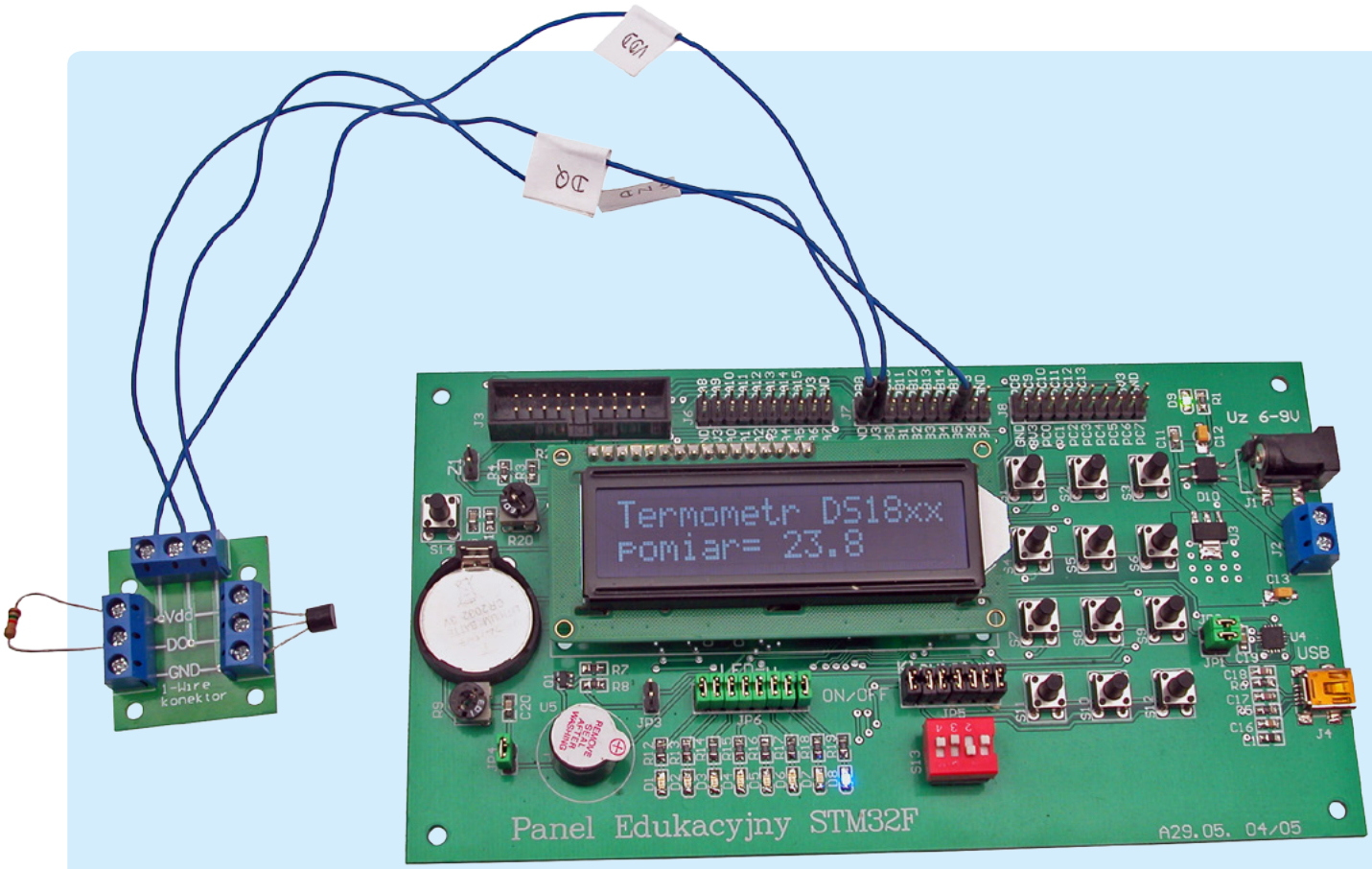


Połączenie DS18B20 z STM32. Projekt dla środowiska CooCox

Artykuł stanowi uzupełnienie kursu „STM-y nie tylko dla początkujących". Pokazane zostaną przykłady procedur pozwalających kontrolerowi STM32 obsłużyć magistralę 1-Wire i podłączony termometr np. popularny DS18B20. Drugim celem będzie przybliżenie darmowego środowiska programistycznego CooCox. Za jego pomocą i korzystając z opisywanych procedur będzie można szybko napisać własny program dla STM32.

Do wykonania oprogramowania dla kontrolera potrzebne jest środowisko pozwalające na przetworzenie plików źródłowych w binarny kod wynikowy nadający się do zapisania do pamięci Flash. Pełne środowisko programistyczne to: edytor kodu źródłowego, oprogramowanie do zarządzania plikami projektu, kompilator, debuger, obsługa programatora do zapisu do pamięci Flash kontrolera. Takim bezpłatnym IDE (Integrated Development Enviroment) jest CooCox. Projekt procedur obsługujących magistralę 1-Wire będzie wykonany z jego użyciem, jednak pliki mogą być łatwo przystosowane dla dowolnego, innego kompilatora języka C dla STM32.

Jako baza sprzętowa do eksperymentu posłuży Panel Edukacyjny AVT5465 z kontrolerem STM32F103RC. Procedury magistrali 1-Wire dadzą się uruchomić także na dowolnym innym kontrolerze z rodziny STM32.

Co to jest CooCox?

