Interfejs modułu kamery (1)

Procedury umożliwiające współpracę modułu kamery z mikrokontrolerem

W EP 4/2013 opisaliśmy projekt interfejsu modułu kamery. Aby móc w pełni wykorzystać jego możliwości, trzeba wykonać odpowiednie oprogramowanie i mieć wiedzę, jak się za to zabrać. Ten artykuł jest poświęcony wewnętrznym mechanizmom mikrokontrolera ułatwiającym współpracę z modułem kamery. Zamiast schematów będą listingi przykładowych procedur dla mikrokontrolerów STM32F2xx.

Opis procedur dotyczy mikrokontrolera STM32F2xx takiego, jak zastosowany w interfejsie. Dla tych, którzy nie czytali poprzedniego artykułu lub zdążyli już o nim zapomnieć, na początek, krótkie przypomnienie. Interfejs współpracuje z modułem kamery dołączanym z zewnątrz. Na gnieździe łączącym interfejs z modułem wyprowadzono wszystkie sygnały niezbędne do sterowania modułem i odczytu danych obrazka (ramki obrazu). Dla uproszczenia konstrukcji dane są zapisywane w buforze, w wewnętrznej pamięci RAM mikrokontrolera. Do sterowania płytką interfejsu oraz odczytu danych obrazka wykorzystuje się drugie złącze, do którego – oprócz zasilania – dołączono sygnały RxD i TxD portu szeregowego mikrokontrolera. Poprzez to złącze interfejs może komunikować się z zewnętrznym komputerem, na którym można uruchomić przykładowy program sterujący.

W celu przechwycenia danych obrazu, oprogramowanie mikrokontrolera wykorzystuje wewnętrzny interfejs DCMI, mechanizm DMA oraz pamięć RAM, która służy do przechowywania danych odczytanych z modułu. Ponadto, płytka interfejsu dostarcza do modułu napięcie zasilające oraz steruje jego działaniem poprzez 2-przewodowy interfejs I^2C.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 20637, pass: 7430ukcs

Ogólny schemat budowy oprogramowania sterującego modułem kamery

Na **rysunku 1** pokazano schemat blokowy przepływu danych obrazu z wyjścia modułu kamery do bufora w pamięci RAM mikrokontrolera. Strumień danych na wyjściu kamery jest informacją o kolejnych pikselach obrazu pogrupowanych w linie tworzących ramkę obrazu. Interfejs DCMI – za pomocą sygnałów synchronizacji generowanych przez moduł – odczytuje dane kolejnych pikseli i tworzy z nich 32-bitowe słowo danych. Dane mogą być zapamiętane w buforze obrazu w pamięci RAM mikrokontrolera. Do ich transmisji najlepiej użyć sprzętowego mechanizmu DMA mikrokontrolera, ponieważ jest szybki, nie angażuje mocy obliczeniowej CPU i może działać w tle programu głównego. Blok danych zapamiętany w buforze może być wykorzystany np. do stworzenia pliku obrazu w formacie BMP czy JPEG.

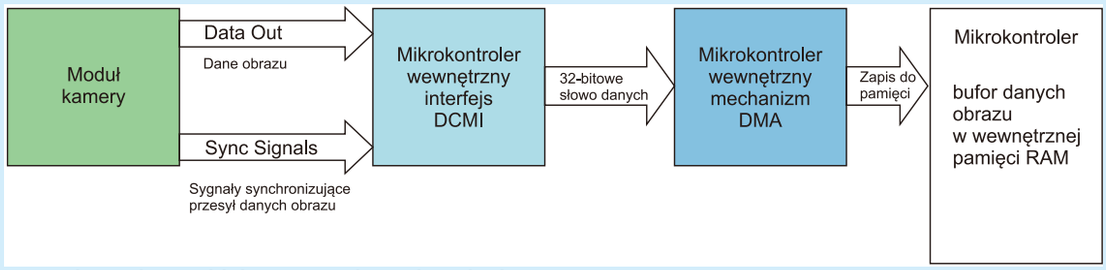
Tworzenie programu sterującego składa się z kilku etapów wymagających utworzenia następujących procedur:

- inicjującej porty mikrokontrolera połączone z wyprowadzeniami modułu kamery,
- inicjującej wewnętrzne rejestry modułu kamery,
- inicjującej rejestry mikrokontrolera konfigurujące interfejs DCMI,
- inicjującej rejestry mikrokontrolera sterujące pracą DMA,
- uruchamiającej interfejsy DCMI i DMA mikrokontrolera oraz umożliwiającej wykonanie zdjęcia przez moduł kamery.

Dołączenie kamery do mikrokontrolera

Po pierwsze, należy fizycznie połączyć wyprowadzenia sygnałów modułu z odpowiednimi wyprowadzeniami mikrokontrolera, co pokazano na schemacie interfejsu opublikowanym w EP 4/2012. Wybór niektórych linii może być dowolny, jednak dla portów mikrokontrolera obsługujących interfejs DCMI istnieją pewne ograniczenia.

