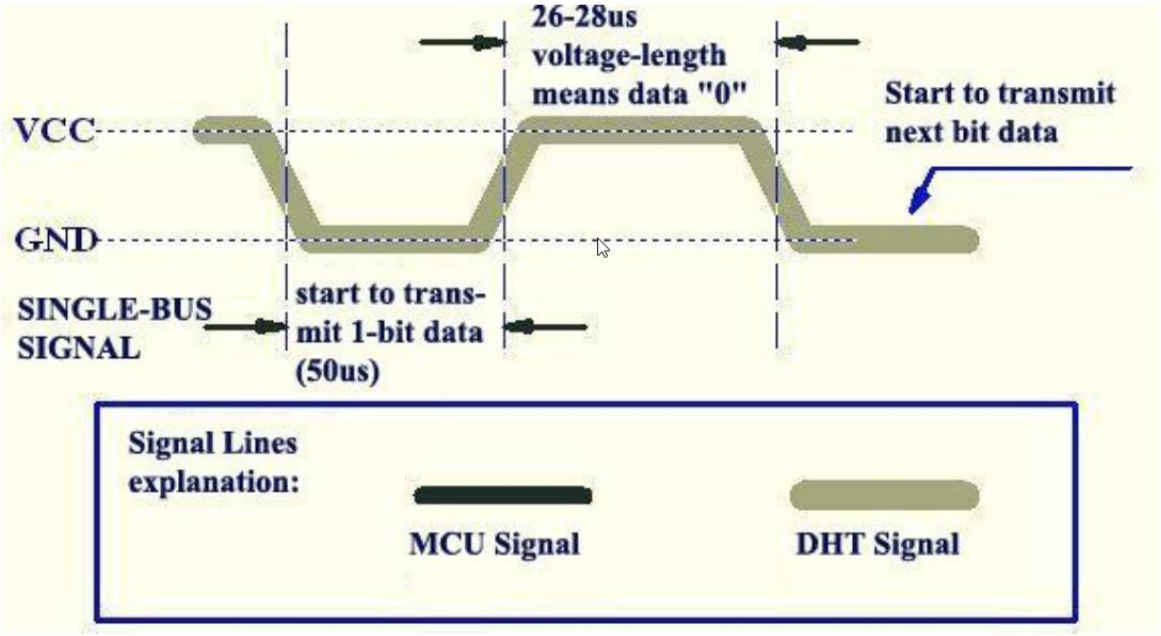

```
void DHT11_Rst(void)
{
  HAL_GPIO_WritePin(objSP_sensor_port, u16L_sensor_pin, GPIO_PIN_RESET);
  delay_ms(20);
  HAL_GPIO_WritePin(objSP_sensor_port, u16L_sensor_pin, GPIO_PIN_SET);
  delay_us(30);
}
```

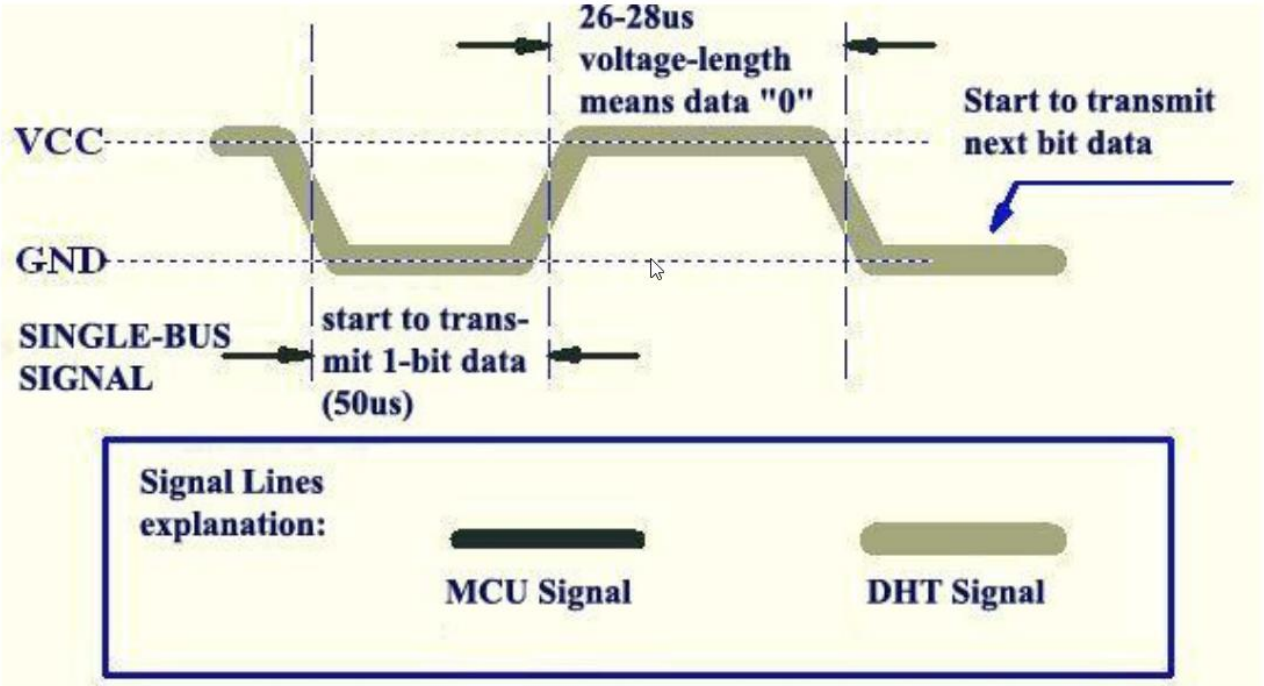
Та перевірити чи сенсор розпочав роботу. Відповідно до специфікації, дані DHT11 надсилає не частіше ніж 1 раз в секунду. Проте практично перевірено, що сенсору потрібно більше ніж 1с, а точніше ~1.2 секунди.

```
uint8_t DHT11_Check(void)
{
  uint16_t u16L_retry = 0;
  DHT11_config_to_in();
  while ((HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin) == GPIO_PIN_RESET) &&
u16L_retry < 100)
  {
    u16L_retry++;
    delay_us(1);
  };

  if(u16L_retry >= 100)
  {
    return 1;
  }

  u16L_retry = 0;

  while ((HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin) == GPIO_PIN_SET) &&
u16L_retry < 100)
  {
    u16L_retry++;
    delay_us(1);
  };

  if(u16L_retry >= 100)
  {
    return 1;
```

```
  }

  return 0;
}
```

Після цього датчик починає надсилання даних. Для цього напишемо наступні функції для читання даних.





```
uint8_t DHT11_Read_Bit(void)
{
  uint16_t u16L_retry = 0;

  while ((HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin) == GPIO_PIN_RESET) &&
u16L_retry < 60)
  {
    u16L_retry++;
