STM32 dla początkujących (i nie tylko)

Nieco więcej niż podsumowanie

Ostatnia część kursu STM32 dla początkujących będzie zawierała krótkie podsumowanie dotychczas przekazanych informacji oraz znacznie bardziej obszerną listę tego, o czym jeszcze można by było napisać.

W pierwszej części opisano Panel Edukacyjny z STM32F103RC mający być tanią sprzętową bazą do eksperymentowania i umożliwienia zapoznania się z mikrokontrolerem STM32. Dodatkowo, krótko omówiono przydatne programy narzędziowe: dostępne kompilatory, firmową bibliotekę *STM32F10x Standard Peripherals Library*, oprogramowanie do zapisu do pamięci Flash. W kolejnym artykule omówiono sterowanie wyprowadzeniami wejścia/wyjścia. Podano podstawowe informacje ułatwiające pracę z pakietem kompilatora Keil oraz firmową biblioteką funkcji. Następnie przyszedł czas na opis układów czasowo-licznikowych (timerów). Przedstawiono użyteczny program narzędziowy *STM32CubeMX*. Opis timerów byłyby niepełny bez wiedzy na temat przerwań, więc w kolejnym artykule krótko omówiono idę funkcjonowania mechanizmu przerwań, sposobu ich uruchamiania i obsługi. Tematem następnego artykułu był interfejs szeregowy USART. Oprócz procedur obsługi omówiona została idea buforów kołowych wykorzystywanych do chwilowego przechowywania danych wysyłanych i odbieranych. W szóstej części pokazano przykładowe sposoby zaprzęgnięcia do pracy wewnętrznego zegara czasu rzeczywistego RTC do budowy czasomierza z alarmem. Bohaterem przedostatniej części cyklu był przetwornik analogowo-cyfrowy. Poglądowe programy demonstracyjne wiążące się z omawianym zagadnieniem miały pełnić funkcje dodatkowej pomocy.

Poruszone tematy to krótkie wprowadzenie w podstawy świata STM-ów. Każdy kontroler zwykle wyposażony jest w opisane peryferia. Ale rozbudowana rodzina STM32F ma znacznie więcej ciekawych funkcji.

Oznaczenia kontrolerów

Rodzina STM32 w ostatnich latach dynamicznie rozrasta się, co skutkuje skomplikowanym nazewnictwem. Nawet mikrokontroler o takim samym oznaczeniu głównym, ale innej obudowie może charakteryzować się nie tylko większa liczbą wyprowadzeń wejścia/wyjścia, ale również dostępnością dodatkowych modułów interfejsów. Pewien obraz sytuacji daje strona firmowa *http://goo.gl/jgqOSU*, na której zebrano wszystkie typy mikrokontrolerów grupując je według kilku kryteriów.

Linki do ciekawych miejsc w Internecie z informacjami nt. STM32

Garść przykładowych linków do interesujących miejsc w Internecie związanych z mikrokontrolerami STM32. Można tam znaleźć zarówno informacje, jak i gotowe przykłady użytecznych programów:
http://www.st.com/web/en/catalog/mmc/FM141 – firmowa strona STM.
http://stm32.eu/ – polska strona poświęcona STM32.
http://stm32f4-discovery.com/ – ciekawa strona z gotowymi przykładami, głównie dla STM32F4.

Podstawowym jest typ zastosowanego rdzenia Cortex, Aktualnie M0…M4, M7. Z reguły rdzenie o wyższym numerze mogą być taktowane szybszym zegarem i mają większe moce obliczeniowe. Drugim kryterium jest wciąż stosowany podział według wielkości wewnętrznej pamięci Flash:

- Low-density devices: 16…32 kB.
- Medium-density devices: 64…128 kB.
- High-density devices: 256…512 kB.
- XL-density devices: 768 kB…1 MB.
- Connectivity line devices: wydzielona grupa z interfejsami USB-OTG i Ethernet.

Ponieważ pojawiają się układy o pamięci Flash większej niż 1 MB, a interfejsy USB i Ethernet są dostępne także w innych liniach kontrolerów, ten podział powoli traci sens. Jest jednak używany w licznych przykładach programów demonstracyjnych.

