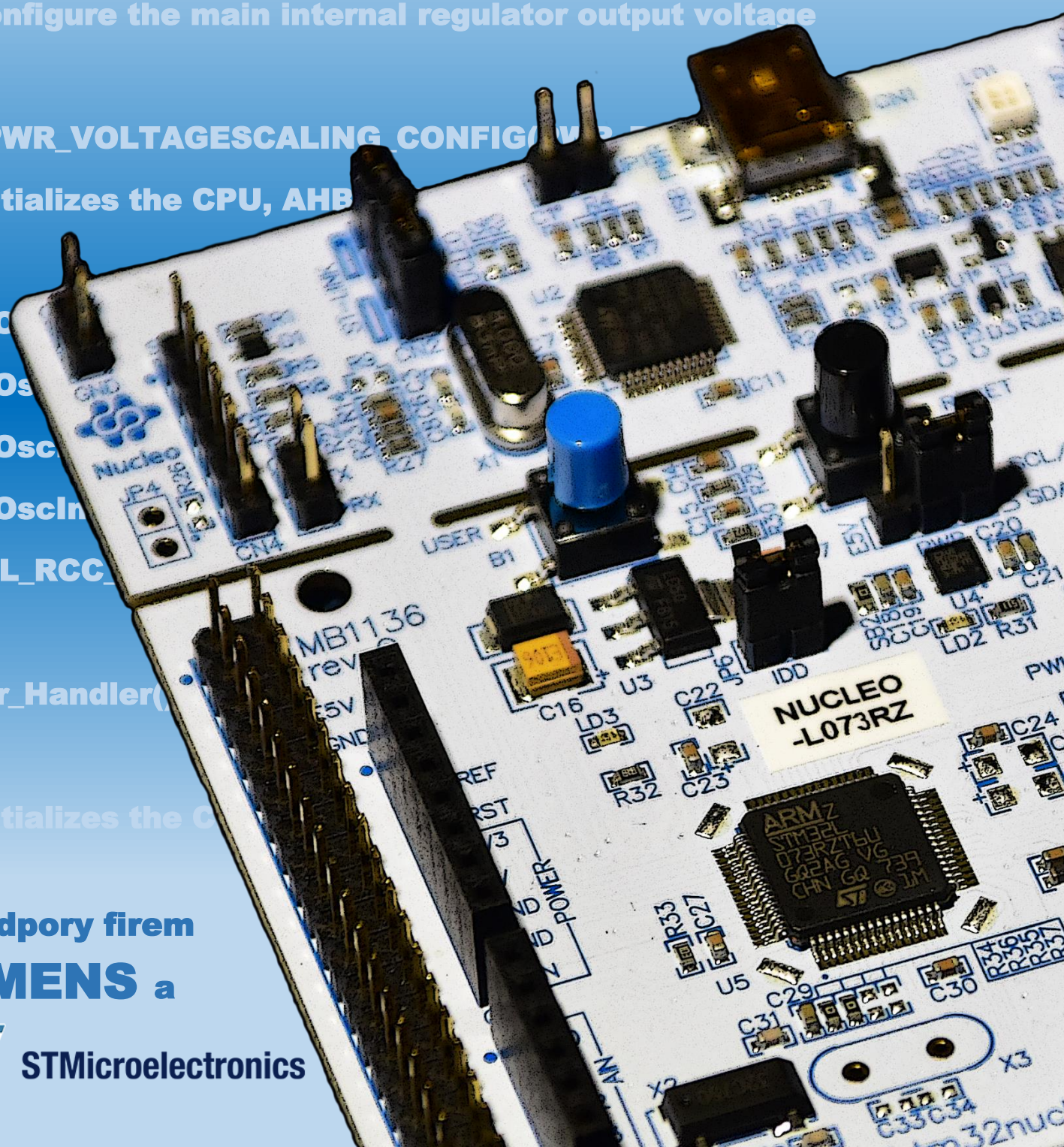
Programujeme STM32

zdolejte jednočipy profesionálů

Ing. Vojtěch Skřivánek

```
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
/** Configure the main internal regulator output voltage
*/
HAL_PWR_VOLTAGESCALING_CONFIG(
/** Initializes the CPU, AHB
*/
RCC_O
RCC_Os
RCC_Osc
RCC_OscIn
if (HAL_RCC_
{
  Error_Handler(
}
/** Initializes the C
```



Za podpory firem

SIEMENS a

STMicroelectronics

Poděkování

Děkuji firmám

SIEMENS

a

za podporu při psaní této knihy.

