POLITECHNIKA CZĘSTOCHOWSKA,
WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI
Instytut Inteligentnych Systemów Informatycznych

Laboratorium: Systemy Wbudowane w Układach Sterowania

Ćwiczenie nr 3

TEMAT: Kontroler portów równoległych PIO (ang. *Paraller Input/Output Controler*) mikrokontrolera AT91SAM7X256.

1. Podstawowe właściwości

Port mikrokontrolera to grupa jego wyprowadzeń zewnętrznych (ang. *pin*-ów). W mikrokontrolerach 8-bitowych jeden port grupuje najczęściej osiem pinów, w 16-bitowych 16 a w 32-bitowych 32 piny mikrokontrolera (wynika to głównie z rozmiaru rejestrów specjalnych kontrolera). Każde wyprowadzenie portu może służyć do wysyłania lub odbierania sygnałów przez mikrokontroler do/z urządzeń zewnętrznych.

1.1 Kontroler wejść/wyjść równoległych (PIO)

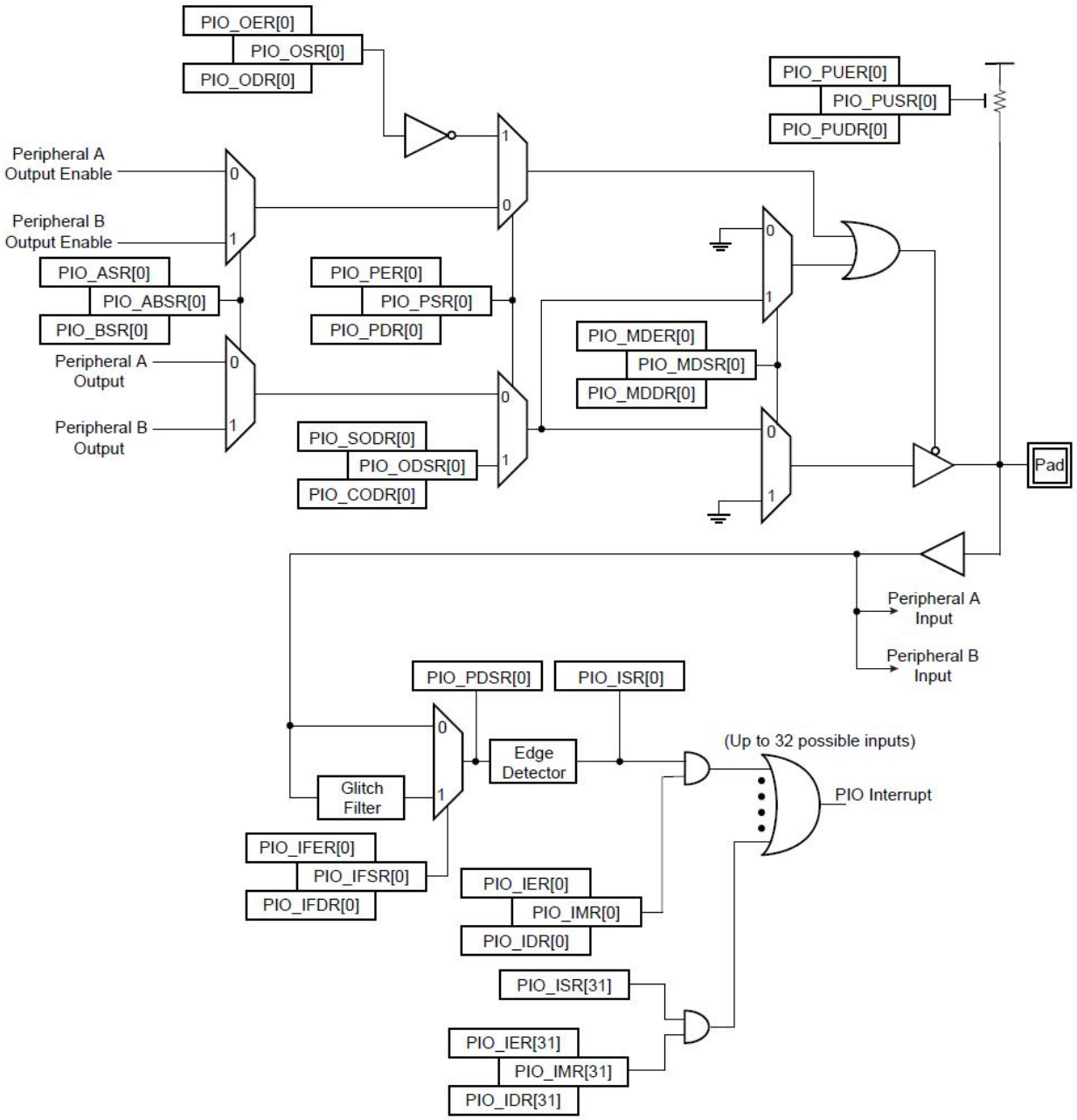
Poszczególne linie portu mikrokontrolera mogą być:
- wejściowe, do których podłączona jest klawiatura, przetwornik analogowo-cyfrowy lub inne sygnały wejściowe z różnorodnych czujników
- wyjściowe – wysyłające sygnały na zewnątrz np. na wyświetlacz, diody świecące LED, sygnalizator dźwiękowy lub głośnik (ang. *buzzer*) itp.
- wejściowo/wyjściowe – **konfigurowalne przez użytkownika poprzez specjalne** rejestry w taki sposób, aby mogły pełnić w danej chwili rolę wejść lub wyjść oraz **spełniać inne specyficzne funkcje**.

