Ćwiczenie 2
Podstawy GPIO i PWM

Bibliografia

- [1] Specyfikacja mikrokontrolera LM4F232H5QD - http://www.ti.com/lit/ds/symlink/tm4c123gh6pge.pdf
- [2] Instrukcja obsługi płyty EasyMxPROv7 https://www.scribd.com/document/311182202/Easymx-Pro-v7-Stellaris-Manual-v102
- [3] Materiały z warsztatów TI http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-TM4C123G-LaunchPad/TM4C123G_LaunchPad_Workshop_Workbook.pdf
- [4] Instrukcja obsługi ROM LM4F232H5QD https://www.ti.com/lit/pdf/spmu270
- [5] Instrukcja obsługi biblioteki TivaWare Peripheral Driver Library https://www.ti.com/lit/pdf/spmu298

GPIO (*General Purpose Input Output*)

Pod pojęciem GPIO kryją się linie wejść/wyjść ogólnego przeznaczenia. Są to wyprowadzenia z mikroprocesora, które domyślnie nie mają żadnych funkcji. Użytkownik może nadać im odpowiednie znaczenie podczas konfiguracji i w etapie programowania mikroprocesora.
Moduł GPIO mikroprocesora składa się z 14 bloków, każdy odpowiada jednemu portowi GPIO (Porty A-P). Moduł pozwala na skonfigurowanie do 105 programowalnych linii wejść/wyjść. Moduł GPIO mikroprocesora TM4C123GH6PGE ma wiele użytecznych funkcji, np.:

- Każdy moduł GPIO może być źródłem przerwania,
- GPIO może być użyte do inicjalizacji sekwencji próbkowania konwertera analogowo-cyfrowego lub transferu µDMA,
- Każdy moduł GPIO może pracować z szybkością przełączania odpowiadającą szybkości zegara (przy wykorzystaniu szyny Advanced High-Performance Bus) bądź połowie szybkości zegara (przy wykorzystaniu standardowej szyny)
- Stan nóżek mikroprocesora może być zachowany w trybie hibernacji.

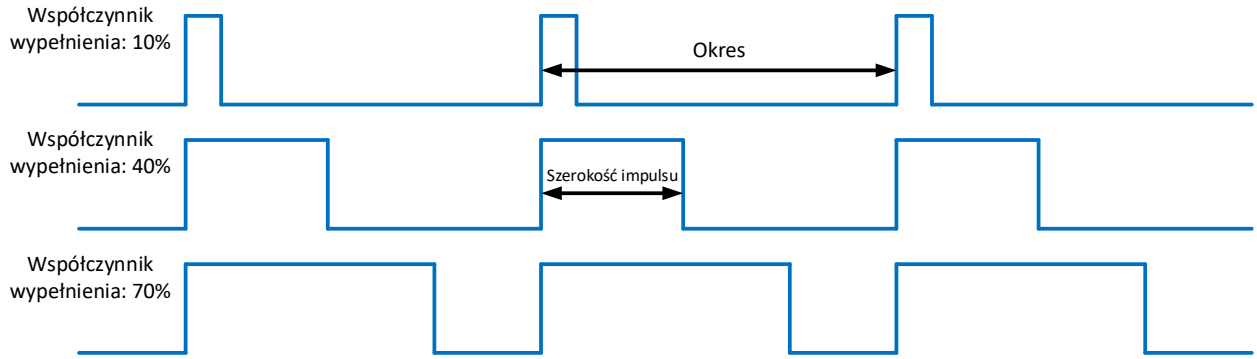
Na płycie EasyMxPro wykorzystane jest 9 portów GPIO (Porty A-J), przy czym kilku nóżkom zostały przypisane specjalne funkcje:

- PORTB[6,7] – nieobecne w mikroprocesorze
  - Linia PB7 na płycie podłączona jest do linii PK7 mikroprocesora
  - Linia PB6 na płycie podłączona jest do linii PK6 mikroprocesora
- PORTC[0:3] – linie wykorzystane do podłączenia złącza JTAG
- PORTD[0:3]
  - Linia PD0 na płycie podłączona jest do linii PN0 mikroprocesora
  - Linia PD1 na płycie podłączona jest do linii PN1 mikroprocesora
  - Linia PD2 na płycie podłączona jest do linii PK4 mikroprocesora

- Linia PD3 na płycie podłączona jest do linii PK5 mikroprocesora
- Linie PD7 i PF0 są wykorzystane do NMI (*Non-maskable interrupt*).