    delay_us(1);
  }

  if (u16L_retry > 60)
  {
    return 0xFF;
  }

  u16L_retry = 0;
  while ((HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin) == GPIO_PIN_SET) &&
u16L_retry < 80)
  {
    u16L_retry++;
    delay_us(1);
  }

  return u16L_retry < 30 ? 0 : 1;
}

uint8_t DHT11_Read_Byte(void)
{
    uint8_t u8L_data;
    u8L_data = 0;
  for (uint8_t u8L_i = 0; u8L_i < 8; u8L_i++)
  {
    u8L_data <<= 1;
       u8L_data |= HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin);
    }

    return u8L_data;
}

uint8_t DHT11_Read_Data(uint8_t* u8P_temp, uint8_t* u8P_humi)
{
  uint8_t u8PL_buf[5];
  uint16_t u16L_retry = 0;

  DHT11_config_to_out();
  DHT11_Rst();

  if (DHT11_Check() != 0)
  {
    return 1;
  }

  for (uint8_t u8_i = 0; u8_i < 5; u8_i++)
  {
    u8PL_buf[u8_i] = DHT11_Read_Byte();
  }

  bool bL_has_error = true;
```

```
  if((u8PL_buf[0] + u8PL_buf[1] + u8PL_buf[2] + u8PL_buf[3]) == u8PL_buf[4])
  {
    *u8P_humi = u8PL_buf[0];
    *u8P_temp = u8PL_buf[2];
    bL_has_error = false;
  }

  if (!bL_has_error)
  {
    while ((HAL_GPIO_ReadPin(objSP_sensor_port, u16L_sensor_pin) == GPIO_PIN_RESET) &&
u16L_retry < 60)
    {
      u16L_retry++;
      delay_us(1);
    }
    if (u16L_retry > 60)
    {
      bL_has_error = true;
    }
  }

  return bL_has_error ? 1 : 0;
}
```