

Fotografia 1. Wygląd zestawu STM32L0538-DISCO

# STM32L0538-DISCO + e-paper

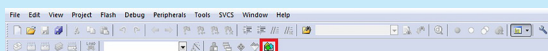
## Konfigurowanie środowiska MDK ARM/Keil MDK oraz wyświetlanie bitmap

**Zestaw STM32L0538-DISCO jest niezwykle rzadkim zjawiskiem na rynku. Jako jedyny tani zestaw startowy jest wyposażony w wyświetlacz e-paper, który wyświetla obraz także po odłączeniu zasilania. Jest to tak interesujące rozwiązanie, że postanowiliśmy je nieco przybliżyć Czytelnikom EP.**

Wygląd zestawu STM32L0538-DISCO pokazano na **fotografii 1**. Zastosowano w nim mikrokontroler STM32L053, który jest wyposażony w rdzeń Cortex-M0+, ponadto 64 kB pamięci Flash i 8 kB RAM. Jednostka centralna mikrokontrolera może być taktowana sygnałem zegarowym do 32 MHz, średnia prędkość wykonywania programów wynosi 0,95 DMIPS/MHz. Użyty w zestawie wyświetlacz ma przekątną 2,04" przy rozdzielczości matrycy 172x72 piksele. Jak każdy z zestawów Discovery, także STM32L0538-DISCO ma wbudowany programator-debugger ST-Link/v2-1.

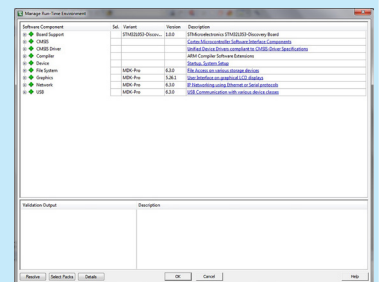
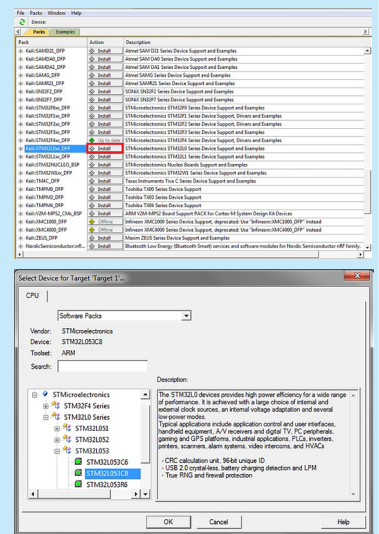
Przygotowanie środowiska ARM-MDK przebiega krok-po-kroku w następujący sposób:

1. Pobieramy oraz instalujemy środowisko MDK-ARM Keil w wersji 5.15 lub nowszej <https://goo.gl/QhzLcn>.
2. Pobieramy oraz instalujemy sterowniki ST-LINK/V2 <https://goo.gl/hYW7nU>.
3. Podłączamy mikrokontroler poprzez złącze USB komputera i złącze USB Mini-B mikrokontrolera USB ST-LINK.
4. Otwieramy środowisko MDK-ARM Keil i otwieramy *Pack Installer* za pomocą ikony na pasku.



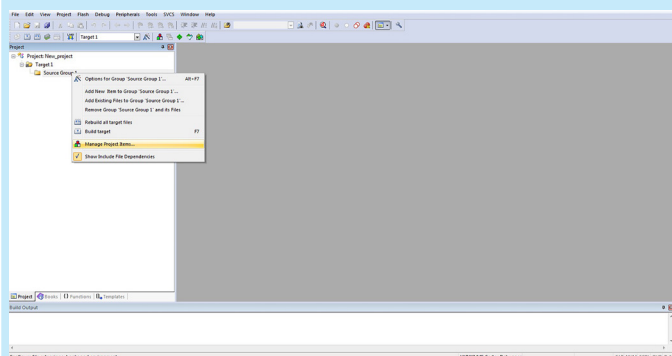
5. Instalujemy najnowsze biblioteki w Pack Installer poprzez znalezienie w zakładce *Packs* bibliotek Keil: *STM32L0xx\_DFP* i kliknięcie przycisku *Install* obok nazwy. Po zakończeniu procesu (wyświetla się tekst *Up to date*) zamykamy okno *Pack Installer*. Po instalacji należy zaktualizować lub zainstalować biblioteki ARM::CMSIS w taki sam sposób.

6. Wybieramy opcję *New μVision Project* z zakładki *Projects*.
7. Zapisujemy nowy projekt w wybranym folderze i wpisujemy nazwę projektu, np. *New\_project.uvproj*.
8. Wybieramy z listy mikrokontroler, który powinien się pojawić po zainstalowaniu odpowiednich bibliotek poprzez *Pack Installer* (*STMicroelectronics* → *STM32L0 Series* → *STM32L053* → *STM32L053C8*). Po podświetleniu nazwy klikamy przycisk OK.
9. Zostanie wyświetlone okno *Manage Run-Time Environment*. Klikamy przycisk OK i zakończenie tworzenia nowego projektu.
10. Otwieramy folder w którym został zapisany projekt, powinien zawierać pliki *nazwa\_projektu.uvproj* i *nazwa\_projektu.uvoptx* oraz dwa foldery



– Listings i Objects. Tworzymy dwa nowe foldery i nazywamy je *Inc* oraz *Src*.

11. Pobieramy paczkę bibliotek *STM32CubeL0* (<http://goo.gl/632jKy>).
12. Kopiujemy folder *Drivers* z paczki *STM32CubeL0* do folderu z projektem.
13. Przechodzimy do ekranu głównego środowiska MDK-ARM Keil. Prawym przyciskiem myszy klikamy na *Target 1* w oknie *Project* i wybieramy *Manage Project Items*.



14. W oknie *Manage Project Items* można zmienić domyślną nazwę folderu *Source Group 1*, *Target 1* oraz dodać inne foldery w oknie *Groups*. Tworzymy kilka grup, np. *User*, *HAL Drivers*, *CMSIS*, *BSP*, *Components* oraz dodajemy do odpowiednich folderów wcześniej skopiowanych plików bibliotek za pomocą przycisku *Add files*.

