Miniaturowa płytka uruchomieniowa i programator dla mikrokontrolerów LPC810

W artykule opisano propozycję miniaturowej płytki uruchomieniowej dla mikrokontrolerów LPC810 oraz projekt działającego na niej programu demonstracyjnego korzystającego z timera SCT do generowania przebiegów PWM oraz z interfejsu UART mikrokontrolera. W programie zaprezentowano również nietypową technikę wywoływania bootloadera mikrokontrolera LPC810 z poziomu aplikacji.

Mikrokontroler LPC810 jest jedynym układem z serii LPC8xx umieszczonym w obudowie DIP. Jest to jednocześnie prawdopodobnie jedyny dostępny na rynku mikrokontroler 32-bitowy w 8-nóżkowej obudowie. Może on skutecznie zastępować niewielkie mikrokontrolery 8-bitowe, dysponując wielokrotnie większą od nich mocą obliczeniową i bogatym zestawem modułów funkcjonalnych, w tym zaawansowanym timerem i dwoma interfejsami UART.

Mikrokontroler jest wyposażony we wbudowany program bootloadera umożliwiający programowanie w urządzeniu docelowym za pomocą interfejsu szeregowego UART. Bootloader może być uruchomiony na kilka sposobów:

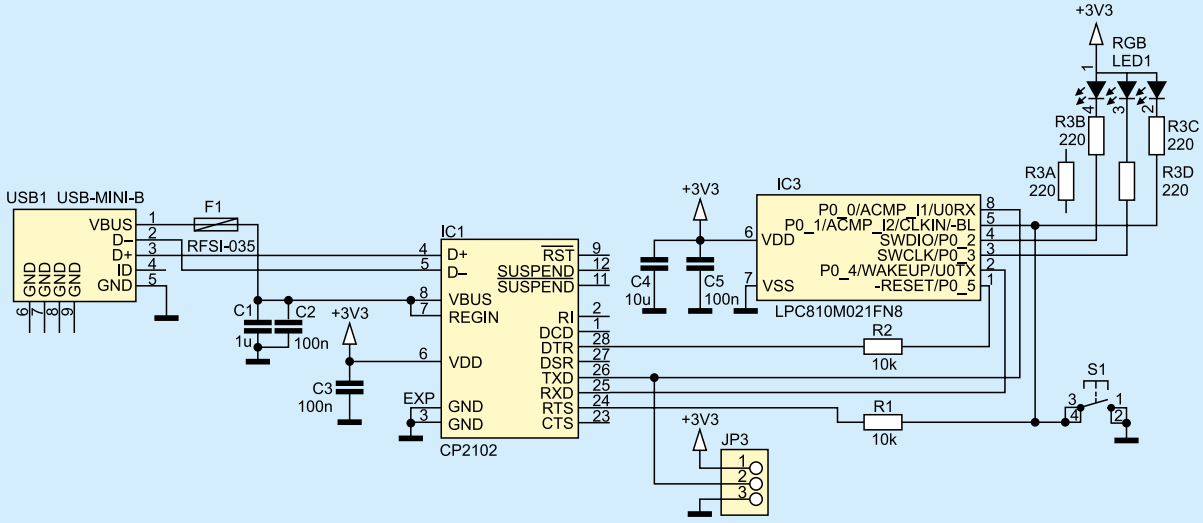
- Samoczynnie przy starcie mikrokontrolera, o ile nie zawiera on ważnej tablicy adresów procedur obsługi wyjątków (np. gdy pamięć Flash jest czysta).
- Po zwarciu do masy linii P0.1 przy włączeniu zasilania lub zwolnieniu sygnału *Reset*.
- Programowo, przez wywołanie odpowiedniej usługi z pamięci ROM mikrokontrolera.

Płytka uruchomieniowa – programator

Prezentowana płytka jest prawdopodobnie najmniejszym modułem uruchomieniowym i programatorem dla 32-bitowych mikrokontrolerów LPC810 firmy NXP. Jako moduł uruchomieniowy umożliwia ona uruchamianie prostych programów dla 32-bitowego mikrokontrolera LPC810, obsługujących komunikację z PC, sterowanie trójkolorową diodą LED i reagowanie na przycisk. Skonfigurowana jako programator może służyć do wygodnego programowania układów mikrokontrolerów przeznaczonych do umieszczenia w innym urządzeniu docelowym. Schemat elektryczny płytki może być zastosowany w dowolnym urządzeniu wyposażonym w mikrokontroler serii LPC8xx o podobnym zastosowaniu.