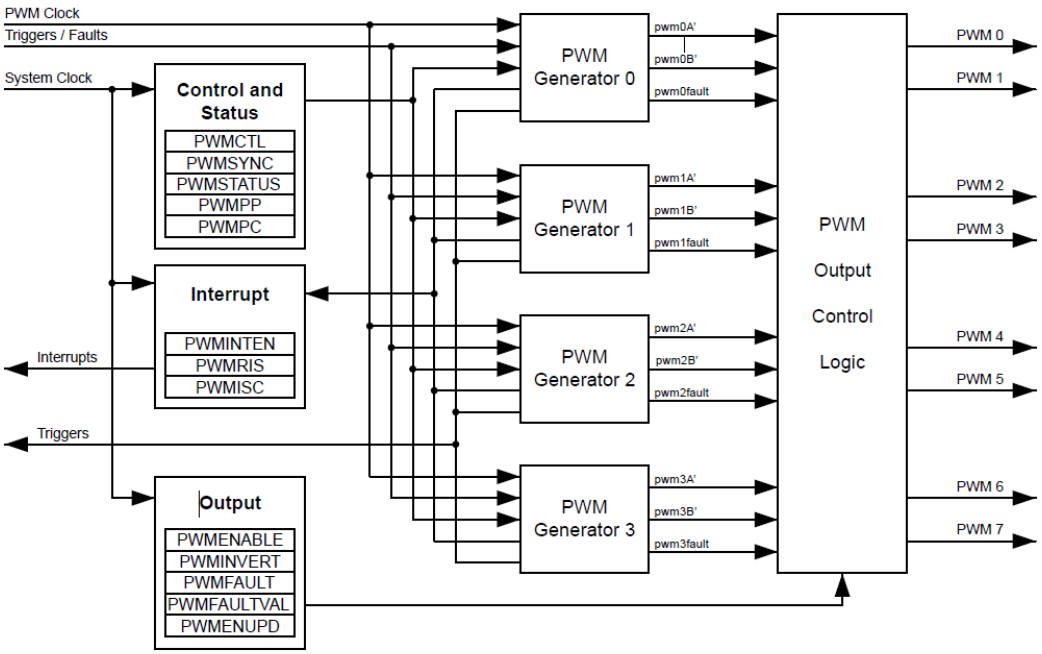
PWM

Modulacja PWM (*Pulse Width Modulation*) jest metodą pozwalającą na cyfrowe zakodowanie poziomów sygnału analogowego. Wykorzystywane są do tego liczniki cyfrowe wysokiej rozdzielczości, które generują sygnał prostokątny o zadanej częstotliwości. Wypełnienie (szerokość) sygnału prostokątnego jest określone poziomem sygnału analogowego. Typowe zastosowania modulacji PWM to sterowanie pracy silników elektrycznych, sterowanie światłem, czy też przełączanie źródeł zasilania. Przykładowy przebieg sygnału PWM dla różnych współczynników wypełnienia przedstawia rys.1.



Rys. 1. Sygnał PWM (Współczynnik wypełnienia = szerokość impulsu / okres)

Mikroprocesor TM4C123GH6PGE wyposażony jest w dwa moduły PWM. Każdy z modułów składa się z czterech bloków generatora oraz bloku sterowania określającego m.in. polaryzację sygnału oraz to które z sygnałów mają być przekazane na wyjścia. Każdy generator może wytworzyć dwa niezależne sygnały wyjściowe o tej samej częstotliwości. W rezultacie każdy z modułów PWM może wytworzyć 8 sygnałów. Schemat blokowy modułu PWM mikroprocesora przedstawiony jest na rys. 2.



