REPORT

Εργαστήριο Μικροεπεξεργαστές και Περιφερειακά

ΓΚΟΥΜΑ ΒΑΣΙΛΙΚΗ 9755

ΚΩΣΤΑΣ ΑΝΔΡΟΝΙΚΟΣ 9754

Περιεχόμενα

Εισαγωγή

Στο συγκεκριμένο παραδοτέο χρησιμοποιούμε τον αισθητήρα DHT11 (αισθητήρας θερμοκρασίας και υγρασίας) για να μετρήσουμε την θερμοκρασία με χρήση ISR καθώς και να τυπώνουμε την θερμοκρασία με διαφορετική περίοδο από την περίοδο που διαβάζουμε τον αισθητήρα. Για τη χρήση του συγκεκριμένου αισθητήρα χρειάστηκε η υλοποίηση των αντίστοιχων drivers. Αρχικά θα περιγραφούν οι συναρτήσεις που είναι απαραίτητες για να λάβουμε δεδομένα από τον αισθητήρα και έπειτα η λογική του κύριου προγράμματος του εργαστηρίου

DHT DRIVERS

Υλοποίηση συνάρτησης *start_signal()*

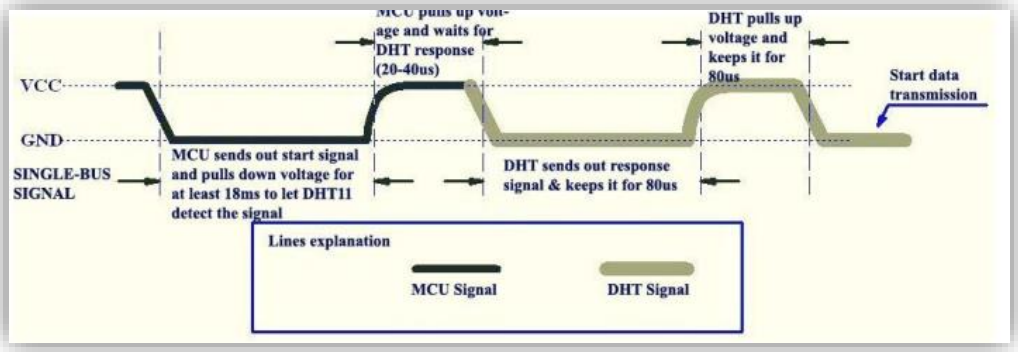
Η επικοινωνία μεταξύ του μικροελεγκτή και του αισθητήρα DHT11 ξεκινάει όταν θέλουμε να λάβουμε δεδομένα από τον αισθητήρα. Για να επιτευχθεί αυτό πρέπει να ενημερώσουμε τον αισθητήρα το οποίο σύμφωνα με το datasheet πρέπει να τεθεί το voltage level από HIGH σε LOW για 20ms της γραμμής μεταφοράς δεδομένων. Έπειτα τον θέτουμε σε HIGH και περιμένουμε 30us (μεταξύ 20 και 40us) για την απάντηση του.

```
////////////////////////////////////////////////////////////////////////////
void start_signal(void){
  gpio_set_mode(DHT_11_Pin, Output);
  // Set the DHT LOW for 20ms to ensure DHT's detection of MCU's signal
  gpio_set(DHT_11_Pin, 0);
  delay_ms(20);

  // Set the DHT HIGH and wait 30us (20-40us) for DHT's response
  gpio_set(DHT_11_Pin, 1);
  delay_us(30);
}
```