Zaprojektowany moduł jest wyposażony w interfejs USB zapewniający zasilanie oraz umożliwiający komunikację z komputerem PC i programowanie mikrokontrolera. Płytka prototypowa ma wymiary około 20 mm×24 mm. Wykonano ją jako dwustronną, z jednostronnym montażem elementów. Projekt płytki prototypowej przewiduje dwie możliwości jej montażu: moduł uruchomieniowy z możliwością programowania mikrokontrolera w układzie lub programator mikrokontrolerów LPC810, który może być wygodnie używany do programowania serii mikrokontrolerów. Schemat ideowy płytki pokazano na **rysunku 1**.

Układ interfejsu USB-UART typu CP2102 zapewnia zasilanie mikrokontrolera napięciem 3,3 V oraz komunikację z komputerem PC poprzez wirtualny port szeregowy np. przy użyciu dowolnego programu emulatora terminala. Do połączenia z PC służy złącze USB typu mini-B.



Rysunek 1. Schemat ideowy płytki demonstracyjnej

Mikrokontroler LPC810 jest umieszczony w podstawce. W celu zabezpieczenia portu USB komputera przed potencjalnym uszkodzeniem wynikającym z przypadkowego zwarcia, na płytce umieszczono bezpiecznik polisilikonowy PTC. Zastosowane kondensatory służą do filtrowania napięcia zasilającego i eliminowania zaburzeń. Wygląd płytki pokazano na **fotografii 2**.

Płytka zmontowana w konfiguracji modułu uruchomieniowego zawiera diodę RGB w obudowie PLCC4 i rezystor ograniczający natężenie prądu diody oraz – opcjonalnie – przycisk. W konfiguracji tej nie są stosowane rezystory R1 i R2. Taki wariant montażu umożliwia uruchamianie programów sterujących diodą LED, komunikujących się z komputerem PC poprzez interfejs UART oraz reagujących na stan przycisku.

Programowanie mikrokontrolera w tej konfiguracji płytki wymaga ręcznego wprowadzenia go w tryb bootloadera, np. poprzez naciśnięcie przycisku lub zwarcie pincetą odpowiednich otworów montażowych podczas podłączania przewodu USB. Botloader może być również wywołany przez oprogramowanie, np. w odpowiedzi na przesłanie odpowiedniego polecenia przez interfejs UART.

Płytka zmontowana w konfiguracji programatora nie zawiera przycisku i nie musi zawierać diody LED. Obecne w tej konfiguracji rezystory R1 i R2 umożliwiają sterowanie wejściami RESET i ISP (P0.1) mikrokontrolera przez układ CP2102 przez linie DTR i RTS interfejsu COM, zgodnie z typową konwencją, zaimplementowaną w popularnych programach do programowania mikrokontrolerów rodziny LPC, np. FlashMagic lub LPC21ISP. Dzięki temu mikrokontroler może zostać wprowadzony w tryb programowania bez odłączania przewodu ani naciskania przycisku przez użytkownika. Płytka w tej konfiguracji umożliwia wygodne programowanie układów LPC810.

Działanie płytki w obu konfiguracjach zostało przetestowane z oboma wymienionymi programami służącymi do programowania mikrokontrolerów serii LPC. Po usunięciu mikrokontrolera z podstawki płytki można również używać jako programatora lub interfejsu USB-UART.

Program demonstracyjny

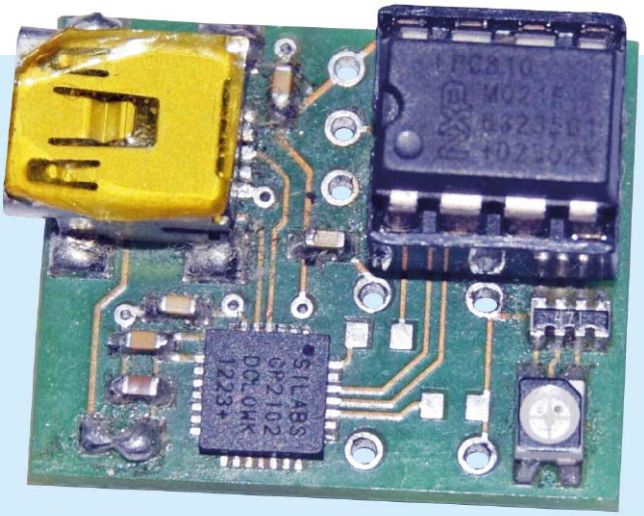
Program demonstracyjny realizuje następujące funkcje:

- Obsługę komunikacji z PC przez port szeregowy mikrokontrolera, w tym zadawanie jasności diody i programowe wejście w tryb programowania.
- Sterowanie diody RGB przebiegami PWM.
- Automatyczną zmianę koloru diody przy dłuższym braku poleceń z PC.