Rys. 2. Schemat blokowy modułu PWM.

Ustawienia zegara mikroprocesora

Częstotliwość taktowania zegara mikroprocesora może być ustawiana manualnie, przy czym podstawowymi źródłami zegara są:

- Precyzyjny Oscylator Wewnętrzny (PIOSC) o częstotliwości 16 MHZ (± 3%)
- Główny oscylator (MOSC) np. zewnętrzny rezonator kwarcowy.
- Wewnętrzny oscylator o częstotliwości 30 kHz przeznaczony do pracy w trybie oszczędzania energii
- Zegar modułu hibernacji sterowany kryształem o częstotliwości 32,768 kHz przeznaczony do zapewnienia źródła czasu rzeczywistego

Schemat połączeń zegara systemowego przedstawiony jest na rys. 3.

Do ustawienia wartości zegara mikroprocesora służy funkcja `SysCtlClockSet(uint32_t ui32Config)` gdzie argument funkcji zawiera pożądaną konfigurację. Parametr ui32Config jest suma logiczną kilku różnych wartości:

- Dzielnik zegara systemowego (ang. *System clock divider*). Wybierany za pomocą jednej z wartości:
  SYSCTL_SYSDIV_1, **SYSCTL_SYSDIV_2**, **SYSCTL_SYSDIV_3**, … **SYSCTL_SYSDIV_64**.
- Wykorzystanie PLL (*Phase Lock Loop*) jest określone poprzez **SYSCTL_USE_PLL** lub **SYSCTL_USE_OSC**
- Częstotliwość zewnętrznego oscylatora krystalicznego wybierana jest poprzez jedną z wartości podanych w bibliotece TivaWare. W przypadku procesora wykorzystywanego na zajęciach zewnętrzne źródło taktowania ma częstotliwość równą 8 MHz w związku z czym prawidłowa wartość to **SYSCTL_XTAL_8MHZ**.
- Źródło oscylatora jest wybierane za pomocą jednej z następujących wartości: **SYSCTL_OSC_MAIN**, **SYSCTL_OSC_INT**, **SYSCTL_OSC_INT4**, **SYSCTL_OSC_INT30**, lub **SYSCTL_OSC_EXT32**. Przy czym **SYSCTL_OSC_EXT32** jest dostępne tylko w urządzeniach z modułem hibernacji i tylko wtedy gdy moduł ten został aktywowany.
- Wewnętrzny oscylator można wyłączyć za pomocą flagi **SYSCTL_INT_OSC_DIS**. Natomiast oscylator główny za pomocą flagi **SYSCTL_MAIN_OSC_DIS**.