Jak to często bywa w wypadku mikrokontrolerów, możliwe konfiguracje połączeń zależą od obudowy i liczby dostępnych wyprowadzeń. Przy większych obudowach istnieje możliwość przypisania określonej



Rysunek 1. Schemat blokowy przepływu danych obrazu

funkcji jednemu z dostępnych portów. Te informacje są podawane w dokumentacji technicznej zastosowanego typu mikrokontrolera w rozdziale *Alternate function mapping*. Najczęściej zawiera on zestawienie w formie tabeli wszystkich dodatkowych funkcji, które można przypisać wyprowadzeniu każdego z portów. W tym momencie najbardziej interesujące są dla nas funkcje związane z interfejsem DCMI. W tabeli są one poprzedzone przedrostkiem *DCMI_* i dotyczą magistrali danych oraz linii sygnałów sterujących. Ponieważ w interfejsie wybrano magistralę o szerokości 8 bitów, w tabeli należy odszukać funkcje nazwane *DCMI_D0…DCMI_D7*. Jak można przekonać się, tę samą funkcję mogą pełnić różne wyprowadzenia. Np. sygnał interfejsu *DCMI_D0* można przypisać albo do portu *PA9*, albo *PC6*. Dzięki temu można łatwiej dopasować schemat urządzenia do możliwości technicznych mikrokontrolera. Oprócz portów magistrali danych należy wybrać te, które będą współpracowały z sygnałami sterującymi: *DCMI_HSYNC, DCMI_VSYNC, DCMI_PIXCK*. Dodatkowo, należy wybrać porty do obsługi szeregowej magistrali I²C, za pomocą której są wysyłane rozkazy sterujące do modułu. Jeżeli zdecydujemy się na własną, programową obsługę protokołu I²C, do obsługi magistrali można wybrać dowolne porty. Jeżeli zaś będziemy chcieli skorzystać ze sprzętowego wsparcia protokołu I²C, które mają mikrokontrolery STM32F, znów trzeba odszukać w sekcji *Alternate function mapping* informację o tym, które porty związane są z obsługą portu I²C. Oprócz tego, do dołączenia pozostał jeszcze jeden sygnał – zegara taktującego, który trzeba doprowadzić do modułu kamery. Zazwyczaj jego częstotliwość powinna zawierać się w granicach 6…80 MHz. Można w tym celu wykorzystać jedno z dostępnych wyprowadzeń MCO wewnętrznego sygnału zegarowego mikrokontrolera. Częstotliwość przebiegu prostokątnego na tym wyprowadzeniu może być programowo zmieniana i doskonale nadaje się jako sygnał zegarowy dla modułu. Jedynym ograniczeniem jest przypisanie funkcji MCO do konkretnego wyprowadzenia.

Szukanie w tabeli interesujących nas funkcji portów i unikanie możliwych kolizji, gdy wybór jednej z funkcji blokuje możliwość wykorzystania innej bywa kłopotliwe. Innym rozwiązaniem jest posłużenie się graficznym programem *MicroXplorer*. *T*o darmowy program pozwalający na łatwiejsze poruszanie się w gąszczu alternatywnych funkcji wyprowadzeń mikrokontrolera. Po zainstalowaniu i uruchomieniu programu, na początku należy zadeklarować typ i obudowę mikrokontrolera, który będzie używany w konstruowanym urządzeniu. Następnie należy wskazać typ interfejsu, który nas interesuje np. DCMI. Na rysunku obudowy zostaną podświetlone na zielono wyprowadzenia, interfejsu. Jednocześnie na liście dostępnych interfejsów mikrokontrolera na czerwono zostaną zaznaczone wszystkie, które w tej konfiguracji staną się niedostępne np. I2C3. Żółtymi znaczkami ostrzeżeń zostaną oznaczone interfejsy o funkcjonalności ograniczonej dla takiej konfiguracji. Ponieważ interfejsy mają zdublowane funkcje dla różnych wyprowadzeń mikrokontrolera, posługując się dokumentacją i programem *MicroXplorer* można wszystko odpowiednio skonfigurować. Zamieszczony w poprzednim artykule schemat interfejsu pokazuje jak ostatecznie przypisano funkcje do wyprowadzeń mikrokontrolera.

Listing 1. Deklaracja zmiennych symbolicznych DCMI

```
#define KAM_DATn  8
typedef enum
{
  KAM_DAT0=0,
  KAM_DAT1=1,
  KAM_DAT2=2,
  KAM_DAT3=3,
  KAM_DAT4=4,
  KAM_DAT5=5,
  KAM_DAT6=6,
  KAM_DAT7=7
}Kam_DAT_TypeDef;
//Linia0 danych
#define KAM_DAT0_PORT  GPIOC
#define KAM_DAT0_CLK   RCC_AHB1Periph_GPIOC
#define KAM_DAT0_PIN   GPIO_Pin_6
#define KAM_DAT0_PIN_SOURCE  GPIO_PinSource6
//Linia1 danych
#define KAM_DAT1_PORT  GPIOC
#define KAM_DAT1_CLK RCC_AHB1Periph_GPIOC
#define KAM_DAT1_PIN GPIO_Pin_7
#define KAM_DAT1_PIN_SOURCE GPIO_PinSource7
```

Inicjowanie portów mikrokontrolera

Porty mikrokontrolera używane przez interfejs DCMI należy odpowiednio zainicjować. Procedurę ich inicjowania najłatwiej napisać wykorzystując odpowiednie funkcje biblioteki standardu CMSIS udostępnianej przez firmę ST dla konkretnego typu mikrokontrolera. Funkcje są dobrze udokumentowane i mają opisowe nazwy procedur. Dlatego wszystkie kolejne przykłady będą wykorzystywały funkcje takiej biblioteki.