W AT91SAM7X256 programowalny kontroler wejść/wyjść (PIO) obsługuje do 32 w pełni konfigurowalnych linii wejściowo-wyjściowych. Każda linia może być ogólnego przeznaczenia linią wejścia/wyjścia (GPIO - ang. *general purpose input output*) lub być przypisana do pełnienia specyficznej funkcji układu peryferyjnego mikrokontrolera. Poglądowy schemat logiki działania pojedynczej linii portu PIO przedstawiono na rys. 1.

Bardziej szczegółowe informacje można znaleźć w dokumentacji procesora doc6120.pdf http://www.atmel.com/Images/doc6120.pdf (rozdział „27. Parallel Input/Output Controller (PIO)").

Każda z linii kontrolera PIO ma następujące możliwości:
- generowanie przerwań w reakcji na wykrycie zmiany poziomu logicznego
- filtrowanie zakłóceń o czasie trwania krótszym niż połowa okresu systemowego sygnału zegarowego
- konfiguracji wyjścia jako tzw. otwarty dren (ang. *open drain*) – dzięki czemu mozliwa jest praca wielu nadajników na jedną linię
- możliwości „podciągnięcia" niesterowanej (przez żaden nadajnik) linii do wysokiego poziomu logicznego (tzw. *pull-up*)

Ponadto, kontroler PIO umożliwia synchroniczne sterowanie (tzw. w jednym takcie następuje jednoczesna zmiana) do 32 linii wyjściowych w pojedynczej operacji zapisu.



Rys. 1 Schemat logiczny PIO dla pojedynczej linii.

1.2 Multipleksowanie linii portu

W przypadku, gdy linie we/wy sa tylko liniami ogólnego przeznaczenia, tzn. nie multipleksowane z funkcjami we/wy układów peryferyjnych, tylko kontroler PIO decyduje o stanie linii we/wy.

1.3 Zewnętrzne linie przerwań

Sygnały przerwań **FIQ** i **IRQ0** do **IRQn** (n=0..31) sa najczesciej multipleksowane z liniami PIO. Do obsługi przerwań kontrolery PIO są traktowane jako elementy peryferyjne użytkownika. Oznacza to, że linie przerwań kontrolera PIO są podłaczone do zródeł przerwan od 2 do 31. Przerwanie kontrolera PIO może być generowane tylko wtedy, gdy włączony jest zegar dla PIO.

1.4 Zarządzanie mocą w PIO

Kontroler zarządzania mocą (PMC) steruje dostarczaniem sygnału zegarowego dla kontrolera PIO po to, by ograniczyć pobór mocy w przypadku gdy użycie zegara nie jest konieczne. W operacjach zapisu do rejestrów kontrolera PIO nie jest potrzebne włączenie sygnału zegarowego. Jakkolwiek niektóre operacje na liniach PIO, np. zmiana stanu linii portu poprzez odczytanie rejestru statusowego (TOGGLE) lub generacja przerwania wymaga włączenia sygnału zegarowego.

1.5 Sterowanie liniami we/wy

Włączanie lub wyłączanie linii portu odbywa sie poprzez zapis do rejestrów **PIO_OER** (*Output Enable Register*) i **PIO_ODR** (*Output Disable Register*). Stan każdej linii można odczytać w **PIO_OSR** (*Output Status Register*). Wartosc „0" na wybranej linii oznacza, że może ona być tylko wykorzystana jako wejście. Natomiast „1" oznacza, że dana linia jest sterowana przez kontroler PIO.

O poziomie logicznym na danej linii decyduja dwa rejestry: **PIO_SODR** (*Set Output Data Register*) i **PIO_CODR** (*Clear Output Data Register*). Zapis do tych rejestrów powoduje odpowiednio: ustawienie 1 lub 0 na wybranej linii PIO. Operacje te maja swoje odzwierciedlenie w rejestrze **PIO_ODSR** (*Output Data Status Register*), który reprezentuje dane wpisane na liniach we/wy.