Ważna jest także obudowa, a ściślej liczba dostępnych wyprowadzeń. I tak w mikrokontrolerze STM32F103RC zastosowanym w panelu, mającym 64 wyprowadzenia, nie ma dostępu do wewnętrznego przetwornika cyfrowo-analogowego, natomiast już mikrokontroler STM32F103VG w obudowie LQFP100 ma wyprowadzenia dwóch przetworników.

Unikatowy numer seryjny i sprzętowa suma kontrolna

Każdy mikrokontroler STM32F ma nadany fabrycznie, unikatowy numer seryjny. Numer jest zakodowany w postaci 96 bitów tylko do odczytu, umieszczonych

Listing 1. Przykładowa procedura odczytująca unikatowy numer mikrokontrolera

```
#define U_ID_BASE  0x1FFFF7E8  //adres bazowy
#define U_ID_NUM_REG  96 / 8   //liczba bajtów numeru bazowego do odczytu

uint8_t u_id_tab[U_ID_NUM_REG];
uint8_t *p_tab;
char x;

p_tab =0;
p_tab =p_tab +U_ID_BASE;
for (x=0; x<U_ID_NUM_REG; x++)
    u_id_tab[x] = __LDREXB (p_tab+x);
```

w przestrzeni adresowej począwszy od adresu 0x1FFF F7E8. Dostęp do unikatowego numeru stwarza szereg ciekawych możliwości, takich jak identyfikacja konkretnego urządzenia, zabezpieczenie przed nieautoryzowanym dostępem, wykorzystanie numeru jako adresu MAC w sieciach ethernetowych itp. Na listingu 1 zamieszczono przykładową procedurę odczytującą unikatowy numer mikrokontrolera.

Inną ciekawą funkcją, w którą są wyposażone mikrokontrolery STM32F jest sprzętowy generator sumy kontrolnej CRC. Suma może być używana do kontroli poprawności przesyłanych bloków danych. Jeżeli odebrana suma kontrolna jest identyczna z wyliczoną na podstawie odebranych danych, to oznacza duże prawdopodobieństwo, że dane przesłane nie zostały zafałszowane.

Do obliczenia sumy kontrolnej używa się wyrażenia:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

Przy obliczaniu sumy kontrolnej wykorzystywane są trzy rejestry:

- **CRC_DR**. Rejestr 32-bitowy. Zapis do rejestru inicjuje obliczanie nowej sumy z uwzględnieniem zapamiętanej w nim dotychczasowej wartości. Odczyt pobiera bieżącą wartość dotychczas obliczonej sumy CRC.

Tabela 1. Orientacyjne wartości poboru prądu w każdym z trybów oszczędnościowych oraz sposób wyjścia do trybu aktywnego

Tryb	Orientacyjny pobór prądu	Wybudzania
SLEEP	15 mA	Każde z przerwań
STOP	25 µA	Każda linia skonfigurowana jako źródło przerwania EXTI, narastające zbocze linii WKUP, alarm RTC, reset kontrolera
STANDBY	3,8 µA	narastające zbocze linii WKUP, alarm RTC, reset kontrolera

- **CRC_IDR**. Rejestr 8-bitowy do chwilowego przechowywania dowolnej wartości np. związanej z obliczeniami.
- **CRC_CR**. Ustawienie bitu 0 powoduje wyzerowanie całego mechanizmu obliczeń i zapis do rejestru CRC_DR wartości 0xFFFFFFFF.

Do obliczeń można wykorzystać funkcje biblioteki *STM32F10x Standard Peripherals Library*. Przy obliczeniu sumy dla danych znajdujących się w buforze wywołanie funkcji będzie wyglądało następująco:

