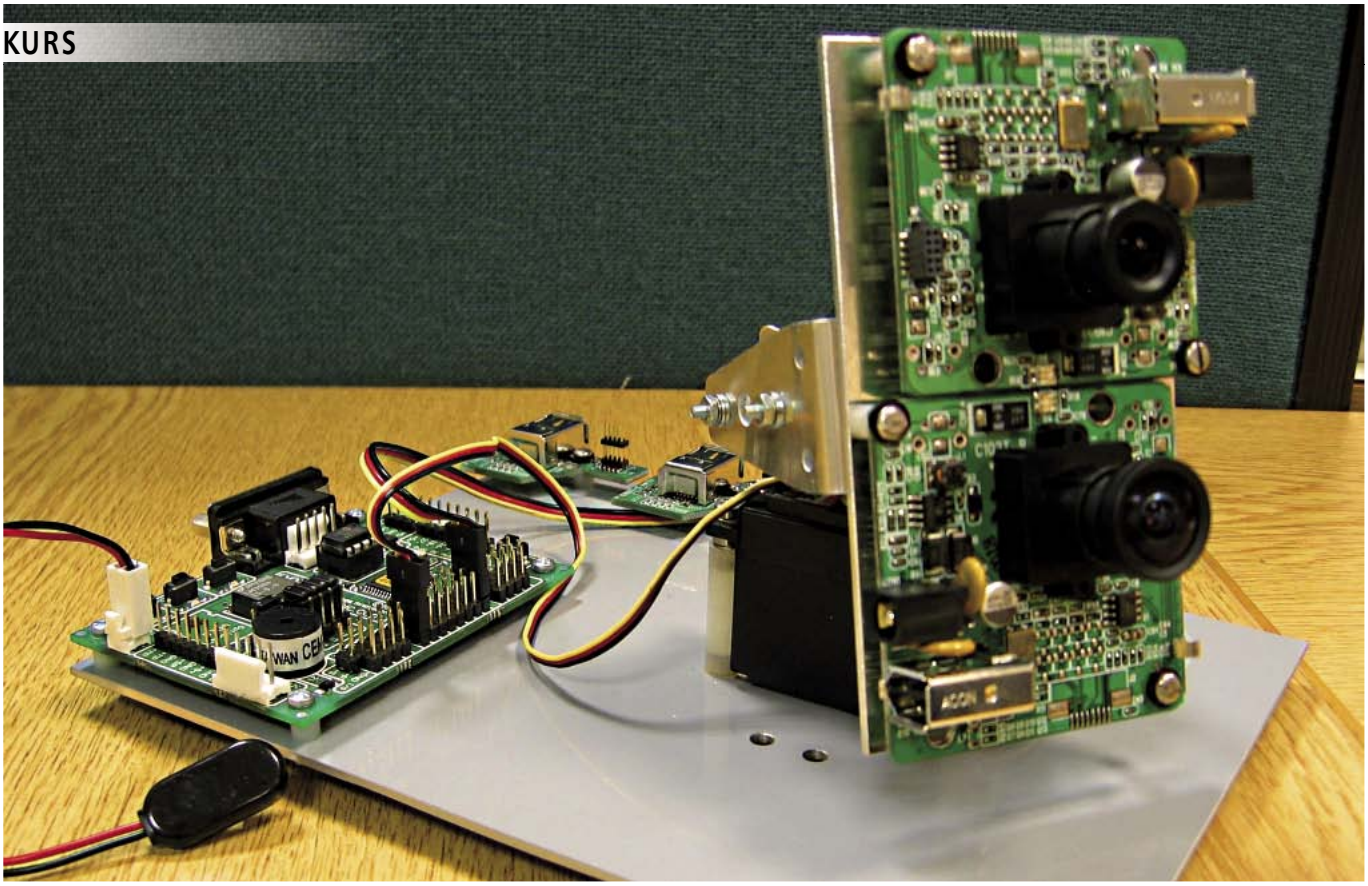


Kamera cyfrowa w urządzeniach mikroprocesorowych (2)

Zapewne wielu Czytelników słyszało opinie, że mikrokontrolery firmy Atmel mają bogaty zestaw układów peryferyjnych. Nawet pomimo wyposażenie niektórych mikrokontrolerów tej firmy może pozytywnie zaskoczyć. Trudno bowiem znaleźć wielu producentów mikrokontrolerów ogólnego przeznaczenia integrujących w swoich produktach układ peryferyjny zapewniający dedykowany interfejs dla kamery cyfrowej.

Dodatkowe materiały na CD i FTP:
host: ep.com.pl, user: 12235, pass: 60u61csy
- pierwsza część kursu
- listingi