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Offset	Register	Name	Access	Reset
0x0070	Peripheral A Select Register[5]	PIO_ASR	Write-only	–
0x0074	Peripheral B Select Register[5]	PIO_BSR	Write-only	–
0x0078	AB Status Register[5]	PIO_ABSR	Read-only	0x00000000
0x007C to 0x009C	Reserved			
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved			

Tabela. 1. Mapa rejestrów kontrolera PIO zaczerpnięta z dokumentacji procesora..

2. Przykład programu wykorzystującego port równoległy

2.1 Informacje ogólne

W przypadku SAM7X dysponujemy dwoma kontrolerami PIO: PIOA i PIOB. Każdy kontroler PIO, tak jak wszystkie inne moduły peryferyjne, ma swój identyfikator (ID) oraz adres bazowy (*Base address*), czyli najniższy adres swojego fragmentu przestrzeni adresowej mikrokontrolera (odpowiadający rejestrowi z offsetem zero) – tabela 1.
Konfiguracja kontrolerów PIO odbywa sie za pomocą rejestrów 32-bitowych. Przyjęto przy tym założenie, że każdy bit rejestru kontrolera PIO odpowiada jednemu z jego wyprowadzeń zewnętrznych. Przykładowo, bit **20** w rejestrach kontrolera **PIOA** będzie odpowiadał wyprowadzeniu zewnętrznemu **PA20** mikrokontrolera.

2.2 Włączenie sygnału zegarowego kontrolera PIO

Zanim zaczniemy prace z większością układów peryferyjnych mikrokontrolerów SAM7, trzeba najpierw włączyć ich sygnał zegarowy. Robi sie to przez wpis do rejestru układu **PMC** (*Power Management Controller*). W przypadku właczania taktowania modułu PIOB ten wpis
to:

```
AT91C_BASE_PMC->PMC_PCER  = (1 << AT91C_ID_PIOB);
```

W ten sposób ustawiamy bit o numerze **AT91C_ID_PIOB** w rejestrze **PCER** modułu **PMC** (o adresie bazowym **AT91C_BASE_PMC**), czyli po wykonaniu powyższej linii kodu, kontroler **PIOB** będzie taktowany sygnałem zegarowym i będzie można korzystać z jego pełnej funkcjonalności. Jak wspomniano wcześniej, **AT91C_ID_PIOB** to liczba oznaczająca numer identyfikacyjny danego układu peryferyjnego. W przypadku AT91SAM7X256 identyfikator modułu PIOB wynosi 3. Wartosc identyfikatora to także miejsce danego układu w rejestrach modułów PMC i AIC. Oczywiscie, jesli bedziemy chcieli właczyc zegar dla kontrolera PIOA, analogicznie napiszemy:

```
AT91C_BASE_PMC->PMC_PCER = (1<<AT91C_ID_PIOA);
```

Dla ścisłości należy dodać, że liczba kryjąca sie pod makrodefinicja **AT91C_ID_PIOA** i stanowiąca identyfikator modułu PIOA to 2.

2.3 Konfiguracja jako wyjście: sterowanie diodami LED

Diodę LED podłączamy zazwyczaj przez rezystor ograniczający maksymalny prąd. Jego wartość typowo wynosi od kilkuset omów do kilku kiloomów. Porty PIO moga zarówno dostarczać, jak i pochłaniać prad (*current sink, current source*), wiec dwa przykładowe sposoby podłaczenia diod LED to:
· anoda diody w kierunku portu i katoda podłączona do masy zasilania (ujemny biegun) - wtedy stan wysoki na porcie spowoduje przepływ prądu i świecenie diody LED (rysunek 6. la)) i wykorzystujemy zdolnosc portu PIO do current source (prad płynie z portu przez diode do masy), · anoda w kierunku napiecia zasilania 3,3 V, a katoda w

kierunku portu - w takiej konfiguracji dioda bedzie swieciła, gdy na wyjsciu mikrokontrolera bedzie stan niski i wykorzystujemy zdolnosc portu PIO do current sink (prad płynie ze zródła napiecia 3,3 V przez diode do portu).
W przypadku zestawu ewaluacyjnego diody sa podłaczone wg drugiego sposobu, wiec wyzerowanie linii PIO odpowiadajacej diodzie LED w stan wysoki spowoduje zaswiecenie diody. Linie sterujace diodami LED podłaczonymi do gniazda **EXT** na płytce mikrokontrolera to PB23 i PB30 (bity nr 23 i 30 w odpowiednich rejestrach kontrolera PIOB).