*CRCSuma = CRC_CalcBlockCRC((uint32_t *)buforDanych, ILE_DANYCH);*

Listing 2. Przykładowe procedury obsługi przetwornika C/A

```
//dołączenie sygnału zegarowego do DAC1
void DAC1_RCC_Configuration(void)
{
  // załączenie zegara GPIOA
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
  // załączenie zegara przetwornika C/A
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
}

// PA.4 jako wyjście przetwornika DAC1
void DAC1_GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

/* Once the DAC channel is enabled, the corresponding GPIO pin
is automatically connected to the DAC converter. In order to
avoid parasitic consumption, the GPIO pin should be configured
in analog */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}

//procedura inicjacji przetwornika DAC1 z wyjściem PA.4
void DAC1_Inicjacja(void)
{
DAC_InitTypeDef  DAC_InitStructure;
    DAC1_RCC_Configuration();
/* Once the DAC channel is enabled, the corresponding GPIO
pin is automatically connected to the DAC converter. In order
to avoid parasitic consumption, the GPIO pin should be configured
in analog */
    DAC1_GPIO_Configuration();
// DAC channel1 Configuration
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software;
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    DAC_Init(DAC_Channel_1, &DAC_InitStructure);
/* Enable DAC Channel1: Once the DAC channel1 is enabled, PA.04 is automatically connected to the DAC
converter. */
  DAC_Cmd(DAC_Channel_1, ENABLE);
}

//procedura startu konwersji przetwornika DAC1
void DAC1_Start_Konwersji(uint16_t wartosc_do_konwersji)
{
// Stop DAC Channel1 conversion by software
    DAC_SoftwareTriggerCmd(DAC_Channel_1, DISABLE);
// Set DAC Channel1 DHR12R register
    DAC_SetChannel1Data(DAC_Align_12b_R, wartosc_do_konwersji);
// Start DAC Channel1 conversion by software
    DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);
}

//procedura startu konwersji przetwornika DAC1
uint16_t DAC1_Odczyt_Biezacej_Wartosc_Konwersji(void)
{
    uint16_t wartosc;
    wartosc =DAC_GetDataOutputValue (DAC_Channel_1);
    return wartosc;
}
```

Listing 3. Przykładowe procedury dla SPI pracującego w trybie master

```
//podłączenie sygnału zegarowego do SPI1
void SPI1_RCC_Configuration(void)
{
// PCLK2 = HCLK/2
    RCC_PCLK2Config(RCC_HCLK_Div2);
// Enable peripheral clocks
// SPI1 clock enable
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
}

//ustawienie PA.5 jako SCK, PA.6 jako MISO i PA.7 jako MOSI
void SPI1_GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
// Configure SPI1 pins: SCK, MISO and MOSI
// Confugure SCK and MOSI pins as Alternate Function Push Pull
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
// Confugure MISO pin as Input Floating
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

//inicjacja SPI w trybie Master
void SPI1_Inicjacja(void)
{
/* SPI1 configuration */
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;
    SPI_Init(SPI1, &SPI_InitStructure);
// Enable SPI1
    SPI_Cmd(SPI1, ENABLE);
}

//procedura transmisji i odbioru danych w trybie duplex
uint16_t Procedura_Transmisji_SPI1(uint16_t TxDat)
{
    uint16_t RxDat;
// Wait for SPI1 Tx buffer empty
    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
// Send SPI1 data
    SPI_I2S_SendData(SPI1, TxDat);
// Wait for SPI1 data reception
    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
// Read SPI1 received data
    RxDat = SPI_I2S_ReceiveData(SPI1);
    return  RxDat;
}
```

Przetwornik C/A

Wiele mikrokontrolerów z serii STM32F wyposażono w dwa działające niezależnie przetworniki cyfrowo-analogowe o rozdzielczości 12-bitowej. Na wyjściu przetwornika można uzyskać napięcie w zakresie określonym przez poziomy ustawione na wyprowadzeniach mikrokontrolera VSSA (zwykle GND) i VREF+ (nie wyższe od napięcia zasilania kontrolera, najczęściej z zakresu 2,4…3,3 V). Przykładowe, nieskomplikowane procedury obsługi przetwornika C/A pokazano na **listingu 2**.

Interfejsy szeregowe SPI, I²C

Oprócz interfejsu UART kontrolery posiadają jeden lub więcej interfejsów SPI i I²C. Każdy z interfejsów SPI można ustawić do pracy w trybie master lub slave. Przykładowe procedury dla SPI pracującego w trybie master mogą wyglądać, jak zaprezentowano na **listingu 3**.