15. Kopiujemy pliki z katalogów *Inc* i *Src* z paczki *STM32CubeL0* z folderu *Projects/STM32L053C8-Discovery/Templates* z do tak samo nazwanych folderów w katalogu naszego projektu. (*main.h*, *stm32l0xx\_hal\_conf.h*, *stm32l0xx\_it.h*, *main.c*, *stm32l0\_hal\_msp.c*, *stm32l0xx\_it.c*). Pliki z rozszerzeniem *.c* dołączamy do grupy *User*. Następnie dołączamy wszystkie pliki z folderu *Drivers/STM32L0xx\_HAL\_Driver/Src* do grupy *HAL Drivers*.

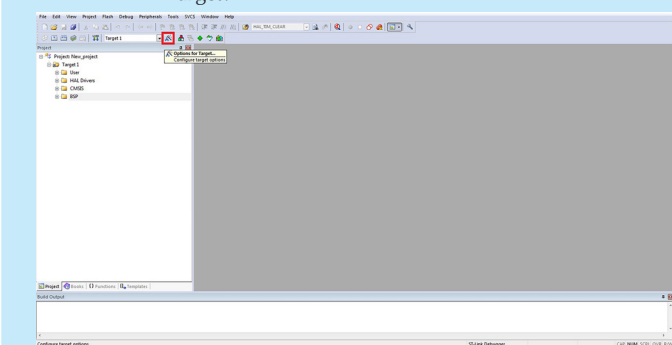
16. Znajdujemy plik *system\_stm32l0xx.c* w katalogu *Drivers/CMSIS/Device/ST/STM32L0xx/Source/Templates* oraz *startup\_stm32l053xx.s* w katalogu *Drivers/CMSIS/Device/ST/STM32L0xx/Source/Templates/arm* i dodajemy te pliki do grupy *CMSIS*.

17. Znajdujemy pliki *stm32l0538\_discovery.c* i *stm32l0538\_discovery\_epd.c* w katalogu *Drivers/BSP/STM32L0538-Discovery* i dodajemy ten plik do grupy *BSP*.

18. Znajdujemy plik *gde021a1.c* w katalogu *Drivers/BSP/Components/gde021a1* i dodajemy ten plik do grupy *Components*.

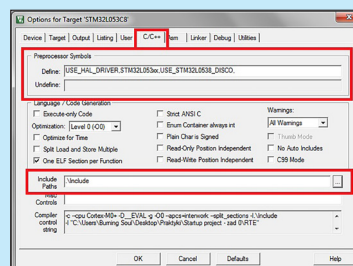
19. Tworzymy nowy plik klikając na zakładkę *File* oraz *New...* i kopiujemy do niego kod, jak na **listingu 1**. Zapisujemy utworzony plik pod nazwą *bsp.c* klikając *File* oraz *Save as...* i dodajemy ten plik do grupy *User*. Zamykamy okno *Manage Projects Items* za pomocą przycisku *OK*.

20. Otwieramy okno konfiguracji projektu przyciskiem *Options for Target 'nazwa'* (domyślnie *Target 1*) lub wybieramy z zakładki *Project* polecenie *Options for Target*.

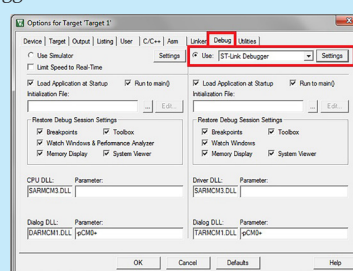


21. Dodajemy w zakładce *C/C++* w polu *Define* w ramce *Preprocessor Symbols* następującą definicję: *USE\_HAL\_DRIVER,STM32L053xx,USE\_STM32L0538\_DISCO*. Za pomocą przycisku po prawej stronie pola *Include Paths* dodajemy ścieżki do następujących folderów:

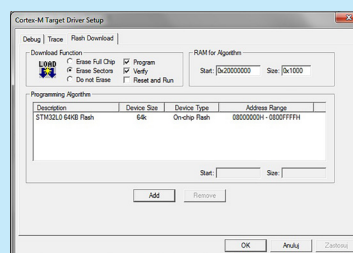
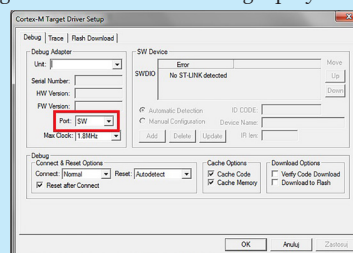
- *.\Inc*
- *.\Drivers\STM32L0xx\_HAL\_Driver\Inc*
- *.\Drivers\CMSIS\Device\ST\STM32L0xx\Include*
- *.\Drivers\BSP\STM32L0538-Discovery*
- *.\Drivers\BSP\Components\gde021a1*



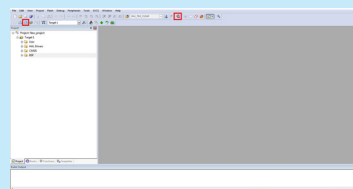
22. Wybranie w zakładce *Debug* opcji *Use* i *ST-Link Debugger*.



23. Następnie wejście w ustawienia za pomocą przycisku *Settings*, wybranie w zakładce *Debug* portu SW oraz konfiguracja pamięci Flash w zakładce *Flash Download* na *STM32L0 64KB Flash*. Zamknięcie okna ustawień debugowania oraz ustawień *Target* przyciskiem *OK*.



24. Kompilacja stworzonego projektu za pomocą opcji *Rebuild all target files* w zakładce *Project* lub wciśnięcie odpowiadającego tej opcji przycisku na ekranie głównym.