Uwaga: Podświetlenie typu LED wyświetlacza LCD sterowane jest linią PB20 kontrolera PIOB. W tym przypadku sterowane jest ono w taki sposób, że ustawienie stanu wysokiego na porcie PB20 powoduje załączenie podświetlenia LED wyświetlacza.
Szczegółowe dane oraz schemat płytki ewaluacyjnej SAM7-EX256 można pobrać ze strony: https://www.olimex.com/Products/ARM/Atmel/SAM7-EX256/resources/SAM7-EX256.pdf

W analogiczny sposób sterowane jest wyjście wbudowanego na płytce małego głośniczka (***buzzera***). Do jego sterowania należy użyć linii **PB19**. Ustawienie tej linii portu w stan wysoki powoduje wypchnięcie membrany głośnika, a ustawienie stanu niskiego cofnięcie membrany. Aby wygenerować dźwięk należy cyklicznie (np. w odstępach 1 milisekundy dla generowanej częstotliwości dźwięku wynoszącej 500Hz) zmieniać stan wyjścia PB19 na przeciwny.
Uwaga: należy pamiętać o ustawieniu jumpera SPEAKER, w taki sposób aby połączone były piny o numerach 1-2.
Dodatkowo możliwa jest regulacja głośności dźwięku buzzera poprzez potencjometr obrotowy (pokrętło) oznaczony opisem *volume*.

2.3.1 Pierwszy sposób sterowania portami I/O

Polegał bedzie on na ustawianiu wyjsc kontrolera PIO za pomoca wpisów do rejestru **PIO_SODR** (w stan wysoki) oraz **PIO_CODR** (w stan niski). Aby kontroler PIO rzeczywiscie ustawiał dane wyprowadzenie w stan wysoki i niski, należy wczesniej skonfigurowac sam kontroler i odpowiednie bity jego rejestrów. W przypadku opisywanego tutaj sposobu, musimy wpisac odpowiednie wartosci do poniżej wymienionych rejestrów.

a) **PIO_PER**. czyli *Peripheral Enable Register* (rejestr właczenia układu peryferyjnego).
Wpisanie „1" do któregos bitu tego rejestru spowoduje uaktywnienie kontrolera PIO na wyprowadzeniu odpowiadajacemu temu bitowi. Przykładowo, uaktywnienie kontrolera sterowania wyprowadzeniem PB23 przeprowadzimy nastepujacym wpisem do bitu 23 rejestru PER układu PIO o adresie bazowym AT91C_BASE_PIOB:

```
AT91C_BASE_PIOB->PIO_PER = AT91C_PIO_PB23;
```

Jesli nie ustawimy odpowiednio bitu PB23 w rejestrze PIO_PER, kontrole nad wyprowadzeniem PB23 moga miec inne specyficzne układy peryferyjne mikrokontrolera.

b) **PIO_OER**, czyli Output Enable Register (rejestr właczajacy wyjscie). Funkcja tego rejestru jest adekwatna do jego nazwy: wyprowadzenia zewnetrzne kontrolera PIO odpowiadajace poszczególnym bitom rejestru zostana skonfigurowane jako wyjscia, jesli wpiszemy wartosci „1" do tych bitów. Przykładowo: chcemy, aby wyprowadzenie PB23, do którego mamy podłaczona czerwona diode LED, było wyjsciem, wiec piszemy:

```
AT91C_BASE_PIOB->PIO_OER = AT91C_PIO_PB23;
```

Jesli wyprowadzenie **PB23** jest kontrolowane przez PIO (czyli dokonalismy wczesniej wpisu do rejestru PIO_PER), zostanie ono skonfigurowane do pracy jako wyjscie. Na koniec właczanie i wyłaczanie diody LED. Dokonujemy tego tak, jak wspomniano wczesniej, przez wpisy do rejestrów **PIO_SODR** oraz **PIO_CODR** w sposób analogiczny jak podczas konfiguracji:

```
AT91C_BASE_PIOB->PIO_CODR = AT91C_BASE_PB23; // zaswiecamy diode LED
AT91C_BASE_PIOB->PIO_SODR = AT91C_BASE_PB23; // wyłaczamy diode LED
```

2.3.2 Drugi sposób sterowania portami I/O

Tym razem posłużymy sie inną możliwoscia kontroli sygnału wyjściowego, mianowicie za pomocą rejestru **PIO_ODSR**, czyli *Output Data Status Register*. Jeśli skonfigurujemy któreś wyprowadzenie portu PIO tak jak w poprzednim podpunkcie, odczytując rejestr PIO_ODSR będziemy mogli dowiedzieć sie, które wyprowadzenia ustawilismy w stan 1, a które w stan 0.