Przykładowe procedury obsługi interfejsu I²C można znaleźć w pliku *STM32F10x Standard Peripherals Library* w podkatalogu Examples.

Tryby obniżonego poboru mocy

Mikrokontrolery STM32 mają 3 tryby obniżonego poboru mocy: Sleep, Stop, Standby. W każdym z wymienionych trybów dla obniżenia poboru prądu kontroler zostaje zatrzymany i wprowadzony w stan uśpienia. Dodatkowo, dla obniżenia zużycia energii zostają wyłączone niektóre bloki wewnętrzne. Zależnie od trybu, redukcja poboru prądu jest większa lub mniejsza. Inne są także sposoby wybudzenia i powrotu do normalnej pracy. W **tabeli 1** zestawione zostały orientacyjne wartości poboru prądu w każdym z trybów i sposoby na wyprowadzenie kontrolera z trybu uśpienia.

Każdy z trybów ma różny czas powrotu do normalnej pracy (wybudzania). Tryby o większej oszczędności poboru prądu, w którym zostaje wyłączona znaczna część wewnętrznych bloków, potrzebują dłuższego czasu na powrót do normalnej pracy. Zależnie od trybu mogą być utracone dane przechowywane w wewnętrznej pamięci RAM po wejściu w stan uśpienia. Dokładne informacje na ten temat można znaleźć w sekcji *Low-power modes* danych technicznych.

Programowe wprowadzenie kontrolera w tryb Stop z obniżonym poborem mocy wewnętrznego regulatora napięcia wykonuje się następująco:

```
// Request to enter STOP mode with regulator in low power mode
PWR_EnterSTOPMode(PWR_Regulator_LowPower, PWR_STOPEntry_WFI);
// od tego momentu kontroler znajduje się w trybie uśpienia STOP
```

Programowe wprowadzenie kontrolera w tryb Standby wygląda następująco:

Tabela 2. Moduły peryferyjne, z którymi współpracuje DMA1							
Moduł	**Kanał 1**	**Kanał 2**	**Kanał 3**	**Kanał 4**	**Kanał 5**	**Kanał 6**	**Kanał 7**
ADC1	ADC1						
SPI		SPI1 RX	SPI1 TX	SPI2 RX	SPI2 TX		
USART		USART3 TX	USART3 RX	USART1 TX	USART1 RX	USART2 TX	USART2 RX
I2C				I2C 2 TX	I2C 2 RX	I2C 1 TX	I2C 1 RX
TIM1		TIM1 CH1	TIM1 CH2	TIM1 CH4 TIM1 TRIG TIM1 COM	TIM1 UP	TIM1 CH3	
TIM2	TIM2 CH3	TIM2 UP			TIM2 CH1		TIM2 CH2 TIM2 CH4
TIM3		TIM3 CH3	TIM3 CH4 TIM3 UP			TIM3 CH1 TIM3 TRG	
TIM4	TIM4 CH1			TIM4 CH2	TIM4 CH3		TIM4 UP

Tabela 3. Moduły peryferyjne, z którymi współpracuje DMA2					
Moduł	**Kanał 1**	**Kanał 2**	**Kanał 3**	**Kanał 4**	**Kanał 5**
ADC3					ADC3
SPI	SPI 3 RX	SPI 3 TX			
UART4			UART4 RX		UART4 TX
SDIO				SDIO	
TIM5	TIM5 CH4 TIM5 TRIG	TIM5 CH3 TIM5 UP		TIM5 CH2	TIM5 CH1
TIM6/ DAC1			TIM6 UP DAC1 CH1		
TIM7				TIM7 UP/ DAC1 CH2	
TIM8	TIM8 CH3 TIM8 UP	TIM8 CH4 TIM8 TRIG TIM8 COM	TIM8 CH1		TIM8 CH2

```
    PWR_EnterSTANDBYMode();
    // od tego momentu kontroler znajduje
się w trybie uśpienia STANDBY
```