Na początku należy zadeklarować zmienne symboliczne związane ze wszystkimi liniami wykorzystywanymi przez interfejs DCMI. Deklarację najlepiej umieścić w dołączanym pliku nagłówkowym. Do nazw symbolicznych będą się potem odwoływały wszystkie procedury operujące na liniach interfejsu. Deklaracja może wyglądać jak na **listingu 1**.

Dalej powinna być umieszczona deklaracja dla kolejnych linii magistrali danych, sygnałów sterujących. Przykładową procedurę inicjującą linie interfejsu DCMI pokazano na **listingu 2**.

Proces inicjowania interfejsu składa się z trzech funkcji. Najpierw są deklarowane tabele nazw symbolicznych, które odwołują się do wewnętrznych rejestrów konfiguracyjnych wyprowadzeń GPIO mikrokontrolera. Następnie, procedura Inicjacja_linii_DCMI w kolejnych krokach inicjuje linie interfejsu DCMI. Dla większej przejrzystości inicjowanie linii magistrali danych i linii sygnałów sterujących podzielono na dwa podprogramy KAM_Magistrala_danych_inicjacja i KAM_IO_Linie_Inicjacja. Dodatkowo, jest uruchamiane wyprowadzenie MCO i ustawiane parametry sygnału zegarowego dla modułu. Od tego momentu porty interfejsu DCMI gotowe są do obsługi sygnałów i współpracy z liniami modułu kamery.

Na koniec należy jeszcze zainicjować linie szeregowej magistrali sterującej. Jeżeli do sterowania linii zostanie użyty wewnętrzny interfejs I²C mikrokontrolera, linie powinny zostać standardowo zainicjowane do pracy w funkcji alternatywnej. Jeżeli będzie stosowana programowa obsługa protokołu I²C, linie powinny zostać zainicjowane w trybie standardowych portów I/O w trybie otwartego drenu z wewnętrznym podciąganiem do plusa. Przy takiej konfiguracji procedury użytkownika obsługujące transmisję I²C będą miały możliwość zapisu i odczytu poziomu portów.

Procedura inicjacji wewnętrznych rejestrów modułu kamery

Następnym krokiem jest inicjacja modułu kamery poprzez zapisanie do jego wewnętrznych rejestrów odpo-