CooCox jest przeznaczony dla kontrolerów ARM z rdzeniem Cortex M0/M0+/M3/M4. Obsługiwane są układy takich firm jak Atmel, Energy Micro, Feescale, Holtek, Nuvoton, NXP, Spansion, ST, TI, Toshiba. CooCox bazuje na platformie Eclipse napisanej w Javie. W istocie jest nakładką na Eclipse z obsługą wtyczek do kompilatora, debugera itd. Integruje wszystkie składniki tworząc gotowe narzędzie o dość intuicyjnym działaniu. Dodatkowo oferuje zestaw gotowych szablonów do budowy nowych projektów dla różnych typów kontrolerów a także dostęp do bazy przykładowych projektów. Dzięki temu można szybko opanować podstawy jego obsługi.

Instalacja CooCoxa

CooCox pracuje z systemem operacyjnym Windows. Został przetestowany ze starym Windows XP jak

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11877, pass: ragjkdt9

i z nowszym Windows 7. Instalacja pakietu przebiega w kilku prostych krokach i zazwyczaj jest bezproblemowa. Opiszę kolejne etapy zwracając uwagę na kluczowe momenty.

1. Instalowanie pakietu kompilatora GNU Tools for ARM.

CooCox z definicji nie jest wyposażony w kompilator natomiast bezproblemowo może współpracować z pakietem kompilatorów GNU Tools for ARM. Pakiet należy zainstalować w pierwszej kolejności. Pod adresem *https://launchpad.net/gcc-arm-embedded/+download* znajduje się strona projektu GNU GCC. Należy ściągnąć plik oznaczony „Windows installer". Plik „waży" ponad 90 MB i automatycznie zainstaluje w systemie pakiet kompilatora. W momencie pisania tekstu była to wersja gcc-arm-none-eabi-4_9-2015q1-20150306-win32.exe.

Po uruchomieniu plik będzie instalował oprogramowanie. Najbezpieczniej zgodzić się na domyślne lokalizacje proponowane przez instalator. Uwaga! Wśród dostępnych powinna być zaznaczona opcja „Add path to enviroment variable". Pozwala to na automatyczne dodanie do zmiennej środowiskowej „path" systemu Windows ścieżki dostępu do plików kompilatora.

2. Instalowanie CooCox CoIDE lub CoCenter.

W następnej kolejności należy ze strony CooCox ściągnąć IDE, czyli środowisko programistyczne. Aby to zrobić należy najpierw się zarejestrować na stronie *http://www1.coocox.org/CooCox_CoIDE.htm*. Jak na razie jest to jedyne ograniczenie w dostępie.

Samo środowisko można zainstalować na dwa sposoby – poprzez plik programu CoCenter automatycznie instalującego różne dodatkowe programy z grupy CooCox lub bezpośrednio. Jeżeli kogoś nie interesuje oprogramowanie dodatkowe lub napotkamy trudności przy pracy z CoCenter, można pobrać plik CoIde korzystając na stronie *http://www1.coocox.org/CooCox_CoIDE.htm* z opcji *Download the latest CoIDE directly*. Plik ma wielkość ponad 450 MB.

3. Ścieżka dostępu do pakietu kompilatora.

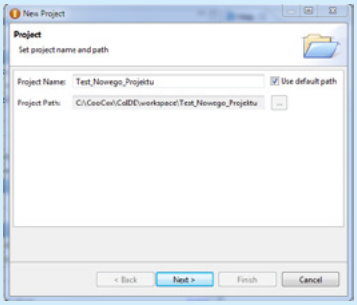
Po zainstalowaniupakietu programistycznego CoIde można go uruchomić i ustawić ścieżkę dostępu do kompilatora GNU GCC. Zależnie od ustawień systemu operacyjnego Windows CooCox CoIde może dać się uruchomić bezpośrednio z zainstalowanego podczas instalacji na pulpicie skrótu, czasami trzeba to będzie zrobić z uprawnieniami administratora. Sam proces uruchamiania trwa dosyć długo, niekiedy pojawiają się ostrzeżenia o problemach którymi zazwyczaj można się nie przejmować. Należy wybrać z menu opcję: *Project->Select Toolchain Path* i podać ścieżkę dostępu do podkatalogu /bin kompilatora GNU GCC. Przykładowo może ona wyglądać tak: *C:\Program Files\GNU Tools ARM Embedded\4.8 2014q3\bin*.

Od tego momentu CooCox CoIDE nadaje się do pracy.

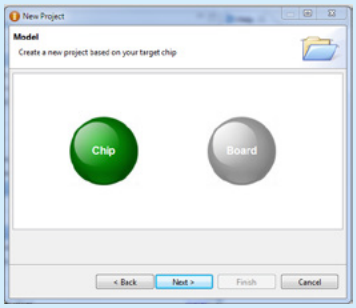
Nowy projekt

Pracując z pakietem CooCox CoIDE utworzenie nowego projektu jest proste i intuicyjne. Pokażę w punktach jak to się robi.