Υλοποίηση συνάρτησης *check_response()*

Όταν ο DHT ανιχνεύσει το σήμα εισόδου στέλνει ένα σήμα χαμηλής τάσης για περίπου 80 us και μετά ένα σήμα υψηλής τάσης για 80us όπως φαίνεται στην Εικόνα 1. Στη συγκεκριμένη συνάρτηση εξετάζουμε αν αυτό επιτελείται.

```
uint8_t check_response(void){
  // keep the time that we are waiting for something to happen //
  uint8_t time = 0;
  // Set the DHT Pin as an Input //
  gpio_set_mode(DHT_11_Pin, Input);

  while (gpio_get(DHT_11_Pin) && (time < 100)){
    time++;
    // sampling of the state of the DHT with lus
    delay_us(1);
  }

  /*
  While ends in the following cases : 1) DHT LOW or 2) time >= 100

    1) DHT on LOW and time < 100   // time = time that DHT starts the response
    2) time >= 100 and DHT on HIGH // never got response from the sensor

  */
  if(time >= 100)
    // 2) error case
    return 1;
  else
    // 1) // correct case
    // we go the time to zero to measure the width of the response in time to check if it is correct ( 80us )
    time = 0;

  while(!gpio_get(DHT_11_Pin) && time < 100){
    time++;
    // sampling of the state of the DHT with lus
    delay_us(1);
  }
    /*
    While ends in the following cases : 1) DHT HIGH or 2) time >= 100
    1) DHT on HIGH and time < 100      // time = time that DHT ends the response cause it becomes HIGH
    2) time >= 100 and DHT on LOW      // never ends the respose ( must end in at 80us in normal conditions )
    */
  if(time >= 100)
     // 2) error case
    return 1;
  else
    // 1) works fine
    return 0; //
```



Εικόνα 1 : σωστή ανταπόκριση του DHT

Υλοποίηση συνάρτησης *readBit()*

Εφόσον όλα λειτουργούν σωστά τώρα μπορούμε να διαβάσουμε δεδομένα. Πριν ο αισθητήρας από κάθε bit πληροφορίας, στέλνει το 0 για 50us ως start bit . Έπειτα έχουμε υψηλή τάση όπου αν είναι για διάρκεια 70 us παίρν1 ενώ αν είναι μεταξύ 26 - 28us παίρνουμε 0,όπως φαίνεται και στις παρακάτω εικόνες .Στην συγκεκριμένη συνάρτηση περιμένουμε 40us και ανάλογα με το αν έχουμε υψηλή ή χαμηλή τάση την στιγμή 40 us τότε έχουμε 1 ή 0 αντίστοιχα.

```
uint8_t readBit(void){

  while(gpio_get(DHT_11_Pin))
  {
    // sampling of the state of the DHT with 1us
    delay_us(1);
  }
  /*
    While ends when becomes LOW
  */
  // Here started the 50us low-voltage-level signal (something like start bit)
  while(!gpio_get(DHT_11_Pin))
  {
    // sampling of the state of the DHT with 1us
    delay_us(1);
  }
  /*
    While ends when becomes HIGH ( the 50us came to its end so we can start to read )
  */

  // Reading Technique //

  // wait 40us and then read the DHT //

  delay_us(40);

  // if it is still HIGH after 40us we are in the range of (40 - 70us ), so the bit is 1
  // else if it is not HIGH, that means that the DHT has started the next 50us low-voltage-level pulse, so the bit is 0

  if(gpio_get(DHT_11_Pin))
    return 1;
  else
    return 0;

  // Reading Technique //
}
```

Υλοποίηση συνάρτησης *readByte()*

Αποθηκεύονται σε μία μεταβλητή byte τα bit που διαβάσαμε με την συνάρτηση *readBit()* .

```
uint8_t readByte(void){
  uint8_t byte = 0x0;
  // A loop to read 8 bit continuously
  for(int i = 0; i < 8; i++){
    // The sensor sends higher data bit first, so we left shift the bits each time
    byte <<= 1;
    byte |= readBit();


    // BITWISE OR
    // cases :
    // 1) 0bxxxxxxx0 | 0b00000000 = 0bxxxxxxx0
    // 2) 0bxxxxxxx0 | 0b00000001 = 0bxxxxxxx1
    // we have two cases because each time we perform left shifting
  }
  return byte;
}
```

Υλοποίηση συνάρτησης *get_packet()*

Αποθηκεύουμε σε έναν πίνακα τα byte με τα δεδομένα για θερμοκρασία και υγρασία που πήραμε από τον αισθητήρα.

```
uint8_t * get_packet(void){

  // static to retain its memory after the execution of the function and to be processed in the main
  static uint8_t sensor_data[5];
  // read 5 bytes that form a complete packet

  for(uint8_t i = 0; i < 5; i++)
    sensor_data[i] = readByte();

  // sensor_data[0] = integral RH
  // sensor_data[1] = decimal RH data
  // sensor_data[2] = integral T data
  // sensor_data[3] = decimal T
  // sensor_data[4] = check sum

  return sensor_data;
}
```

Υλοποίηση συνάρτησης *check_sum*()

