Konkurs dla Czytelników Elektroniki Praktycznej

W artykule zostało pokazane nietypowe zachowanie programu i jest postawione pytanie o jego przyczynę: „Dlaczego debugger nie zatrzymał wykonania programu na pułapce w linii 98?”.Autorzy trzech pierwszych poprawnych odpowiedzi, przesłanych w terminie do dnia 30 maja 2014 na adres konkursy@ep.com.pl, zostaną uhonorowani kompletem książek:

- Henryk A. Kowalski, Procesory DSP dla praktyków, BTC, Warszawa, 2011
- Henryk A. Kowalski, Procesory DSP w przykładach, BTC, Warszawa, 2012

C2000 Piccolo LanuchPad (13)

Łatwe programowanie systemów czasu rzeczywistego

Układ procesorowy serii Piccolo F2802x dobrze nadaje się do budowania sterowników cyfrowych do układów zasilających. Płytka rozszerzeniowa C2000 LED BoosterPack firmy Texas Instruments, dołączana do zestawu C2000 LaunchPad, jest dobrą platformą dla wszystkich, którzy chcą poznać techniki programowania układów sterowania w czasie rzeczywistym. Załączony do zestawu projekt demonstracyjny LED_Boost_PC umożliwia precyzyjne i efektywne sterowanie systemem z czasem dyskretnym.

Nowoczesne cyfrowe sterowniki dla zasilaczy stosują technikę nazywaną multipleksowaniem czasowym –TDM (*Time Domain Multilexing*). Technika TDM pozwala na dzielenie pojedynczego procesora pomiędzy wiele zadań [14]. Tworzenie dla zasilaczy ze sterowaniem cyfrowym oprogramowania wykorzystującego zalety techniki TDM jest dużym wyzwaniem. Jednak przy dobrym zrozumieniu kluczowych zasad i przy postępowaniu zgodnie z odpowiednim wzorcem można tą pracę znacząco uprościć. Dobrym wzorcem tworzenia struktury programu dla taniego sterownika cyfrowego oferującego wysoką wydajność pracy jest aplikacja *LED_Boost_PC* omówiona w artykule.

No dobrze, przyznaję się, że z tym tytułem artykułu to lekka przesada. Wszyscy praktycy w dziedzinie programowania systemów czasu rzeczywistego, którym pokazywałem ten tytuł, zaczynali się śmiać. „Łatwo nie jest nigdy” – mówili. Może to i prawda, ale przy zastosowaniu prezentowanego w artykule sposobu organizacji systemu, programowanie może stać się łatwiejsze. Oraz łatwiejsze będzie późniejsze uruchamianie i strojenie tego systemu.

Systemy z czasem dyskretnym

Podstawowym parametrem sterowania aplikacją TDM jest szybkość próbkowania danych, określona przez odstęp czasu pomiędzy kolejnymi próbkami danych T_{SAMPE}. Gdy jest dostępna bieżąca próbka danych, to procesor ma na wykonanie obliczeń tylko przedział czasu do następnej próbki. Gdy nie zdąży, to występuje opóźnienie fazowe lub błąd pętli sterowania. Czas wykonania kodu sterowania jest krótszy niż odstęp próbkowania. Typowo pozostały czas jest zbyt krótki do wykonania kolejnej pętli sterowania, ale można go wykorzystać do wykonania wolnych działań kodu podstawowego BG (Background).

Wybór języka programowania

Wybór języka programowania to trudne pytanie, na które nie ma dobrej lub złej odpowiedzi. Wybór zależy od konkretnego zastosowania. Dla struktury systemu z jedną procedurą obsługi przerwania ISR (*Interrupt Service Routines*) i jednym kodem BG wybór jest stosunkowo prosty.

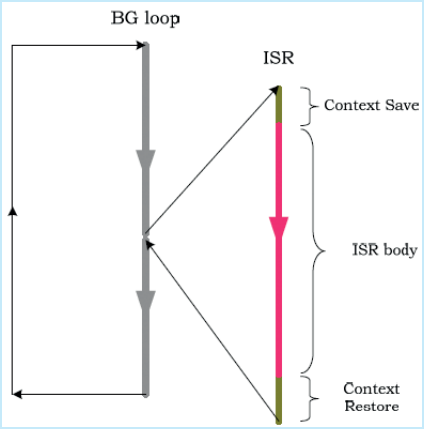
Pętla BG zawiera wolne i inteligentne funkcjonalności systemu. Typowo stanowią one 90% całego kodu. Najbardziej sensowne jest zastosowanie dla pętli BG języka wysokiego poziomu, jak C/C++.

Procedura ISR może być napisana w języku C lub w asemblerze. Wybór zależy od tego jak duża wydajność wykonania jest konieczna. Typowo kompilator języka C/C++ nie korzysta w efektywny sposób z zasobów sprzętowych procesora, jak rejestr przesuwający, tryby mnożenia, tryby adresowe itd.

Język asemblera jest dopasowany do architektury procesora i daje kompletną kontrolę wszystkich jego zasobów. Dlatego pozwala osiągnąć najlepszą wydajność dla danego procesora.