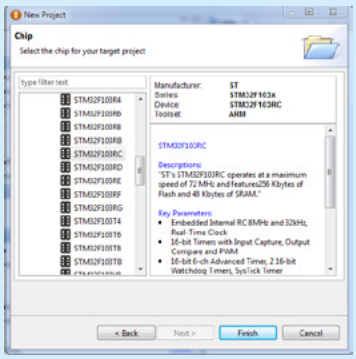
1. Wybór z menu opcji *Project–>New Project* otwiera okno, w którym należy podać katalog docelowy do zapamiętywania plików nowego projektu. Jeżeli pozostanie zaznaczona opcja „Use default path", pliki zostaną zapisane w standardowym podkatalogu programu „C:\CooCox\CoIDE\workspace". Jeżeli opcja nie zostanie zaznaczona, można wybrać dowolną lokalizację. Dodatkowo, trzeba podać nazwę wybraną dla nowego projektu, tak jak na **rysunku 1**.
2. Następnie należy określić czy projekt ma dotyczyć płytki ewaluacyjnej np. typu „Discovery" czy będzie tworzony dla samego kontrolera, tak jak na **rysunku 2**.
3. Jeżeli (jak w przykładzie) projekt dotyczy kontrolera, należy wskazać na liście jego typ, co pokazano na **rysunku 3**.
4. Korzystając z menu *View → Repository* można otworzyć listę plików związanych z biblioteką standardową dla danego typu kontrolera. Zaznaczone pliki zostaną automatycznie dodane do projektu. W najprostszym przypadku, gdy oprogramowanie będzie tylko korzystało tylko z portów GPIO, należy wybrać następujące pozycje:
   - C Library,
   - CMSIS core,
   - CMSIS Boot,
   - RCC,
   - GPIO,
   - MISC.



Rysunek 1. Podanie nazwy projektu



Rysunek 2. Wybór płytki ewaluacyjnej

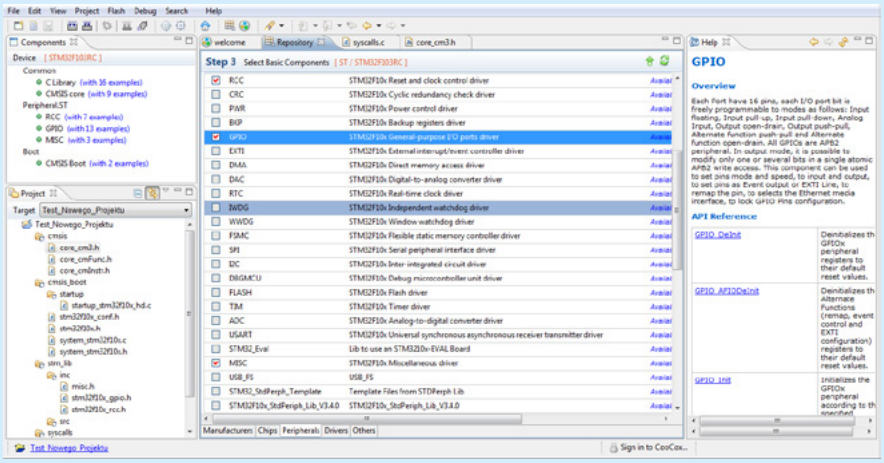


Rysunek 3. Wybór typu mikrokontrolera

Do projektu automatycznie zostaną dodane pliki nagłówkowe i pliki biblioteki między innymi konfigurujące sygnały zegarowe kontrolera, pozwalające na sterowanie wyprowadzeniami GPIO, konieczne do obsługi samej biblioteki standardowej itd. Pulpit CooCox CoIDE będzie wyglądał podobnie do pokazanego na **rysunku 4**. W katalogu projektu zostaną automatycznie utworzone podkatalogi z dodanymi plikami. Struktura podkatalogów będzie identyczna jak w umieszczonym z lewej stronie pulpitu oknie „Project". Kliknięcie na nazwę pliku w tym okienku powoduje jego otwarcie do edycji w centralnej części pulpitu. Do struktury plików dodany został także plik ze szkieletem procedury *main()*.

Wybór programatora

Wybór z menu *View → Configuration → Debugger* pozwala na wybranie z listy *Adapter* posiadanego programator. Jeżeli to będzie np. ST-Link będzie służył zarówno do programowania jak i do debugowania. W opcji *Download* można zmienić ustawienia związane ze sposobem zapisu do pamięci FLASH kontrolera.



Rysunek 4. Wygląd pulpitu CooCox CoIDE

Ważne pliki projektu

Pewne ustawienia w konkretnych plikach projektu mają szczególne znaczenie dla kompilacji i utworzenia pliku wynikowego.

- ***cmsis_boot/stm32f10x.h*** – w pliku tym znajduje się deklaracja wartości stałej HSE_VALUE. W deklaracji przypisuje się wartość w hercach częstotliwości zewnętrznego rezonatora kwarcowego. Powinna ona odpowiadać rzeczywiście zastosowanemu rezonatorowi. Żeby uniknąć zmieniania pliku stm32f10x.h można zadeklarować wartość stałej HSE_VALUE w oddzielnym pliku nagłówkowym utworzonym przez użytkownika dla np. własnych globalnych definicji programu. Deklaracja może wyglądać następująco: *#define HSE_VALUE ((uint32_t)8000000) /*!< Value of the External oscillator in Hz */.*
- ***cmsis_boot/stm32f10x_conf.h*** – plik zawiera listę plików nagłówkowych, które powinny być dołączone do programu podczas kompilacji. Wstępnie wszystkie pliki są opatrzone komentarzem Należy usunąć komentarz z nazw dołączanych plików np.

```
#include „stm32f10x_gpio.h”
#include „stm32f10x_rcc.h”
#include „misc.h”
```

- ***cmsis_boot/system_stm32f10x.c*** – w tym pliku można wybrać częstotliwość głównego zegara systemowego. Dokonuje się tego przez usunięcie komentarza z wybranej deklaracji np.

```
/* #define SYSCLK_FREQ_HSE    HSE_VALUE */
/* #define SYSCLK_FREQ_24MHz  24000000 */
/* #define SYSCLK_FREQ_36MHz  36000000 */
/* #define SYSCLK_FREQ_48MHz  48000000 */
/* #define SYSCLK_FREQ_56MHz  56000000 */
#define SYSCLK_FREQ_72MHz  72000000
```

