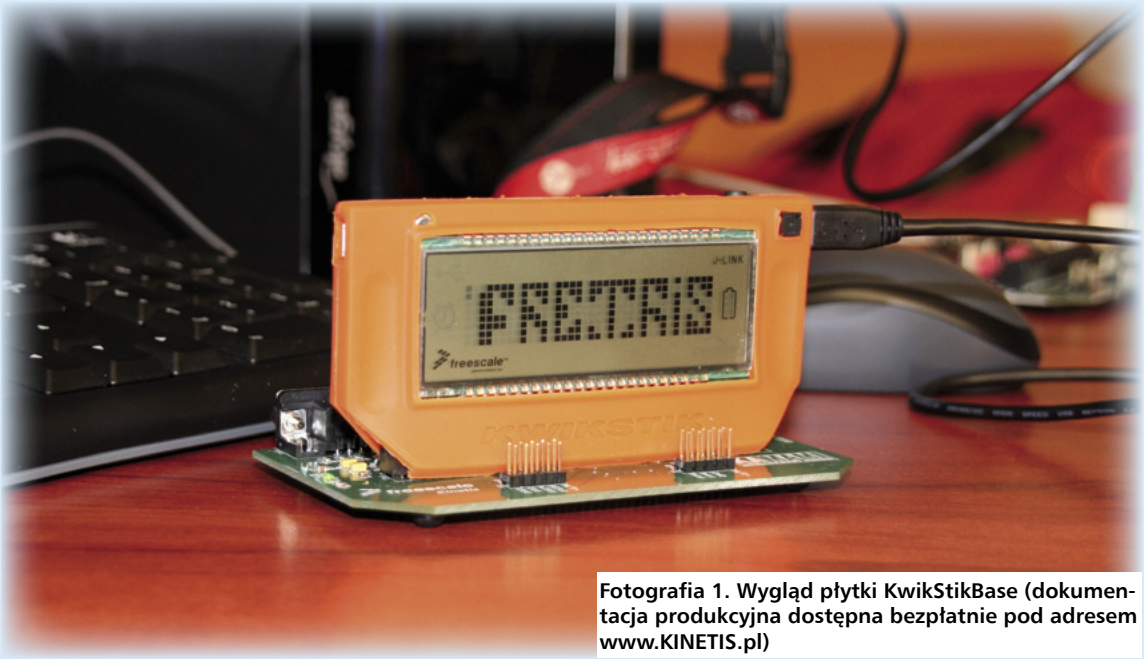


Fotografia 1. Wygląd płytki KwikStikBase (dokumentacja produkcyjna dostępna bezpłatnie pod adresem www.KINETIS.pl)

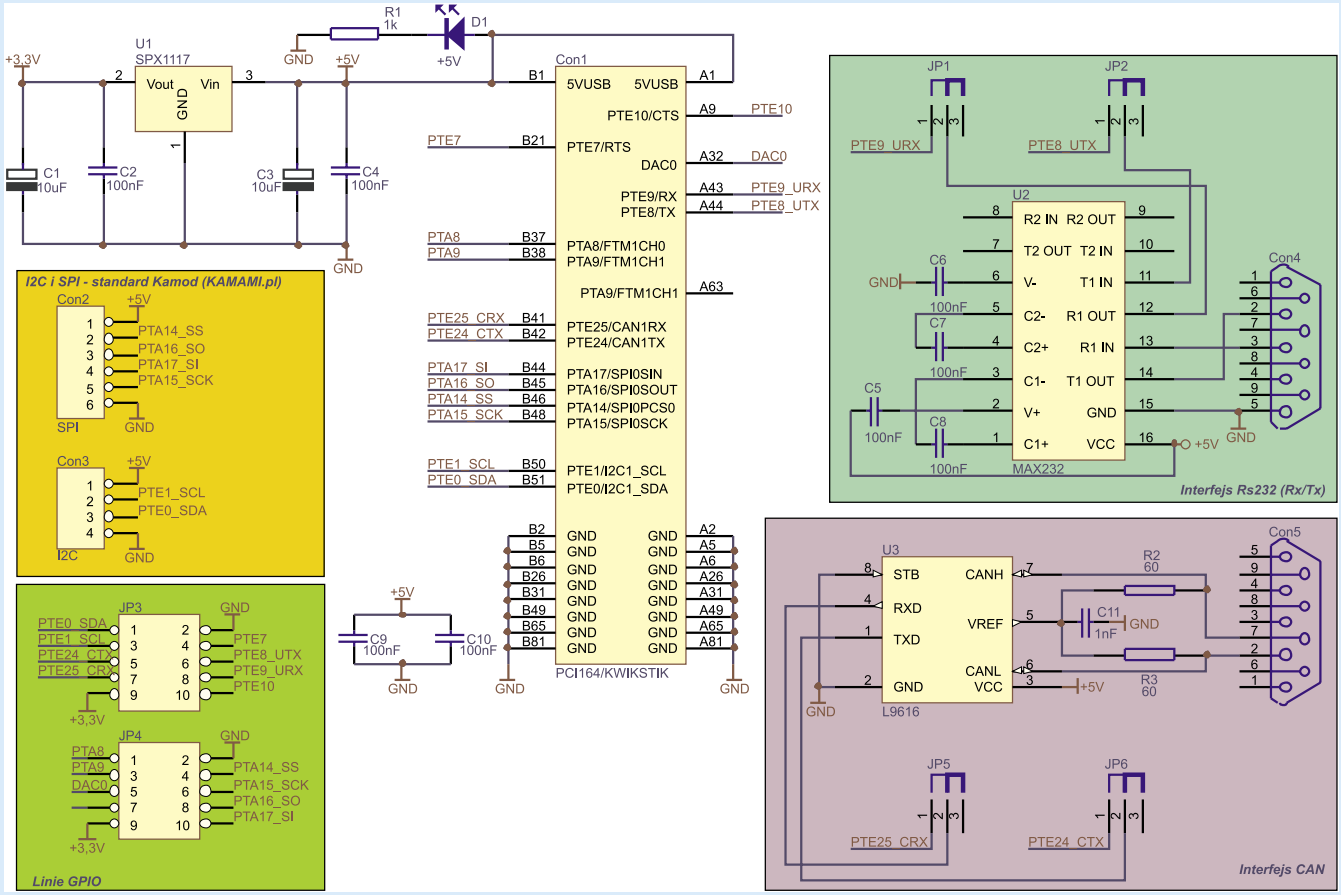
Freescale Kinetis KwikStik: ćwiczenia z Cortex-M4