Szkielet programowy

Szkielet programowy (software framework) jest pojęciem określającym „infrastrukturę” zwierającą kod aplikacji.



Rysunek 1. Szkielet programowy z pojedynczą procedurą ISR i kodem BG [14]

Określa on przepływ sterowania oraz sposób szeregowania zadań. Kluczowe zagadnienia do rozpatrzenia to:

- Ile można zastosować procedur obsługi przerwania ISR?
- Czy procedury ISR są synchroniczne czy asynchroniczne ?
- Jaki procent czasu pozostaje dla kodu BG ?
- Czy zastosować do kodowania język wysokiego poziomu (C/C++) czy asembler, a może oba?
- Czy jest potrzebny system operacyjny czasu rzeczywistego?
- Czy komunikacja jest sterowana przerwaniami?

Pewne wybory bezpośrednio wpływają na efektywność kodu, stopień wykorzystania czasu procesora, złożoność aplikacji, łatwość programowania i debugowania. Najlepszy wynik można osiągnąć stosując jak najprostszy szkielet programowy: pojedyncza procedura ISR oraz kod BG z realizacją TDM z zastosowaniem techniki periodycznego podziału czasowego (*time slicing*).

W tym schemacie procedura ISR ma najwyższy priorytet i przerywa działanie kodu BG synchronicznie z działaniem modułu PWM (**rysunek 1**).

Procedura ISR

Czas pracy procedury ISR zawiera fragmenty ze stałym narzutem czasowym:

- IL (*interrupt latency*) – opóźnienie rozpoczęcia obsługi przerwania.

Tabela 1. Czas wykonania (Clock Cycles) operacji sterowania [14]

Operacja	Jedna pętla	Dwie pętle	Trzy pętle	LED_Boost_PC
Context Save + Int. latency	16	16	16	9(+8)
ADC servicing + Ack	4	5	6	6(x7)+4
2P/2Z controller	25	46	69	47(x3)
DPWM access	4	8	12	16(+11)
Context Restore + Int. Return	16	16	16	9(+8)
Razem	65	92	119	248

- CS (*context save*) – zapisanie kontekstu.
- IA (*interrupt acknowledge*) – potwierdzenie obsługi przerwania.
- CR (*context restore*) – odtworzenie kontekstu.
- IR (*interrupt return*) – powrót z przerwania.

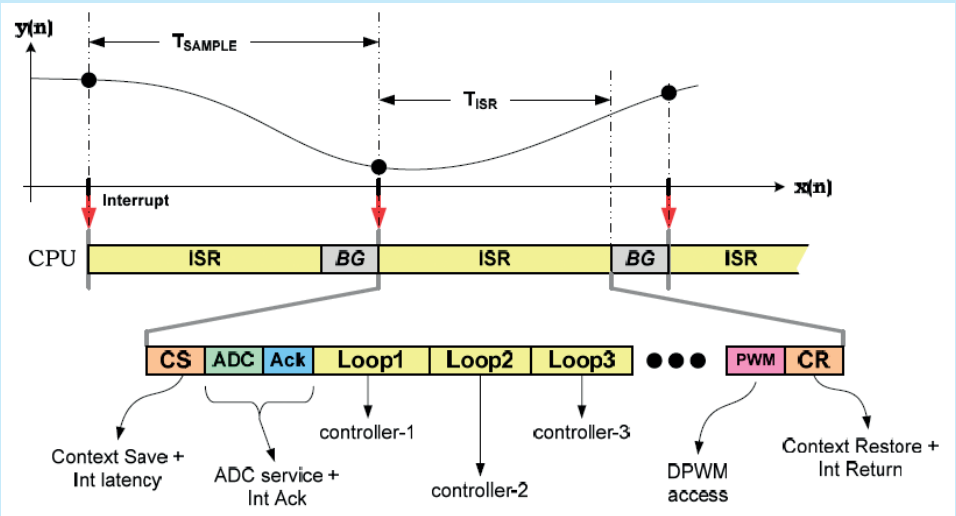
Wszystkie te operacje są wykonywane jeden raz podczas obsługi przerwania i mają stały (niezmienny) czas wykonania (**rysunek 2**).

Określenie czasu T_{ISR} potrzebnego do wykonania procedury ISR jest jednym z najważniejszych zagadnień podczas projektowania systemu zasilania z cyfrowym sterowaniem. Wyliczanie tego czasu z wystarczającą dokładnością określa czy pętla sterowania może być zrealizowana, jak dużo czasu pozostanie dla kodu BG oraz jak dużo czasu wolnego pozostanie na dalszą rozbudowę systemu.

Detale konieczne do oszacowania systemu dobrze pokazuje przykład z zamkniętą pętlą sterowania. Budowa systemu z napięciową lub prądową pętlą sterowania jest praktycznie taka sama. Elementy systemu ze sterowaniem przetwornicy DC/DC w topologii „back" zostały pokazane w artykule [12].

W pętli sterowania został zastosowany programowy filtr IIR (2 bieguny/2 zera) zrealizowany jako asemblerowa funkcja ze standardowej biblioteki TI. Podsumowanie zależności czasowych procedury ISR jest pokazane na rys. 2.