Dodawanie do projektu własnych plików użytkownika

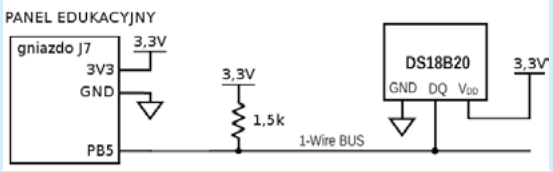
Użytkownik może dodawać do projektu własne podkatalogi grupujące dodatkowe pliki z własnymi procedurami. Dla utrzymania porządku drzewo katalogów w okienku „Project” na pulpicie powinno odzwierciedlać rzeczywistą strukturę dodawanych podkatalogów w katalogu projektu. Kliknięcie prawym przyciskiem myszy w obrębie okienka „Project” wyświetla menu pozwalające dodawać podkatalogi, włączać do nich istniejące pliki, usuwać itp.

Rysunek 5. Oznaczenie wyprowadzeń DS18B20 (obudowa TO-92)

Założenia do projektu testującego procedury 1-Wire

Kiedy wiadomo już jak zainstalować CooCox CoIDE, pora na stworzenie projektu do testu procedur magistrali 1-Wire przyłączonej do kontrolera STM32. Ponieważ jako płytka testowa posłuży Panel Edukacyjny kontrolerem docelowym projektu będzie STM32F103RC. W przypadku tworzenia projektu dla innej płytki należy zadeklarować właściwy typ kontrolera. W opisywanych dalej procedurach magistrala 1-Wire obsługiwana jest przez port PB5. Numer portu może być w zasadzie dowolny, trzeba jedynie w pliku nagłówkowym zadeklarować inny wybrany port. Kontroler poprzez magistralę będzie się komunikował z czujnikiem DS18B20 lub DS18S20. Oznaczenie końcówek tego elementu w obudowie TO-92 pokazane zostało na **rysunku 5**. Z pozostałych dwu wyprowadzeń termometru jedno należy podłączyć do masy a drugie oznaczone Vdd do napięcia +3,3 V. Schemat połączenia elementu DS18B20 z Panelem Edukacyjnym pokazano na **rysunku 6**. Dodatkowym elementem jest rezystor 1,5 kΩ podciągający magistralę do napięcia zasilania +3,3 V. Połączenie końcówki Vdd do napięcia zasilającego nie jest niezbędne, jednak poprawia zasięg i stabilność pracy szczególnie, jeśli do magistrali miało by być dołączonych więcej czujników niż jeden. Jeżeli wyprowadzenie Vdd nie będzie podłączone do zasilania, musi zostać zwarte do masy.

W przygotowanym projekcie o nazwie *Termometr_LCD* założono, że kontroler będzie czytał temperaturę z jednego czujnika dołączonego do magistrali. Jednak bardzo łatwo projekt można przystosować do obsługi



Rysunek 6. Schemat połączenia DS18B20 z Panelem Edukacyjnym

Listing 1. Deklaracje portu mikrokontrolera, do którego będzie dołączona magistrala 1-Wire

```
#define LINIE_1Wn         2
typedef enum
{
    LINIA_1_WIRE_WY = 0,
    LINIA_1_WIRE_WE = 1
}Linie_1W_TypeDef;

//LINIA_1_WIRE_WY linia wyjściowa sterująca magistralą 1-Wire
#define LINIA_1_WIRE_WY_PORT        GPIOB
#define LINIA_1_WIRE_WY_PORT_NUM    GPIO_PortSourceGPIOB
#define LINIA_1_WIRE_WY_CLK         RCC_APB2Periph_GPIOB
#define LINIA_1_WIRE_WY_PIN         GPIO_Pin_5
#define LINIA_1_WIRE_WY_PIN_SOURC   GPIO_PinSource5
//LINIA_1_WIRE_WE linia wejściowa czytająca z magistral1 1-Wire
#define LINIA_1_WIRE_WE_PORT        GPIOB
#define LINIA_1_WIRE_WE_PORT_NUM    GPIO_PortSourceGPIOB
#define LINIA_1_WIRE_WE_CLK         RCC_APB2Periph_GPIOB
#define LINIA_1_WIRE_WE_PIN         GPIO_Pin_5
#define LINIA_1_WIRE_WE_PIN_SOURC   GPIO_PinSource5
```