Etap 2: obsługa GPIO

Możliwości zestawu KwikStik K40 są duże dzięki bogatemu „wyposażeniu pokładowemu". Prezentację możliwości zastosowanego w zestawie mikrokontrolera pokażemy od obsługi linii GPIO.

Podstawowe cechy płytki KwikStikBase:
- stabilna mechanicznie podstawa dla KwikStika umożliwiająca wygodne korzystanie z wyświetlacza LCD
- zasilanie z USB (via KwikStik)
- interfejs RS232 (Tx/Rx) z gniazdem DB9F
- interfejs CAN z gniazdem DB9M
- wbudowany stabilizator +3,3 V
- sygnalizacja włączonego zasilania LED
- 14 linie GPIO wyprowadzone na złącza IDC10 w rastrze 2,54 mm
- 1 linia DAC wyprowadzona na złącze IDC10 w rastrze 2,54 mm
- I2C i SPI wyprowadzone na złącza KAmod (uniwersalne, umożliwiają wygodne dołączenie do KwikStika modułów KAmod firmy KAMAMI.pl)
- możliwość zasilania układów zewnętrznych napięciem +3,3 V

Przykład przygotowano środowisku Keil uVision, którego wersja ewaluacyjna (z ograniczeniem wielkości kodu wynikowego) jest dostępna na stronie internetowej producenta: *www.keil.com*. Ponieważ dostęp do linii GPIO w KwikStiku jest trudny, zastosowano płytkę bazową KwikStikBase (**fotografia 1**), której opis i dokumentację produkcyjną można na stronie *www.KINETIS.pl*. Monitorowanie stanu GPIO zrealizowano za pomocą modułu KAmodLED8, który został dołączony do złącza KwikStikBase z wyprowadzonymi 8 liniami GPIO. Schemat elektryczny płytki KwikStikBase pokazano na **rysunku 2**. Na schemacie wydzielono poszczególne bloki funkcjonalne zestawu, obydwa zastosowane interfejsy mogą zostać odłączone od mikrokontrolera zastosowanego na KwikStiku za pomocą jumperów, które zalecamy przełączać parami:
- interfejs RS232: JP1 i JP2, odpowiednio linie Rx i Tx interfejsu UART mikrokontrolera,



Rysunek 2. Schemat elektryczny zestawu KwikStikBase

- interfejs CAN: JP5 i JP6, odpowiednio linie Rx i Tx interfejsu CAN mikrokontrolera.

Rezygnacja z wyprowadzania na złącze interfejsu RS232 sygnałów synchronizujących transmisję danych CTS i RTS spowodowała, że nie ma konieczności uwzględniania błędu wyprowadzenia sygnału RTS, opisanego w artykule. Z powodu specyficznych rozwiązań zastosowanych na płytce KwikStik, nie może on być zasilany napięciem podawanym na złącze krawędziowe, jedyną możliwą drogą zasilania jest jedno ze złącz USB.