Listing 2. Procedura inicjująca linie interfejsu DCMI

```
const uint32_t GPIO_KAM_DAT_CLK[KAM_DATn]=
{
  KAM_DAT0_CLK, KAM_DAT1_CLK, KAM_DAT2_CLK, KAM_DAT3_CLK,
  KAM_DAT4_CLK, KAM_DAT5_CLK, KAM_DAT6_CLK, KAM_DAT7_CLK
};
const uint16_t GPIO_KAM_DAT_PIN[KAM_DATn]=
{
  KAM_DAT0_PIN, KAM_DAT1_PIN, KAM_DAT2_PIN, KAM_DAT3_PIN,
  KAM_DAT4_PIN, KAM_DAT5_PIN, KAM_DAT6_PIN, KAM_DAT7_PIN
};
GPIO_TypeDef* GPIO_KAM_DAT_PORT[KAM_DATn]=
{
  KAM_DAT0_PORT, KAM_DAT1_PORT, KAM_DAT2_PORT, KAM_DAT3_PORT,
  KAM_DAT4_PORT, KAM_DAT5_PORT, KAM_DAT6_PORT, KAM_DAT7_PORT
};
const uint32_t GPIO_KAM_LINE_CLK[KAM_IOn]={KAM_HSYNC_CLK, KAM_VSYNC_CLK, KAM_PCLK_CLK};
const uint16_t GPIO_KAM_LINE_PIN[KAM_IOn]={KAM_HSYNC_PIN, KAM_VSYNC_PIN, KAM_PCLK_PIN};
GPIO_TypeDef* GPIO_KAM_LINE_PORT[KAM_IOn]={KAM_HSYNC_PORT, KAM_VSYNC_PORT, KAM_PCLK_PORT};

//procedura inicjacji linii interfejsu DCMI
void Inicjacja_linii_DCMI(void)
{
//inicjowanie linii MCO1 impulsów zegara KAM_MCLK dla modułu kamery
  KAM_IO_Linie_Inicjacja(KAM_MCLK, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_MCO1Config(RCC_MCO1Source_PLLCLK, RCC_MCO1Div_3);
  Delay_ms(100);
//inicjacja 8 linii równoległej magistrali danych obrazu
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT0], ENABLE);
  GPIO_PinAFConfig (KAM_DAT0_PORT, KAM_DAT0_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT0, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT1], ENABLE);
  GPIO_PinAFConfig (KAM_DAT1_PORT, KAM_DAT1_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT1, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT2], ENABLE);
  GPIO_PinAFConfig (KAM_DAT2_PORT, KAM_DAT2_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT2, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT3], ENABLE);
  GPIO_PinAFConfig (KAM_DAT3_PORT, KAM_DAT3_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT3, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT4], ENABLE);
  GPIO_PinAFConfig (KAM_DAT4_PORT, KAM_DAT4_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT4, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT5], ENABLE);
  GPIO_PinAFConfig (KAM_DAT5_PORT, KAM_DAT5_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT5, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT6], ENABLE);
  GPIO_PinAFConfig (KAM_DAT6_PORT, KAM_DAT6_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT6, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[KAM_DAT7], ENABLE);
  GPIO_PinAFConfig (KAM_DAT7_PORT, KAM_DAT7_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_Magistrala_danych_inicjacja(KAM_DAT7, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);

//inicjacja linii sygnałów sterujących
//wejście sygnału HSYNC
  RCC_AHB1PeriphClockCmd(GPIO_KAM_LINE_CLK[KAM_HSYNC], ENABLE);
  GPIO_PinAFConfig(KAM_HSYNC_PORT, KAM_HSYNC_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_IO_Linie_Inicjacja(KAM_HSYNC, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
//wejście sygnału VSYNC
  RCC_AHB1PeriphClockCmd(GPIO_KAM_LINE_CLK[KAM_VSYNC], ENABLE);
  GPIO_PinAFConfig (KAM_VSYNC_PORT, KAM_VSYNC_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_IO_Linie_Inicjacja(KAM_VSYNC, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
//wejście sygnału PIXCLK
  RCC_AHB1PeriphClockCmd(GPIO_KAM_LINE_CLK[KAM_PCLK], ENABLE);
  GPIO_PinAFConfig (KAM_PCLK_PORT, KAM_PCLK_PIN_SOURCE, GPIO_AF_DCMI);
  KAM_IO_Linie_Inicjacja  (KAM_PCLK, GPIO_Mode_AF, GPIO_OType_PP, GPIO_PuPd_UP);
}

//procedura inicjowania linii magistrali danych kamery
void KAM_Magistrala_danych_inicjacja(Kam_DAT_TypeDef Linia, GPIOMode_TypeDef tryb_pracy, GPIOOType_TypeDef
typ_wyjscia, GPIOPuPd_TypeDef typ_podciagania)
{
  GPIO_InitTypeDef  GPIO_InitStructure;
/* Enable Clock */
  RCC_AHB1PeriphClockCmd(GPIO_KAM_DAT_CLK[Linia], ENABLE);
/* Configure the GPIO_KAM_DAT pin */
  GPIO_InitStructure.GPIO_Pin = GPIO_KAM_DAT_PIN[Linia];
  GPIO_InitStructure.GPIO_Mode =tryb_pracy;
  GPIO_InitStructure.GPIO_OType = typ_wyjscia;
  GPIO_InitStructure.GPIO_PuPd = typ_podciagania;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIO_KAM_DAT_PORT[Linia], &GPIO_InitStructure);
}

//procedura inicjowania linii sterujących KAM
void KAM_IO_Linie_Inicjacja(Kam_IO_TypeDef Linia, GPIOMode_TypeDef tryb_pracy, GPIOOType_TypeDef typ_wyjscia,
GPIOPuPd_TypeDef typ_podciagania)
{
  GPIO_InitTypeDef  GPIO_InitStructure;
/* Enable the GPIO_KAM_LINIE Clock */
  RCC_AHB1PeriphClockCmd(GPIO_KAM_LINE_CLK[Linia], ENABLE);
/* Configure the GPIO_KAM_LINIE pin*/
  GPIO_InitStructure.GPIO_Pin = GPIO_KAM_LINE_PIN[Linia];
  GPIO_InitStructure.GPIO_Mode = tryb_pracy;
  GPIO_InitStructure.GPIO_OType = typ_wyjscia;
  GPIO_InitStructure.GPIO_PuPd = typ_podciagania;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIO_KAM_LINE_PORT[Linia], &GPIO_InitStructure);
}
```

wiednich ustawień. Najczęściej po włączeniu zasilania w rejestrach są już jakieś sensowne wartości. Jednak nawet w takim przypadku użytkownik może chcieć zaprogramować własne początkowe ustawienia np. formatu, wymiarów obrazu, jaskrawości, nasycenia itp. Zapis ewentualnie odczyt rejestrów następuje poprzez szeregową magistralę sterującą. Sposób adresowania rejestrów, ich funkcje, format danych określa dokumentacja konkretnego modelu modułu. Dane do rejestrów przesyłane są magistralą w formacie I²C z wykorzystaniem wewnętrznego interfejsu mikrokontrolera lub za pomocą własnych procedur użytkownika obsługi protokołu I²C. Przykład inicjowania modułu testowanego z płytką interfejsu zostanie podany w innym artykule.