Obsah

Kapitola 7

Časovač

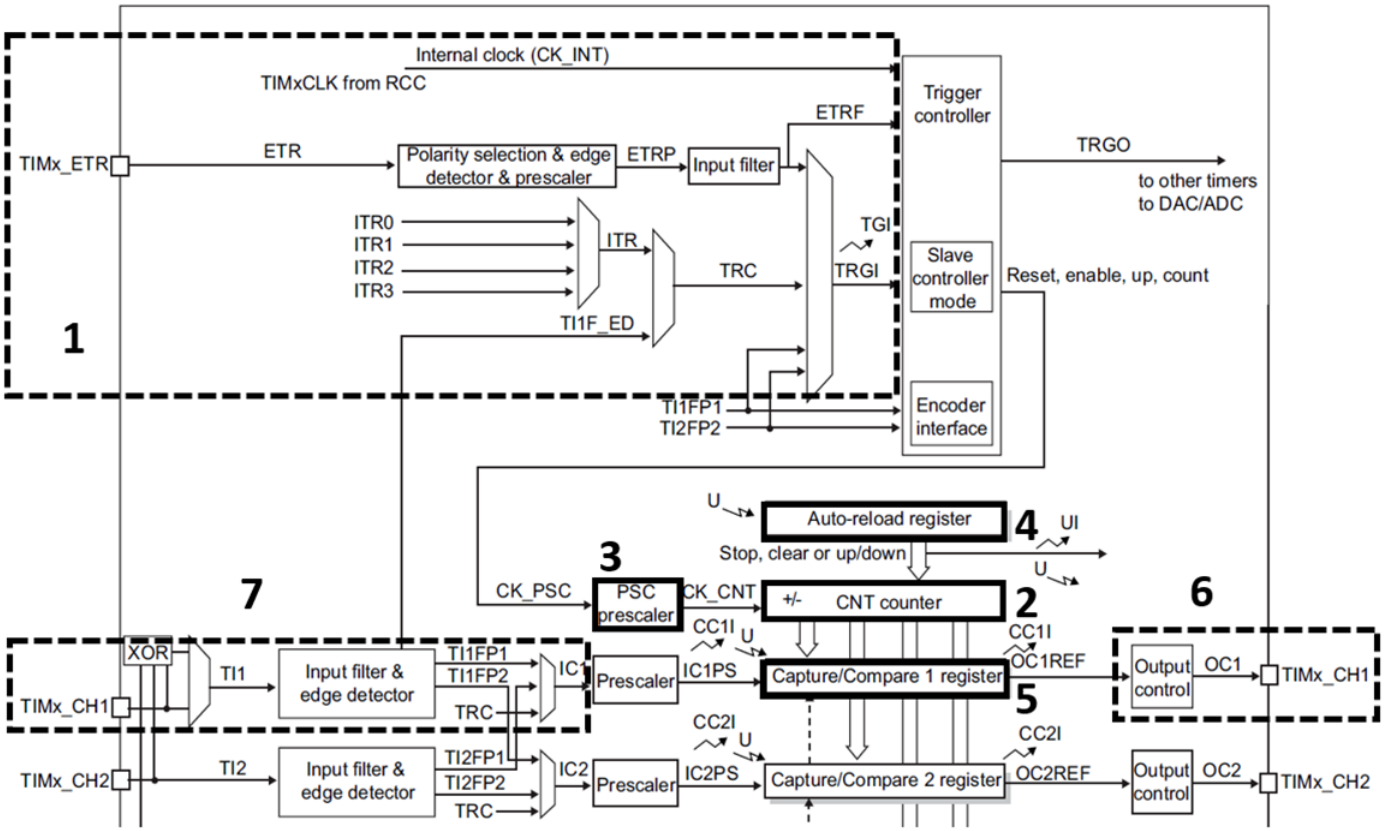
Stejně jako v životě i v běhu programu často rozhoduje o úspěchu to, zda se nachází ve správný čas na správném místě. Na rozdíl od života, kdy je tato podmínka většinou dílem náhody, v programu je vyžadováno, aby přesné načasování proběhlo pokaždé. Hardwarový časovač, který není nikterak ovlivněn během programu, je nástroj, který nám přesné načasování umožňuje.