FSMC dołączanie zewnętrznych pamięci

Bardzo interesującą opcją oferowaną przez niektóre typy kontrolerów STM32F jest możliwość współpracy z pamięciami różnego typu wyposażonymi w interfejs równoległy. Są to między innymi: SRAM, ROM, NOR, PSRAM. Możliwość przechowywania większych bloków danych np. w dodatkowej pamięci RAM bardzo rozszerza możliwości kontrolera, którego wewnętrzny RAM nigdy nie będzie zbyt wielki w stosunku do potrzeb, a dodatkowo jest wykorzystywany przez procedury programu. Interfejsem pozwalającym na zarządzanie zewnętrzną pamięcią jest FSMC (*Flexible Static Memory Controler*). Najważniejsze parametry FSMC są następujące:

- A0…A25 – możliwość sterowania 26 liniami adresowymi.
- D0…D15 – zapis i odczyt z pamięci danych poprzez 16 linii. Dostępny jest również tryb z 8 liniami danych.
- NE(4) – możliwość sterowania 4 liniami wyboru pamięci (CS).
- NOE – linia zezwolenia na odczyt z pamięci (OE).
- NWE – linia zezwolenia zapisu do pamięci (WE).
- Kilka dodatkowych linii sterujących pamięciami wymagającymi dostępu w trybie multipleksowanym.

Jednostka FSMC zarządza 4 bankami danych, z których każdy jest przeznaczony do obsługi określonego rodzaju pamięci. W trybie danych 8-bitowych jest możliwa obsługa pamięci o pojemności 64 MB. W trybie 16-bitowym pojemność obsługiwanej pamięci jest o połowę mniejsza.

Dostęp do FSMC i jego możliwości zależą od typu mikrokontrolera. Pamięci równoległe wymagają do sterowania wielu linii i wyprowadzeń, których w nie ma w mniejszych obudowach. Informacji o obsługiwanych przez dany kontroler pamięciach zewnętrznych należy szukać w danych technicznych lub można w tym celu posłużyć się programem *STM32CubeMX*.

Zamontowany na Panelu Edukacyjnym mikrokontroler STM32F103RC (LQFP64) nie daje dostępu do FSMC. W mikrokontrolerze STM32F103VCT (LQFP100) obsługiwane są pamięci NAND w trybie 16-bitowym i multipleksowane NOR. W STM32F103ZCT (LQFP144), oprócz poprzednio wspomnianych, obsługiwane są standardowe pamięci ulotne RAM. W *STM32F10x Standard Peripherals Library* znajduje się sekcja procedur obsługujących FSMC dostępne są także przykłady sterowania zewnętrznymi pamięciami RAM.

DMA bezpośrednie przesłanie do pamięci

Mechanizm DMA po zainicjowaniu pozwala przesyłać dane automatycznie bez udziału programu kontrolera. Taki transfer możliwy jest pomiędzy dwoma obszarami pamięci RAM kontrolera (także z udziałem pamięci zewnętrznej o ile jest zamontowana), albo pomiędzy pamięcią a konkretnym interfejsem. Przy transmisji dużych bloków danych lub danych zapisywanych w sposób ciągły (np. z przetwornika A/C) w pamięci pozwala to na odciążenie jednostki centralnej. Za pomocą DMA można przesyłać albo określoną liczbę bajtów danych, albo w sposób ciągły zapisywać dane w buforze kołowym. Maksymalny wielkość pojedynczego bloku danych lub pojemność bufora to 65536 bajtów.

Zwykle w STM32 są dwa kontrolery DMA. Każdy ma niezależne kanały, które jednocześnie mogą realizować

kilka sesji DMA. Do każdego kanału przypisane są na stałe interfejsy, które obsługuje. Natomiast transmisję danych pomiędzy obszarami pamięci RAM można realizować za pomocą dowolnego kanału DMA.

W **tabelach 2** i **3** wymieniono połączenia poszczególnych kanałów jednostek DMA1 oraz DMA2 z modułami peryferyjnymi. Uwaga: ten sam kanał nie może jednocześnie realizować transmisji danych z dwóch różnych modułów peryferyjnych.