Program został napisany w konwencji automatu bez pętli zdarzeń – cała funkcjonalność jest realizowana w procedurach obsługi przerwań. Do zainicjowania mikrokontrolera użyto struktury danych zawierającej adresy rejestrów sterujących i ich wartości. Program demonstracyjny przygotowano w środowisku LPCxpresso.

Inicjowanie linii portów

Inicjowanie rozpoczyna się od włączenia w rejestrze SYSAHBCLKCTRL bloków IOCON i SWM, sterujących charakterystyką elektryczną i funkcjami wyprowadzeń mikrokontrolera. Zapis rejestru PINENABLE0 wyłącza domyślne funkcje specjalne wyprowadzeń, poza wejściem RESET (P0_5). W rejestrach PINASSIGNx następuje przypisanie funkcji wyprowadzeń – linii TX i RX interfejsu UART0 oraz wyjść PWM timera SCT. Zapisy rejestrów



Fotografia 2. Wygląd płytki prototypowej po zmontowaniu

bloku IOCON służą do zdefiniowania charakterystyki elektrycznej – program wyłącza rezystory podciągające oraz ustawia część wyjść w tryb z otwartym drenem, w celu zabezpieczenia przed ich zwarciem do masy z zewnątrz. Drugi zapis rejestru SYSAHBCLKCTRL wyłącza niepotrzebne już po skonfigurowaniu wyprowadzeń bloki IOCON i SWM i włącza moduły UART0 i timer SCT.

Programowanie UART0

Programowanie modułu UART wymaga zapisu sześciu rejestrów. Rozpoczynamy je od zapisu ułamkowego dzielnika częstotliwości, umożliwiającego uzyskanie typowych częstotliwości „transmisyjnych" przy dowolnej częstotliwości przebiegu zegarowego mikrokontrolera. Odpowiednie definicje umieszczone na początku pliku powodują wyliczenie przez kompilator wartości dzielnika ułamkowego tak, by z częstotliwości 12 MHz wygenerować używaną przez moduły UART częstotliwość bliską 7372800 Hz, z której łatwo uzyskać można wszystkie typowe szybkości transmisji do 460800 b/s. Ustawienie głównego dzielnika częstotliwości w rejestrze UARTCLKDIV powoduje włączenie przebiegu zegarowego dla modułów UART. Następnie, zapisujemy dzielnik częstotliwości transmisji do rejestru BRG oraz ustawiamy długość słowa danych i włączamy moduł poprzez zapis rejestru CFG. Ostatnim etapem jest włączenie przerwania odbioru danych.

Programowanie timera SCT

Warto poświęcić nieco więcej uwagi timerowi SCT. Jest to blok o bardzo dużych możliwościach funkcjonalnych, znacznie przekraczających potrzeby programu demonstracyjnego. W konsekwencji zaprogramowanie go do wykonywania prostych funkcji, takich jak generowanie przebiegów PWM, jest znacznie bardziej złożone niż w przypadku prostszych timerów dostępnych w innych mikrokontrolerach.

Programowanie timera rozpoczynamy od jego podstawowej konfiguracji – wybieramy tryb 32-bitowy oraz automatyczne zerowanie przy osiągnięciu wartości rejestru MATCH0. Okres timera programujemy w 32-bitowym rejestrze MATCHREL[0].