Listing 2. Procedury wykorzystywane do obsługi transmisji 1-Wire

```
const uint32_t GPIO_LINIA_1W_CLK[LINIE_1Wn]={
        LINIA_1_WIRE_WY_CLK, LINIA_1_WIRE_WE_CLK};

const uint16_t GPIO_LINIA_1W_PIN[LINIE_1Wn]={
        LINIA_1_WIRE_WY_PIN, LINIA_1_WIRE_WE_PIN};

GPIO_TypeDef*  GPIO_LINIA_1W_PORT[LINIE_1Wn]={
        LINIA_1_WIRE_WY_PORT, LINIA_1_WIRE_WE_PORT};

//-------------------------------------------------------------
//procedura inicjacji linii 1-Wire
void GPIO_Linie_1W_Konfig(Linie_1W_TypeDef Linia, GPIOMode_TypeDef tryb_pracy)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    /* Enable the GPIO_Linie_1W Clock */
    RCC_APB2PeriphClockCmd(GPIO_LINIA_1W_CLK[Linia], ENABLE);
    /* Configure the GPIO_Linie_1W pin*/
    GPIO_InitStructure.GPIO_Pin = GPIO_LINIA_1W_PIN[Linia];
    GPIO_InitStructure.GPIO_Mode =tryb_pracy;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIO_LINIA_1W_PORT[Linia], &GPIO_InitStructure);
}

//---------------------------------------
//ustawienie stanu wysokiego na linii 1-Wire
void GPIO_Linie_1W_High(Linie_1W_TypeDef Linia)
{
    GPIO_LINIA_1W_PORT[Linia]->BSRR =GPIO_LINIA_1W_PIN[Linia];
}
//---------------------------------------
//ustawienie stanu niskiego na linii 1-Wire
void GPIO_Linie_1W_Low(Linie_1W_TypeDef Linia)
{
    GPIO_LINIA_1W_PORT[Linia]->BRR =GPIO_LINIA_1W_PIN[Linia];
}
//---------------------------------------
//procedura odczytu stanu linii 1-Wire
uint32_t GPIO_Linie_1W_Odczyt(Linie_1W_TypeDef Linia)
{
    return GPIO_ReadInputDataBit(GPIO_LINIA_1W_PORT[Linia], GPIO_LINIA_1W_PIN[Linia]);
}
//---------------------------------
//procedura inicjacji linii portów
void GPIO_1W_Inicjacja(void)
{
    //GPIO_Linie_1W_Konfig(Linie_1W_TypeDef Linia, GPIOMode_TypeDef tryb_pracy)
    GPIO_Linie_1W_Konfig(LINIA_1_WIRE_WY, GPIO_Mode_Out_OD);
    GPIO_Linie_1W_High(LINIA_1_WIRE_WY);
}
```

jednocześnie wielu układów DS18B20. Po odczycie danych z czujnika podlegają one konwersji na standardową postać temperatury w °C i są wyświetlane na wyświetlaczu LCD Panelu. Dodatkowo świecenie diod LED sygnalizuje fakt odczytu danych z czujnika a po prawidłowej konwersji sukces zakończenia operacji. Kolejne odczyty temperatury odbywają się w niekończącej pętli o czasie trwania cyklu ok. 2 sekund.

Struktura plików procedur obsługi magistrali 1-Wire

Pliki procedur zgrupowane zostały w dwu podkatalogach: „Procedury 1Wire" dla plików źródłowych i „procedury_1Wire" dla plików nagłówkowych. Dzięki temu powinno być ułatwione przenoszenie plików do innych projektów, w których będą wykorzystywane. Poniżej pokazano strukturę katalogów procedur:

/inc/procedury_1Wire
- Definicje_1Wire.h
- GPIO_1W_procedury.h
- Procedury_1W.h

/Procedury 1Wire
- GPIO_1W_procedury.c
- Procedury_1W.c

Deklaracje portu sterującego magistralą 1-Wire

W pliku *Definicje_1Wire.h* znajdują się deklaracje portu mikrokontrolera, do którego będzie dołączona magistrala 1-Wire – pokazano je na **listingu 1**.

Formalnie zadeklarowane zostały dwie linie: wyjściowa i wejściowa. Jednak obie wskazują na ten sam port PB5. Dwa oznaczenia pomagają łatwiej pokazać sytuacje, w których port odczytuje dane z magistrali lub je wysyła.

Listing 3. Podstawowe procedury obsługi transmisji 1-Wire

```
//procedura inicjacji obsługi magistrali 1-Wire
void Inicjacja_1_Wire(void)
{
    GPIO_1W_Inicjacja();
}
//-----------------------------------------------------
//Generowanie impulsu Reset na magistrali 1-Wire
uint8_t OW_reset(void)
//wy: OW_PRESENCE -odebrane potwierdzenie PRESENCE,
//    OW_NOPRESENCE -brak potwierdzenia PRESENCE
{
uint8_t presence;

    GPIO_Linie_1W_Low(LINIA_1_WIRE_WY);  //start impulsu RESET
    Delay_us(480);  //pauza 480us
    GPIO_Linie_1W_High(LINIA_1_WIRE_WY);
    Delay_us(70);  //pauza 70us
    presence =GPIO_Linie_1W_Odczyt(LINIA_1_WIRE_WE);  //test impulsu Presence
    if (presence ==0)
    {
        Delay_us(480);
        return(OW_PRESENCE);
    }
    Delay_us(30);  //pauza 30us
    presence =GPIO_Linie_1W_Odczyt(LINIA_1_WIRE_WE);  //2 test impulsu Presence
    Delay_us(480);
    if (presence ==0) return(OW_PRESENCE); else return(OW_NOPRESENCE);
}
//-----------------------------------------------------
//Zapisanie bitu na magistralę 1-Wire
void OW_write_bit(unsigned char bitval)
//we:    wartość bitu 0 lub 1
{
    if (bitval == 1)  //zapisanie 1 na 1wire
    {
        GPIO_Linie_1W_Low(LINIA_1_WIRE_WY);  //impuls bitu
        Delay_us(6);  //pauza 6us dla bitu ,1'
        GPIO_Linie_1W_High(LINIA_1_WIRE_WY);  //koniec impulsu
        Delay_us(64);  //pauza 64us
    }
    else  //zapisanie 0 na 1wire
    {
        GPIO_Linie_1W_Low(LINIA_1_WIRE_WY);  //impuls bitu
        Delay_us(70);  //pauza 70us dla bitu ,0'
        GPIO_Linie_1W_High(LINIA_1_WIRE_WY);  //koniec impulsu
    }
    Delay_us(1);  //pauza 1us
}
//-----------------------------------------------------
//Odczyt bitu z magistrali 1-Wire
unsigned char OW_read_bit(void)
//wy: wartość bitu 0 lub 1
{
unsigned char val;
    GPIO_Linie_1W_Low(LINIA_1_WIRE_WY);  //impuls bitu
    Delay_us(6);  //pauza 6us
    GPIO_Linie_1W_High(LINIA_1_WIRE_WY);  //koniec impulsu
    Delay_us(9);
    val=GPIO_Linie_1W_Odczyt(LINIA_1_WIRE_WE);  //odczyt stanu magistrali 1-Wire
    val &= 0x01;  //zerowanie bitów z wyjątkiem b.0
    Delay_us(44);  //pauza 44us
    return (val);
}
//-----------------------------------------------------
//Zapis bajtu na magistralę 1-Wire
void OW_write_byte(unsigned char val)
//we: bajt 0-255
{
    u8 a;
    u8 temp;
    Delay_us(1);  //pauza 1us
    for(a=0;a<8;a++)  //pętla dla zapisu 8 bitów
    {
        temp = val >> a;  //kolejny bit bajtu
        temp &=0x01;
        OW_write_bit(temp);  //zapis bitu na 1wire
    }
    Delay_us(1);  //pauza 1us
}
//-----------------------------------------------------
//Odczyt bajtu z magistrali 1-Wire
unsigned char OW_read_byte(void)
//wy: odczytany bajt 0-255
{
    unsigned char i;
    unsigned char value=0;
    for (i=0;i<8;i++)  //pętla dla odczytu 8 bitów
    {
        if(OW_read_bit()) value |= (0x01<<i);  //jeżeli odczytany bit to ,1' ustawianie bitu
w bajcie
    }
    Delay_us(10);  //pauza 10us
    return(value);
}
```