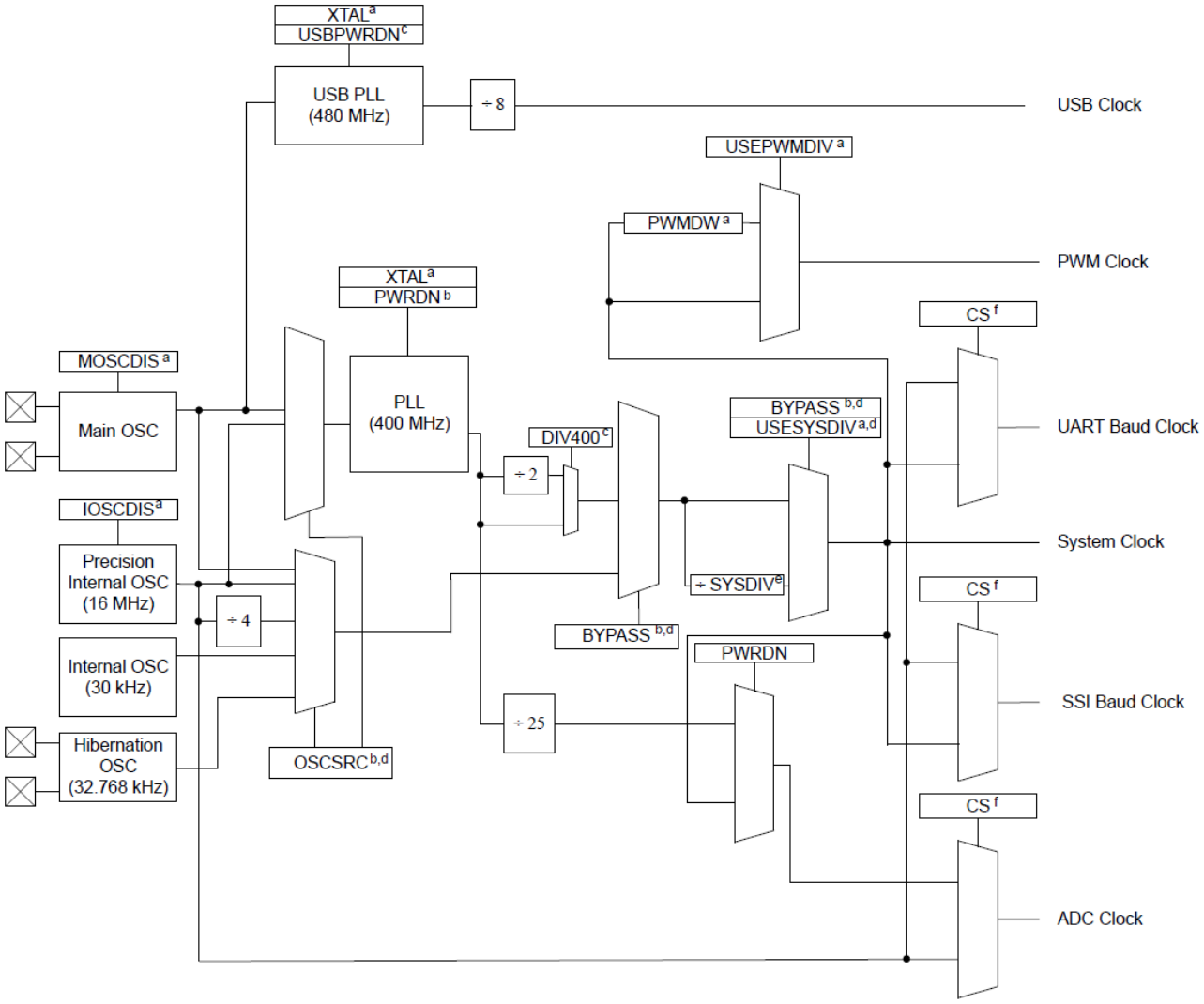
Dla przykładu aby korzystać z głównego zegara należy użyć flag **SYSCTL_USE_OSC** | **SYSCTL_OSC_MAIN**. Aby skorzystać z PLL należy użyć flag **SYSCTL_USE_PLL** | **SYSCTL_OSC_MAIN** oraz wybrać odpowiednią wartość oscylatora krystalicznego poprzez **SYSCTL_XTAL_8MHZ.**

W rozdziale 5.2.5 dokumentu [1] znajdują się szczegółowe informacje dotyczące ustawień zegara. Istotne są w szczególności tabelki 5-5 oraz 5-6, w których pokazane jest jak wybrać ustawienia dzielnika zegara systemowego aby uzyskać oczekiwaną wartość zegara. Na przykład, aby ustawić zegar na maksymalną wartość 80 MHz należy jako argument funkcji `SysCtlClockSet` podać wartość:

```
SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ |SYSCTL_OSC_MAIN
```

W tabeli poniżej znajdują się inne przykładowe argumenty funkcji `SysCtlClockSet`.