W **tabeli 1** pokazano typowe liczby cykli zegara systemowego procesora potrzebne do wykonania procedury ISR. Czas wykonania obliczeń procedur z biblioteki jest bardzo dokładnie określony i znany. Również znane są czasy obsługi modułu ADC i modułu PWM. W tab. 1 podane są czasy dla obsługi jednej pętli jak również dla dwóch i trzech pętli. Narzut czasowy obsługi przerwania



Rysunek 2. Zestawienie czasu próbkowania i wykonania operacji przykładowego systemu [14]

jest w tych trzech przypadkach taki sam. Rośnie tylko (wolno) czas związany z obsługą pętli sterowania.

Zmierzone wartości czasów operacji dla procedury *_DPL_ISR* projektu *LED_Boost_PC* wyglądają bardzo podobnie. Czas wykonania procedury ISR jest powiększony przez wywołanie pomocniczej procedury *comp* (język C) o czasie wykonania ok. 117 cykli zegara. Łączny czas wykonania procedury ISR wynosi ok. 361 cykli zegara systemowego.

Wyliczenie obciążenia ISR jest łatwe – *ISR_Load* = $(T_{ISR}/T_{SAMPLE}) \times 100\%$.

Procesory rodziny Piccolo F2802x mają zegar systemowy 60 MHz z okresem TCLK=16,6 ns.

Dla sygnału PWM 50 kHz okres wynosi TPWM=20 µs. Jednak procedura *_DPL_ISR* jest wykonywana co trzeci okres sygnału PWM, dlatego efektywny okres T_{SAMPLE} wynosi 60 µs (ok 3600 cykli zegara systemowego).

Dla procedury *_DPL_ISR* projektu *LED_Boost_PC* czas wykonania wynosi ok. 361 cykli zegara, co daje czas ok. 6,01 µs i obciążenie *ISR_Load* wynosi 6,1 µs/60 µs=10,02%.

Pętla BG

Pętla BG realizowana w ramach kodu BG nie ma określonego precyzyjnie, deterministycznego czasu wykonania. Typowo jest ona zrealizowana z zastosowaniem kodu decyzji („if then else") zwykle napisanego w języku wysokiego poziomu.

Pasmo pętli BG można określić jako *BG_BW=100% – ISR_Load*.

Dla procedury *DPL_ISR* projektu *LED_Boost_PC* pasmo BG_BW=89,97%.

Jest możliwe określenie średniego czasu wykonania pętli BG.

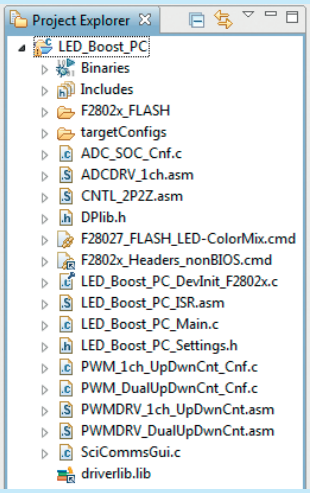
Średnia prędkość pętli BG można obliczyć jako *BG_LR=BG_MIPS/LPI*,
gdzie:

- I*BG_MIPS = BG BW x CPU MIPS*
- I*CPU_MIPS* = wydajność procesora
- I*LPI* = liczba instrukcji najdłuższej ścieżki kodu

Dla procesora rodziny Piccolo F2802x można przyjąć, że *CPU_MIPS* = 60 MIPS [7, 14, 18], wtedy *BG MIPS*= 60×89.97%=53.98 MIPS.

Jeśli założymy (arbitralnie), że najdłuższa pętla ma 300 instrukcji (jednocyklowych) to BG_LR= 53.98 MIPS/300=17,99 kHz.

Dla określonego czasu wykonania procedury ISR balans pomiędzy obciążeniem ISR_Load a pasmem BG_BW jest określany poprzez odstęp próbkowania T_{SAMPLE}.



Rysunek 3. Organizacja projektu *LED_Boost_PC*

Organizacja projektu LED_Boost_PC

Opis sposobu pracy programu jest zamieszczony w nocie aplikacyjnej *Multi-DC/DC Conversion & Color LED Control Integrated on a C2000 Microcontroller* [17]. Projekt *LED_Boost_PC* może być budowany w dwóch wariantach: z otwartą pętlą lub z zamkniętą pętlą ze sterowaniem prądu. Dla diod LED każdego koloru zastosowana jest osobna pętla sterowania.

W projekcie *LED_Boost_PC* zostało zastosowane jedno przerwanie EPWM1_INT generowane przez moduł ePWM1 dla co trzeciego zdarzenia CTR=PRD (zawartość licznika podstawy czasu zgodna z rejestrem okresu).