25. Jeśli kompilacja przebiegła bezbłędnie, można uruchomić tryb debugowania za pomocą opcji *Start/Stop Debugging Session* w zakładce *Debug* oraz wrócić do pamięci mikrokontrolera i uruchomić program za

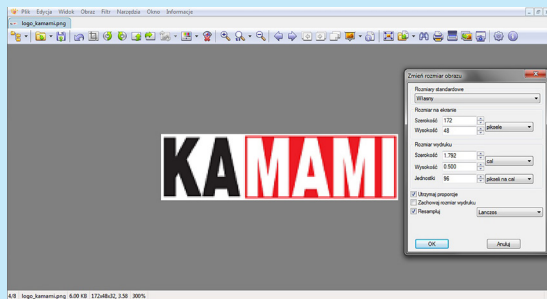


pomocą opcji *Run* w zakładce *Debug* lub za pomocą przycisków na ekranie głównym. Jeśli program został uruchomiony poprawnie, dioda kontrolna w zestawie zamiga i zaświeci się na zielono.

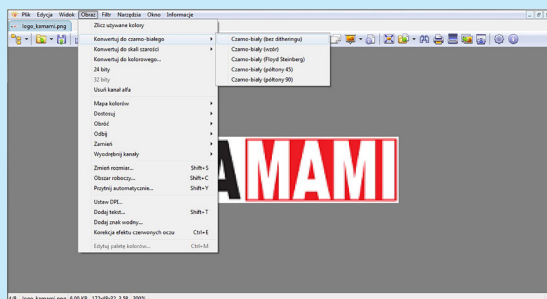
## Wyświetlenie bitmapy na wyświetlaczu e-paper

Wbudowany w zestaw STM32L0538-DISCO wyświetlacz GDE021A1 wyświetla obrazy monochromatyczne o wymiarach 172×72 pikseli. Aby wyświetlić bitmapę na wyświetlaczu, należy odpowiednio sformatować obraz i skonwertować go na kod możliwy do przesłania do pamięci RAM mikrokontrolera.

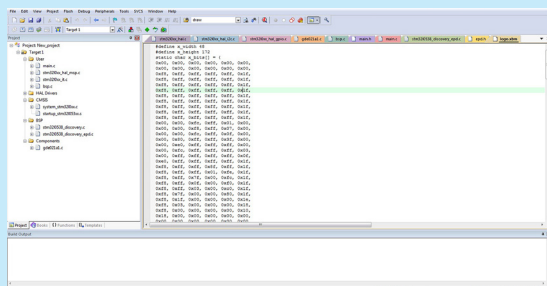
Wstępną edycję obrazu możemy dokonać w dowolnym programie graficznym. Programem, który umożliwi konwersję bitmapy na kod oraz edycję zapewniającą dobrą jakość wyświetlonego obrazu jest darmowy XnView. Rozmiar obrazu nie może przekroczyć rozdzielczości wyświetlacza, czyli 172x72 pikseli, możemy tego dokonać wybierając opcję *Zmień rozmiar...* z zakładki *Obraz*.



Następnym krokiem jest zapisanie obrazu w czarno-białej skali barw, w zależności od obrazu wybieramy rodzaj konwersji zapewniający najlepszy efekt z zakładki *Obraz i Konwertuj do czarno-białego*.



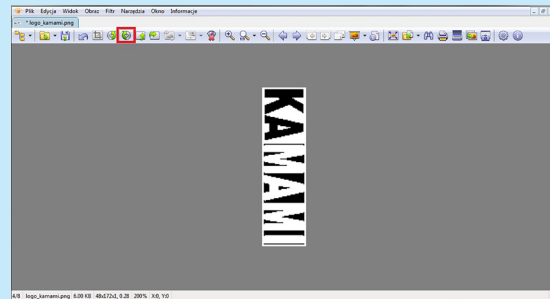
Po konwersji część detali może zniknąć, możemy w tym momencie zapisać plik i dokonać jego edycji w dowolnym programie graficznym, np. Paint. Aby otrzymać obraz na wyświetlaczu odpowiadający naszej bitmapie, musimy obrócić go o 90° zgodnie ze wskazówkami zegara (jego rozmiar zmieni się ze 172×72 na 72×172 pikseli).



Ostatnią częścią przygotowania obrazu jest zapisanie go w formacie XBM-X11 korzystając z opcji *Zapisz jako...* i wybraniu formatu z listy.

W utworzonym wcześniej projekcie w środowisku ARM-MDK otwieramy bitmapę zapisaną w formacie *.xbm*.

Kopiujemy cały kod oprócz definicji rozmiarów (możemy je skopiować i skomentować, aby pamiętać o rozmiarze) do *main.c* jako prywatną funkcję, zmieniając *static char x\_bits[] =* na *static uint8\_t x\_bits[] =*. W miejscu



*x\_bits* wpisujemy nazwę naszej funkcji, np. *static uint8\_t Logo[] =*. W funkcji *main()* deklarujemy użycie wyświetlacza za pomocą funkcji *BSP\_EPD\_Init()*; a następnie używamy następujących funkcji, aby wyświetlić obraz na wyświetlaczu:

```
BSP_EPD_Clear(EPD_COLOR_WHITE);
BSP_EPD_DrawImage(uint16_t Xpos, uint16_t Ypos,
uint16_t Xsize, uint16_t Ysize, uint8_t *pdata);
BSP_EPD_RefreshDisplay();
```

Funkcja *BSP\_EPD\_Clear(EPD\_COLOR\_WHITE)*; zmienia wszystkie piksele wyświetlacza na białe pozwalając na wyświetlenie nowego obrazu.

Funkcja *BSP\_EPD\_DrawImage(uint16\_t Xpos, uint16\_t Ypos, uint16\_t Xsize, uint16\_t Ysize, uint8\_t \*pdata)*; służy do przesłania danych naszego obrazu do wyświetlacza. Parametry *uint16\_t Xpos, uint16\_t Ypos* określają gdzie zostanie wyświetlony obraz. Aby obraz został wyświetlony poprawnie wartość parametru *Ypos* musi być podzielna przez 4. Parametry *uint16\_t Xsize, uint16\_t Ysize* określają rozmiar wyświetlanego obrazu, muszą być zgodne z rozmiarami naszego obrazu (maksymalnie *uint16\_t Xsize = 72* i *uint16\_t Ysize = 172*). Parametr *uint8\_t \*pdata* wskazuje na dane o naszym obrazie, więc należy to wpisać nazwę funkcji pod jaką znajdują się dane skopiowane z pliku *.xbm*. Przykładowe wywołanie funkcji ma postać: *BSP\_EPD\_DrawImage(0, 3, 48, 172, Logo)*.