Drugi etap programowania SCT – to konfiguracja zdarzeń. Korzystamy z czterech zdarzeń – początku okresu oraz osiągnięcia wartości wypełnień zaprogramowanych w trzech kanałach dla trzech kolorów diody. Zdarzenie początku okresu ułatwia sterowanie wypełnieniem – dzięki niemu wygaszenie diody może być osiągnięte przez zapis

Listing 1. Program demonstrujący działanie płytki

```
#ifdef __USE_CMSIS
#include „LPC8xx.h”
#endif

#include <cr_section_macros.h>
#include <NXP/crp.h>

__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
//=========================================================
#include „LPC8xxx.h”
//=========================================================
// Defs for LPC8n board

#define SYSCLK     __IRC_OSC_CLK
#define RXD_BIT    (1 << 0)
#define BTN_BIT    (1 << 1)
#define BLUE_BIT   (1 << 1)
#define RED_BIT    (1 << 2)
#define GREEN_BIT  (1 << 3)
#define TXD_BIT    (1 << 4)
#define BTN_BIT    (1 << 1)
//=========================================================
#define PWM_STEPS  250
#define PWM_FREQ   400
#define SCT_PRE   (SYSCLK / PWM_STEPS / PWM_FREQ)
#define LED_DARK   1
#define LED_MED   20
#define LED_ON    200
//=========================================================
#define U_PCLK   7372800ul
#define U_DIV   256ul
#define U_MULT   ((SYSCLK * U_DIV + U_PCLK / 2) / U_PCLK - U_DIV)
#define U0_BAUD   115200
#define U0_BRG   ((U_PCLK / 16 + U0_BAUD / 2) / U0_BAUD)
//=========================================================
// init table:
// each entry consists of a pair - control register address and its value
// all listed registers are loaded with their values at system init
struct init_entry_ {
    volatile uint32_t *loc;
    uint32_t value;
};

__attribute__ ((section(„.after_vectors.rodata”)))
static const struct init_entry_ init_table[] = {
    {&LPC_SYSCON->SYSAHBCLKCTRL, SYS_ACC_RSTVAL | SYS_ACC_IOCON | SYS_ACC_SWM},
    {&LPC_SWM->PINENABLE0, 0b1101111111},  // leave reset, disable all other SF
    {&LPC_SWM->PINASSIGN0, 0x0ffff0004},  // U0_RXI, U0_TXO
    {&LPC_SWM->PINASSIGN6, 0x02ffffff},  // CTOUT_0 -> PIO0_7/RED
    {&LPC_SWM->PINASSIGN7, 0xffff0103},  // CTOUT_1 -> PIO0_17/green, CTOUT_2 -> PIO0_16/blue, CTOUT_3 -
    // LEDs
    {&LPC_IOCON->PIO0_1, IOCON_RSVD | IOCON_OD},        // button - should be OD
    {&LPC_IOCON->PIO0_2, IOCON_RSVD | IOCON_OD},        // remove pullups
    {&LPC_IOCON->PIO0_3, IOCON_RSVD | IOCON_OD},
    {&LPC_IOCON->PIO0_4, IOCON_RSVD},
    {&LPC_SYSCON->SYSAHBCLKCTRL, SYS_ACC_RSTVAL | SYS_ACC_UART0 | SYS_ACC_SCT},
    // UART
    {&LPC_SYSCON->UARTFRGMULT, U_MULT - 1},
    {&LPC_SYSCON->UARTFRGDIV, U_DIV - 1},
    {&LPC_SYSCON->UARTCLKDIV, 1},
    {&LPC_USART0->BRG, U0_BRG - 1},
    {&LPC_USART0->CFG, USART_CFG_DL8 | USART_CFG_EN},
    {&LPC_USART0->INTENSET, USART_STAT_RXRDY},
    // SCT
    {&LPC_SCT->CONFIG, SCT_CFG_AUTOLIMIT_L | SCT_CFG_UNIFY},  // unified, autolimit on match 0
    //{&LPC_SCT->OUTPUT, 7},  // outputs initially high
    {&LPC_SCT->MATCHREL[0].U, PWM_STEPS - 1},
    // event 0 - pwm cycle start
    {&LPC_SCT->EVENT[0].CTRL, 0x1004},
    {&LPC_SCT->EVENT[0].STATE, 0xFFFFFFFF},
    // event 1 - red match
    {&LPC_SCT->EVENT[1].CTRL, 0x1001},
    {&LPC_SCT->EVENT[1].STATE, 0xFFFFFFFF},
    // event 2 - green match
    {&LPC_SCT->EVENT[2].CTRL, 0x1002},
    {&LPC_SCT->EVENT[2].STATE, 0xFFFFFFFF},
    // event 3 - blue match
    {&LPC_SCT->EVENT[3].CTRL, 0x1003},
    {&LPC_SCT->EVENT[3].STATE, 0xFFFFFFFF},
    // output behavior
    {&LPC_SCT->OUT[0].SET, 0x02},  // high at match
    {&LPC_SCT->OUT[0].CLR, 0x01},  // low at cycle start
    {&LPC_SCT->OUT[1].SET, 0x04},
    {&LPC_SCT->OUT[1].CLR, 0x01},
    {&LPC_SCT->OUT[2].SET, 0x08},
    {&LPC_SCT->OUT[2].CLR, 0x01},
    {&LPC_SCT->RES, 0b010101},  // set on conflict
    // enable event int and start
    {&LPC_SCT->EVEN, 1},  // enable event 0 interrupt
    {&LPC_SCT->CTRL_U, (SCT_PRE - 1) << 5},   // start SCT
    // enable ints
    {&NVIC->ISER[0], 1 << UART0_IRQn | 1 << SCT_IRQn},
        {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},
    {0, 0}
};
//=========================================================
static inline void writeregs(const struct init_entry_ *p)
{
    for (; p->loc; p ++) *p->loc = p->value;
}
//=========================================================
// IAP stuff
typedef void (*IAP)(const uint32_t [], uint32_t []);
#define iap_entry ((IAP)(0x1FFF1FF1))
static const  uint32_t command[] = {57};
//=========================================================
__attribute__ ((section(„.after_vectors”)))
int main(void)
{
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 07641, pass: yus9jv2r

Listing 1. c.d.