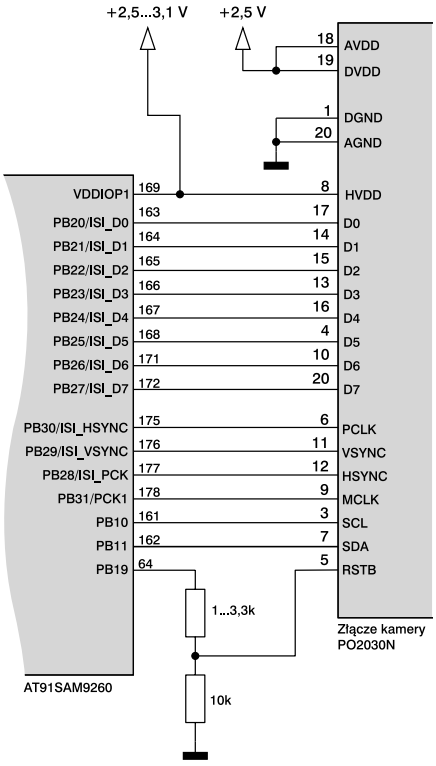
AT91SAM9 (SAM9) to względnie nowa rodzina mikrokontrolerów z rdzeniem ARM. Są one mniej popularne niż np. AT91RM9200, lecz mogą być naprawdę godnymi uwagi następcami tego modelu przy zbliżonej cenie, nowszym rdzeniu (ARM926EJ-S) i o wiele większych możliwościach funkcjonalnych. Mikrokontrolery te mają wiele przydatnych układów peryferyjnych, nie tylko takich, jak omawiany w niniejszym artykule interfejs sensora obrazu, ale także:

– interfejsy służące do sterowania matrycami LCD (np. takimi, które stosuje się w monitorach LCD lub laptopach),
– sprzętowe akceleratory grafiki 2D,
– interfejsy audio (w podstawowych modelach można zaimplementować popularny I²S, a w bardziej rozbudowanych jest 6-kanałowy, sprzętowy układ wspomagający kodowanie AC97).

W niektórych mikrokontrolerach SAM9 istnieje możliwość utworzenia dwóch niezależnych magistral równoległych do komunikacji m.in. z pamięciami SDRAM. Można wtedy jedną pamięć przeznaczyć na dane graficzne, a drugą na pozostałe dane (zmienne, program), dzięki czemu system może działać jeszcze szybciej.

Częstotliwość taktowania rdzenia w mikrokontrolerach SAM9 jest stosunkowo wysoka, ponieważ w najpopularniejszych modelach wynosi typowo ok. 200 MHz, natomiast w wybranych może sięgać nawet 400 MHz (np. AT91SAM9G20). Do tego pamięci cache i szybkie wewnętrzne pamięci RAM (w tym TCM) powodują, że mikrokontrolery SAM9 mogą być postrzegane jako naprawdę uniwersalne, a może nawet multimedialne.

W niniejszym artykule koncentruję się nad bardzo ciekawym układem peryferyjnym znajdującym się w mikrokontrolerach SAM9, jakim jest interfejs sensora obrazu (*Image Sensor Interface*, w skrócie **ISI**). Moduł ten może odbierać, wstępnie przetwarzać i automatycznie zapisywać do pamięci dane uzyskane z typowych, cyfrowych sensorów



Rys. 1. Sposób podłączenia kamery do mikrokontrolera AT91SAM9260

obrazu. Jako przykład sensora obrazu, który można podłączyć do interfejsu ISI, wybrałem moduł kamery cyfrowej PO2030N – jej krótki opis znajduje się w pierwszej części kursu.

Jak podłączyć moduł kamery do mikrokontrolera SAM9?

Dysponując podstawowymi wiadomościami na temat sygnałów elektrycznych kamery PO2030N, możemy zająć się podłączeniem jej do najprostszego mikrokontrolera SAM9 – AT91SAM9260 (**rys. 1**). Jak wspomniałem w poprzedniej części kursu, kamera ma interfejs elektryczny o napięciach niższych (2,5...3,1 V) niż typowy zakres napięć na wyprowadzeniach mikrokontrolerów SAM9 (3,3 V). Z pomocą tutaj przychodzi możliwość zasilania wyprowadzeń zewnętrznych modułu ISI z innego napięcia niż pozostałe porty. Napięcie ustalające poziomy logiczne wyprowadzeń PB10, PB11, PB20... PB31 mikrokontrolerów AT91SAM9260 (do nich można podłączać kamery) podawane jest na nóżkę o nazwie VDDIOP1.

Wyjścia wideo kamery (D7...D0, HSYNC, VSYNC, PCLK) najlepiej jest podłączyć do dedykowanych wyprowadzeń interfejsu ISI mikrokontrolera. Z sygnałów synchronizacji HSYNC i VSYNC można w swoim projekcie zrezygnować, jeśli do synchronizacji zastosujemy kody SAV, EAV i *blanking* (jednak przedstawiony dalej kod testowy korzysta z HSYNC i VSYNC). Wg producenta ważne jest jedynie, aby sensor obrazu wysyłał w każdej klatce obrazu co najmniej jedną linię wypełnioną sygnałami *blank*, dzięki czemu mikrokontroler będzie mógł rozpoznać początek i koniec kolejnych obrazów. Zarówno moduł ISI jak i omówiony wcześniej sensor obrazu PO2030 mają dość szerokie możliwości konfiguracji, więc można poprawnie przesyłać dane pomiędzy tymi układami na kilka różnych sposobów: zarówno jeśli chodzi o różne formaty kodowania koloru w pikselach, jak i sposoby synchronizacji.