Funkcja *BSP\_EPD\_RefreshDisplay()*; odświeża wyświetlacz i wyświetla nasz obraz po załadowaniu jego danych do pamięci RAM mikrokontrolera.

Jakub Górnicki

### Listing 1. Oprogramowanie testowe

```
#include „bsp.h”
void Bsp_Init(void)
{
    /* Configure LED3 and LED4 */
    BSP_LED_Init(LED3);
    BSP_LED_Init(LED4);

    /* Initialize the User button */
    BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);

    /* Initialize the EPD */
    BSP_EPD_Init();

    /* Initialize the Touch sensor */
    BSP_TSL_Init();
}
/**
 * @brief Configure the touch sensor.
 * @param None
 * @retval None
 */
void BSP_TSL_Init(void)
{
    /* Configure the TSC peripheral */
    /* All channel, shield and sampling IOs must be declared below */
    /* Initialize the STMTouch driver */
}
/**
 * @brief Initializes LEDs, Button, and EPD.
 * @param None
 * @retval None
 */
void BSP_USB_Init(void)
{
    /* Init Device Library */

    /* Register the HID class */

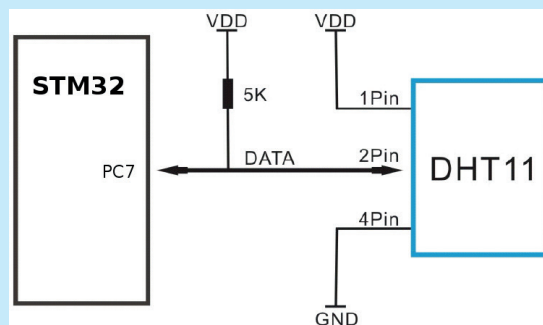
    /* Start Device Process */
}
```

Czujnik DHT11 jest członkiem większej rodziny zintegrowanych czujników wilgotności i temperatury. Różnią się one precyzją pomiarów, natomiast są kompatybilne pod względem rozmieszczenia wyprowadzeń i sposobu komunikacji z systemem nadrzędnym. W tabeli 1 umieszczono najważniejsze parametry 3 typów czujników.

Sensor DHT11 ma 4 wyprowadzenia. Mają one następujące funkcje:

1. VDD – zasilanie czujnika.
2. DATA – linia danych.
3. Nieużywane.
4. GND – masa zasilania.

Komunikacja czujnika z kontrolerem STM32 odbywa się za pośrednictwem linii DATA. Typowy schemat połączenia SHT11 do systemu nadrzędnego pokazano na rysunku 1.



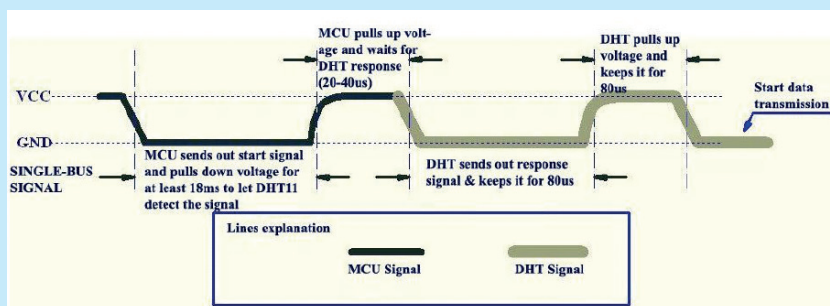
Rysunek 1. Schemat połączenia czujnika SHT11 z systemem nadrzędnym

## Komunikacja z czujnikiem wilgotności i temperatury

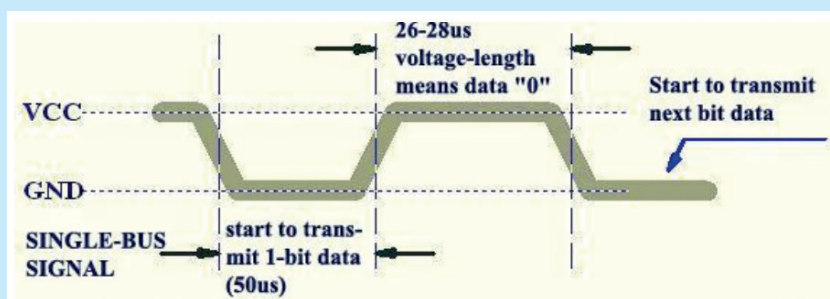
Protokół komunikacji pomiędzy czujnikiem a kontrolerem podobny jest do protokołu 1-Wire. Dane przesyłane są szeregowo magistralą DATA bit po bicie. Bity o wartości logicznej 0 i 1 różnią się czasem trwania. Magistrala w stanie nieaktywnym utrzymywana jest na poziomie wysokim poprzez opornik zasilający (5 kΩ). Kolejną transmisję inicjuje kontroler wymuszając na magistrali poziom niski przez 18 ms. Jeżeli czujnik jest dołączony do magistrali, powinien w czasie nie dłuższym niż 40 μs odpowiedzieć impulsem ujemnym o czasie około 80 μs. Po zakończeniu impulsu i czasie przerwy trwającej 80 μs czujnik rozpoczyna wysyłanie kolejnych bajtów danych zmierzonej wilgotności i temperatury. Na rysunku 2 pokazano sekwencję inicjującą transmisję. Logiczne 0 i 1 różnią się czasem trwania sekwencji stanu niskiego i wysokiego. Transmisję bitu 0 rozpoczyna ujemny impuls o czasie około 50 μs wysyłany przez czujnik. Następnie, magistrala jest utrzymywana na poziomie wysokim przez 26...28 μs

i czujnik rozpoczyna wysłanie kolejnego bitu. Opisaną sytuację pokazano na rysunku 3. Transmisję bitu '1' także rozpoczyna ujemny impuls o czasie około 50 μs wysyłany przez czujnik. Następnie magistrala utrzymywana jest w stanie wysokim przez 70 μs i czujnik inicjuje przesłanie kolejnego bitu. Na rysunku 4 pokazano przebiegi podczas transmisji logicznej 1. Transmisję każdego bajta danych czujnik rozpoczyna od wysłania najstarszego bitu. Każda transmisja składa się z sekwencji 5 bajtów zawierających komplet informacji o zmierzonej wilgotności względnej (RH) w procentach, temperatury w stopniach i sumy kontrolnej. Bajty transmisji wysyłane są w następującej kolejności: wilgotność w procentach, dziesiąta część wilgotności, temperatura w stopniach Celsjusza, temperatura w dziesiątych częściach stopnia, bajt sumy kontrolnej.