```
    static const  uint32_t command[] = {57};
    if (LPC_SYSCON->SYSRSTSTAT & SYS_SYSRSTSTAT_SYSRST)
    {
        LPC_SYSCON->SYSRSTSTAT = SYS_SYSRSTSTAT_SYSRST;
        iap_entry (command, 0);      // invoke bootloader
    }
    writeregs(init_table);
    __WFI();
    return 0;  // will NOT arrive here, just to suppress warning
}
//=======================================================
#define NPHASES 3
#define DIVR 800
//=======================================================
static uint16_t div = PWM_FREQ * 2;
static uint8_t target[3] = {0, 0, 0};
//=======================================================
void SCT_IRQHandler(void)
{
    static uint8_t phase = 0;
    LPC_SCT->EVFLAG = 1;
    if (-- div == 0)
    {
        div = PWM_FREQ * 2;
        // change target intensity
        target[0] = 0;
        target[1] = 0;
        target[2] = 0;
        target[phase] = LED_ON;
        if (++ phase == NPHASES) phase = 0;
    }
    uint32_t i, mval;
    for (i = 0; i < 3; i ++)
        if ((mval = LPC_SCT->MATCHREL[i + 1].U) != target[i])
            LPC_SCT->MATCHREL[i + 1].U = mval > target[i] ? mval - 1 : mval + 1;
}
//=======================================================
void UART0_IRQHandler(void)
{
    static uint32_t val = 0;
    uint32_t c;
    if (LPC_USART0->INTSTAT & USART_STAT_RXRDY)
    {
        c = LPC_USART0->RXDATA;  // echo
        if ((c >= ,0' && c <= ,9') || (c >= ,a' && c <= ,f'))
        {
            LPC_USART0->TXDATA = c;
            if (c >= ,a') c -= ,a' - ,9' - 1;
            val = (val << 4) + c - ,0';
        }
        else if (c == ,\r')
        {
            LPC_USART0->TXDATA = c;  // echo
            LPC_USART0->INTENSET = USART_STAT_TXRDY;
            target[0] = val >> 16;
            target[1] = val >> 8;
            target[2] = val;
            div = PWM_FREQ * 15;  // 15 seconds delay
        }
        else if (c == ,\\')
        {
            SCB->AIRCR = 0x05FA0004;
            SCB->SCR = 0;  // continue to boot loader
        }
    }
    if (LPC_USART0->INTSTAT & USART_STAT_TXRDY)
    {
        LPC_USART0->INTENCLR = USART_STAT_TXRDY;
        LPC_USART0->TXDATA = ,\n';
    }
}
```

wypełnienia o wartości 0. Do wykrycia zdarzenia początku okresu posłuży rejestr MATCHREL[4], o domyślnej wartości 0. Wypełnienia dla trzech kanałów będą zadawane w rejestrach MATCHREL[1], MATCHREL[2] i MATCHREL[3]. W SCT zdarzenia mogą być uwarunkowane wartością zmiennej stanu timera. Ponieważ nie korzystamy z tego mechanizmu, maski zdarzeń we wszystkich rejestrach EVENT[n].STATE zostaną ustawione tak, by zdarzenia występowały w każdym stanie. Rejestry EVENT[n].CTRL zawierają na najmniej znaczących bitach numery rejestrów MATCHREL odpowiadających poszczególnym zdarzeniom, a bit 12 powoduje włączenie prostej generacji zdarzenia tylko na podstawie wartości rejestru porównania.

W sekwencji inicjującej nie programujemy rejestrów porównań – mają one domyślną wartość 0, co odpowiada wygaszeniu diod.

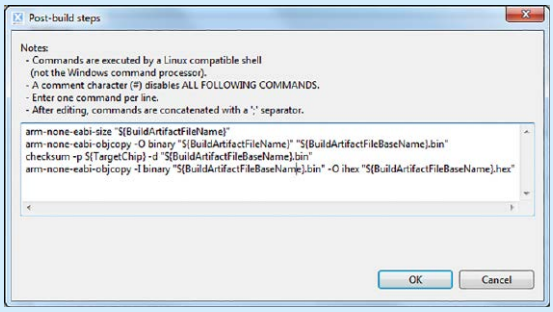