Wartość zegara	Argument funkcji **SysCtlClockSet**
4 MHz	SYSCTL_SYSDIV_2 | SYSCTL_USE_OSC | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN
8 MHz	SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN
8 MHz	SYSCTL_SYSDIV_25 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN
12,5 MHz	SYSCTL_SYSDIV_16 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN
20 MHz	SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN
50 MHz	SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ | SYSCTL_OSC_MAIN



Rys. 3. Schemat połączeń zegara systemowego.

Przebieg ćwiczenia

GPIO

Pierwszy program (przypomnienie)

1. Otworzyć projekt z laboratorium nr 1.
2. Ponownie przeanalizować kod i zapoznać się z funkcjami wykorzystywanymi w programie.

Pliki nagłówkowe

`stdint.h`: Definicje zmiennych zgodnych ze standardem C99

`stdbool.h`: Definicje zmiennych boolowskich zgodnych ze standardem C99

`hw_memmap.h` : Zawiera makra określające mapę pamięci urządzeń Tiva. Znajdują się tutaj definicje adresów bazowych dla różnych urządzeń mikroprocesora (np. `GPIO_PORTA_BASE`).

`hw_types.h` : Zawiera definicje często wykorzystywanych typów zmiennych oraz makr.

`sysctl.h` : Zawiera definicje oraz makra dla System Control API pochodzącej z biblioteki driverlib. Są to m.in. funkcje `SysCtlClockSet` i `SysCtlClockGet`

`gpio.h` : Zawiera definicje oraz makra dla GPIO API pochodzącej z biblioteki driverlib. Są to m.in. funkcje `GPIOPinTypeGPIOOutput` i `GPIOPinWrite`

`rom.h`: Zawiera definicje i makra procedur dostępnych z pamięci ROM mikroprocesora (np. `ROM_SysCtlClockSet`)

Ustawienie zegara

Korzystając z informacji z wprowadzenia określ ustawienia zegara:

- **Z jaką częstotliwością pracuje mikroprocesor?**
- **Co jest źródłem zegara?**

Konfiguracja GPIO

Przed skorzystaniem z jakiegokolwiek urządzenia mikroprocesora należy zawsze uruchomić dla niego zegar poprzez metodę `SysCtlPeripheralEnable`. Jeśli operacja ta nie zostanie wykonana to pojawi się błąd Fault ISR i program zakończy działanie.

By móc korzystać z linii portu GPIO należy też określić jaki jest ich typ. W przypadku tego programu wszystkie linie portu A są ustawione na wyjście (`GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PINS_ALL)`), co pozwala na zapalanie odpowiednich diod podłączonych do tego portu. W wywołaniu funkcji `GPIOPinTypeGPIOOutput` pierwszy argument jest adresem portu, który chcemy skonfigurować. Adres ten jest zdefiniowany w pliku nagłówkowym `hw_memmap.h`. Każdy z portów GPIO ma dwa rodzaje adresów bazowych są to adres typu APB (*Advanced Peripheral Bus*) oraz AHB (*Advanced High-Performacne Bus*). Dostęp poprzez AHB jest szybszy niż dostęp przez APB ale jest jednocześnie bardziej wymagający energetycznie.

Aby wpisać wartość do portu A skorzystaliśmy z funkcji `GPIOPinWrite`. Zapoznaj się z jej opisem w dokumentacji biblioteki *Peripheral driver* [5]. Zwróć uwagę na użyte w wywołaniu argumenty.

- Który z argumentów określa, który pin jest adresowany?

- W jaki sposób dokonuje się wyboru portu, na którym będzie wykonana operacja zapisu?