Główne pliki zastosowane w projekcie *LED_Boost_PC* to (**rysunek 3**):

- *LED_Boost_PC_Main.c* – plik odpowiada za inicjalizację, uruchamianie i sterowanie aplikacją. Zawiera całą „logikę" aplikacji.
- *LED_Boost_PC_DevInit_F2802x.c* – plik odpowiada za inicjalizację i konfigurowanie układu procesorowego (TMS320F28027), ustawianie generacji zegara systemowego, konfigurowanie wejść GPIO itd.
- *LED_Boost_PC_Settings.h* – plik zawiera globalne definicje dla projektu. Jest linkowany do pliku *LED_Boost_PC_Main.c* oraz pliku *LED_Boost_PC_ISR.asm*.
- *ADC_SOC_Cnf.c* – plik zawiera funkcję *ADC_SOC_CNF* konfiguracji modułu ADC procesora
- *PWM_DualUpDwnCnt_Cnf.c* – zawiera funkcję *PWM_DualUpDwnCnt_CNF* konfiguracji modułu ePWM procesora do pracy dwukanałowej
- *PWM_1ch_UpDwnCnt_Cnf.c* – zawiera funkcję *PWM_1ch_UpDwnCnt_CNF* konfiguracji modułu ePWM procesora do pracy jednokanałowej
- *SciCommsGui.c* – zawiera funkcje obsługi komunikacji szeregowej z komputerem PC poprzez moduł SCI procesora.
- *LED_Boost_PC_ISR.asm* – plik zawiera asemblerowy kod krytyczny czasowo. Odpowiada za jednokrotną inicjalizację obsługi przerwania w procedurze *_DPL_Init*. Zawiera procedurę obsługi przerwania *_DPL_ISR*.
- *ADCDRV_1ch.asm* – zawiera asemblerowe makro *ADCDRV_1ch* obsługi modułu ADC przez procedurę obsługi przerwania *_DPL_ISR*. Kopiuje rezultat przetwarzania a/c ze wskazanego rejestru ADCResult do tabeli NetBus.
- *PWMDRV_1ch_UpDwnCnt.asm* – zawiera asemblerowe makro *PWMDRV_1ch_UpDwnCnt_INIT* inicjalizacji obsługi jednokanałowej modułu ePWM oraz asemblerowe makro *PWMDRV_1ch_UpDwnCnt* jednokanałowej obsługi modułu ePWM przez procedurę obsługi przerwania *_DPL_ISR*.
- *PWMDRV_DualUpDwnCnt.asm* – zawiera asemblerowe makro *PWMDRV_DualUpDwnCnt_INIT* inicjalizacji obsługi dwukanałowej modułu ePWM oraz asemblerowe makro *PWMDRV_DualUpDwnCnt* dwukanałowej obsługi modułu ePWM przez procedurę obsługi przerwania *_DPL_ISR*.
- *CNTL_2P2Z.asm* – zawiera asemblerowe makro *CNTL_2P2Z_INIT* inicjalizacji obliczeń sterowania jednego kanału koloru oraz asemblerowe makro *CNTL_2P2Z* obliczeń sterowania jednego kanału koloru przez procedurę obsługi przerwania *_DPL_ISR*.

Organizacja pracy kodu ASM

Impulsy SOC (rozpoczęcia przetwarzania ADC) są generowane przez moduł ePWM1 dla co trzeciego zdarzenia CTR=PRD. Tak samo, co trzeci okres, generowane jest przerwanie *EPWM1_INT* obsługiwane przez procedurę *_DPL_ISR*.

Procedurę *_DPL_ISR* (**listing 1**) obsługi przerwania *EPWM1_INT* zrealizowano w języku asemblera proceso-

rów z rodziny C2000 z rdzeniem C28x. Procedura ISR zawiera przede wszystkim obliczenia matematyczne i interfejs do przetwornika ADC oraz modułów PWM. Po napisaniu i uruchomieniu kodu typowo nie ma potrzeby jego modyfikowania.

Procedura *_DPL_ISR* wykorzystuje zrealizowane w języku asemblerowym drajwery (zorganizowane jako makra): obsługi dwukanałowej PWMDRV_DualUpDwnCnt i jednokanałowej PWMDRV_1ch_UpDwnCnt modułów ePWM (aktualizacja zawartości rejestrów porównania) oraz drajwer ADCDRV_1ch obsługi modułu ADC (odczyt rezultatu z rejestru). Używane jest również makro CNTL_2P2Z obliczeń sterowania (2 bieguny i 2 zera).

Procedura *_DPL_ISR* jest przepisywana z pamięci Flash do pamięci RAM i stamtąd wykonywana. Dokładny opis takiego postępowania jest zamieszczony w książce [19].

Organizacja pracy kodu BG

Zorganizowanie pracy pętli BG w dobrze zestrukturalizowany sposób jest bardzo krytyczne. Można to zrobić w różny sposób, ale najprościej jest zastosować automat stanów (*state machine*).

Implementacja automatu stanów w języku C jest bardzo prosta. Jest to zorganizowane w postaci podobnej do instrukcji „case”. Najlepszym sposobem jest zastosowanie wskaźnika do funkcji realizujących stany (**listing 2**). Kompilator języka C używa wtedy pośredniego trybu adresowania procesora i wywołanie jest realizowane poprzez wykonanie instrukcji skoku. Pozwala to oszczędzić bardzo dużo czasu.