Na rys. 1 wyprowadzenia magistrali I²C (SDA i SCL) podłączone są do mikrokontrolera i w założeniach sterowane mają być w pełni programowo jako uniwersalne wejścia-wyjścia (odpowiednio PB11 i PB10). Wybierając wyprowadzenia PB11 i PB10 kontrolera PIO jako przykład utworzenia I²C sugerowałem się tym, że napięcie ich poziomu logicznego „1" definiowane jest nóżką VDDIOP1, więc mogą one bez dodatkowych buforów pracować przy niskim napięciu interfejsu kamery (są to także wyprowadzenia ISI, lecz tutaj nieużywane ze względu na węższą magistralę kamery). Niskie napięcie VDDIOP1 definiuje także stan logiczny wyprowadzenia PB31, które co prawda nie wchodzi w skład wyprowadzeń ISI, lecz jest multipleksowane z wyjściem zewnętrznego sygnału zegarowego PCK1 (*Peripheral Clock 1*). W omawianym sposobie podłączenia kamery sygnał zerujący (RSTB) sensora podłączony został do wyprowadzenia PB19 mikrokontrolera. Z racji, że wyprowadzenie to ma poziomy 0...3,3 V, zastosowany jest prosty dzielnik rezystorowy ograniczający napięcie do poziomu bezpiecznego dla kamery. Zależności czasowe sygnału RSTB nie są wyjątkowo krytyczne czasowo, więc zastosowanie dzielnika nie wpływa negatywnie na pracę całego układu. **Uwaga**: dla przejrzystości, na rys. 1 nie zaznaczono elementów filtrujących zakłócenia napięcia zasilania. Sposób filtracji napięć zasilających można znaleźć w nocie katalogowej konkretnego modułu kamery.

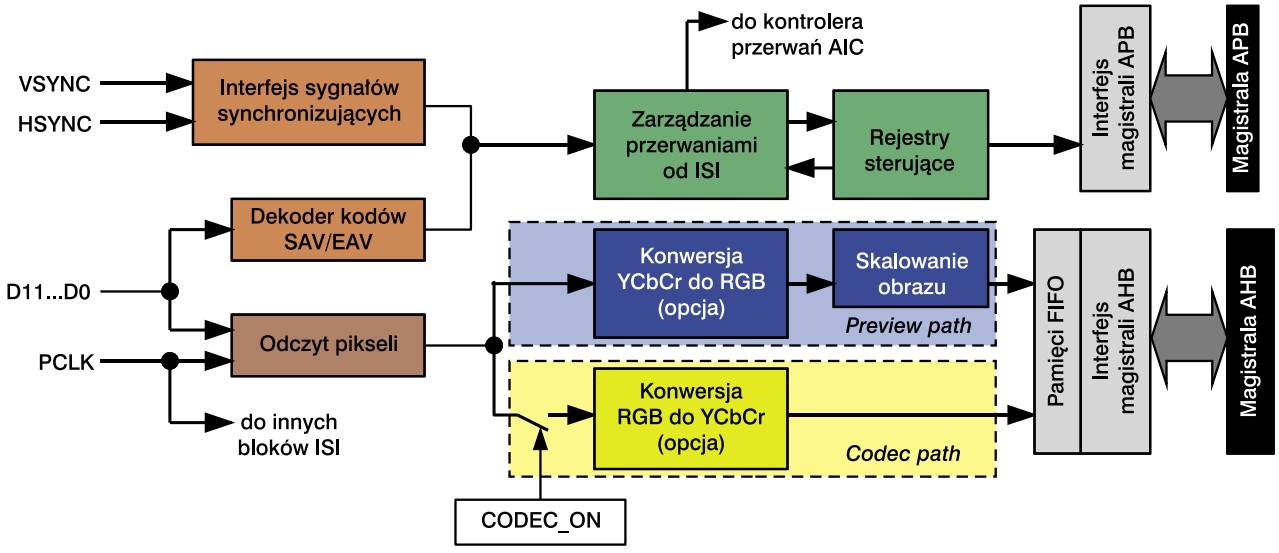
Moduł ISI

Moduł ISI pozwala na odbiór i wstępne przetwarzanie danych nadchodzących z sensora obrazu. Jego uproszczony schemat blokowy znajduje się na **rys. 2**. Pełny schemat blokowy modułu ISI znajdziemy w nocie katalogowej mikrokontrolera z tym interfejsem (choćby AT91SAM9260).

Mikrokontrolery rodziny SAM9 mają dwa rodzaje wewnętrznych magistral:

- APB (*Advanced Peripheral Bus*) służącą głównie do komunikacji pomiędzy rdzeniem (wykonującym program) a układami peryferyjnymi,
- AHB (*Advanced High Performance Bus*) mające zastosowanie w komunikacji blokami mikrokontrolera wymagającymi najszybszych transferów danych. W mikrokontrolerach AT91SAM9260 zaimplementowano specjalną matrycę (*Bus Matrix*) sześciu magistral AHB.

Moduł ISI korzysta z obu wymienionych magistral: dostęp do jego rejestrów kontrolnych odbywa się przez APB (jak w każdym układzie peryferyjnym mikrokontrolerów SAM), natomiast do transmisji wideo używana jest bezpośrednio magistrala AHB.



Rys. 2. Uproszczony schemat blokowy modułu ISI

List. 1.
```
uint8_t camWriteReg(uint8_t reg, uint8_t value)
{
        i2cStart();
        uint8_t ack = OK;
        if(i2cWrite(0xDC)==I2C_NACK) ack=FAIL;
        if(i2cWrite(reg)==I2C_NACK) ack=FAIL;
        if(i2cWrite(value)==I2C_NACK) ack=FAIL;
        i2cStop();
        return(ack);
}
```

List. 2.
```
uint8_t camReadReg(uint8_t reg)
{
        i2cStart();
        i2cWrite(0xDC);
        i2cWrite(reg);
        i2cStart();
        i2cWrite(0xDD);
        uint8_t ret = i2cRead(I2C_NACK);
        i2cStop();
        return(ret);
}
```

Wiąże się to z faktem, że obraz jest zapisywany przez przeznaczony dla ISI kanał DMA. Kanał ten można określić jako „pełnoprawny”, ponieważ nie jest nadzorowany w żaden sposób przez kontroler PDC (*Peripheral DMA Controller*). Interfejs wideo modułu ISI zawiera dodatkowe bufory FIFO redukujące wpływ różnic pomiędzy różnymi sygnałami zegarowymi (sygnał synchronizacji pikseli PCLK i sygnał zegara AHB). Szybkość działania modułu ISI jest na tyle duża, że bez problemu można uzyskać płynny (praktycznie nieopóźniony) podgląd obrazu z kamery na wyświetlaczu LCD.