3. Zweryfikować działanie programu.

Drugi program

1. Utworzyć nowy projekt poprzez kopię szablonu (wg opisu z lab. 1).
2. W pliku main.c wpisać kod:

```
1.  /*
2.  main.c
3.  */
4.  #include <stdint.h>
5.  #include <stdbool.h>
6.  #include "inc/hw_gpio.h"
7.  #include "inc/hw_memmap.h"
8.  #include "driverlib/sysctl.h"
9.  #include "driverlib/gpio.h"
10. #include "driverlib/rom.h"

11. #define GPIO_PINS_ALL
12. GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7

13. int main(void)
14. {
15. uint8_t a = 0;
16. //
17. // Set the clocking to run directly from the crystal.
18. //
19. ROM_SysCtlClockSet (SYSCTL_SYSDIV_20 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
20. //
21. // Enable GPIOA, GPIOJ, GPIOC
22. //
23. SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOA);
24. SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOJ);
25. SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOC);
26. //
27. // Set all GPIOA and GPIOC pins as outputs
28. //
29. GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PINS_ALL);
30. GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PINS_ALL);
31. //
32. // Set all GPIOJ pins as inputs
33. //
34. GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_6|GPIO_PIN_7);
35. for (;;)
36. {
37. //
38. // Check button press and write data to port
39. //
40. while(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_7) == GPIO_PIN_7)
41. GPIOPinWrite (GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80);
42. while(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_6) == GPIO_PIN_6)
43. GPIOPinWrite (GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40);
44. //
45. // Show some patterns on LEDs
46. //
47. GPIOPinWrite (GPIO_PORTA_BASE, 0xFF, a++);
48. //
49. // Delay for a while so changes can be visible
50. //
51. SysCtlDelay(SysCtlClockGet() / 2);
52. }
53. return 0;
54. }
```

Program ten jest rozszerzeniem programu pierwszego. Rozszerzenie polega na dodaniu reakcji na klawisze (linie 39-42). Aby reakcja na klawisze portu J była możliwa należy ustawić typ nóżek portu J na wejście (linia 29). W wyniku naciskania przycisków portu J (PJ7 i PJ6) program zapala odpowiednie diody podłączone do portu C.

3. Sprawdzić czy przełączniki diod LED dla portów A i C są ustawione na pozycję włączoną. Ustawić przełącznik poziomu przycisków dla portu J (SW16.1) na pozycję włączoną.
4. Uruchomić program przyciskiem Debug.
5. Zweryfikować działanie programu.
6. Zapoznać się z dokumentacją metody `GPIOPinRead` w dokumentacji [5]

   - **Ile argumentów przyjmuję ta metoda?**
   - W jaki sposób wybiera się nóżki, z których zostaną odczytane dane?

Zadanie

1. Napisać program, który w sposób binarny będzie wyświetlał na diodach portów A i E liczbę naciśnięć jednego z przycisków portów H i J oraz na diodach portu G różnicę liczby naciśnięć.
2. Rozszerzyć zadanie 1 w taki sposób, aby procesor w sposób ciągły wyliczał wartości NWD i NWW liczby naciśnięć.