Procedury inicjowania rejestrów interfejsu DCMI

Sprzętowy interfejs DCMI służy do wychwytywania z ciągłego strumienia pikseli i linii wysyłanego przez moduł kamery danych obrazu i formowania ich w logicznie ułożone bloki. Interfejs może współpracować z różnymi modułami i dlatego jego parametry są konfigurowalne. Dotyczy to między innymi:

- Szerokości magistrali danych obrazu, która może składać się z 8, 10, 12 lub 14 bitów.
- Określenia aktywnego zbocza sygnału *PIXCLK* służącego do synchronizacji kolejnych bajtów danych oraz określenia aktywnego poziomu sygnałów *HSYNC*, *VSYNC*. W przypadku tych ostatnich, aktywny poziom wskazuje na przerwę pomiędzy kolejnymi liniami obrazu i przerwę pomiędzy kolejnymi ramkami obrazu.
- Określenia trybu rejestracji danych: kilka kolejnych pojedynczych obrazków (ramek) – *Snapshot mode*, rejestracja ciągła (rejestracja filmu) – *Continuous grab mode*.
- Ustalenia formatu danych obrazu wysyłanych przez moduł: prostego, gdy wysyłane są dane kolejnych pikseli następujących po sobie linii lub kodowanego *JPEG format*, gdy dane stanowią zakodowane bloki.

Oprócz tego, interfejs może być zaprogramowany do automatycznego „przycinania" odbieranego rozkazu. Może także pracować bez sygnałów *HSYNC*, *VSYNC* o ile moduł jest w stanie wysyłać dane z kodami synchronizacji zastępującymi wspomniane sygnały. Na potrzeby tego opisu zostanie podany przykład zaprogramowania interfejsu DCMI do pracy z 8-bitową magistralą obrazu, sygnałami *HSYNC*, *VSYNC* aktywnymi przy poziomie niskim, rejestrujący pojedyncze, niekodowane ramki obrazu. Przykład konfigurowania rejestrów interfejsu DCMI zamieszczono na **listingu 3**.

Najpierw podłączony zostaje wewnętrzny zegar mikrokontrolera do obwodów interfejsu DCMI. Następnie wypełnione zostają kolejne pola struktury wartościami konfiguracyjnymi o następującym znaczeniu:

- *DCMI_CaptureMode_SnapShot* – interfejs będzie pracował przy obsłudze pojedynczych zdjęć (ramek),
- *DCMI_SynchroMode_Hardware* – interfejs będzie współpracował z liniami sterującymi sygnałów HSYNC, VSYNC,
- *DCMI_PCKPolarity_Rising* – będzie aktywne zbocze narastające sygnału taktującego dane PIXCL,
- *DCMI_VSPolarity_Low* – poziomem aktywnym sygnału VSYNC jest poziom niski (linia VSYNC przyjmuje poziom aktywny pomiędzy kolejnymi ramkami obrazów),
- DCMI_HSPolarity_Low -poziomem aktywnym sygnału HSYNC jest poziom niski (linia HSYNC przyjmuje poziom aktywny pomiędzy kolejnymi liniami ramki obrazu),
- *DCMI_CaptureRate_All_Frame* – interfejs prześle do bufora dane wszystkich kolejnych ramek (parametr istotny w wypadku transmisji z modułu więcej niż 1 obrazu),
- DCMI_ExtendedDataMode_8b – interfejs będzie współpracował z 8-bitową magistralą danych obrazu.

Listing 3. Przykład konfigurowania rejestrów interfejsu DCMI

```
//Konfigurowanie interfejsu DCMI do pracy z modułem
MT9D111
//Enable DCMI clock
  RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_DCMI, ENABLE);
//DCMI configuration
  DCMI_DeInit();
  DCMI_InitStructure.DCMI_CaptureMode=DCMI_CaptureMode_
SnapShot;
  DCMI_InitStructure.DCMI_SynchroMode=DCMI_SynchroMode_
Hardware;
  DCMI_InitStructure.DCMI_PCKPolarity=DCMI_PCKPolarity_
Rising;
  DCMI_InitStructure.DCMI_VSPolarity=DCMI_VSPolarity_
Low;
  DCMI_InitStructure.DCMI_HSPolarity=DCMI_HSPolarity_
Low;
  DCMI_InitStructure.DCMI_CaptureRate=DCMI_CaptureRate_
All_Frame;
  DCMI_InitStructure.DCMI_ExtendedDataMode=DCMI_
ExtendedDataMode_8b;
//DCMI configuration
  DCMI_Init(&DCMI_InitStructure);
  DCMI_JPEGCmd( DISABLE );
//mask interrupt for DCMI
  DCMI_ITConfig(DCMI_IT_VSYNC, ENABLE);
  DCMI_ITConfig(DCMI_IT_LINE, ENABLE);
        DCMI_ITConfig(DCMI_IT_FRAME, ENABLE);
        DCMI_ITConfig(DCMI_IT_OVF, ENABLE);
        DCMI_ITConfig(DCMI_IT_ERR, ENABLE);
```