Rejestr ODSR ma tak_e drugie zastosowanie: dane wpisane do rejestru PIO_ODSR pojawia sie na wyjsciach kontrolera PIO. Jednak, aby pełnił on te_ swoja druga funkcje, należy wcześniej dokonać odpowiedniego wpisu do rejestru **PIO_OWER** (*Output Write Enable Register*, czyli „rejestr zezwolenia na wpisywanie danych do rejestru wyjsciowego") . Innymi słowy: po uaktywnieniu wybranych wyprowadzeń wyjsciowych przez wpis do rejestru OWER bedziemy mogli zmieniac ich stan wpisami do rejestru ODSR. W praktyce wyglada to nastepujaco:

```
//inicjalizacja PB23 taka jak poprzednio
//właczenie sterowania PB23 przez PIO
AT91C_BASE_PIOB->PIO_PER = AT91C_PIO_PB23;

//konfiguracja PB23 jako wyjscie
AT91C_BASE_PIOB->PIO_OER = AT91C_PIO_PB23;

//zezwolenie na nowy sposób zmiany stanu PB23:
AT91C_BASE_PIOB->PIO_OWER = AT91C_PIO_PB23;

//teraz mozna modyfikowac stan PB23 za pomoca rejestru ODSR
//ustawienie PB23 w stan wysoki, dioda swieci
AT91C_BASE_PIOB->PIO_ODSR = AT91C_PIO_PB23;
```

Rozwiazanie polegajace na sterowaniu wyjsciem za pomoca instrukcji przypisania jest właściwie podstawowa metoda radzenia sobie z wyjsciami . Dzieki temu, w przypadku sterowania dioda LED mo_emy zaimplementowac polecenia „czytaj-modyfikuj-zapisz" znane tak_e pod nazwa RMW (Read-Modify-Write) , czyli np. zmiane stanu diody LED (toggle) na przeciwny bez u_ycia instrukcji warunkowych:

```
AT91C_BASE_PIOB->PIO_ODSR = AT91C_BASE_PIOB->PIO_ODSR^AT91C_PIO_PB23;
```

Jak widać, w jednej linijce kodu dokonaliśmy:
- odczytu stanu rejestru ODSR (czyli „w jakim stanie aktualnie są wyjścia", operacja read),
- modyfikacji zawartosci ODSR {modify), realizujac funkcje EXOR operatorem „^" ,
- zapisu zmodyfikowanej wartosci z powrotem do ODSR (write).

Skrócony (lecz równorzędny) zapis powyższego kodu wygląda jeszcze przyjaźniej :

```
AT91C_BASE_PIOB->PIO_ODSR ^= AT91C_PIO_PB23;
```

Za pomocą wpisów do rejestrów SODR i CODR, operacji zmiany stanu diody LED nie dało by sie tak prosto wykonac - musielibyśmy radzić sobie za pomocą instrukcji warunkowych, ponieważ rejestry SODR i CODR służą tylko do zapisu (write-only).

2.4 Praca jako wejście cyfrowe: odczyt stanu przycisków

Podłaczenie przycisków do mikrokontrolerów SAM7 można wykonać w sposób najbardziej typowy, czyli umieszczając przycisk tak, aby zwierał wyprowadzenie mikrokontrolera z masa układu, gdy zostanie wciśnięty. Wyprowadzenie, do którego dołączony jest przycisk, warto podciągnąć do napiecia 3,3 V rezystorem o wartości tym mniejszej , im większe sa zakłócenia w pobli_u mikrokontrolera. Zastosowanie zewnetrznych rezystorów podciagajacych jest dobrym pomysłem nawet pomimo tego, _e mikrokontrolery SAM7 maja wewnetrzne rezystory podciagające. Rozsadne wartości zewnetrznych rezystorów podciagajacych zawierają sie w przedziale od 1 k do 100 k. Powszechnie stosowana wartosc to 10 k. W takiej konfiguracji przycisk wcisniety spowoduje podanie na wejscie mikrokontrolera stanu logicznego „0", natomiast zwolnienie przycisku spowoduje powrót wspomnianego

wejscia mikrokontrolera do stanu logicznej 1. Oczywiscie w trakcie zmiany stanu wejscia mikrokontrolera (wciskanie i zwalnianie przycisku) wystapia pewne zakłócenia (tzw. drgania styków), których wpływ redukuje sie, stosujac np. niewielkie opóznienie.
Na płytce ewaluacyjnej podstawowy zestaw przycisków SW1 –SW2 jest dołaczony kolejno do wyprowadzen: PB24 i PB25, a styki joysticka do PA7, PA8, PA9, PA14 i PA23.