Sygnał wideo z sensora może przechodzić przez moduł ISI jedną z dwóch „ścieżek”. Jedna z nich nazywa się ***Preview path*** (ścieżka podglądu), natomiast druga nazywana jest ***Codec path*** (ścieżka kodeka).

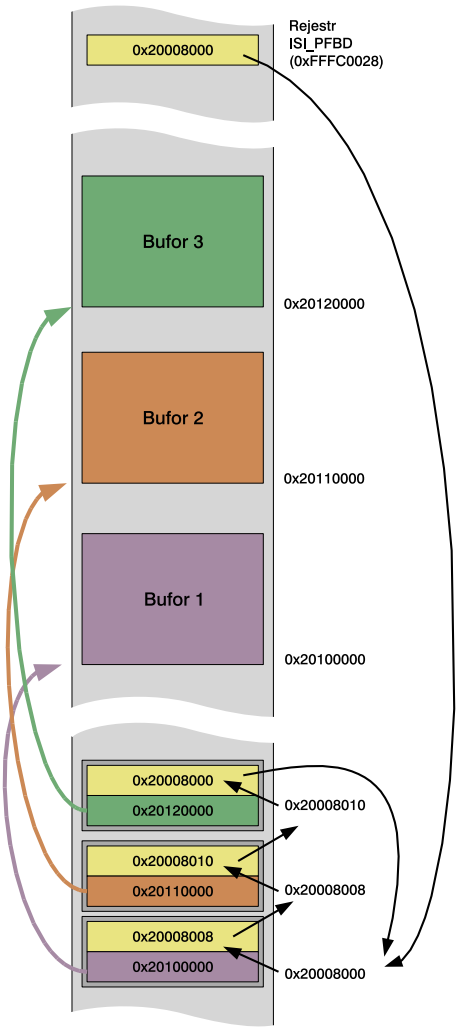
Ścieżka kodeka ma służyć do przetwarzania i przechowywania obrazu. Jak wspomniałem wcześniej, w tym celu przydatny jest format YCbCr. Nawet jeśli sensor obrazu nie posiadałby możliwości generowania sygnału w tym formacie, w ścieżce kodeka istnieje możliwość konwersji obrazu RGB na YCbCr.

Ścieżka podglądu służy do wyświetlania podglądu obrazu z kamery na wyświetlaczu LCD dołączonym do mikrokontrolera. Umożliwia ona konwersję sygnału wideo z formatu YCbCr na format RGB przydatny przy wyświetlaniu oraz skalowanie obrazu z sensora (pomniejszanie). Pomniejszanie odbywa się sprzętowo z zadanymi współczynnikami dla rozmiarów X i Y obrazu tak, że w pamięci znajduje się od razu obraz pomniejszony do zadanych rozmiarów. Dodatkowo warto wspomnieć, że pomniejszanie odbywa się w sposób „inteligentny”, czyli przy zastosowaniu specjalnego filtru decymacyjnego, dzięki czemu pomniejszony obraz nie jest nienaturalnie poszarpany.

Przyjrzyjmy się bliżej sposobowi zapisu poszczególnych klatek obrazu podglądu. Moduł ISI zapewnia hardware'owe wspomaganie przechowywania obrazów za pomocą listy (*linked list*). W celu zapisania obrazów podglądu w pamięci musimy poinformować moduł ISI, pod którym adresem chcemy zapisać podgląd. W tym celu należy utworzyć w pamięci operacyjnej mikrokontrolera proste struktury danych składające się z dwóch słów – w dokumentacji nazywane są one deskryptorami klatek **FBD** (*Frame Buffer Descriptor*). Pierwsze ze słów wcho-

List. 3.
```
#define ISI                         AT91C_BASE_HISI
#define ISI_ID              AT91C_ID_HISI

#define CAM_RESET_MASK      AT91C_PIO_PB19

#define CAM_PIO             AT91C_BASE_PIOB
#define CAM_HW_MASK         ( 0x7FF00000 )
#define CAM_MCK_MASK        AT91C_PIO_PB31
#define CAM_MCK_ABSEL       PIO_ASR

#define CAM_IMAGE_X_SIZE   640
#define CAM_IMAGE_Y_SIZE   480
#define CAM_PREVIEW_X_SIZE 128
#define CAM_PREVIEW_Y_SIZE 96
#define CAM_BYTES_PER_PIXEL         2
#define CAM_DECIMATION              (5*16)
```