Ελέγχουμε ότι στάλθηκαν σωστά τα δεδομένα σύμφωνα με το datasheet.

```
uint8_t check_sum(uint8_t *packet)
{
  if(packet[0] + packet[1] + packet[2] + packet[3] == packet[4])
    return OK;
  else
    return ERROR;
}
```

Main Program

Δηλώσεις και Αρχικοποιήσεις Μεταβλητών

```
static uint8_t packet[5];
static uint8_t check_sum_var = OK;
static char _temp_[2];
static uint8_t temperature = 0;

static uint8_t button_pressed = 0;
static uint8_t AEM[5];
static uint8_t AEM_SUM = 0;
static uint8_t aem_index = 0;

// The counter for the temperature measurement

static uint8_t temperature_timing_counter = 0;
static uint8_t print_timing_counter = 0;
static uint8_t PRINT_FREQUENCY = 0;
static char refresh_rate[2];

// ISR to process the user's AEM
void AEM_ISR(uint8_t uart_byte);
// ISR to process the temperature
void timer_ISR(void);
// LED ISRs
// called when temperature > 25
void led_on(void);
// called when temperature < 20
void led_off(void);
// called when temperature  = [20, 25]
void toggle_led(void);
// Init the USER Button
void BUTTON_INIT(void);
// Called when the button is pressed
void BUTTON_ISR(void);
```

UART Communication

Αρχικοποιούμε την επικοινωνία στα 115200bps και θέτουμε τον επεξεργαστή να είναι σε θέση να ακούσει το interrupt από τον UART transceiver μέσω της *uart_set_rx_callback()*

```
uart_init(115200);
uart_enable();

// Set the uart callback function
uart_set_rx_callback(AEM_ISR);
uart_print("Give me your AEM \n");
```

Ακολουθεί η ISR που καλείται κάθε φορά που λαμβάνεται από τον μικροελεγκτή ένα byte από τον χρήστη που στέλνει μέσω σειριακής κονσόλας . Αποθηκεύουμε τον κάθε χαρακτήρα σε έναν πίνακα AEM αφού ελέγξουμε ότι λαμβάνουμε από τον χρήστη αριθμούς. Μόλις πατηθεί το «enter» από τον χρήστη θα σταλεί το '10', οπότε γνωρίζουμε ότι έχουμε λάβει σωστά όλο το AEM. Υπολογίζουμε και το άθροισμα των ψηφίων καθώς θα χρειαστεί στην *BUTTON_ISR()* όπως θα δούμε παρακάτω.

```
// ISR to process the user's AEM
void AEM_ISR(uint8_t uart_byte){
  // Save the AEM until the enter character was recognized
  if(uart_byte != 10 && (uart_byte >= 48 && uart_byte <= 57)){
    AEM[aem_index] = uart_byte;
    aem_index++;
  }
  else if(uart_byte == 10){ // which is the "enter"{
    aem_index--;
    AEM_SUM = (AEM[aem_index] - 48) + (AEM[aem_index-1] - 48);
    aem_index = 0;
  }
  else{
    uart_print("Wrong AEM. Type Again\n");
    AEM_SUM = 0;
  }
}
```

Timer Initialization and Logic

Στην εργασία πρέπει να χειριστούμε διαφορετικά events και όπου η εκτέλεση του αντίστοιχου κώδικα για το κάθε event πρέπει να εκτελείται με διαφορετική συχνότητα. Για παράδειγμα, η θερμοκρασία πρέπει να διαβάζεται ανά 2s, ενώ η τύπωση της δεν είναι κατά ανάγκη να συμβαίνει στον ίδιο χρόνο αλλά εξαρτάται από το AEM και το πάτημα του user_button. Επειδή θα χρησιμοποιήσουμε έναν timer, τον αρχικοποιούμε να εκτελεί την ISR του ανά 1s και με κατάλληλους counters μπορούμε να ξέρουμε πότε μεσολάβησαν 2s ή όποια άλλη συχνότητα επιθυμούμε.