2.4.1 Sposób najprostszy

W celu odczytania wartości przycisku rejestry modułu PIO powinny zostać odpowiednio zainicjalizowane, a sygnał taktujący mikrokontroler powinien być włączony. W najprostszym przypadku inicjalizacja uniwersalnego wyprowadzenia mikrokontrolera do pracy jako wejście cyfrowe jest bardzo prosta, ponieważ najczęściej wcale nie musimy jej przeprowadzać. Jest tak dlatego, że wyprowadzenia kontrolerów PIO w mikrokontrolerach SAM7X po resecie są skonfigurowane jako wejścia z włączonym wewnętrznym rezystorem podciągającym.
Odczyt stanu przycisku często wykonuje sie w instrukcji warunkowej:

```
if ((AT91C_BASE_PIOA_PIO_PDSR & AT91C_PIO_PA7)==0)
{
        //tu umiescimy kod który sie wykona, gdy przycisk bedzie wcisniety
}
```

Do odczytu stanu przycisku wykorzystaliśmy w powyższym fragmencie rejestr **PIO_PDSR** (*Pin Data Status Register*). Wartość odczytana z tego rejestru reprezentuje stan wszystkich linii kontrolera PIO niezależnie od konfiguracji. Można zatem odczytać stan logiczny na wyprowadzeniu mikrokontrolera, nawet jeśli wyprowadzenie to nie jest kontrolowane przez PIO, tylko przez układ peryferyjny.

Wbudowane rezystory podciągające. Co prawda nie zastąpią one w pełni tych zewnętrznych, silnie podciągających wejścia do szyny zasilania, lecz w niektórych sytuacjach mogą być przydatne. Załączenie wewnętrznego rezystora podciągającego na danym wyprowadzeniu uzyskuje sie przez wpis wartosci 1 do bitu odpowiadajacemu temu wejsciu w rejestrze **PIO_PPUER** (*Pullup Enable Register*). Wpisu tego dokonujemy w dokładnie ten sam sposób jak czyniliśmy to np. podczas włączania portu PIO na danym wyprowadzeniu.
Identycznie wygląda sytuacja z wyłaczaniem podciagania: wystarczy wpis do rejestru **PIO_PPUDR** (*Pullup Disable Register*).

Filtracja zakłóceń przy odczycie wejść cyfrowych. Zestaw rejestrów kontrolujacy te funkcje to **PIO_IFER** (*Input Filter Enable Register*), **PIO_IFDR** (*Input Filter Disable Register*) oraz **PIO_IFSR** (*Input Filter Status Register*). Ta trójka rejestrów działa oczywiscie tak samo jak wiele innych zestawów enable-disable-status. Jeśli natomiast chodzi o samo działanie filtru zakłócen na wejsciach, to polega ono na wpuszczaniu do rejestru **PIO_PDSR** jedynie impulsów dłuższych niż jeden okres sygnału zegarowego taktującego mikrokontroler (czyli ok. 20ns, gdyż długość jednego okresu sygnału zegarowego wynosi w przybliżeniu T=1/f = 1 /50MHz = 20ns).
Jeśli impuls będzie krótszy, nie zmieni on ani na chwilę stanu rejestru **PIO_PDSR**, czyli rejestru, z którego odczytujemy stan linii kontrolera PIO. Dzięki temu typowe (tj. krótkotrwałe) zakłócenia na danej linii wejściowej zostaną wyfiltrowane.

Przykład 1.
Do 24 linii portu PB podpięty jest klawisz (SW1) w ten sposób, że jego wciśniecie powoduje ustawienie stanu niskiego na tej linii. Napisać program zmieniający każdorazowo stan sygnału na linii PB20 (sterującej podświetleniem LED wyświetlacza LCD) na przeciwny gdy nastąpi wciśnięcie klawisza.
Zmiana stanu linii PB20 powinna nastąpić JEDNORAZOWO w reakcji na POJEDYŃCZĄ sekwencję wciśnięcia i puszczenia klawisza.

Rozwiązanie:
Sekwencja instrukcji realizująca zadanie wyglądała będzie następująco:

```
while(1)
{
      //wykrycie wcisniecia klawisza (PB24)i zmiana stanu wyjścia PB20
      if (((AT91C_BASE_PIOB->PIO_PDSR) & AT91C_PIO_PB24)==0)
```