Listing 4. Złożone procedury magistrali 1-Wire

```
//-----------------------------------------------------
//Wysłanie 8 bajtowego kodu ROM na magistralę 1-Wire
void Send_rom(uint8_t* adr)
//we:    wskaźnik do buforu z kodem ROM
{
uint8_t a;
    for(a=0;a<8;a++)  //pętla 8 bajtów (ID ROM)
        OW_write_byte(adr[a]);  //wysłany kolejny bajt (ID ROM)
}
//-----------------------------------------------------
//inicjacja wyszukania i odczytu pierwszego ROM elementu dołączonego do magistrali
uint8_t OW_search_first(uint8_t *ROM)
//wy:    status wyszukania
{
    OW_LastDiscrepancy = 0;  //zerowanie zmiennych
    OW_LastDevice = 0;  // -//-
    OW_LastFamilyDiscrepancy = 0;  // -//-
    return OW_search_next(ROM);  //pierwszy odczyt
}
//-----------------------------------------------------
uint8_t OW_search_next(uint8_t *ROM)
{
    uint8_t     bit_test, search_direction;
    uint8_t bit_number = 1;
    uint8_t last_zero = 0;
    uint8_t rom_byte_number = 0;
    uint8_t rom_byte_mask = 1;
    uint8_t lastcrc8 = 0;
    uint8_t crcaccum = 0;
    int8_t next_result = OW_NOMODULES;
    // if the last call was not the last one
    if(!OW_LastDevice) {
        if(OW_reset() == OW_NOPRESENCE) {
            OW_LastDiscrepancy = 0;
            OW_LastFamilyDiscrepancy = 0;
            return OW_NOPRESENCE;
        }
        OW_write_byte(OW_SEARCH_ROM);
        Delay_us(100);
        do {
            bit_test = OW_read_bit() << 1;
            bit_test |= OW_read_bit();
            if(bit_test == 3) {
                return(OW_BADWIRE);
            } else {
                if(bit_test > 0) {
                    search_direction = !(bit_test & 0x01);  // bit write value for search
                } else {
// if this discrepancy is before the Last Discrepancy
//on a previous OWNext then pick the same as last time
                    if(bit_number < OW_LastDiscrepancy) {
                        search_direction = ((*(ROM+rom_byte_number) & rom_byte_mask) > 0);
                    } else {
// if equal to last pick 1, if not then pick 0
                        search_direction = (bit_number == OW_LastDiscrepancy);
                    }
// if 0 was picked then record its position in LastZero
                    if (search_direction == 0) {
                        last_zero = bit_number;
// check for Last discrepancy in family
                        if (last_zero < 9) {
                            OW_LastFamilyDiscrepancy = last_zero;
                        }
                    }
                }
// set or clear the bit in the ROM byte rom_byte_number
// with mask rom_byte_mask
                if(search_direction == 1) {
                    *(ROM+rom_byte_number) |= rom_byte_mask;
                } else {
                    *(ROM+rom_byte_number) &= ~rom_byte_mask;
                }
// serial number search direction write bit
                OW_write_bit(search_direction);
// increment the byte counter bit_number and shift the mask rom_byte_mask
                bit_number++;
                rom_byte_mask <<= 1;
// if the mask is 0 then go to new ROM byte rom_byte_number
// and reset mask
                if(rom_byte_mask == 0) {
                    OW_crc(*(ROM+rom_byte_number), &crcaccum);  // accumulate the CRC
                    lastcrc8 = crcaccum;
                    rom_byte_number++;
                    rom_byte_mask = 1;
                }
            }
        } while(rom_byte_number < 8);  // loop until through all ROM bytes 0-7
// if the search was successful then
        if(!(bit_number < 65) || lastcrc8) {
```