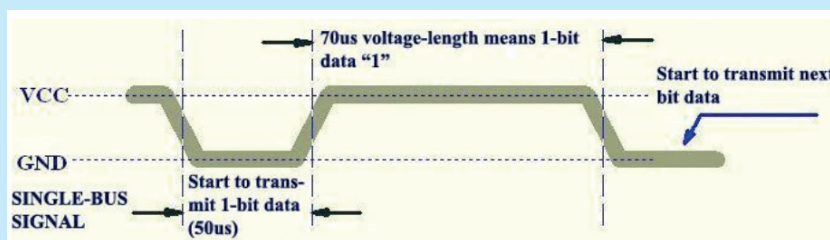
Protokół transmisji dla wszystkich wymienionych w tabeli typów czujników jest identyczny. Dla DHT11 bajty dziesiętne pomiarów będą zawsze wyzerowane, ponieważ jego dokładność ogranicza się do pełnych



Rysunek 2. Sekwencja inicjująca transmisję



Rysunek 3. Transmisja logicznego 0



Rysunek 4. Transmisja logicznej 1

procentów pomiaru wilgotności i pełnych stopni temperatury. Sumę kontrolną oblicza się dodając do siebie wartości kolejnych bajtów wilgotności i temperatury. Bajtem kontrolnym jest LSB tak obliczonej sumy.

## Założenia projektu testowego

Przykładowe procedury obsługi czujnika wykonano jako część projektu testowego termometru z higrometrem. Zmierzone wartości wilgotności względnej i temperatury mają być wyświetlane na 2-liniowym wyświetlaczu alfanumerycznym LCD. Cykle pomiaru i wyświetlania mają być dodatkowo sygnalizowane świeceniem diod LED. Błąd komunikacji z czujnikiem jest sygnalizowany za pomocą diody LED. Jako platformę sprzętową zastosowano Panel Edukacyjny z dołączonym przewodami czujnikiem DHT11. Listę połączeń sensora z Panelem Edukacyjnym umieszczono w tabeli 2. Przełącznik S13 powinien być usunięty lub ustawiony w pozycji

„otwarty”, zwory na JP6 założone, a wyświetlacz LCD umieszczony w gnieździe J5.

## Przygotowanie projektu dla środowiska CooCox

Projekt testowy opracowano za pomocą środowiska CooCox, jednak wydzielone pliki procedur obsługi czujnika bez problemu dadzą się przenieść do innego zintegrowanego IDE np. Keil.

Po otwarciu środowiska CooCox rozpoczynamy nowy projekt od wybrania z menu *Project* opcji *New Project*. Następnie, podajemy nazwę i ścieżkę dostępu do katalogu, w którym projekt ma być umieszczony. Wybieramy typ kontrolera, dla którego projekt ma być kompilowany. Dla Panelu Edukacyjnego będzie to STM32F103RC. Kolejnym krokiem jest dodanie niezbędnych bibliotek. Otwieramy *View* → *Repository* i zaznaczamy: C Library, CMSIS Core, CMSIS Boot, RCC, GPIO, TIM, MISC. Zaznaczone składniki zostaną dodane automatycznie do projektu. Na koniec otwieramy *View* → *Configuration* i wybieramy opcję *Link* → *Use base C Library*. W tym momencie szkielet programu został utworzony. Pozostało dopisać własne procedury, w tym te obsługujące komunikację i odczyt pomiarów z czujnika.

### Listing 2. Manipulowanie poziomem linii PC7

```
//-----
//ustawienie linii SINGLE-WIRE
void GPIO_Linia_S_W_High(void)
{
    LINIA_S_WIRE_PC7_PORT->BSRR =LINIA_S_WIRE_PC7_PIN;
}
//-----
//zerowanie linii SINGLE-WIRE
void GPIO_Linia_S_W_Low(void)
{
    LINIA_S_WIRE_PC7_PORT->BRR =LINIA_S_WIRE_PC7_PIN;
}
//-----
//procedura odczytu stanu linii SINGLE-WIRE
uint32_t GPIO_Linia_S_W_Odczyt(void)
{
    return GPIO_ReadInputDataBit(LINIA_S_WIRE_PC7_PORT, LINIA_S_WIRE_PC7_PIN);
}
}
```

### Listing 3. Inicjowanie Timera 3

```
/* TIMER3 taktowany wewnętrznym zegarem z częstotliwością 1MHz dla
rozdzielczości pomiaru lus TIM_Channel_2 (PC7) zatrzaszkuje stan
licznika TIM3 zboczem narastającym i opadającym procedury inicjacji
TIMER3 */
void TIMER3_Inicjacja(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_ICInitTypeDef TIM_ICInitStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;
    uint16_t PrescalerValue = 0;
    #define TAKTOWANIE_1MHz 1000000

    //TIM3 clock enable
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    //TIM3 disable counter
    TIM_Cmd(TIM3, DISABLE);
    //konfiguracja TIM3
    TIM_TimeBaseStructure.TIM_Period =0xFFFF;//do pomiaru wykorzystano 16 bitów
    TIM_TimeBaseStructure.TIM_Prescaler = 0;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;//zliczanie „do góry”
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
    // Prescaler configuration
    PrescalerValue = (uint16_t) (SystemCoreClock / (TAKTOWANIE_1MHz)) - 1;
    TIM_PrescalerConfig(TIM3, PrescalerValue, TIM_PSCReloadMode_Immediate);
    //konfiguracja kanału 2 do przechwycenia stanu TIMER3 w momencie podania
    //padającego zbocza na wejście PC7
    TIM_ICInitStructure.TIM_Channel =TIM_Channel_2;
    TIM_ICInitStructure.TIM_ICFilter =0x0F;
    TIM_ICInitStructure.TIM_ICPolarity =TIM_ICPolarity_Falling;//TIM_ICPolarity_BothEdge
    TIM_ICInitStructure.TIM_ICPrescaler =TIM_ICPSC_DIV1;
    TIM_ICInitStructure.TIM_ICSelection =TIM_ICSelection_DirectTI;
    TIM_ICInit(TIM3, &TIM_ICInitStructure);
    //konfiguracja kanału 4 do pracy jako Stoper mierzący odcinki czasu 1-65535us
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 1000;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Disable);
    /* TIM CC2 IT enable */
    flaga_zbocza =FALSE;
    TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
    TIM_ITConfig(TIM3, TIM_IT_CC2, ENABLE);
    //włączenie licznika
    TIM_Cmd(TIM3, ENABLE);
}
}
```