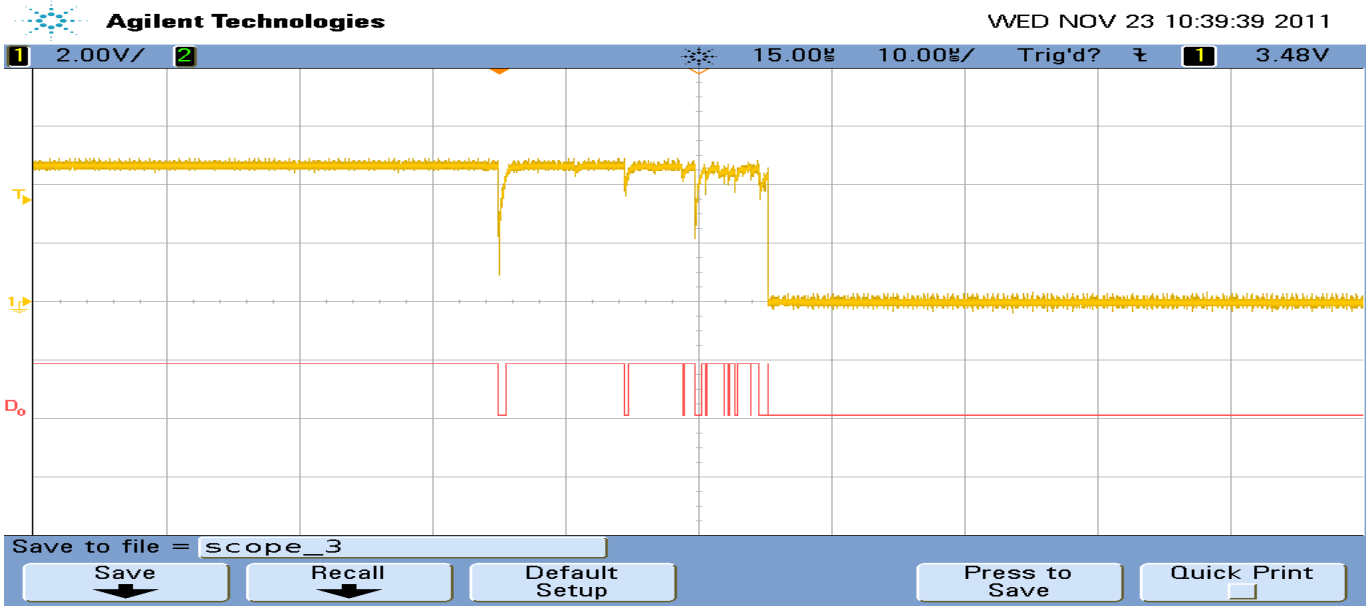
```
        {
                AT91C_BASE_PIOB->PIO_ODSR = AT91C_BASE_PIOB->PIO_ODSR^AT91C_PIO_PB20;

                //oczekiwanie na puszczenie klawisza
                while (((AT91C_BASE_PIOB->PIO_PDSR) & AT91C_PIO_PB24)==0)
                {};
        }
}
```

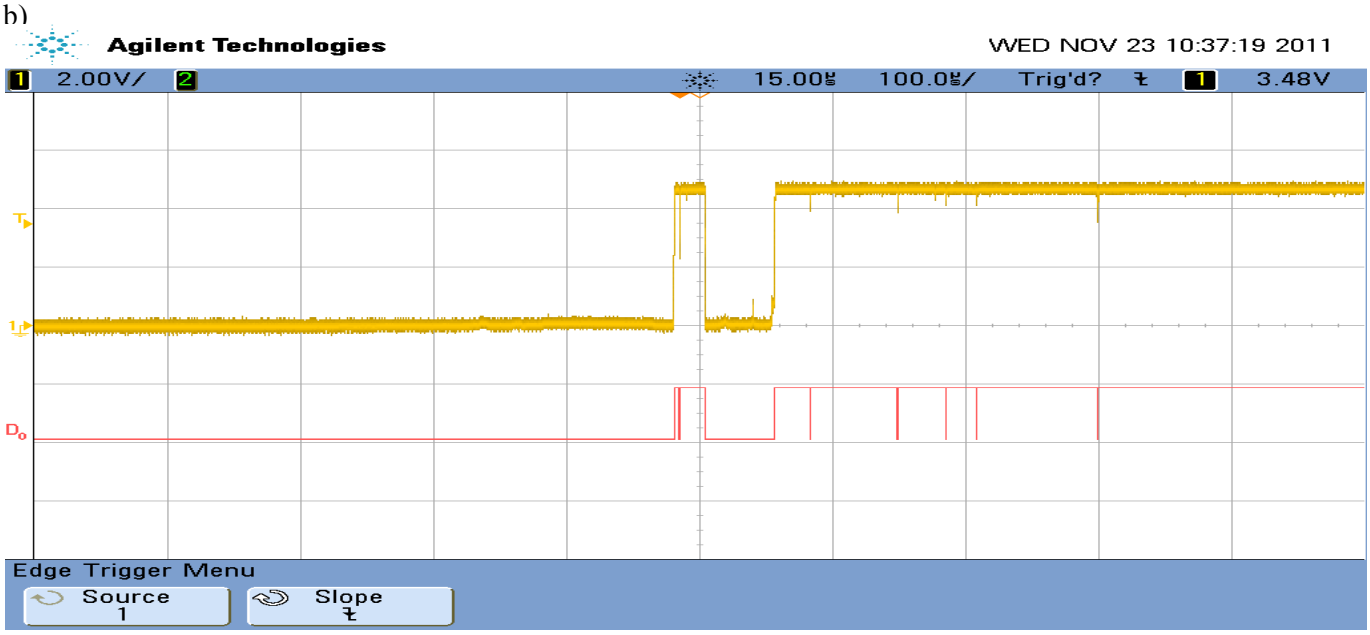
Powyższy program dwukrotnie sprawdza stan klawisza (stan linii portu PB24) aby zapewnić jednorazową reakcję na jednokrotne wciśnięcie i puszczenie klawisza.