Γνωρίζουμε ότι κάθε κύκλος που εκτελεί ο επεξεργαστής μια συγκεκριμένη εντολή επιτελείται με συχνότητα 16MHz (System CPU Clock). Για να περάσει 1s πρέπει να 6μεσολαβήσουν 16 * 10^6 κύκλοι ρολογιού. Επιλέγουμε ως όρισμα το 10^6 το οποίο προκύπτει αν κατανοήσουμε την *timer_init()* που φαίνεται παρακάτω. Τέλος, αρχικοποιούμε κα την callback *timer_ISR().*

```
void timer_init(uint32_t timestamp) {

    uint32_t tick_us = (SystemCoreClock)/1e6; // SystemCoreClock = 16MHz --> tick_us = 16
    tick_us = tick_us*timestamp; // = 16000000 ticks , tick = 1 / SystemClock
    SysTick_Config(tick_us);
    NVIC_SetPriority(SysTick_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(), 0, 0));

  }
```

```
// Timer Init //
// Set up the timer to interrupt the CPU with a rate of 1sec
timer_init(1000000);
// Enable the timer
timer_enable();
// Set the timer callback ( called every second )
timer_set_callback(timer_ISR);
```

Υλοποίηση συνάρτησης *timer_ISR()*

Εφόσον έχουμε αρχικοποιήσει τον timer να προκαλεί interrupt ανά δευτερόλεπτο και εμείς θέλουμε να ελέγχουμε τον ρυθμό της εκτύπωσης της θερμοκρασίας δημιουργήθηκε η μεταβλητή print_timing_counter. Αντίστοιχα, το διάβασμα του αισθητήρα γίνεται ανεξαρτήτως της τύπωσης ανά 2s μέσω της μεταβλητής temperature_timing_counter. Σημαντικό πριν τυπώσουμε την θερμοκρασία ή πριν τον έλεγχο των LED πρέπει να βεβαιωθούμε ότι το πακέτο που λάβαμε από τον αισθητήρα είναι σωστό. Αυτό ελέγχεται από την *check_sum()* .

```
void timer_ISR(void)
{
  temperature_timing_counter++;
  print_timing_counter++;
  if(temperature_timing_counter == 2)
  {
    temperature_timing_counter = 0;
    start_signal();
    // check_response() returns 0 when its ok to read the sensor //
     if(!check_response()){
       uint8_t *p;
       p = get_packet();
       for(uint8_t i = 0 ; i < 5; i++)
          packet[i] = *(p + i);
        // Control the LED based on the temp value
         if(check_sum(packet) == OK)
         {

           if(temperature > 25 )
             led_on();
           else if(temperature < 20)
             led_off();
           else
             toggle_led();
         }
    }
  }

  if(print_timing_counter == PRINT_FREQUENCY && check_sum(packet) == OK)
  {
    print_timing_counter = 0;
    check_sum_var = OK;
    uart_print("Temperature = ");
    temperature = packet[2];
    sprintf(_temp_, "%d", temperature);
    uart_print(_temp_);
    uart_print("\n");
  }
}
```

Υλοποίηση συναρτήσεων για το control των LED

Χρησιμοποιήθηκε το user_led για την ενημέρωση του χρήστη σχετικά με την θερμοκρασία.

```
// LED init

gpio_set_mode(USER_LED, Output);
gpio_set(USER_LED,0);
```

```
void led_on(void)
{
  gpio_set(USER_LED, 1);
}


void led_off(void)
{
  gpio_set(USER_LED, 0);
}


void toggle_led(void)
{
  gpio_toggle(USER_LED);
}
```

Υλοποίηση της *BUTTON_ISR()*

Εδώ φαίνεται ότι το user button επηρεάζει την συχνότητα τύπωσης της θερμοκρασίας καθώς μεταβάλει την όριο PRINT_FREQUENCY που ελέγχεται εντός της timer_ISR.

```
void BUTTON_INIT(void)
{
  gpio_set_mode(BUTTON, PullUp);
  gpio_set_trigger(BUTTON, Rising);
  gpio_set_callback(BUTTON,(void *)BUTTON_ISR);
}

void BUTTON_ISR(void){
  button_pressed ++;
  print_timing_counter = 0 ;

  if(button_pressed == 1){
    PRINT_FREQUENCY = AEM_SUM;
  }
  else{
    if(button_pressed % 2 == 0){
      PRINT_FREQUENCY = 4;
    }
    else{
      PRINT_FREQUENCY = 3;
    }
  }
  uart_print("\nRefresh rate: ");
  sprintf(refresh_rate, "%d", PRINT_FREQUENCY);
  uart_print(refresh_rate);
  uart_print(" sec");
  uart_print("\n");
}
```