### Listing 1. Deklaracja i inicjowanie portu PC7

```
//LINIA_S_WIRE_PC7 linia I/O magistrali SINGLE-WIRE (PC7)
#define LINIA_S_WIRE_PC7_PORT GPIOC
#define LINIA_S_WIRE_PC7_PORT_NUM GPIO_PortSourceGPIOC
#define LINIA_S_WIRE_PC7_CLK RCC_APB2Periph_GPIOC
#define LINIA_S_WIRE_PC7_PIN GPIO_Pin_7
#define LINIA_S_WIRE_PC7_PIN_SOURC GPIO_PinSource7

//procedura konfiguracji linii SINGLE-WIRE
void GPIO_Linia_S_W_Konfig(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    /* Enable the GPIO Clock */
    RCC_APB2PeriphClockCmd(LINIA_S_WIRE_PC7_CLK, ENABLE);
    /* Configure the GPIO pin*/
    GPIO_InitStructure.GPIO_Pin = LINIA_S_WIRE_PC7_PIN;
    GPIO_InitStructure.GPIO_Mode =GPIO_Mode_Out_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(LINIA_S_WIRE_PC7_PORT, &GPIO_InitStructure);
    /* Enable the Alternate function Clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    //remapowanie funkcji TIMER3 i przypisanie do PC7
    GPIO_PinRemapConfig (GPIO_FullRemap_TIM3 , ENABLE);
}
}
```

Tabela 2. Połączenie czujnika DHT11 z Panelem Edukacyjnym

Wyprowadzenie czujnika	Złącze Panelu Edukacyjnego
1 (VDD)	3V3
2 (DATA)	PC7
4 (GND)	GND



**Listing 4. Procedura ustawienia kanału 4 do pracy jako stoper**

```
//wykorzystany Timer3 i kanał CC4
//odliczanie czasu przerwy z rozdzielczością lus przez Stoper1
void Start_Stoper1(uint16_t odliczany_czas_us)
{
    TIM_OCInitTypeDef TIM_OCInitStructure;
    uint16_t CCR4_Val;
    /* Output Compare Timing Mode configuration: Channel4 */
    TIM_ITConfig(TIM3, TIM_IT_CC4, DISABLE);
    CCR4_Val=TIM_GetCounter(TIM4);
    CCR4_Val = CCR4_Val + odliczany_czas_us;
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = CCR4_Val;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Disable);
    /* TIM CC4 IT enable */
    flaga_stoperal =FALSE;
    TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
    TIM_ITConfig(TIM3, TIM_IT_CC4, ENABLE);
}
}
```

**Listing 5. Obsługa przerwania Timer'a 3 oraz konfigurowanie NVIC**

```
//-----
//TIM3 obsługa przerwania
void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM3, TIM_IT_CC2) != RESET)
    {
        //kanał CC2 wyzwalany zboczem przechwytuje stan TIM3
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
        flaga_zbocza=TRUE;
    }
    if (TIM_GetITStatus(TIM3, TIM_IT_CC4) != RESET)
    {
        //kanał CC4 steruje odliczaniem czasu Stoperal
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
        //wyłączenie przerwania kanału
        TIM_ITConfig(TIM3, TIM_IT_CC4, DISABLE);
        flaga_stoperal=TRUE;
    }
}

//w pliku Procedury_Ogolne.c
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    /*Enable the TIM3 global Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
}
```

## Katalogi projektu

Na pulpicie CooCox-a w okienku *Project* jest wyświetlone drzewo katalogów projektu. Po przygotowaniu szkieletu projektu powinno być już częściowo wypełnione automatycznie utworzonymi katalogami. Przy dodawaniu do projektu własnych plików z procedurami, użytkownik sam tworzy dodatkowe grupy katalogów i plików. Najłatwiej robi się to po kliknięciu prawym przyciskiem myszy na pierwszą pozycję w okienku *Project* będącą nazwą projektu. Z wyświetlonej listy wybiera się pozycję *Add Group* i wpisuje nazwę grupy. Logicznie jest tworzenie grup odzwierciedlających rzeczywisty układ podkatalogów projektu na dysku. Klikając w ten sam sposób na nazwę nowo utworzonej grupy i wybierając z wyświetlonej listy pozycję *Add Files* dodaje się do utworzonej grupy pliki już istniejące na dysku.

Po wstępnej fazie przygotowania projektu środowisko CooCox najprawdopodobniej samo utworzy następujące grupy: *cmsis*, *cmsis\_boot*, *stm\_lib*, *syscalls*, plik z funkcją *main()*. W projekcie testowym przewidziałem następujące dodatkowe grupy, w których znajdują się pliki projektu dodawane przez użytkownika:

- **Procedury\_SINGLE\_WIRE.** Pliki związane z komunikacją i odczytem danych z czujnika DHT11.
- **Procedury\_TIMER.** Pliki związane z użytym w projekcie sprzętowym timerem kontrolera.
- **Procedury\_GPIO.** Pliki związane z portami obsługiwanyymi na Panelu Edukacyjnym, między innymi wyświetlacz LCD i diody LED.
- **Procedury\_LCD.** Pliki procedur pokazujących informację na wyświetlaczu.