Uwaga – często mimo zastosowania wyżej wymienionego zabezpieczenia dość często występuje zjawisko więcej niż jednokrotnej zmiany stanu linii PB20. W prawidłowo działającym programie podświetlenie LED powinno zmieniać się z załączonego na wyłączone i odwrotnie przy jednokrotnej sekwencji wciśnięcia i puszczenia. Zaobserwowane działanie jest to wynikiem tzw. iskrzenia/drgania styków klawiszy. Wynika to z faktu samoczynnego drgania styków mechanicznych (wykonanych najczęściej w formie sprężystej blaszki) jakie następuje przy wciskaniu lub puszczaniu klawisza. Rysunek nr 2 przedstawia zarejestrowane oscyloskopem przebiegi stanu sygnału linii PB24, na których widać w/w efekt.

a)



b)



Rys.2 Zjawisko drgania mechanicznych styków przycisków podłączonych do portu wejściowego mikrokontrolera: a) wciśnięcie przycisku, b) zwolnienie (puszczenie) przycisku. Na obu rysunkach górny przebieg przedstawia rzeczywista wartość napięcia wejściowego na porcie wejściowym, natomiast dolny przebieg przedstawia wartość cyfrową (dwustanową) odczytaną przez port wejściowy.

W celu wyeliminowania ww. efektu należy zastosować kolejne zabezpieczenie. Zarówno po wykryciu stanu niskiego (wciśnięcie klawisza) jak i po wykryciu stanu wysokiego (zwolnienie/puszczenie klawisza) należy odczekać jeszcze kilka-kilkanaście milisekund, nie reagując na pojawiające się zmiany stanu portu wejściowego.

Można to zrobić np. tak:

Przykład 2.

```
#define LEFT_KEY_DOWN (((AT91C_BASE_PIOB->PIO_PDSR) & AT91C_PIO_PB24)==0)
#define LED_BCK_TGL   (AT91C_BASE_PIOB->PIO_ODSR=AT91C_BASE_PIOB->PIO_ODSR ^ AT91C_PIO_PB20)

while(1)
{

      if (LEFT_KEY_DOWN)
      {
            LED_BCK_TGL;
            delay(10); //opóźnienie po wykryciu wciśnięcia
            //(zabezpieczenie na „zboczu opadający” sygnału – rys. 2a)

            //oczekiwanie na puszczenie klawisza
            while (LEFT_KEY_DOWN)
            {};

            delay(10);
            //opóźnienie po wykryciu puszczenia klawisza
            //(zabezpieczenie na „zboczu narastającym” sygnału – rys. 2b)

      };
};
```

3. Program ćwiczenia

1. Utworzyć program na podstawie przykładu 1. Skompilować, wgrać do mikrokontrolera (konfiguracja RAM-ULINK). Zaobserwować (potwierdzić) zjawisko iskrzenia styków. Sprawdzić jak często ono występuje.
2. Utworzyć program na podstawie przykładu 2. Skompilować, wgrać do mikrokontrolera (konfiguracja RAM-ULINK). Zaobserwować przy jakiej najniższej wartości opóźnienia wpływ zjawiska iskrzenia styków na poprawne funkcjonowanie programu jest wyeliminowany?
3. Dla programu z przykładu 1 włączyć sprzętową filtrację na linii PB24, sprawdzić czy wyeliminowała ona zjawisko drgania styków? Przeanalizować rys. 2 zwracając uwagę na podziałkę na osi x wynoszącą odpowiednio 10 i 100us/podziałkę poziomą dla rys. 2a i 2b. Opisać wyniki analizy w sprawozdaniu.

W rozwiązaniu należy podać:

1. **Treść zadań**
2. **Kod zadania 2 wraz z obszernymi komentarzami**
3. **Wnioski i odpowiedzi na postawione pytania**