Listing 4. c.d.

```
            if(lastcrc8) {
                next_result = OW_BADCRC;
            } else {
//  search successful so set LastDiscrepancy,LastDevice,next_result
                OW_LastDiscrepancy = last_zero;
                OW_LastDevice = (OW_LastDiscrepancy == 0);
                next_result = OW_FOUND;
            }
        }
    }
// if no device found then reset counters so next ‚next' will be
// like a first
//if(next_result != OW_FOUND || ROM[0] == 0) {
        if(next_result != OW_FOUND || *(ROM) == 0) {
            OW_LastDiscrepancy = 0;
            OW_LastDevice = 0;
            OW_LastFamilyDiscrepancy = 0;
        }
        if(next_result == OW_FOUND && *(ROM) == 0x00) {
        next_result = OW_BADWIRE;
    }
    return next_result;
}
//--------------------------------------------------
//CRC tabela
uint8_t dscrc_table[] = {
0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65,
157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220,
35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98,
190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255,
70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7,
219, 133,103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154,
101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36,
248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185,
140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242,  172, 47, 113,147, 205,
17, 79, 173, 243, 112, 46, 204, 146, 211,141, 111, 49, 178, 236, 14, 80,
175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82,176, 238,
50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115,
202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139,
87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22,
233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168,
116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53};
//--------------------------------------------------
void OW_crc(uint8_t x, uint8_t *crc)
{
    *crc = (uint8_t)dscrc_table[(*crc) ^ x];
}
```

Inicjacja portu sterującego magistralą 1-Wire

Na początku zanim cokolwiek zaczniemy wysyłać lub odczytywać z magistrali należy zainicjować port sterujący, przykładowo PB.5. Port jest ustawiany w trybie wyjścia z otwartym drenem. Jego obciążeniem jest dołączany na zewnątrz opornik 1,5 kΩ (rys. 6). W tym trybie jest możliwe manipulowanie poziomem napięcia wyjściowego portu oraz odczytanie poziomu na porcie. Odpowiednie procedury znajdują się w pliku *GPIO_1W_procedury.c*. Oprócz inicjacji portu w pliku umieszczone zostały procedury ustawiania portu w stanie wysokim, niskim oraz odczytu stanu portu. Na **listingu 2** zamieszczono wszystkie z wymienionych procedur. Na samym początku znajdują się tabele ze stałymi pozwalające w procedurach posługiwać się zadeklarowanymi w pliku *Definicje_1Wire.h* nazwami symbolicznymi: LINIA_1_WIRE_WY i LINIA_1_WIRE_WE.

Elementarne procedury magistrali 1-Wire

W pliku *Procedury_1W.c* znajdują się zasadnicze procedury komunikacji z termometrem DS18B20 poprzez magistralę 1-Wire. Można je podzielić na procedury elementarne i złożone. Pierwsze bezpośrednio współpracują z procedurami sterującymi portem PB.5. Są to:

- *Inicjacja_1_Wire* – inicjowanie 1-Wire, czyli tylko inicjowanie trybu pracy portu.
- OW_reset – procedura generowanie impulsu RESET i odbioru potwierdzenia PRESENCE z czujnika DS18B20.
- OW_write_bit – zapisanie pojedynczego bitu na magistralę 1-Wire.
- OW_read_bit – odczytanie pojedynczego bitu z magistrali 1-Wire.
- OW_write_byte – wysłanie zmiennej 1-bajtowej na magistralę.
- OW_read_byte –odczytanie z magistrali zmiennej 1-bajtowej.

Procedury do prawidłowego działania wymagają odmierzania odcinków czasu rzędu mikrosekund. Realizuje to zewnętrzna procedura *Delay_us()*, która odmierza czas za pomocą pętli programowej, co zapewnia wystarczającą precyzję. Wymienione procedury zaprezentowano na **listingu 3**.

Złożone procedury obsługi magistrali 1-Wire

Procedury złożone odwołując się do procedur elementarnych obsługują komunikację z elementem dołączonym do magistrali. Są to:

- Send_rom – wysyłanie 8 bajtów adresu wybranego czujnika (termometru).
- OW_search_first – przy pracy z wieloma czujnikami (termometrami) dołączonymi do magistrali inicjacja identyfikacji adresów kolejnych czujników.