PWM

Trzeci program

1. Utworzyć nowy projekt poprzez kopię szablonu (patrz lab. 1).
2. W pliku main.c wpisać kod:

```
1.  #include <stdint.h>
2.  #include <stdbool.h>
3.  #include "inc/hw_memmap.h"
4.  #include "driverlib/pin_map.h"
5.  #include "inc/hw_types.h"
6.  #include "driverlib/sysctl.h"
7.  #include "driverlib/gpio.h"
8.  #include "driverlib/debug.h"
9.  #include "driverlib/pwm.h"
10. #include "inc/hw_gpio.h"
11. #include "driverlib/rom.h"

12. void playSound(uint32_t frequency, uint32_t duration)
13. {
14.         uint32_t PWMclock = ROM_SysCtlClockGet() / 4 ; // get the current PWM clock value
15.         uint32_t period = (PWMclock/frequency) - 1; // calculate the period for PWM signal
16.         float numOfCyc = ROM_SysCtlClockGet()/1000.0; // calculate number of cycles per ms
17.         uint32_t counter = duration * numOfCyc / 3; //calculate the counter for SysCtlDelay function;
18.         ROM_PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, period); // set the period of the PWM signal
19.         // Set the pulse width of PWM1 for a 50% duty cycle:
20.         ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2,  0.5 * period);
21.         ROM_PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true); // enable PWM1 output
22.         ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_1); // enable the PWM signal generation
23.         ROM_SysCtlDelay(counter); // add delay
24.         ROM_PWMGenDisable(PWM1_BASE, PWM_GEN_1); // disable the PWM signal generation
25. }
26. void noteC(int duration){playSound(523, duration);}
27. void noteD(int duration){playSound(587, duration);}
28. void noteE(int duration){playSound(659, duration);}
```

```
29.  void noteF(int duration){playSound(698, duration);}
30.  void noteG(int duration){playSound(784, duration);}
31.  void noteA(int duration){playSound(880, duration);}
32.  void noteB(int duration){playSound(988, duration);}
33.  void Melody()
34.  {
35.          noteC(300);noteD(300);noteE(300);
36.          noteF(300);noteG(300);noteA(300);
37           noteB(600);
38.  }
39.  int main(void)
40.  {
41.          // Set the clocking to run directly from the crystal.
42.          ROM_SysCtlClockSet (SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

43.          // Set the PWM clock configuration.
44.          ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_4);
45.          ROM_SysCtlDelay(ROM_SysCtlClockGet() / 2);

46.          // Enable GPIOA and PWM module 1
47.          ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOA);
48.          ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);

49.          // Configure the PWM generator for count down mode with immediate updates
50.          // to the parameters.
51.          ROM_PWMGenConfigure(PWM1_BASE, PWM_GEN_1,PWM_GEN_MODE_DOWN);

52.          //Configure the pin mux to select module 1 PWM generator 1 for GPIO pin PA6
53.          ROM_GPIOPinConfigure(GPIO_PA6_M1PWM2);

54.          // set the type of PA6 pin to PWM
55.          ROM_GPIOPinTypePWM(GPIO_PORTA_BASE,GPIO_PIN_6);

56.          for (;;)
57.          {
58.                  // play the melody
59.                  Melody();
60.                  ROM_SysCtlDelay(ROM_SysCtlClockGet() / 2); // add delay
61.          }
62.          return 0;
63.  }
```

Powyższy program pozwala na wytworzenie dźwięku z wykorzystaniem brzęczyka płyty EasyMxPRO. Dźwięk powstaję w skutek podania sygnału PWM pochodzącego z wyjścia nr 2 generatora 1 modułu PWM1 na nóżkę PA6 portu GPIO A.

Konfiguracja i działanie PWM:

Aby skonfigurować generator PWM, należy wykonać następujące kroki:

1. **Ustawienie zegara PWM**
   Zegar PWM ustawia się względem zegara systemowego poprzez metodę `ROM_SysCtlPWMClockSet`, która ustawia dzielnik modułu PWM. Funkcje z przedrostkiem ROM pozwalają na zmniejszenie rozmiaru kodu, dzięki wykorzystaniu metod zapisanych w pamięci ROM mikroprocesora. Dzielnik może przyjmować wartości z zakresu 2-64, przy czym są to kolejne potęgi 2. W powyższym przykładzie zegar systemowy ustawiony jest na 40MHz (200 MHz / 5) natomiast dzielnik PWM jest ustawiony na 4, co daje częstotliwość zegara PWM wynoszącą 10 MHz.
2. **Włączenie PWM**
   Aby możliwe było korzystanie z modułu PWM, należy go włączyć poprzez polecenie `ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1)`. W przykładzie korzystamy z modułu

PWM 1. Jak to zostało wspomniane we wstępie, mikroprocesor TM4C123GH6PGE posiada dwa moduły PWM.