Listing 4. Przykład procedury obsługi przerwań DCMI

```
//przerwania DCMI
void DCMI_IRQHandler(void)
{
  extern uint8_t  dcmi_flag_frameri_bak, dcmi_flag_lineri_
bak, dcmi_flag_errri_bak, dcmi_flag_ovfri_bak, dcmi_flag_
vsyncri_bak;
  if (DCMI_GetFlagStatus(DCMI_FLAG_VSYNCRI)==SET) dcmi_
flag_vsyncri_bak =TRUE;
  if (DCMI_GetFlagStatus(DCMI_FLAG_LINERI) ==SET) dcmi_
flag_lineri_bak =TRUE;
  if (DCMI_GetFlagStatus(DCMI_FLAG_LINEMI)==SET) { }
  if (DCMI_GetFlagStatus(DCMI_FLAG_FRAMERI)==SET) dcmi_
flag_frameri_bak =TRUE;
  if (DCMI_GetFlagStatus(DCMI_FLAG_ERRRI) ==SET) dcmi_
flag_errri_bak =TRUE;
  if (DCMI_GetFlagStatus(DCMI_FLAG_OVFRI) ==SET) dcmi_
flag_ovfri_bak =TRUE;
  DCMI_ClearFlag(DCMI_FLAG_VSYNCRI);
  DCMI_ClearFlag(DCMI_FLAG_LINERI);
  DCMI_ClearFlag(DCMI_FLAG_LINEMI);
  DCMI_ClearFlag(DCMI_FLAG_FRAMERI);
  DCMI_ClearFlag(DCMI_FLAG_ERRRI);
  DCMI_ClearFlag(DCMI_FLAG_OVFRI);
}
```

Po zainicjowaniu interfejsu w wypadku odczytu danych obrazu w postaci kolejnych linii, zostaje wyłączona opcja pracy z danymi kodowanymi.

Najwygodniej jest kontrolować pracę interfejsu poprzez przerwania generowane przez różne zdarzenia. Najważniejsze DCMI_IT_FRAME oznacza zakończenie przesyłania kolejnej ramki obrazu, DCMI_IT_OVF oraz DCMI_IT_ERR oznaczają wystąpienie błędów w pracy interfejsu DCMI. Kolejne linie programu włączają poszczególne przerwania. Obsługę przerwań należy umieścić w przeznaczonym do tego pliku *stm32f2xx_it.c*. Przykład procedury obsługi przerwań DCMI pokazano na **listingu 4**.

Każde przerwanie może np. ustawiać flagę deklarowaną w programie użytkownika. Stan flagi będzie infor-

Listing 5. Inicjowanie DMA2

```
//DMA2 Stream1 Configuration
#define DCMI_DR_ADDRESS  0x50050028
DMA_DeInit(DMA2_Stream1);
DMA_InitStructure.DMA_Channel=DMA_Channel_1;
DMA_InitStructure.DMA_PeripheralBaseAddr=DCMI_DR_
ADDRESS;
DMA_InitStructure.DMA_Memory0BaseAddr=(uint32_t)bufor_
RAM_danych_obrazka;
DMA_InitStructure.DMA_DIR=DMA_DIR_PeripheralToMemory;
DMA_InitStructure.DMA_BufferSize=rozmiar_przeslania_DMA;
DMA_InitStructure.DMA_PeripheralInc=DMA_PeripheralInc_
Disable;
DMA_InitStructure.DMA_MemoryInc=DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize=DMA_
PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize=DMA_MemoryDataSize_
Byte;
DMA_InitStructure.DMA_Mode=DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority=DMA_Priority_High;
DMA_InitStructure.DMA_FIFOMode=DMA_FIFOMode_Disable;
DMA_InitStructure.DMA_FIFOThreshold=DMA_FIFOThreshold_
Full;
DMA_InitStructure.DMA_MemoryBurst=DMA_MemoryBurst_
Single;
DMA_InitStructure.DMA_PeripheralBurst=DMA_
PeripheralBurst_Single;
DMA_Init(DMA2_Stream1, &DMA_InitStructure);
//Enable DMA2 stream 1 and DCMI interface*/
DMA_Cmd(DMA2_Stream1, ENABLE);
while(DISABLE == DMA_GetCmdStatus ( DMA2_Stream1 ));
DCMI_Cmd(ENABLE);
```

mował procedurę odbioru ramki obrazu o sukcesie lub błędzie transmisji danych. Na koniec należy pamiętać o ustawieniu wektora przerwania interfejsu DCMI w kontrolerze NVIC.