Pętla BG projektu *LED_Boost_PC* jest podzielona na trzy automaty stanów (listing 3): A, B i C zrealizowane jako osobne funkcje A0, B0 i C0. Pętla główna zawiera wskaźnik na funkcję obsługi aktualnego stanu. Wskaźnik zawiera adres wywołania bieżącej funkcji. Raz na jedno wykonanie pętli BG wywoływana jest jedna funkcja i automat kończy działanie. Nie wykonuje oczekiwania na ustawienie warunku.

Dla każdego stanu jest zaprogramowany osobny licznik Timer CPU 0/1/2 z czasami okresu, odpowiednio, 1 ms, 10 ms i 50 ms. Bit TIF rejestru sterującego TCR licznika jest ustawiany, gdy licznik zliczy do zera. Gdy bit ten jest ustawiony, to po jego wyzerowaniu, wykonywane jest wywołanie funkcji podstanu. Każdy stan jest podzielony na dwa podstany zrealizowane jak osobne funkcje (np. A1, A2). Wywołanie podstanów jest zrealizowane ze wskaźnikiem adresowym osobno dla każdego stanu.

Automat A (co 1 ms) – **listing 3** – obsługuje zabezpieczenia przeciążeniowe i komunikację z komputerem PC. Automat B obsługuje pomiar napięć i prądów. Automat C obsługuje sterowanie kolorem świecenia diod LED. Płytka rozszerzeniowa *C2000 LED BoosterPack* ma trzy osobne przetwornice DC/DC pracujące w topologii „boost” przeznaczone dla szeregu diod LED każdego koloru [12]. Podwyższają one napięcie wejściowe (12 V typ.) do napięcia wystarczająco wysokiego do wprowadzenia diod w stan przewodzenia. Przetwornice DC/DC są sterowane przez sygnały PWM generowane przez moduły ePWM1 i ePWM2 procesora F28027 Piccolo zestawu *C2000 Piccolo LaunchPad*.

Listing 1. Procedura obsługi przerwania ISR

```
;;#####################################################
;; FILE: f2802x_examples/LED_Boost_PC/LED_Boost_PC_ISR.asm
;; TITLE:Interrupt service routines for LED BoosterPack
;;#####################################################
;; $TI Release: LaunchPad f2802x Support Library v100 $
;; $Release Date: Wed Jul 25 10:45:39 CDT 2012 $
;;#####################################################
  ;Gives peripheral addresses visibility in assembly
  .cdecls   C,LIST,"DSP28x_Project.h"

  ;Include C header file - sets INCR_BUILD
  ;(used in conditional builds)
  .cdecls C,NOLIST, "LED_Boost_PC_Settings.h"

  ;Include files for the Power Library Maco's
  ;being used by the system
  .include "ADCDRV_1ch.asm"
  .include "CNTL_2P2Z.asm"
  .include "PWMDRV_DualUpDwnCnt.asm"
  .include "PWMDRV_1ch_UpDwnCnt.asm"

  .global _comp

;*****************************************************
; Variable declaration
;*****************************************************
; All Terminal modules initially point to the ZeroNet
; to ensure a known ; start state.
; Pad extra locations to accomodate unwanted ADC results.
        ; dummy variable for pointer initialisation
ZeroNet .usect "ZeroNet_Section",2,1,1 ;output terminal 1

        .text

        .sect "ramfuncs"
        ; label to DP ISR Run function
        .def    _DPL_ISR

;-----------------------------------------------------
; ISR Run
;-----------------------------------------------------
_DPL_ISR:    ;(13 cycles to get to here from ISR trigger)
    ;CONTEXT_SAVE
    ASP
    PUSH AR1H:AR0H ; 32-bit
    PUSH XAR2 ; 32-bit
    PUSH XAR3 ; 32-bit
    PUSH XAR4 ; 32-bit
;-- Comment these to save cycles --------
    PUSH XAR5 ; 32-bit
    PUSH XAR6 ; 32-bit
    PUSH XAR7 ; 32-bit
;-----------------------------------------
    PUSH XT      ; 32-bit
;    SPM       0                        ; set C28 mode
;    CLRC      AMODE
;    CLRC      PAGE0,OVM
    NOP

        ADCDRV_1ch 1
        ADCDRV_1ch 2
        ADCDRV_1ch 3

        LCR #_comp

        CNTL_2P2Z 1
        CNTL_2P2Z 2
        PWMDRV_DualUpDwnCnt 1

        CNTL_2P2Z 3
        PWMDRV_1ch_UpDwnCnt 2

        ADCDRV_1ch 9
        ADCDRV_1ch 10
        ADCDRV_1ch 11
        ADCDRV_1ch 12

;===================================
EXIT_ISR
;===================================
; Interrupt management before exit
    MOVW     DP,#_EPwm1Regs.ETCLR
    MOV     @_EPwm1Regs.ETCLR,#0x01 ; Clear EPWM1 Int flag
    MOVW     DP,#_PieCtrlRegs.PIEACK
    MOV     @_PieCtrlRegs.PIEACK, #0x4
                    ;Acknowledge PIE interrupt Group 3

; Restore context & return
    POP XT
;-- Comment these to save cycles ---
    POP XAR7
    POP XAR6
    POP XAR5
;-----------------------------------
    POP XAR4
    POP XAR3
    POP XAR2
    POP AR1H:AR0H
    NASP
    IRET
; end of file
```