List. 4.
```
void camInit(void *previewBuffer, void *imageBuffer)
{
  //aktywacja sygnału zerowania kamery
  CAM_PIO->PIO_PER = CAM_RESET_MASK;
  CAM_PIO->PIO_SODR = CAM_RESET_MASK;
  CAM_PIO->PIO_OER = CAM_RESET_MASK;

  //impuls reset
  delay_ms(1);
  CAM_PIO->PIO_CODR = CAM_RESET_MASK;
  delay_ms(1);
  CAM_PIO->PIO_SODR = CAM_RESET_MASK;
  delay_ms(1);
  i2cInit();

  //ustawiamy format danych wyjściowych na RGB565
  camWriteReg(0x51,0x09);

  //uaktywnienie sygnału PCK1 w układzie PMC
  AT91C_BASE_PMC->PMC_SCER = AT91C_PMC_PCK1;

  //konfiguracja sygnału zegarowego PCK1
  AT91C_BASE_PMC->PMC_PCKR[1] = AT91C_PMC_CSS_PLLA_CLK | AT91C_PMC_PRES_CLK_8;

  //wybór układu peryferyjnego ISI (tutaj B)
  CAM_PIO->PIO_BSR = CAM_HW_MASK;

  //wybór układu peryferyjnego PCK1 (tutaj A)
  CAM_PIO->PIO_ASR = CAM_MCK_MASK;

  //odłączenie kontrolera PIO od sygnałów kamery
  CAM_PIO->PIO_PDR = CAM_HW_MASK | CAM_MCK_MASK;

  //włączenie sygnału zegarowego modułu ISI
  AT91C_BASE_PMC->PMC_PCER = 1<<AT91C_ID_HISI;

  //inicjalizacja jedynego używanego FBD
  //ustawiamy adres bufora z obrazem podglądu
  pFbd->currentBufferAddress = (uint32_t) previewBuffer;
  //ustawiamy adres "następnego" FBD
  pFbd->nextFBDAddress = (uint32_t) pFbd;

  //informujemy ISI, gdzie jest zdefiniowany przez nas FBD
  ISI->ISI_PFBD = (uint32_t) pFbd;

  //ustawiamy rozmiar obrazu podglądu
  camPreviewSize(CAM_PREVIEW_X_SIZE,CAM_PREVIEW_Y_SIZE,CAM_DECIMATION);

  //ustawiamy adres bufora właściwego obrazu
  ISI->ISI_CDBA = (uint32_t) imageBuffer;

  //programowy reset modułu ISI
  ISI->ISI_CR1 = AT91C_ISI_RST;
  while(ISI->ISI_SR & AT91C_ISI_SOFTRST);

  //w CR1 odwracamy tylko aktywność sygnału PCLK i włączamy ISI
  ISI->ISI_CR1 = AT91C_ISI_PIXCLK_POL;

  while((ISI->ISI_SR & AT91C_ISI_DIS));

  const uint32_t CR1_IM_HSIZE_MASK = (CAM_IMAGE_X_SIZE-1) << 16;
  const uint32_t CR1_IM_VSIZE_MASK = (CAM_IMAGE_Y_SIZE-1) << 0;

  //w CR1 ustawiamy przestrzeń kolorów, format danych wejściowych
  //i definiujemy rozmiar obrazu ze ścieżki kodeka
  ISI->ISI_CR2 =  AT91C_ISI_COL_SPACE | AT91C_ISI_RGB_MODE_RGB_565 | CR1_IM_
HSIZE_MASK | CR1_IM_VSIZE_MASK;

}
```



Rys. 3. Przykład zastosowania 3 deskryptorów FBD obsługujących 3 bufory z danymi

dzących w skład FBD będzie mówiło, pod jakim adresem znajduje się bufor na dane graficzne. Drugie słowo FBD będzie mówiło, gdzie znajduje się następny FBD z adresem kolejnego bufora obrazu, itd. Wystarczy poinformować moduł ISI, gdzie znajduje się pierwszy FBD, a reszta znajdzie się sama poprzez adresy kolejnych buforów (ISI „wie", że pierwsze słowo to adres bufora, a drugie słowo to adres następnego FBD). Nie będzie złym rozwiązaniem wskazanie w ostatnim FBD jako adresu następnego bufora adresu pierwszego FBD, tak żeby lista wróciła do początku. Na rys. 3 można zobaczyć przykład zastosowania trzech deskryptorów FBD obsługujących trzy bufory z danymi. Adres pierwszego FBD znajduje się w rejestrze ISI_PFBD, natomiast adres „następnego FBD" zawarty w ostatnim FBD zawiera ten sam adres co rejestr ISI_PFBD, czyli adres pierwszego FBD. Można także zastosować jeden bufor: w tym celu należy w drugim słowie FBD (adres „następnego FBD") wskazać adres pierwszego słowa FBD.

Przykład inicjalizacji ISI do współpracy z PO2030N

W tej części omawiam sposób napisania funkcji wykonującej minimalną inicjalizację modułu ISI do współpracy z modułem kamery PO2030N. Przedstawione dalej fragmenty kodu kompilowane były przy użyciu kompilatora GCC wchodzącego w skład pakietu WinARM. Kod testowany był bez użycia systemu operacyjnego, na mikrokontrolerze AT91SAM9260.

Do komunikacji z kamerą cyfrową przydatne będą funkcje realizujące wymianę danych przez zaimplementowaną programowo magistralę I²C. Podstawowymi funkcjami będą *camWriteReg* (**list. 1**) zapisująca bajt (*value*) do wybranego rejestru kamery (*reg*) oraz *camReadReg* (**list. 2**) odczytująca bajt z wybranego rejestru (*reg*) i zwracająca jego wartość. Realizacja software'owej magistrali I²C, czyli używane w list. 1 i list. 2 funkcje *i2cRead*, *i2cWrite*, *i2cStart* oraz *i2cStop*, nie są wyjątkowo ambitne, więc nie będą tutaj omawiane. Przy używaniu kamery PO2030N z domyślnymi ustawieniami (m.in. format obrazu YCbCr) magistrala I²C może nawet nie być potrzebna (wejściowy format YCbCr może być automatycznie konwertowany na RGB w ścieżce podglądu).