Procedury inicjowania rejestrów DMA

Mechanizm interfejsu DCMI odpowiada za współpracę pomiędzy modułem kamery a mikrokontrolerem. Odebrane dane powinny zostać przesłane do buforu w pamięci RAM mikrokontrolera. Takie przesłanie najwygodniej powierzyć mechanizmowi DMA mikrokontrolera. Ponieważ jest to wewnętrzny mechanizm sprzętowy w trakcie pracy będzie w minimalny sposób obciążał główny program użytkownika oraz co nie bez znaczenia będzie szybki w działaniu.

Mikrokontroler z rodziny STM32F2xx wyposażono w dwa kontrolery DMA. Każdy z nich obsługuje określoną liczbę umownych kanałów i strumieni. Taki podział wynika z faktu, że DMA może obsługiwać wiele różnych źródeł i odbiorców danych. Mogą to być np. porty mikrokontrolera, timery/liczniki, przetworniki oraz właśnie interfejs DCMI. Obsługa każdego źródła danych do przesłania jest powiązana z określonym kanałem i strumieniem. Przypisanie obsługi interfejsu DCMI do konkretnego kanału i strumienia można znaleźć w dokumentacji technicznej mikrokontrolera w tabeli *DMA2 request mapping*. Po decyzji, z którego strumienia DMA chcemy korzystać, należy zainicjować DMA wypełniając pola struktury i wywołując funkcję inicjującą. Na potrzeby tego przykładu poczynione zostały następujące założenia co do obrazu przesyłanego z modułu kamery:

- wielkość obrazka to 160×120 pikseli,
- dane przesyłane będą w formacie 565RGB bez kodowania, co oznacza 2 bajty na piksel,
- każdorazowo przesyłana i zapisywana do buforu będzie 1 ramka obrazu.

Przyjęcie takich założeń oznacza, że obsługiwana transmisja będzie miała stałą wielkość 160×120×2 =38400 bajtów. Bufor w pamięci RAM mikrokontrolera, do którego DMA będzie zapisywało dane obrazu, musi mieć co najmniej taką pojemność. Procedurę inicjowania DMA2 zamieszczono na **listingu 5**.

Znaczenie najważniejszych ustawień wartości inicjujących DMA:

- *DMA_Channel_1* – do transmisji danych obrazu będzie wykorzystany kanał 1 DMA2.
- *DCMI_DR_ADDRESS* – zmienna symboliczna określająca adres urządzenia peryferyjnego (w tym przypadku interfejsu DCMI) będącego źródłem danych do przesłania. Dane do przesłania przez DMA pobierane są z rejestru DCMI_DR będącym chwilowym buforem po odczycie danych z magistrali modułu. Zmienna symboliczna wskazuje na fizyczny adres rejestru w przestrzeni adresowej mikrokontrolera.
- *(uint32_t)bufor_RAM_danych_obrazka* – adres odbiorcy danych (w tym przypadku adres początku bufora dla danych obrazu w pamięci RAM mikrokontrolera).
- *DMA_DIR_PeripheralToMemory* – kierunek transmisji danych (w tym przypadku z urządzenia peryferyjnego do bufora w pamięci).
- *rozmiar_przeslania_DMA* – liczba bajtów danych do przesłania (w tym przypadku rozmiar danych obrazka).
- *DMA_PeripheralInc_Disable* – po kolejnym przesłaniu bazowy adres urządzenia peryferyjnego nie będzie zwiększany automatycznie.
- *DMA_MemoryInc_Enable* – po kolejnym przesłaniu bazowy adres odbiorcy danych będzie automatycznie zwiększany (i będzie wskazywał na miejsce do zapisu w buforze dla kolejnych bajtów danych).
- *DMA_PeripheralDataSize_Word* – liczba bajtów danych pobieranych z interfejsu DCMI. Dane odbierane z magistrali danych modułu zawsze są pakowane do postaci 4 bajtowego słowa. Tabela pokazuje rozmieszczenie poszczególnych bitów koloru w słowie danych obrazu.

Byte address	31:27	26:21	20:16	15:11	10:5	4:0
0	Red n + 1	Green n + 1	Blue n + 1	Red n	Green n	Blue n
4	Red n + 3	Green n + 3	Blue n + 3	Red n + 2	Green n + 2	Blue n + 2

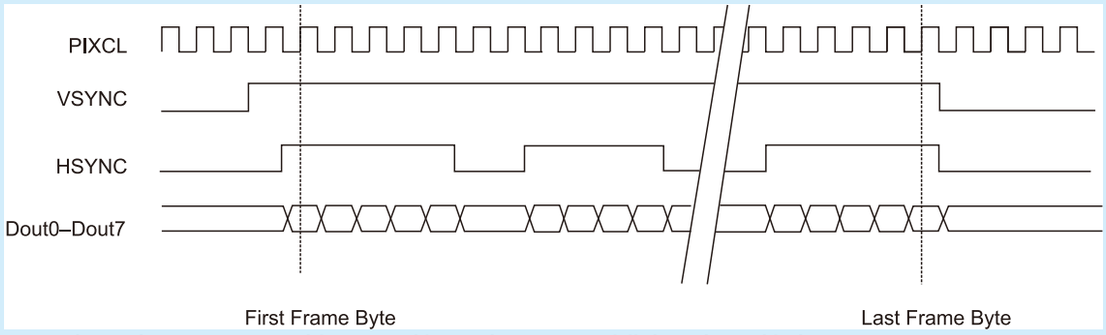
Po przesłaniu nowych ustawień inicjujących do rejestrów DMA może on zostać otwarty podobnie jak interfejs DCMI.