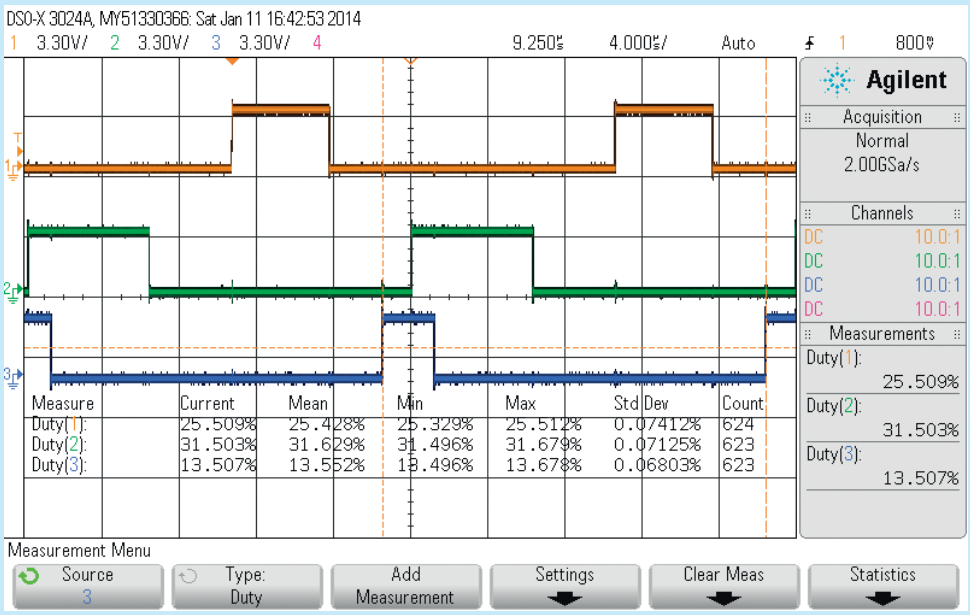
Moduły ePWM pracują w trybie zliczania w górę i w dół z okresem TPWM=20 µs. Dla modułu ePWM1, gdy zawartość licznika podstawy czasu TBCTR przy zliczaniu w górę jest równa zawartości rejestru CMPA wykonywana jest akcja zerowania – ustawienie poziomu wysokiego na wyjściu EPWM1A [10, 18]. Gdy zawartość licznika podstawy czasu TBCTR przy zliczaniu w dół jest równa zawartości rejestru CMPA wykonywana jest akcja zerowania – ustawienia poziomu niskiego na wyjściu EPWM1A. Poziom wysoki sygnału EPWM1A jest symetryczny względem największej wartości licznika podstawy czasu (rys. 4). Generowanie sygnału EPWM1B wygląda tak samo z zastosowaniem rejestru CMPA i z odwróceniem akcji. Poziom wysoki sygnału EPWM1B jest symetryczny względem zerowej wartości licznika podstawy czasu (**rysunek 4**).

Generowanie sygnału EPWM2A wygląda tak samo jak sygnału EPWM1A lecz z wykorzystaniem synchronizacji fazy – jest on opóźniony o 400 cykli (6.66 µs). Aktywny poziom jest wysoki. Takie rozsunięcie wysterowania w kanałach pozwala na zmniejszenie amplitudy poboru mocy chwilowej przez układ.

Okres wszystkich sygnałów PWM wynosi TPWM=20 µs. Regulowane jest wypełnienie. Szereg diod LED koloru niebieskiego jest sterowany sygnałem EPWM1A. Dla sterowania koloru zielonego jest sygnał EPWM1B a czerwonego sygnał EPWM2A. Na rys. 4 jest pokazany przykład sterowania diodami LED dla koloru białego z dokładnym pomiarem wypełnienia. Sygnały są pokazane następująco: 1 – blue (15 mA), 2 – green (20 mA), 3 – red (25 mA).

Listing 2. Pętla BG

```
// State Machine function prototypes
//-------------------------------------
// Alpha states
void A0(void);     //state A0
void B0(void);     //state B0
void C0(void);     //state C0

// A branch states
void A1(void);     //state A1
void A2(void);     //state A2

// B branch states
void B1(void);     //state B1
void B2(void);     //state B2

// C branch states
void C1(void);     //state C1
void C2(void);     //state C2

// Variable declarations
void (*Alpha_State_Ptr)(void);    // Base States pointer
void (*A_Task_Ptr)(void);         // State pointer A branch
void (*B_Task_Ptr)(void);         // State pointer B branch
void (*C_Task_Ptr)(void);         // State pointer C branch

...

    // Timing sync for background loops
    // Timer period definitions found
    // in device specific F2802x_device.h
    CpuTimer0Regs.PRD.all =  mSec1;          // A tasks
    CpuTimer1Regs.PRD.all =  mSec10;         // B tasks
    CpuTimer2Regs.PRD.all =  mSec50;         // C tasks

    // Tasks State-machine init
    Alpha_State_Ptr = &A0;
    A_Task_Ptr = &A1;
    B_Task_Ptr = &B1;
    C_Task_Ptr = &C1;

...

//=================================================================
//    BACKGROUND (BG) LOOP
//=================================================================
  for(;;)
  {
  // State machine entry & exit point
  //===================================================
  (*Alpha_State_Ptr)(); // jump to an Alpha state (A0,B0,..)
  //===================================================
  }
} //END MAIN CODE
```