Przed przystąpieniem do dalszych działań warto utworzyć makrodefinicje mówiące o podłączeniu najważniejszych sygnałów, dla kamery cyfrowej (**list. 3**). Makrodefinicje *ISI* oraz *ISI_ID* to odpowiednio alias adresu bazowego oraz identyfikatora układu peryferyjnego ISI. Makrodefinicja *CAM_PIO* mówi o tym, do którego kontrolera PIO podłączone są wszystkie sygnały modułu kamery. *CAM_HW_MASK* definiuje, które wyprowadzenia wybranego kontrolera PIO używane są do podłączenia samego sygnału wideo. Dodatkowe makrodefinicje to *CAM_MCK_MASK* i *CAM_RESET_MASK* definiujące wyprowadzenia używane jako wyjście sygnału zegarowego oraz zerującego moduł kamery. W przykładowym kodzie rozdzielczość obrazu wynikowego (ze ścieżki kodeka) zdefiniowana jest w makrodefinicjach *CAM_IMAGE_X_SIZE* i *CAM_IMAGE_Y_SIZE* (640×480 pikseli). Natomiast w makrodefinicjach *CAM_PREVIEW_X_SIZE* i *CAM_PREVIEW_Y_SIZE* wybrano rozmiar obrazu podglądu (w przykładzie ustawiamy rozdzielczość 128 na 96 pikseli). Ważne jest, by razem ze zmianą rozdzielczości obrazu podglądu zmieniać także omówiony dalej współczynnik decymacji (tutaj zadany makrodefinicją *CAM_DECIMATION*).

Proponowana, przykładowa funkcja inicjalizująca *camInit* będzie pobierała dwa parametry: adres bufora dla obrazu uzyskanego ze ścieżki kodeka (*imageBuffer*) oraz adres bufora dla obrazu podglądu (*previewBuffer*). Oba te parametry najwygodniej będzie przekazać za pomocą wskaźników. Kod przykładowej funkcji inicjalizującej przedstawiony został na **list. 4**. Pierwszymi czynnościami wykonywanymi w niej jest inicjalizacja wyprowadzenia zerującego moduł kamery, a następnie podanie impulsu zerującego. Dalej inicjalizowana jest magistrala I²C i wpisywana jest wartość *0x08* do rejestru kamery o adresie *0x51* (jest to *ISP Control Register 2*). Ten wpis ustawi format wyjściowy kamery na RGB565. Ustawianie formatu przeprowadzone jest wyłącznie dla pokazania, w jaki sposób można to uczynić – z tego polecenia można zrezygnować i wtedy kamera będzie wysłała dane w trybie YCbCr. W dalszej części funkcji *camInit* uaktywniany jest sygnał zegarowy dla kamery generowany na wyjściu PCK1 mikrokontrolera oraz inicjalizowane są wyprowadzenia kontrolera PIOB, do których przyłączona jest kamera. Po włączeniu PCK1, w kontrolerze PMC uaktywniany jest także sygnał zegarowy taktujący moduł ISI. Przeglądając notę katalogową AT91SAM9260 niewyposażonego w moduł APMC (*Advanced Power Manager Controller*), warto mieć na uwadze, że sygnał dla kamery cyfrowej nie może być generowany przez APMC (jednak producent twierdzi inaczej). **W AT91SAM9260 sygnał zegarowy na wyprowadzeniu PB31 można uzyskać tylko jako wyjście PCK1 (moduł PMC), natomiast sygnał ISI_MCK nie występuje.**

Potrzebny w dalszej części funkcji *camInit* deskryptor FBD można definiować np. za pomocą prostej struktury danych, takiej jak na **list. 5**. Zdefiniowana struktura użyta jest także do utworzenia deskryptora FBD (*fbd*) oraz – dla wygody – wskaźnika do niego

List. 5.

```
typedef struct
{
        uint32_t currentBufferAddress;
        uint32_t nextFBDAddress;
} TFrameBufferDescriptor;

static TFrameBufferDescriptor fbd;
static TFrameBufferDescriptor *pFbd = (TFrameBufferDescriptor *) &fbd;
```

(*pFbd*). Z racji, że zajmujemy się najprostszym przypadkiem, tworzony jest tylko jeden deskryptor FBD.

W dalszej części funkcji inicjalizującej przechodzimy do konfiguracji samego modułu ISI. Te czynności rozpoczynamy od ustawienia wskaźników w jedynym deskryptorze FBD (danym wskaźnikiem *pFbd*). Jako adres bufora z obrazem (*pFbd –>currentBufferAddress*) wpisujemy adres zadany wskaźnikiem *previewBuffer* przekazanym jako argument do funkcji *camInit*. Natomiast w miejscu adresu następnego bufora FBD (*pFbd–>nextFBDAddress*) wpisujemy adres aktualnego FBD, przez co dane będą zapisywane tylko do jednego bufora. Aby struktura danych *fbd* rzeczywiście stała się deskryptorem FBD, należy poinformować moduł ISI, na jakim adresie umieszczony jest pierwszy bufor FBD – czynimy, to wpisując adres ze wskaźnika *pFbd* do rejestru ISI_PFBD. Ważne jest, aby inicjalizacja wskaźnika do pierwszego FBD odbyła się jeszcze przed włączeniem odczytu obrazu z sensora (wyzerowanie bitu ISI_DIS w rejestrze ISI_CR1). Jeśli zaniechamy ustawienia FBD, a rozpoczniemy odczyt obrazu, aktualny FBD będzie przemieszczał się po przestrzeni adresowej w sposób niemający sensu.