Listing 5. Procedury odczytu i konwersji temperatury z czujników DS18B20

```
//procedura rejestracji podłączonego termometru: czujnika DS18B20
char Rejestracja_termometru(void)
{
uint8_t presence;
    presence =OW_search_first(&ROM[0]);
    if (presence ==OW_FOUND) return TRUE; else return FALSE;
}
//-----------------------------------------------------
//procedura pomiaru temperatury przez dołączony czujnik
//adres czujnika w tabeli ROM[]
uint8_t Procedura_pomiaru_temperatury(float *p_temperatura_term)
//wy: TRUE -sukces, FALSE -błąd
{
    #define TYP_DS18B20  0x28
    #define TYP_DS18S20  0x10
    char typ_termometru;
    uint8_t status;
    uint8_t scratchpad_tab[9], y, crc;

    typ_termometru =ROM[0];  //najmłodsze 8 bitów adresu ROM
    status =OW_reset();
    if (status ==OW_NOPRESENCE) return FALSE;  //błąd zarejestrowany czjnik nie daje się odczytać
    OW_write_byte(OW_SKIP_ROM);  //Skip ROM command
    OW_write_byte(OW_CONVERT_T);  //Convert T command
    GPIO_Linie_1W_Konfig(LINIA_1_WIRE_WY, GPIO_Mode_Out_PP);  //mocne podciągnięcie 1-Wire do Vdd
    Delay_ms(1000);  //czas konwersji 1s
    GPIO_Linie_1W_Konfig(LINIA_1_WIRE_WY, GPIO_Mode_Out_OD);  //powrót do trybu pracy wyprowadzenia
jako otwarty dren
    status =OW_reset();
    if (status ==OW_NOPRESENCE) return FALSE;  //błąd zarejestrowany czujnik nie daje się odczytać
//odczyt danych temperatury z czujnika
    OW_write_byte(OW_MATCH_ROM);  //Match ROM command
    Send_rom(&ROM[0]);  //64-bit ROM code
    OW_write_byte(OW_READ_SCRATCHPAD);  //Read Scratchpad command
//Read Scratchpad 9 data bytes
    crc =0;
    for (y=0; y<9; y++)
    {
        scratchpad_tab[y] =OW_read_byte();
        if (y<8)
        {
            OW_crc(scratchpad_tab[y], &crc);
        }
    }
    if (crc ==scratchpad_tab[8] & crc !=0)
    {
    //sumy kontrolne zgodne
        if (typ_termometru ==TYP_DS18B20)
        {
            *p_temperatura_term =
            Konwersja_rejestry_temperatura_DS18B20(scratchpad_tab[1], scratchpad_tab[0]);
        }
        if (typ_termometru ==TYP_DS18S20)
        {
            *p_temperatura_term =
            Konwersja_rejestry_temperatura_DS18S20(scratchpad_tab[1], scratchpad_tab[0]);
        }
        if ((typ_termometru !=TYP_DS18B20) && (typ_termometru !=TYP_DS18S20))
        {
            //nie obsługiwany typ termometru, sygnalizacja błędu
            return FALSE;
        }
    }
    else
    {
        //błąd sumy kontrolnej
        return FALSE;
    }
return TRUE;
}
//-----------------------------------------------------
//konwersja odczytanych rejestrów na wartość zmierzonej temperatury dla termometru DS18B20
float Konwersja_rejestry_temperatura_DS18B20(char rejestr_MSB, char rejestr_LSB)
{
    #define STALA_TEMPERATURY_12B  0.0625
    float stala, obliczona_temperatura;
    int wartosc_int;

    wartosc_int =rejestr_MSB <<8;
    wartosc_int =wartosc_int | rejestr_LSB;
    if ((rejestr_MSB &0x80) !=0)
    {
        stala =-STALA_TEMPERATURY_12B;
        wartosc_int =(~wartosc_int) & 0xFFFF;
        wartosc_int++;
    }
    else stala =STALA_TEMPERATURY_12B;
    obliczona_temperatura =wartosc_int;
    obliczona_temperatura =obliczona_temperatura * stala;
```

Listing 5. c.d.

```
    return obliczona_temperatura;
}
//---------------------------------------------------
//konwersja odczytanych rejestrów na wartość zmierzonej temperatury dla termometru DS18S20 i DS1820
float Konwersja_rejestry_temperatura_DS18S20(char rejestr_MSB, char rejestr_LSB)
{
    #define STALA_TEMPERATURY_9B    0.5
    float stala, obliczona_temperatura;
    int wartosc_int;

    wartosc_int =rejestr_MSB <<8;
    wartosc_int =wartosc_int | rejestr_LSB;
    if ((rejestr_MSB &0x80) !=0)
    {
        stala =-STALA_TEMPERATURY_9B;
        wartosc_int =(~wartosc_int) & 0xFFFF;
        wartosc_int++;
    }
    else stala =STALA_TEMPERATURY_9B;
    obliczona_temperatura =wartosc_int;
    obliczona_temperatura =obliczona_temperatura * stala;
    return obliczona_temperatura;
}
```

- OW_search_next – przy pracy z wieloma czujnikami (termometrami) dołączonymi do magistrali procedura identyfikacji kolejnego czujnika.
- OW_crc – procedura obliczania sumy kontrolnej odebranych danych z magistrali 1-Wire.

Wymienione procedury pokazano na **listingu 4**.

Procedury odczytu i konwersji temperatury z czujników DS18B20

Opisane dalej procedury nie są właściwie związane z obsługą magistrali 1-Wire. Przeznaczone są do obsługi konkretnych czujników -termometrów poprzez odwołanie do procedur magistrali:

- Rejestrowanie_termometru – rejestrowanie czujnika DS18B20 dołączonego do magistrali. Właściwie zadaniem procedury jest odczytanie numeru czujnika i zapamiętanie go w statycznej tabeli ROM[]. Można ją łatwo dostosować do odczytu numerów wielu dołączonych do magistrali czujników i zapamiętania ich numerów w dwu wymiarowej tabeli ROM[].
- Procedura_pomiaru_temperatury – właściwa procedura inicjacji i odczytu zmierzonej przez czujnik temperatury.
- Konwersja_rejestry_temperatura_DS18B20 – konwersja danych odczytanych z czujnika DS18B20 na stopnie Celsiusa.
- Konwersja_rejestry_temperatura_DS18S20 – konwersja danych odczytanych z czujnika DS18S20 na stopnie Celsiusa.

Wymienione procedury przestawiono na **listingu 5**.

Wszystko razem

W głównej procedurze *main()* najpierw powinna nastąpić inicjacja procedur 1-Wire przez wywołanie *Inicjacja_1_Wire()*. Następnie powinien zostać odczytany numer dołączonego do magistrali termometru i zapisany w statycznej tabeli ROM[] poprzez wywołanie procedury *Rejestracja_termometru()*. Potem w pętli wywoływana jest procedura odczytu temperatury z dołączonego termometru przez

status =Procedura_pomiaru_temperatury (&temperatura_term);

Jeżeli status = TRUE, przesłana w zmiennej typu float *temperatura_term* może zostać wyświetlona na wyświetlaczu panelu LCD, po czym sytuacja się powtarza.

Ryszard Szymaniak, EP