Trzeci etap programowania SCT polega na zdefiniowaniu zmian stanów wyjść w zależności od zdarzeń; służą do tego rejestry OUT[n].SET i OUT[n].CLR. Ponieważ diody świecą przy niskim stanie wyjść, dla każdego z trzech wyjść programujemy ustawienie w stan 0 na początku okresu (zdarzenie 0) i ustawienie w stan 1 przy wystąpieniu zdarzeń 1..3, dla każdego kanału oddzielnie. Aby zapewnić wyłączenie diod przy wypełnieniu równym 0, programujemy rejestr rozstrzygania konfliktów RES tak, by przy równoczesnym wykryciu żądania zaświecenia i wygaszenia diody wyższy priorytet miało wygaszenie.

Programowanie timera kończymy poprzez włączenie przerwania na końcu okresu (rejestr EVEN) i włączenie timera poprzez zapis rejestru preskalera (rejestr CTRL_U).

Na końcu sekwencji inicjującej włączamy przerwania timera SCT i UART0 w sterowniku przerwań oraz włączamy tryb usypiania procesora przy wyjściu z obsługi przerwania.

Obsługa przerwania UART

UART transmituje dane z szybkością 115200 b/s. Program reaguje na polecenia przesyłane z programu terminala działającego na komputerze PC. Po zainicjowaniu modułu UART jest włączane przerwanie odbioru znaku. Jeżeli



Rysunek 3. Postać finalna polecenia Post-build steps

odebrany znak jest cyfrą szesnastkową – zachodzi akumulacja wartości w zmiennej *val*, a odebrany znak jest odsyłany zwrotnie.

Odebranie kodu końca wiersza CR powoduje ustawienie jasności diod RGB na podstawie wartości trzech bajtów zmiennej val, odesłanie kodu CR i włączenie przerwania nadawania, którego późniejsze zgłoszenie spowoduje wysłanie kodu LF. Automatyczna zmiana koloru świecenia diody zostaje zablokowana na 15 sekund.

Wprowadzenie na terminalu znaku ukośnika powoduje programowe zresetowanie mikrokontrolera. Funkcję *main()* napisano w taki sposób, że po wykryciu takiego restartu następuje programowe wywołanie wbudowanego botloadera mikrokontrolera. Dzięki temu jest możliwe wprowadzenie w tryb programowania mikrokontrolera na płytce zmontowanej w konfiguracji demonstracyjnej (a nie programatora) przez polecenie z terminala, bez odłączania zasilania.

Przerwanie timera SCT

Przerwanie timera jest zgłaszane z częstotliwością 400 Hz. W każdym przerwaniu jest dekrementowany zmienna timera programowego, a przy osiągnięciu wartości 0 następuje nastawienie timera na czas 2 sekund i zmiana zadanego koloru światła na kolejny z tablicy. Jasności diod RGB są zmieniane płynnie poprzez modyfikację bieżącej wartości prz ykażdym przerwaniu timera tak, aby zbiegała ona do wartości zadanej.

Listing całego programu przedstawiono na **listingu 1**.

Generowanie pliku .hex w środowisku LPCxpresso

Do zaprogramowania układu mikrokontrolera przy użyciu programu FlashMagic jest potrzebny plik w formacie .hex. Przy domyślnej konfiguracji środowiska LPCxpresso proces kompilowania kończy się wygenerowaniem pliku .axf. W celu wygenerowania pliku .hex należy otworzyć dialog własności projektu, wejść w ścieżkę C/C++ Build – Settings i w zakładce Build steps wyedytować polecenie Post-build steps, usuwając znak ‚#' przed poleceniem:

```
arm-none-eabi-objcopy -O binary
„${BuildArtifactFileName}"
„$BuildArtifactFileBaseName}.bin"
```

oraz dopisując na końcu polecenie:

```
ęarm-none-eabi-objcopy -I binary
„$BuildArtifactFileBaseName}.bin" -O ihex
„${BuildArtifactFileBaseName}.hex"
```

Finalną postać linii polecenia *Post-build steps* przedstawiono na **rysunku 3**.

Programowanie układu przy użyciu FlashMagic

Program FlashMagic można ściągnąć z witryny *http://www.flashmagictool.com*. Wygodne programowanie mikrokontrolera na płytce w konfiguracji programatora wymaga skonfigurowania programu FlashMagic do sterowania wprowadzaniem mikrokontrolera w tryb bootloadera. W tym celu, po otwarciu programu FlashMagic, wybieramy z menu *Options → Advanced options* i w zakładce *Hardware Config* zaznaczamy *Use DTR and DST to control RST and ISP pin* (**rysunek 4**).

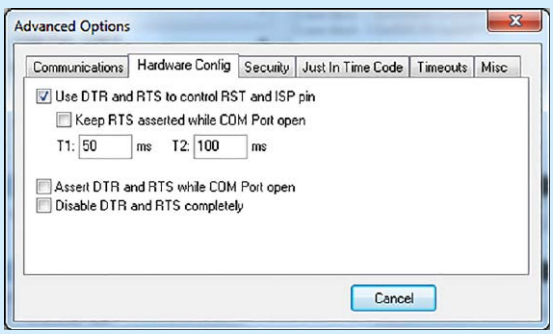
Jeżeli płytka została zmontowana w konfiguracji demonstracyjnej, bez sterowania liniami RESET i ISP, a mikrokontroler ma już zaprogramowaną pamięć Flash, należy zastosować jedną z opisanych wcześniej metod wywołania bootloadera – zewrzeć wejście ISP do masy przy włączaniu zasilania lub użyć odpowiedniej funkcji oprogramowania (w programie demonstracyjnym – wysłać z terminala znak ukośnika). Następnie w głównym oknie programu wybieramy plik .hex, którym chcemy zaprogramować mikrokontroler i naciskamy przycisk *Start* (**rysunek 5**).

Po zaprogramowaniu mikrokontrolera można uruchomić nowy program, naciskając przycisk Execute/Go (niebieska strzałka w pasku narzędzi u góry okna)

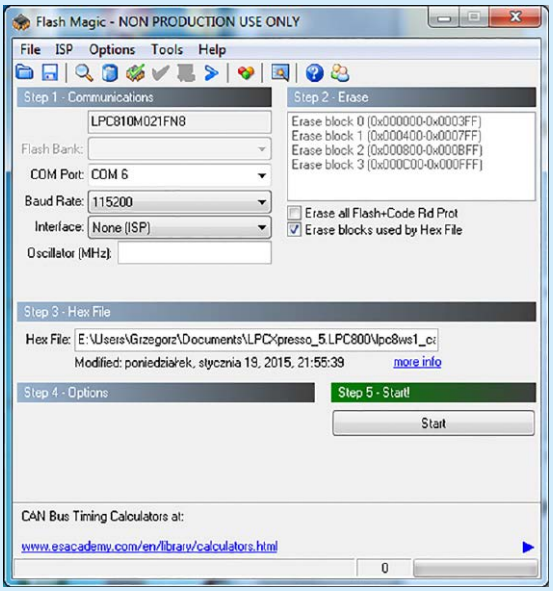
Grzegorz Mazur
g.mazur@ii.pw.edu.pl

Bibliografia

UM10601 – LPC81x User Manual, NXP 2014
LPC81xM Product Data Sheet, NXP 2014



Rysunek 4. Wygląd zakładki Hardware config



Rysunek 5. Okno programu Flash magic