3. **Konfiguracja generatora**

   Kolejnym krokiem jest konfiguracja generatora PWM. Odpowiedzialna jest za to funkcja `ROM_PWMGenConfigure(PWM1_BASE, PWM_GEN_1,PWM_GEN_MODE_DOWN)`. W tym przypadku ustawiamy pierwszy z czterech generatorów modułu PWM1 aby działał w trybie zliczania w dół. Dostępne tryby oraz inne parametry konfiguracji można odnaleźć w dokumentacji [5].

4. **Konfiguracja linii**

   Brzęczyk piezoelektryczny płyty EasyMxPro podłączony jest do linii PA6 mikroprocesora. W nocie technicznej mikroprocesora możemy odnaleźć informację, że do linii PA6 podłączone jest wyjście nr 2 pierwszego generatora modułu PWM1(rys.4).

M1PWM2	45 51 139	PA6 (5) PG4 (5) PE4 (5)	O	TTL	Motion Control Module 1 PWM 2. This signal is controlled by Module 1 PWM Generator 1.

Rys.4. Podłączenie modułu PWM1

Należy więc odpowiednio skonfigurować tą linię następującymi metodami:

```
ROM_GPIOPinConfigure(GPIO_PA6_M1PWM2);
ROM_GPIOPinTypePWM(GPIO_PORTA_BASE,GPIO_PIN_6);
```

Metoda `ROM_GPIOPinConfigure` łaczy linie PA6 z wyjściem nr 2 generatora 1 modułu PWM1. Metoda `ROM_GPIOPinTypePWM` ustawia typ linii PA6 na PWM.

5. **Wyznaczenie okresu i współczynnika wypełnienia**

   Sygnał PWM określony jest przez swój okres oraz współczynnik wypełnienia impulsu. Okres wyznacza się na podstawie częstotliwości zegara modułu PWM. Zegar, jak wiadomo z informacji powyżej, ustawiany jest względem zegara systemowego poprzez dzielnik. W związku z tym, aby wyznaczyć okres sygnału PWM, dzielimy częstotliwość zegara modułu PWM przez żądaną częstotliwość i odejmujemy od tego 1 (licznik liczy do zera). W przykładzie powyżej za wyznaczenie i ustawienie okresu odpowiedzialne są następujące linie:

```
uint32_t PWMclock = ROM_SysCtlClockGet() / 4 ;
uint32_t period = (PWMclock/frequency) - 1;
ROM_PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, period);
```

   Drugim parametrem określającym sygnał PWM jest współczynnik wypełnienia. Ustawienie współczynnika wypełnienia odbywa się poprzez metodę `ROM_PWMPulseWidthSet`. W naszym przykładzie współczynnik wypełnienia wynosi 50% (0.5 * period):

```
ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2,  0.5 * period);
```

6. **Włączenie wyjść PWM**

   Po określeniu parametrów sygnału PWM kolejnym krokiem jest włączenie wyjść generatora poprzez metodę `PWMOutputState`. W przykładzie korzystamy z wyjścia nr 2 generatora 1 modułu PWM1:

```
ROM_PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true);
```

7. **Włączenie generatora PWM**

   Uruchomienie generator odbywa się poprzez wywołanie metody:

```
ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_1);
```

8. **Wyłączenie Generatora PWM**

Wyłączenie genertora odbywa się poprzez wywołanie metody:

```
ROM_PWMGenDisable(PWM1_BASE, PWM_GEN_1);
```

3. Ustawić przełącznik SW12.1 na pozycję włączoną (włączenie brzęczyka)
4. Uruchomić program przyciskiem Debug i zweryfikować jego działanie.
5. Zmień głośność poprzez zmianę współczynnika wypełnienia PWM.
6. Utwórz nową melodię.

Zadanie

1. Napisać program, który pozwoli na wytwarzanie różnych dźwięków za pomocą przycisków portów H i J.
2. Zmodyfikować powyższy program poprzez dodanie możliwości sterowania głośnością oraz częstotliwością generowanego dźwięku za pomocą wybranych dwóch przycisków.