Podczas inicjowania jednostki DMA ustawiane są następujące parametry:

- Adres rejestru (do zapisu lub odczytu) danych modułu, który będzie korzystał z DMA.
- Adres początku obszaru w pamięci RAM (do zapisu lub odczytu), do którego będą przesyłane dane z interfejsu.
- Liczba bajtów danych do przesłania.
- Priorytet DMA.
- Tryb pracy, w tym kierunek transmisji (z lub do pamięci), wielkość słowa danych (8-, 16-, 32-bitowe), wybór przesłania jednokrotnego lub ciągłego z wykorzystaniem bufora kołowego.
- Wybór generowanych przerwań przez mechanizm DMA, np. po przesłaniu połowy lub całego bloku danych.

Po uruchomieniu DMA automatycznie rozpoczyna transmisję inkrementując adres pamięci RAM.

Interfejs SDIO dla kart SD/microSD

Innym obsługiwanym nośnikiem danych są karty SD (microSD). Takie karty zazwyczaj mają dwa typy interfejsów komunikacyjnych: SPI i SDIO. Drugi jest bardziej wydajny i zapewnia dostęp do większości funkcji karty SD.

Interfejs SDIO mikrokontrolerów STM32F zapewnia niskopoziomową transmisję i transmisję komend oraz danych pomiędzy kontrolerem a kartą w trybie SDIO. Interfejs jest zgodny ze specyfikacją *SD I/O Card Specification Version 2.0*. Możliwa jest transmisja w trybie 1- lub 4-bitowym. Maksymalna częstotliwość zegarowa interfejsu to 48 MHz. W trybie 4 bitowym kartę SD przyłącza się do wyprowadzeń mikrokontrolera wymienionych w **tabeli 4**.

Z poziomu interfejsu SDIO jest możliwa transmisja danych, czyli odczyt i zapis na kartę. Najwygodniej, aby na karcie dane były przechowywane w sposób uporządkowany np. w formie plików FAT. Dzięki temu będę one przenośne i dostępne z poziomu powszechnie używanych systemów operacyjnych. O to musi zadbać oprogramowanie kontrolera. Nie ma jednak potrzeby

Tabela 4. Wyprowadzanie interfejsu karty SD

Nr złącza karty SD	Sygnał	Port kontrolera
1	CD/DAT3	PC11
2	CMD	PD2
3	VSS1	GND
4	VDD	3.3V
5	CLK	PC12
6	VSS2	GND
7	DAT0	PC8
8	DAT1	PC9
9	DAT2	PC10

samemu tworzyć oprogramowanie zarządzające plikami. Wystarczy sięgnąć po któryś z darmowych pakietów, chociażby po *FatFs – Generic FAT File System Module* dostępny na stronie *http://elm-chan.org/fsw/ff/00index_e.html*. Oprogramowanie jest napisane w języku C. Za pomocą kilku procedur należy je tylko dostosować do interfejsu SDIO kontrolera. Te procedury będą się odwoływały do funkcji SDIO odpowiedzialnych za zapis i odczyt na karcie i do zegara RTC udostępniającego aktualny czas systemowy. Na stronie znajdują się linki z opisami jak samodzielnie stworzyć takie procedury oraz do konkretnych przykładów, w tym dla STM32. Natomiast wszystkim, co dotyczy zarządzania systemem plików i folderów zajmie się FatFs.

Nowe kontrolery i interfejsy

Wraz z nowymi mocniejszymi kontrolerami STM32 zwiększają się ich możliwości i dostępne interfejsy. Przykładem mogą być STM32F4xx z jądrem Cortex M4 znane z wielu zestawów ewaluacyjnych pod firmową nazwą Discovery. Oprócz wcześniej opisanych różne typy STM32F4 wyposażono w takie interfejsy jak:

- Interfejs kolorowego wyświetlacza graficznego. Dostępne są gotowe biblioteki funkcji do tworzenia i zarządzania typowymi obiektami graficznymi jak figury geometryczne, ale także przyciski, tabele, okna itp.
- Interfejs DCMI do podłączania modułów kamer cyfrowych. Interfejs obsługuje automatyczny przechwyt danych z magistrali kamery w postaci binarnej.
- Interfejs USB OTG.
- Interfejs Ethernet.

W niektóre STM32F4xx wbudowano sprzętowy koprocesor matematyczny przyśpieszający obliczenia zmiennoprzecinkowe.

Ryszard Szymaniak, EP

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 75421, pass: tkuyg3b9