- *DMA_MemoryDataSize_Byte* – liczba bajtów danych zapisywanych w buforze.
- *DMA_Mode_Normal* – tryb pracy DMA normalny, po zakończeniu przesyłania określonej wcześniej liczby bajtów danych DMA automatycznie zakończy pracę.

Wykonanie zdjęcia i odczyt danych ramki obrazu

Jeżeli moduł kamery, interfejs DCMI oraz DMA zostały prawidłowo zainicjowane można przejść do procedury wykonania zdjęcia. Do tego jest niezbędne wykonanie następujących kroków:

1. Przygotowanie rejestrów modułu oraz w razie konieczności DCMI i DMA. Jeżeli parametry rejestrowanego obrazu ulegają zmianie (np. zmienia się format koloru lub wymiary obrazka) należy skorygować ustawienia wewnętrznych rejestrów modułu oraz w razie potrzeby rejestrów DCMI i DMA.
2. Wysłanie do modułu rozkazu wykonania zdjęcia. Zależnie od typu zastosowanego modułu rozkaz wy-



Rysunek 2. Schematyczny przebieg sygnałów podczas transmisji danych ramki obrazu

konania zdjęcia może mieć różną składnię. W przypadku modułu MT9D111/9D131 wysłany rozkaz uruchomi moduł do rejestracji określonej liczby ramek (w tym przypadku jednej) począwszy od następnego aktywnego poziomu linii VSYNC. Moment pojawienia się impulsu, którego zakończenie oznacza start procesu rejestracji danych obrazu można określić odczytując ustawienie wewnętrznego rejestru statusu modułu.

3. Uruchomienie interfejsu DCMI do odczytu danych jednej ramki obrazu.
4. Oczekiwanie na zakończenie przechwytywania danych ramki obrazka.

Przykładową procedurę rozpoczęcia rejestracji i oczekiwanie na jej zakończenie pokazano na **listingu 6**.

Listing 6. Przykładową procedurę rejestracji ramki obrazu

```
void Start_DMA_MT9D111_do_Int_RAM(void)
{
  DCMI_ClearITPendingBit(DCMI_IT_FRAME | DCMI_IT_OVF |
DCMI_IT_ERR | DCMI_IT_VSYNC | DCMI_IT_LINE);
  DCMI_ClearFlag (DCMI_FLAG_FRAMERI | DCMI_FLAG_OVFRI |
DCMI_FLAG_ERRRI | DCMI_FLAG_VSYNCRI | DCMI_FLAG_LINERI);
  dcmi_flag_frameri_bak =FALSE;
  dcmi_flag_errri_bak =FALSE;
  dcmi_flag_ovfri_bak =FALSE;
  DCMI_CaptureCmd(ENABLE);
  while ( dcmi_flag_frameri_bak ==FALSE )
  {
    if (dcmi_flag_errri_bak ==TRUE || dcmi_flag_ovfri_bak
==TRUE)
    {
      break;
    }
  }
  DCMI_CaptureCmd ( DISABLE );
  DCMI_Cmd ( DISABLE );
}
```

Na początku procedury o ile wcześniej nie zostało to zrobione są zerowane flagi związane z przerwaniami interfejsu DCMI. Następnie zostaje uruchomiony interfejs DCMI do odbioru danych z modułu. Odbiór danych rozpocznie się automatycznie w chwili zakończenia sygnału linii sterującej VSYNC. Włączony wcześniej kanał DMA także w tej chwili rozpocznie automatyczny transfer do buforu kolejnych odbieranych danych. Zarówno interfejs DCMI jak i DMA samoczynnie zawieszają pracę podczas aktywnych poziomów sygnału linii HSYNC rozdzielających transmisję danych kolejnych linii obrazu. Na **rysunku 2** przedstawiono schematyczny przebieg sygnałów podczas transmisji danych ramki obrazu.

Kolejny aktywny poziom sygnału linii VSYNC oznacza zakończenie transmisji. Praca interfejsów DCMI i DMA zostaje zawieszona. Interfejs DCMI może zostać wyłączony. Od tej chwili dane obrazu znajdują się w buforze i są dostępne do wykorzystania przez program użytkownika.

Podsumowanie

Opisane procedury każdy użytkownik może dostosować do swoich potrzeb. Jak zawsze, możliwe są różne rozwiązania, także lepsze od wcześniej opisanych. Mam jednak nadzieję, że zaprezentowane przykładowe procedury pozwolą lepiej zrozumieć sposób działania mechanizmu DCMI, który w przypadku mikrokontrolerów STM-32F2XX i STM32F4XX bardzo ułatwia pracę z modułami kamer.

Ryszard Szymaniak, EP