Rysunek 4. Sygnały PWM przy wyświetlaniu koloru białego

Ćwiczenie

Dokładne omówienie zestawu ewaluacyjnego *C2000 Piccolo LaunchPad* jest zamieszczone w artykule [1] a modułu ePWM w artykule [10]. Dokładne omówienie środowiska CCSv5 oraz pakietu *controlSUITEv3* jest zamieszczone w artykule [3]. Opis instalowania najnowszej wersji środowiska CCS i pakietu *controlSUITE* jest zamieszczony w artykule [9].

1. Dołącz zestaw *C2000 LaunchPad* z płytką rozszerzeniową *C2000 LED BoosterPack* do komputera PC według opisu w artykule „C2000 Piccolo LanuchPad (11) – Łatwe sterowanie diodami LED-RGB mocy" [12].
2. Uruchom CCSv5 i wykonaj trzy pierwsze kroki zgodnie z opisem w artykule [12].

W celu debugowania programu projektu *LED_Boost_PC* nie trzeba ponownie programować wewnętrznej pamięci Flash procesora Piccolo F28027. Po pierwszym zaprogramowaniu wystarczy dołączyć się do procesora i wykonać ładowanie symboli (do debugera).

3. W perspektywie *CCS Edit* wybierz z menu *View* → *Target Configurations*. Rozwiń linie *Project* → *LED_Boost_PC* → *targetConfigs*.
4. Kliknij prawym klawiszem myszy na linię *TMS320F278027.ccxml [Default]* i na rozwiniętej liście kliknij lewym klawiszem myszy na *Launch Se-*

lected Configuration. Zostanie uruchomiony debuger i otwarta perspektywa *CCS Debug*.

5. Kliknij na ikonę *Connect Target*. Procesor zostanie dołączony poprzez port JTAG do debugera.

6. Z menu wybierz *View* → *Disassembly*. Zostanie otwarte okno *Disassembly* w którym zostanie pokazany aktualny kod asemblerowy z licznikiem rozkazów PC wskazującym na adres 0x3ff7bf początku procedury bootowania.

7. Kliknij na trójkącik obok ikony *Load*. Z listy wybierz pozycję *Load Symbols*. Kliknij *Browse projekt* i w ścieżce *Led_Boost_PC* → *F2802x_FLASH* zaznacz plik *LED-ColorMix.out* z kodem ładowalnym. Kliknij *OK*. I ponownie *OK*.

8. Kliknij na ikonę *Restart*. Program zostanie uruchomiony i zatrzymany na pierwszej instrukcji funkcji *main*.

9. W perspektywie *CCS Edit* otwórz plik *LED_Boost_PC_ISR.asm*.

10. W perspektywie *CCS Debug* kliknij *Resume*.

11. Ustaw pułapkę na pierwszej instrukcji procedury *_DPL_ISR*, w linii 98 (dwukliknij) z instrukcją *ASP*. Działanie programu zostanie zatrzymane na tej instrukcji.

12. Kliknij *CPU Reset* a następnie *Restart*. Po zatrzymaniu działania na pierwszej instrukcji funkcji *main* kliknij *Resume*.

Zobacz w pliku *LED_Boost_PC_ISR.asm*, że pułapka dalej jest pokazywana.

98 ASP

Ale program nie został na niej zatrzymany. **Dlaczego debuger nie zatrzymał wykonania programu na pułapce w linii 98.?**

13. Zdejmij pułapkę z linii 98.

14. Kliknij *CPU Reset* a następnie *Restart*. Po zatrzymaniu działania na pierwszej instrukcji funkcji *main* kliknij *Resume*.

15. Ponownie ustaw pułapkę na pierwszej instrukcji procedury *_DPL_ISR*, w linii 98 z instrukcją *ASP*. Tym razem działanie programu zostanie zatrzymane na tej instrukcji.