Przygotowując program najpierw musimy stworzyć na dysku katalog, w którym umieścimy nowy projekt (np. *Kinetis_GPIO*). Aby stworzyć projekt wybieramy w menu środowiska Keil uVision pozycję *Project–> New uVision Project*, następnie wskazujemy wcześniej utworzony katalog i wpisujemy nazwę projektu, tu ponownie użyłem nazwy *Kinetis_GPIO*. W kolejnym kroku musimy wskazać typ mikrokontrolera z którego będziemy korzystać (MK40X256VMD100 zastosowany w zestawie KwikStik K40). Po zatwierdzeniu wyboru środowisko zapyta, czy życzymy sobie, aby do projektu został dodany plik z kodem rozruchowym mikrokontrolera – co potwierdzamy. Mamy w tej chwili gotowy pusty projekt. Musimy jeszcze skonfigurować projekt, aby mógł korzystać z debugera *Segger J-Link*, który jest wbudowany w zestaw uruchomieniowy KwikStik. W tym celu musimy otworzyć okno właściwości projektu (w menu *Project–> Options for Target 'Target 1'*), w zakładce *Debug*.

Dzięki temu J-Link będzie wykorzystywany do debugowania. Aby można było za jego pomocą również programować pamięć Flash mikrokontrolera należy przejść do zakładki *Utilities*, tam ponownie wskazać J-Linka, kliknąć przycisk *Settings*, w nowym oknie za pomocą przycisku Add dodać algorytm programowania dla naszego mikrokontrolera. Po dodaniu algorytmu trzeba jeszcze w sekcji *RAM for Algorithm* zwiększyć wielkość pamięci udostępnianej debugerowi do 0x0F00. Ostatnią operacją jest zaznaczenie opcji *Reset and Run* w sekcji *Download Function*, co spowoduje, że po zaprogramowaniu pamięci mikrokontrolera program w niej zapisany będzie automatycznie uruchamiany.

Teraz do katalogu projektu kopiujemy plik *C:\Keil\ARM\INC\Freescale\K40\MK40N512MD100.h*, dzięki któremu będziemy mogli odwoływać się do rejestrów za pomocą ich nazw, co jest znacznie łatwiejsze i bardziej czytelne niż korzystanie z adresów. W końcu tworzymy plik właściwego programu (*File–> New*), zapisujemy go jako main.c w katalogu projektu. Korzystając z menu kontekstowego okienka Project dodajemy plik *main.c* do projektu, w tej chwili jesteśmy gotowi do pisania programu. Polecam zachować taki pusty projekt na przyszłość, żeby nie powtarzać za każdym razem tych samych czynności.

Konfigurowanie mikrokontrolera

Pisanie programu rozpoczniemy od dołączenia pliku nagłówkowego MK40N512MD100.h:

```
#include "MK40N512MD100.h".
```

Kod rozruchowy wymaga, abyśmy utworzyli funkcję *SystemInit*, która skonfiguruje mikrokontroler, która zwykle będzie zawierać konfigurację zegarów, ale w naszym przypadku ograniczymy się do wyłączenia watchdoga, który uniemożliwiałby debugowanie programu.

```
void SystemInit()
{
  // Wyłączenie watchdoga
  WDOG->UNLOCK = (uint16_t)0xC520u;  /* Key 1 */
```

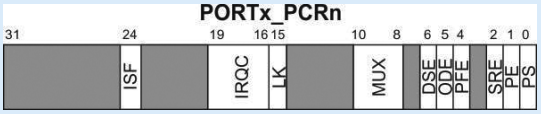
```
  WDOG->UNLOCK  = (uint16_t)0xD928u; /* Key 2 */
  WDOG->STCTRLH = (uint16_t)0x01D2u;
}
```

Konfigurowanie linii GPIO

Procedura konfigurowania linii GPIO składa się z trzech kroków:

1. Odblokowanie zegara portu. Aby wykorzystać linie mikrokontrolera jako GPIO należy najpierw odblokować zegar portu, w naszym przypadku będzie to port E. Zegary peryferii mikrokontrolera Kinetis odblokowuje się ustawiając wartości odpowiednich bitów rejestrów SIM_SCGCx (*System Lock Gating Control*) na wartość 1, w tym przypadku musimy ustawić bit o nazwie PORTE w rejestrze SIM_SCGC5.

2. Konfiguracja linii. Do konfiguracji linii portów służą rejestry PORTx_PCRn (*Pin Control Register*), gdzie x to nazwa portu, np. E, n to numer linii, przykładowo aby skonfigurować linię PTE0 należy dokonać zapisu do rejestru PORTE_PCR0.