Rozmiary obrazu podglądu ustawiane są za pomocą prostej funkcji *camPreviewSize* (**list. 6**). Funkcja pobiera 3 argumenty: dwa pierwsze (*x* i *y*) to wymiary obrazu podglądu w pikselach. Ostatni argument (*decimation*) to współczynnik zależny od stosunku wielkości obrazu z kamery do obrazu podglądu. W przykładzie z list. 5 kamera ma rozdzielczość 640 na 480 pikseli, a chcemy uzyskać obraz podglądu 128 na 96 pikseli. Do argumentu *decimation* wpiszemy wartość 5·16 (list. 3), bo 5=640/128=480/96, a współczynnik decymacji ustawiamy z rozdzielczością 1/16. Jeśli zmienimy proporcje obrazu podglądu w stosunku do obrazu oryginalnego, to współczynniki decymacji dla wymiarów X i Y będą się nieco różniły, ale nie trzeba się tym bardzo przejmować, ponieważ wartość zadawanego współczynnika decymacji nie jest bardzo krytyczna. Oznacza to, że gdy nie będzie się dało ustawić „idealnego" wyliczonego współczynnika decymacji, wtedy także nic strasznego się nie stanie.

Wreszcie, na samym końcu funkcji *camInit* wykonywane są ustawienia rejestrów kontrolnych modułu ISI. W tym celu przeprowadzane są wpisy do rejestrów ISI_CR1 oraz ISI_CR2 (*ISI Control Register*).

Rejestry kontrolne ISI_CR1 i ISI_CR2

W prezentowanym przykładzie funkcji inicjalizujacej korzystamy z najbardziej podstawowych ustawień, dlatego wpis do

Nazwa Rejestru: ISI_CR1
Dostęp do rejestru: R/W

31	30	29	28	27	26	25	24
SFD							
23	22	21	20	19	18	17	16
SLD							
15	14	13	12	11	10	9	8
CODEC_ON	THMASK		FULL	-	FRATE		
7	6	5	4	3	2	1	0
CRC_SYNC	EMB_SYNC	-	PIXCLK_POL	VSYNC_POL	HSYNC_POL	ISI_DIS	ISI_RST

Rys. 4. Rejestr kontrolny układu ISI ISI_CR1

Nazwa Rejestru: ISI_CR2
Dostęp do rejestru: R/W

31	30	29	28	27	26	25	24
RGB_CFG		YCC_SWAP		-	IM_HSIZE		
23	22	21	20	19	18	17	16
IM_HSIZE							
15	14	13	12	11	10	9	8
COL_SPACE	RGB_SWAP	GRAYSCALE	RGB_MODE	GS_MODE	IM_VSIZE		
7	6	5	4	3	2	1	0
IM_VSIZE							

Rys. 5. Rejestr kontrolny układu ISI ISI_CR2

numer bitu	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
kolor	-	B4	B3	B2	B1	B0	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

Rys. 6. Sposób zapisu danych podglądu w pamięci

List. 6.
```
void camPreviewSize(uint16_t x, uint16_t y, uint8_t decimation)
{
        ISI->ISI_PDECF = decimation;
        ISI->ISI_PSIZE = (uint32_t)(x-1)<<16 | (y-1);
}
```

ISI_CR1 (**rys. 4**) oznacza ustawienie tylko jednego bitu o nazwie PIXCLK_POL. Przy okazji także kasujemy bit ISI_DIS, przez co rozpoczynamy przetwarzanie danych przez ISI (włączamy funkcjonalność ISI). Ustawienie bitu PIXCLK_POL spowoduje odczytywanie danych nadchodzących z kamery na opadającym zboczu sygnału PCLK – w takim trybie domyślnie wysyła dane PO2030N. Możemy także wybrać, jaki poziom sygnałów synchronizujących HSYNC i VSYNC będzie uznawany za aktywny, ustawiając bądź zerując bity odpowiednio HSYNC_POL i VSYNC_POL. Bit EMB_SYNC pozwala na włączenie trybu odczytu danych zawierających kody SAV i EAV zastępujące sygnały synchronizacji. Bit CODEC_ON pozwala na włączenie ścieżki kodeka i pobranie właściwego obrazu.

Podstawowe ustawienia w rejestrze ISI_CR2 (**rys. 5**) są bardziej złożone. Definiujemy w nim bowiem rozmiary obrazu uzyskanego ze ścieżki kodeka oraz format wejściowy. Do zdefiniowania rozmiarów obrazu przydatne są dwie stałe reprezentujące maski bitowe pól IM_HSIZE i IM_VSIZE rejestru ISI_CR2. Do tych pól wpisujemy odpowiednio szerokość i wysokość obrazu w pikselach, odejmując 1 piksel od każdego rozmiaru. Za pomocą bitu COL_SPACE możemy wybrać, czy kamera wysyła dane do modułu ISI w jednym z formatów RGB (COL_SPACE=1), czy w YCbCr (COL_SPACE=0). Bitem COL_SPACE włączamy i wyłączamy konwersję obrazu RGB na YCbCr oraz YCbCr na RGB w ścieżkach kodeka i podglądu.