16. Z menu *Run* wybierz *Clock* → *Enable*. Na pasku stanu zostanie pokazana ikonka *Profile Clock* z wartością zero.

17. Ustaw drugą pułapkę na ostaniej linii kodu procedury *_DPL_ISR*, w linii 171 z instrukcją *IRET*.

18. Kliknij *Resume*. Licznik *Profile Clock* :361 pokazuje liczbę cykli instrukcyjnych wykonania procedury (ok. 361).

Listing 3. Automaty stanów

```
//=========================================================
//    STATE-MACHINE SEQUENCING AND SYNCRONIZATION
//=========================================================
void A0(void)
{
  // loop rate synchronizer for A-tasks
  if(CpuTimer0Regs.TCR.bit.TIF == 1)
  {
    CpuTimer0Regs.TCR.bit.TIF = 1;    // clear flag
    //-----------------------------------------------------
    (*A_Task_Ptr)();     // jump to an A Task (A1,A2,A3,...)
    //-----------------------------------------------------
    VTimer0[0]++;       // virtual timer 0, instance 0 (spare)
    SerialCommsTimer++; // used by DSP280x_SciCommsGui.c
  }
  Alpha_State_Ptr = &B0; //Comment out to allow only A tasks
}

void B0(void)
{
  // loop rate synchronizer for B-tasks
  if(CpuTimer1Regs.TCR.bit.TIF == 1)
  {
    CpuTimer1Regs.TCR.bit.TIF = 1;           // clear flag
    //-----------------------------------------------------
    (*B_Task_Ptr)();     // jump to a B Task (B1,B2,B3,...)
    //-----------------------------------------------------
    VTimer1[0]++;       // virtual timer 1, instance 0
                        //(used to control SPI LEDs)
  }
  Alpha_State_Ptr = &C0;
}

void C0(void)
{
  // loop rate synchronizer for C-tasks
  if(CpuTimer2Regs.TCR.bit.TIF == 1)
  {
    CpuTimer2Regs.TCR.bit.TIF = 1;           // clear flag
    //-----------------------------------------------------
    (*C_Task_Ptr)();     // jump to a C Task (C1,C2,C3,...)
    //-----------------------------------------------------
    VTimer2[0]++;       // virtual timer 2, instance 0 (spare)
  }
  Alpha_State_Ptr = &A0;      // Back to State A0
}
```

19. Kliknij *Resume*. Zobacz jaki stan pokazuje licznik. Czy to się zgadza z wartością oczekiwaną?

20. Skasuj licznik – dwukliknij na niego. Kliknij *Resume*.

Sprawdź liczbę cykli wykonania dla sekwencji kliku kolejnych uruchomień programu.

Wskazania licznika są dokładne. Zliczanie jest wykonywane sprzętowo przez moduł emulacji sprzętowej procesora. Dlatego inny jest rezultat pomiaru pomiędzy zatrzymaniami debugowymi oraz dla wykonania tego fragmentu kodu w pracy krokowej.

W ten sam sposób można sprawdzać czas działania różnych fragmentów kodu, np. zapamiętania stanu lub jego odtwarzania.

Henryk A. Kowalski
kowalski@ii.pw.edu.pl

Dodatkowe informacje (materiały do kursu znajdują się na serwerze EP: ftp://ep.com.pl, user: 86341, pass: 54cqkf85):

Dotychczas w EP na temat zestawu ewaluacyjnego C2000 Piccolo LaunchPad:

[1] „Zestaw ewaluacyjny C2000 Piccolo LaunchPad", EP 01/2013
[2] „C2000 Piccolo LanuchPad (1) – Pierwszy program w środowisku programowym CCS v5", EP 02/2013
[3] „C2000 Piccolo LanuchPad (2) – Łatwe programowanie z pakietem controlSUITE", EP 03/2013
[4] „C2000 Piccolo LanuchPad (3) – Łatwe programowanie do pamięci Flash", EP 04/2013
[5] „C2000 Piccolo LanuchPad (4) – Łatwa obsługa szyny SPI", EP 05/2013
[6] „C2000 Piccolo LanuchPad (5) – Łatwa obsługa szyny I²C", EP 07/2013
[7] C2000 Piccolo LanuchPad (6) – Łatwa inicjalizacja systemowa procesora serii Piccolo F2802x", EP 09/2013
[8] „C2000 Piccolo LanuchPad (7) – Łatwa obsługa wyświetlacza LCD", EP 11/2013
[9] „C2000 Piccolo LanuchPad (8) – Budowanie biblioteki drivelib dla procesorów serii Piccolo F2802x", EP 12/2013
[10] „C2000 Piccolo LanuchPad (9) – Łatwa obsługa modułu PWM procesora serii Piccolo F2802x", EP 1/2014
[11] „C2000 Piccolo LanuchPad (10) – Łatwa obsługa modułu eCAP procesora serii Piccolo F2802x", EP 2/2014
[12] „C2000 Piccolo LanuchPad (11) – Łatwe sterowanie diodami LED-RGB mocy", EP 3/2014
[13] „C2000 Piccolo LanuchPad (12) – Łatwy pomiar koloru", EP 4/2014

Bibliografia:

[14] Figoli D., Software Design for Digital Power – Programming 101 for Analog Designers, Topic 6, 2006/07 Power Supply Design Seminar - SEM1700, Texas Instruments
[15] C2000 LED BoosterPack User's Guide, SPRUHH9, 25 Jul 2012
[16] LED BoosterPack Quick Start Guide, SPRZ384, 17 May 2012
[17] Multi-DC/DC Conversion & Color LED Control Integrated on a C2000 Microcontroller, Application Report, SPRABR7, 01 Feb 2012
[18] Henryk A. Kowalski, Procesory DSP dla praktyków, BTC, Warszawa, 2011 http://ii.pw.edu.pl/kowalski/dsp/book,
[19] Henryk A. Kowalski, Procesory DSP w przykładach, BTC, Warszawa, 2012 http://ii.pw.edu.pl/kowalski/dsp/book