- **Procedury\_Podstawowe.** Pliki z procedurami programowych pauz, inicjacją przerwania oraz wydzielony plik na podprogramy przerwania Timera.

## Port GPIO do obsługi komunikacji z czujnikiem DHT11

Dekodowanie danych odczytywanych z linii DATA czujnika polega głównie na precyzyjnym pomiarze czasu trwania impulsów kodujących kolejne bity transmisji. W projekcie testowym zdecydowałem się na użycie do pomiaru jednego z wewnętrznych timerów kontrolera w trybie pracy *capture*. W tym trybie można „w locie” zapamiętywać bieżącą wartość timera, gdy wystąpi kolejne zbocze impulsu na linii DATA. Porównanie wartości odpowiadających kolejnym zboczom pozwala na obliczenie czasu trwania impulsu, a tym samym właściwie odczytanie kolejnych bitów transmisji z czujnika.

Wybór padł na TIMER3 i jego kanał 2. Portem współpracującym z tym kanałem jest PC7. I właśnie do tego portu jest dołączana linia DATA. W podkatalogu *Procedury\_SINGLE\_WIRE* umieszczono pliki *GPIO\_S\_Wire\_Procedure.h* i *GPIO\_S\_Wire\_Procedure.c*, w których są deklaracja i inicjacja portu PC7 (**listing 1**). W efekcie port PC7 będzie pracował jako wyjściowy w trybie otwartego drenu. W tym trybie można zarówno wpływać na poziom sygnału na wyprowadzeniu portu, co jest konieczne do wygenerowania przez kontroler impulsu *Start*, jak i czytać poziom sygnału występującego na porcie, co jest konieczne do sterowania Timerem w trybie *capture*. Dodatkowo, w pliku umieszczono krótkie procedury do ustawienia/zerowania portu i jego odczytu (**listing 2**).

```

Listing 6. Obsługa transmisji czujnika DHT11
//procedura odczytu danych z czujnika DHTxx
char Odczyt_DHTxx(unsigned char *p_bufor)
//we: wskaźnik do 5 bajtowego buforu odczytu
//wy: status operacji
{
//zbczoc inicjujące, H bajt higro, L bajt higro, H bajt temp, L bajt temp, bajt sumy
#define ILE_ZBOCZY_TRANSMISJI_DHTXX (1+8+8+8+8)
extern char flaga_stoperal, flaga_zbczocza;
uint32_t stan_linii;
uint8_t x, numer_bajtu, numer_bitu, suma;
uint16_t zbczocza_licznik[ILE_ZBOCZY_TRANSMISJI_DHTXX], czas_bitu;
unsigned char maska, bajt;
stan_linii =GPIO_Linie_S_W_Odczyt();
if (stan_linii ==0) return STATUS_BLAD_ZWARCIE;
//generowanie impulsu START transmisji
Start_Stoper1(CZAS_SYGNALU_START);
GPIO_Linie_S_W_Low();
while (flaga_stoperal==FALSE)
    stan_linii =GPIO_Linie_S_W_Odczyt();
GPIO_Linie_S_W_High();
//oczekiwanie na początek sygnału potwierdzenia od DHTxx
Start_Stoper1(CZAS_STARTU_TRANS_MAX);
flaga_zbczocza =FALSE;
while (1)
{
//detekcja zbczocza sygnału potwierdzenia od DHTxx
if (flaga_zbczocza ==TRUE) break;
if (flaga_stoperal ==TRUE) return STATUS_BLAD_PRZERWA;
stan_linii =GPIO_Linie_S_W_Odczyt();
}
//oczekiwanie na początek 1 bitu
flaga_zbczocza =FALSE;
Start_Stoper1(CZAS_STARTU_TRANS_MAX);
for (x=0; x<ILE_ZBOCZY_TRANSMISJI_DHTXX; x++)
{
while(1)
{
if (flaga_zbczocza ==TRUE)//kolejne zbczocze opadające
{
zbczocza_licznik[x] =Odczyt_CAPTURE2();
break;
}
if (flaga_stoperal ==TRUE) return STATUS_BLAD_PRZERWA;
}
flaga_zbczocza =FALSE;
Start_Stoper1(CZAS_BITU_MAX);
}
//odtworzenie przesłanych bitów
x=0;
for (numer_bajtu=0; numer_bajtu <5; numer_bajtu++)
{
maska =0x80;
*(p_bufor+numer_bajtu) =0;
for (numer_bitu=0; numer_bitu<8; numer_bitu++)
{
if (zbczocza_licznik[x+1] > zbczocza_licznik[x])
{
czas_bitu =zbczocza_licznik[x+1] - zbczocza_licznik[x];
}
else
{
//pomiędzy 2 zbczoczymi zawartość licznika się przewinęła
czas_bitu =(0xFFFF+1) -zbczocza_licznik[x];
czas_bitu =czas_bitu + zbczocza_licznik[x+1];
}
if (czas_bitu >CZAS_BITU_LOW_MAX)
{
//przesłany bit jest '1'
bajt =(p_bufor+numer_bajtu);
bajt =bajt | maska;
*(p_bufor+numer_bajtu) =bajt;
}
x++;
maska =maska >>1;
}
}
suma =(p_bufor+0) +(p_bufor+1) +(p_bufor+2) +(p_bufor+3);
if (suma !=*(p_bufor+4)) return STATUS_BLAD_SUMY;
return STATUS_SUKCES;
}