Funkcje interesujących nas bitów rejestru PCR są następujące:

- PS (*Pull Select*) – jeśli bit PE ma wartość 1, to bit PS służy do wyboru, czy linia ma być podciągnięta do masy (kiedy PS=0) czy do napięcia zasilania (kiedy PS=1),
- PE (*Pull Enable*) – dołączenie do linii rezystora podciągającego,
- SRE (*Slew Rate Enable*) – wybór czasu narastania sygnału, krótki dla wartości 0, długi dla wartości 1,
- ODE (*Open Drain Enable*) – jeśli ODE=1, to linia jest wyjściem typu otwarty dren,
- MUX (*Pin Mux Control*) – wybór funkcji linii.

Rzeczą, która może zaskoczyć konstruktorów korzystających wcześniej z innych mikrokontrolerów może być fakt, że domyślnie linie I/O są skonfigurowane nie jako linie *general-purpose*, a są dołączone do różnych modułów peryferyjnych, przykładowo linia PTE0 (pierwsza linia portu E) domyślnie jest dołączona do przetwornika analogowo-cyfrowego. Aby wykorzystać linię jako GPIO musimy przełączyć ją w tryb ALT1 (pełna lista funkcji linii I/O znajduje się w rozdziale 10.3 podręcznika użytkownika).

Aby skonfigurować linię jako cyfrowe wyjście typu *push-pull* (zwykłe wyjście podające 0 lub 3,3 V) należy więc wpisać do jej rejestru konfiguracyjnego następującą wartość:

```
PORTE->PCR[n] = 1 << PORT_PCR_MUX(1);
```

Warto zwrócić uwagę na definicję PORT_PCR_MUX w pliku MK40N512MD100.h, która zwalnia nas z konieczności przesuwania bitów i poprawia czytelność kodu.

3. Ustalenie kierunku linii I/O. Aby ustawić kierunek linii GPIO jako wyjście należy nadać bitowi o odpowiadającym jej numerze w rejestrze GPIOx_PDDR (Port Data Direction Register) wartość 1, na przykład dla linii PTE0 należy zapisać jedynkę do bitu zerowego rejestru GPIOE_PDDR.

Kompletna funkcja konfigurująca linie 0, 1, 8, 9, 10, 11, 24, 25 (wykorzystane do sterowania LED) portu E wygląda następująco:

```
void GPIOInit()
{
  // Odblokowanie zegara portu E
  SIM->SCGC5 |= (1 << SIM_SCGC5_PORTE_SHIFT);
  // Ustawienie funkcji linii na GPIO
  PORTE->PCR[0] = PORT_PCR_MUX(1);
  PORTE->PCR[1] = PORT_PCR_MUX(1);
  PORTE->PCR[8] = PORT_PCR_MUX(1);
  PORTE->PCR[9] = PORT_PCR_MUX(1);
  PORTE->PCR[10] = PORT_PCR_MUX(1);
  PORTE->PCR[11] = PORT_PCR_MUX(1);
  PORTE->PCR[24] = PORT_PCR_MUX(1);
  PORTE->PCR[25] = PORT_PCR_MUX(1);

  // Ustawienie linii jako wyjścia
PTE->PDDR|=1|(1<<1)|(1<<8)|(1<<9)|(1<<
10)|(1<<11)|(1<<24)|(1<<25);
}
```

Sterowanie liniami GPIO. Teraz wystarczy w funkcji *main* zaimplementować pętlę, która będzie naprzemiennie zapalać diody dołączone do linii 0, 8, 10, 24 oraz 1, 9, 11, 25. Aby ustawić stan wysoki na linii I/O trzeba wpisać do odpowiadającego jej bitu rejestru PDOR portu jedynkę, aby ustawić stan niski – wyzerować ten bit. Funkcja *Delay* zawiera pustą pętlę opóźniającą.

```
int main()
{
  GPIOInit();
  while(1)
  {
    PTE->PDOR=1|(1<<9)|(1<<11)|(1<<24);
    Delay();
    PTE->PDOR=(1<<1)|(1<<8)|(1<<10)|(1<<25);
    Delay();
  }
}
```

Kompilowanie i uruchomienie programu

Aby skompilować program należy wcisnąć F7, wynik kompilacji widać w oknie *Build Output*, jeśli kompilacja przebiegła pomyślnie możemy uruchomić debugowanie naciskając Ctrl+F5, po uruchomieniu debugowania program zatrzyma się na wywołaniu funkcji SystemInit, po naciśnięciu F5 (*Run*) program powinien się uruchomić, czego efektem są migające na przemian diody o parzystych i nieparzystych numerach. W trybie debugowania można również ustawiać pułapki (*breakpoint*, klawisz F9), czyli zaznaczyć miejsce w kodzie programu po osiągnięciu którego program ma się zatrzymać, można również wykonać pracę krokową (F10), czyli wykonać program instrukcja po instrukcji.

Jacek Zbysiński

Więcej informacji:
Dodatkowe informacje i materiały do pobrania są dostępne pod adresem *www.KINETIS.pl*