Ustawiając bit RGB_MODE, wymuszamy odczytywanie sygnału w formacie RGB565. Gdybyśmy pozostawili RGB_MODE=0, to moduł ISI próbowałby odczytać dane z kamery w formacie RBG888 (po jednym bajcie na składową). W polu RGB_CFG możemy dodatkowo zdefiniować, w jaki sposób mikrokontroler będzie rozkodowywał dane z sygnału wideo (w jakiej kolejności pojawiać się będą poszczególne składowe koloru). W przykładzie korzystamy z pola RGB_CFG ustawionego na 00, dlatego nic nie musimy do niego wpisywać.

Moduł ISI w ścieżce podglądu wykonuje automatyczną konwersję formatu na RGB555 (nawet jeśli kamera wysyła dane w formacie RGB565, RGB888 czy YCbCr). Format RGB555 jest szczególnie przydatny w mikrokontrolerach SAM9, w których znajduje się kontroler matryc LCD (kontrolera LCD nie ma w AT91SAM9260). Dane podglądu zapisywane są w pamięci w sposób przedstawiony na **rys. 6**. Widzimy na nim ułożenie bitów reprezentujących poszczególne kolory, jeśli obraz podglądu re-

prezentowany będzie przez tablicę elementów 16-bitowych (wskaźnik do takiej tablicy można przekazać jako argument *previewBuffer* w wywołaniu funkcji *camInit*). Zwróćmy uwagę, że w formacie RGB555 kompatybilnym z kontrolerami LCD wbudowanymi w mikrokontrolery SAM9, kolory czerwony i niebieski ułożone są inaczej niż w standardzie RGB565 i w rzeczywistości taki format można nazwać „BGR555". Niestety nota katalogowa mikrokontrolerów milczy na temat zamiany kolorów R z B w formacie RGB555 (żeby się o tym dowiedzieć, trzeba otworzyć notę aplikacyjną kontrolera LCD).

Dalsze pomysły i kilka uwag praktycznych

Przedstawiony w artykule sposób zapewnienia interfejsu pomiędzy sensorem obrazu a mikrokontrolerem SAM9 jest tylko jedną z możliwości realizacji tego zadania. Z racji szerokich możliwości konfiguracji modułu ISI mikrokontrolerów SAM9, nie powinno stanowić problemu podłączenie także innych sensorów obrazu (modułów kamer) do tego mikrokontrolera.

Warto docenić możliwości i prostotę implementacji wyświetlania obrazu w trybie podglądu. W wielu prostych projektach tryb podglądu może być wykorzystany do zapisywania docelowych obrazów przede wszystkim ze względu na jego elastyczność, np. możliwość sprzętowego, „inteligentnego" skalowania obrazu oraz wygodną konfigurację buforów za pomocą deskryptorów FBD. Wielkość obrazu podglądu możemy zmieniać zależnie od potrzeb także po inicjalizacji, w czasie działania programu.

Chcąc wykonywać własne eksperymenty z modułem kamery PO2030N, warto pamiętać o wyłączeniu niektórych opcji automatycznego ustawiania parametrów obrazu. Jeśli np. zdecydujemy się na modyfikacje wzmocnienia poszczególnych składowych kolorystycznych, przydaje się wcześniejsza deaktywacja automatycznej regulacji balansu bieli (AWB). AWB jest domyślnie aktywny po włączeniu bądź zresetowaniu modułu kamery, podobnie jak automatyczny dobór czasu ekspozycji.

W większości urządzeń korzystających z modułu ISI prawdopodobnie aktywne będą także pamięci cache mikrokontrolera. Z tego powodu trzeba mieć na uwadze, że dane zapisywane przez kanały DMA do obszarów, na których aktywna jest pamięć DCache (*Data Cache*), muszą być specjalnie traktowane. Można albo poinformować jednostkę MMU, że wybrany obszar zostanie/został zapisany w trybie DMA, albo (najwygodniej) w ogóle nie korzystać z pamięci cache na tych obszarach. Podobnie będzie z tworzeniem deskryptora FBD. Moduł ISI potrzebuje danych w FBD znajdujących się **w pamięci operacyjnej** mikrokontrolera. Jeśli uzupełnimy pola FBD, gdy deskryptor ten będzie znajdował się na obszarze z aktywną pamięcią DCache, zawartość FBD może nie znaleźć się od razu w pamięci operacyjnej (pozostanie w cache) w momencie uaktywnienia transferów danych w ścieżce podglądu. To prowadzi do wspomnianego wcześniej, bezsensownego „przeskakiwania" aktualnego deskryptora FBD po przestrzeni adresowej, tak jakbyśmy w ogóle nie ustawili adresów w FBD.

Zastosowania kamery w urządzeniu z mikrokontrolerem (zarówno te hobbystyczne, jak i przemysłowe) można mnożyć bez końca. Dysponując już gotowym obrazem z kamery, mamy otwartą drogę np. do zbudowania kamery z interfejsem Ethernet (korzystając z modułów ISI i EMAC). Możemy także zaprojektować i zbudować kamerę będącą elementem wkomponowanym w automatykę inteligentnego budynku, np. monitorującą ruch w pomieszczeniach. A może zdalnie sterowany „szpiegowski" pojazd przesyłający zarejestrowane obrazy przez interfejs Bluetooth?

Robert Brzoza-Woch
rabw@poczta.fm