```

### Konfigurowanie licznika Timer 3

W podkatalogu *Procedury\_TIMER* umieszczono pliki *Procedury\_TIMER.c* i *Procedury\_TIMER.h*. Zawierają one procedury inicjowania i obsługi Timera 3, którego kanał 2 funkcjonuje w trybie *capture*. Zamieszczono je na **listingu 3**. W efekcie timer będzie taktowany sygnałem o częstotliwości 1 MHz, co zapewnia rozdzielczość pomiaru impulsów równą 1 μs. Timer będzie liczył nieprzerwanie od wartości 0000h do FFFFh i po przepełnieniu znów od 0000h. Kanał 2 timera zaprogramowano do pracy w trybie *capture*, w którym zatrzymuje aktualny stan licznika po wystąpieniu zbczocza opadającego na wejściu

portu PC7. Procedurę pozwalającą odczytać zatrzaśniętą może być zaimplementowana, jak niżej:

```

//odczyt rejestru CAPTURE2 ze stanem TIMER3
//zatrzaśniętym zbczoczem na porcie PC7
uint16_t Odczyt_CAPTURE2(void)
{
uint16_t zawartosc_CAPTURE2;
zawartosc_CAPTURE2 =TIM_GetCapture2(TIM3);
return zawartosc_CAPTURE2;
}

```

Dodatkowo, kanał 4 Timer'a 3 użyto jako *Stoper1* odmierzający odcinki czasu z zakres 1...65535 μs. Procedurę ustawienia kanału 4 do pracy jako stoper pokazano na **listingu 4**.

Listing 7. Pętla główna programu

```

while(1)
{
    GPIO_Linie_IO_Low(LINIA_LED1_WY);
    GPIO_Linie_IO_Low(LINIA_LED4_WY);
    GPIO_Linie_IO_High(LINIA_LED8_WY);
    status = Odczyt_DHTxx(&bufor_pomiar[0]);
    GPIO_Linie_IO_Low(LINIA_LED8_WY);
    if (status == STATUS_SUKCES)
    {
        sprintf(&bufor_disp[0], 20, "%3d.%1d%c %3d.%1dC \n", bufor_pomiar[0],
            bufor_pomiar[1], '%', bufor_pomiar[2], bufor_pomiar[3]);
        Disp_Wyswietl_txt(&bufor_disp[0], 16);
        GPIO_Linie_IO_High(LINIA_LED4_WY);
    }
    if (status == STATUS_BLAD_ZWARCIE)
    {
        Disp_Wyswietl_txt(LCD_EKRAN_ZWARCIE, 0);
        GPIO_Linie_IO_High(LINIA_LED1_WY);
    }
    if (status == STATUS_BLAD_PRZERWA)
    {
        Disp_Wyswietl_txt(LCD_EKRAN_PRZERWA, 0);
        GPIO_Linie_IO_High(LINIA_LED1_WY);
    }
    if (status == STATUS_BLAD_SUMY)
    {
        Disp_Wyswietl_txt(LCD_EKRAN_SUMA_ER, 0);
        GPIO_Linie_IO_High(LINIA_LED1_WY);
    }
    Delay_ms(1000);
}

```

## Konfigurowanie przerwania Timera

Zatrzaśnięcie chwilowej wartości Timera przy opadającym zboczu i odliczenie czasu przez *Stoper1* będą generowały przerwania. Procedurę przerwania standardowo umieszcza się w pliku *stm32f10x\_it.c*. Na końcu pliku należy dodać procedurę obsługi przerwania oraz zainicjować NVIC – listingu 5. Działanie procedury sprowadza się do ustawienia flag (zmiany wartości rejestrów) sygnalizujących zaistnienie zdarzenia. **Uwaga:** procedurę konfigurującą NVIC umieszczono w w pliku *Procedury\_Ogolne.c*. Pliki *stm32f10x\_it.c* i *Procedury\_Ogolne.c* umieściłem we wspólnym podkatalogu *Procedury\_Podstawowe*.

## Procedura obsługi transmisji z czujnika DHT11

Procedura znajduje się w pliku *DHTxx\_Procedury.c* w podkatalogu *Procedury\_SINGLE\_WIRE*. Cała procedura zamieszczona na listingu 6. Na początku procedura odczytuje poziom na linii DATA. Jeżeli jest on niski, procedura zwraca kod błędu *STATUS\_BLAD\_ZWARCIE*. Potem jest generowany impuls *Start* o czasie trwania 18 ms. Do odmierzenia czasu impulsu użyto czasomierza *Stoper1*. Jeżeli w określonym czasie czujnik nie odpowie oznacza to albo przerwanie magistrali, albo brak dołączonego czujnika. W takim wypadku procedura kończy się zwracając kod błędu *STATUS\_KLAD\_PRZERWA*.

Następnie w pętli procedura oczekuje na wystąpieniu 41 kolejnych zboczy opadających w trakcie transmisji z czujnika. Jeżeli podczas odbioru transmisja zostanie przerwana procedura kończy się z kodem błędu. Następnie procedura przystępuje do odtwarzania odebranych bajtów transmisji z czujnika. Jeżeli czas pomiędzy pojawieniem się kolejnych zboczy opadających był bliski 80  $\mu$ s oznacza to, że przesłany z czujnika bit miał wartość 0. Jeżeli czas był bliski 120  $\mu$ s oznacza to, że przesłany bitem była 1. Na koniec procedura oblicza sumę kontrolną i porównuje z przesłaną. W wypadku niezgodności jest zwracany kod błędu *STATUS\_BLAD\_SUMY*. Jeżeli wszystko przebiegło pomyślnie, procedura zwraca kod *STATUS\_SUKCES*. Odczytane bajty pomiarów po kolei zapisane zostają w buforze, do którego wskaźnik został przesłany w wywołaniu procedury.

## Wszystko razem

W głównej procedurze *main.c* najpierw są inicjowane porty, timer, NVIC i inne. Następnie, w nieskończonej pętli jest wykonywany odczyt z czujnika i wyświetlanie wyniku pomiaru na wyświetlaczu LCD. Świecenie diod LED sygnalizuje aktualny status urządzenia. Pętlę główną programu pokazano na listingu 7. Opisane procedury będą działały z różnymi czujnikami: DHT11, DHT22, AM232, AM2303.

Ryszard Szymaniak, EP

REKLAMA

# ELEKTRONIKA PRAKTYCZNA



Zaprenumeruj na stronie AVT.pl, e-mail: [prenumerata@avt.pl](mailto:prenumerata@avt.pl)  
lub telefonicznie pod numerem: 22 257 84 99  
Bieżący numer zamów na [www.ulubionykiosk.pl](http://www.ulubionykiosk.pl)