Časovač je hardwarový blok mikrokontroleru, který počítá pulzy, jež do něj vstupují. Vstupní pulzy mohou být jednak s konstantním kmitočtem, a jejich zdrojem může být například hodinový signál kontroleru. Nebo mohou pulzy přicházet v náhodných momentech, kdy je často jejich zdrojem vnější signál připojený na pin kontroleru. V takovém případě je vhodnější místo časovače použít název čítač, jelikož periferie počítá pulzy, z jejichž množství nelze určit časový úsek, ve kterém proběhly.

STM32 disponuje třemi verzemi časovače – základní, univerzální a pokročilý. Tyto verze se liší komplexností jejich hardwarového řešení, a tedy i množstvím funkcí, které umožňují. Například základní časovač neumožňuje pokročilé režimy (***Input Capture, Output Compare a PWM***), kterými disponuje univerzální časovač a které budou popsány v následujících kapitolách.

7.1 Nezbytná teorie

Následující blokový diagram zobrazuje řešení univerzálního časovače, se kterým se bude pracovat v praktických příkladech na konci kapitoly.



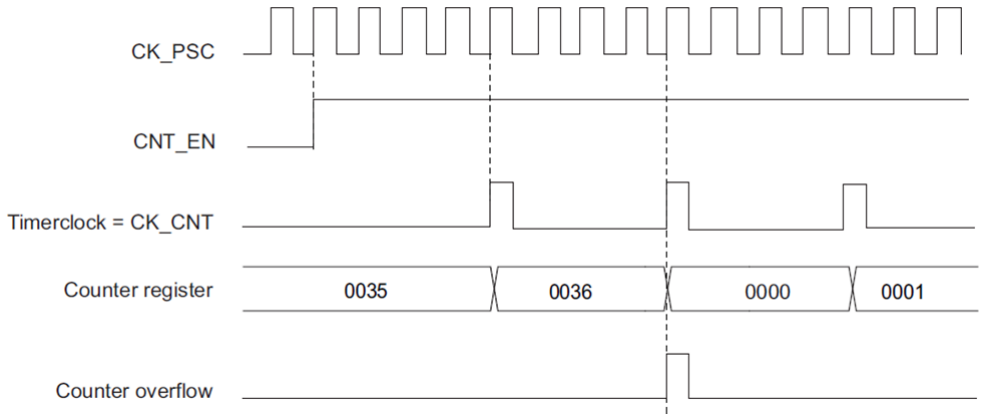
Blokový diagram obvodu univerzálního časovače [1]

V horní části diagramu jsou vidět možnosti nastavení zdroje hodinového signálu a spouštěčů časovače **(1)**. Zdrojem signálu může být interní hodinový signál, nebo externí zdroj připojený na vstupní pin. Spouštěči časovače mohou být vnitřní nebo vnější (připojené na vstupní pin) události. Při každé periodě vstupního signálu se inkrementuje hodnota čítacího registru časovače **(2)**.

Pokud je žádoucí, aby k inkrementaci docházelo méně často, je možné nastavit hodnotu registru předděličky (***Prescaler)*(3)** na hodnotu *((počet period k inkrementaci) - 1)*. Například při nastavení předděličky na hodnotu 3 bude čítací registr inkrementován každou čtvrtou periodu vstupního signálu.

Přetečení časovače (***Overflow***) se nazývá moment, kdy hodnota čítacího registru přesáhne hodnotu uloženou v ***AutoReload*** registru **(4)**. V ten okamžik dojde namísto další inkrementace k vynulování registru čítače. V praxi to znamená, že registr čítače nemůže nikdy nabýt hodnoty vyšší, než je uložená v ***AutoReload*** registru. Nastavení hodnoty ***AutoReload*** registru tedy určuje, po kolika inkrementacích čítacího registru dojde k přetečení časovače. Počet inkrementací nutných k přetečení je dán výrazem *((hodnota AutoReload registru) + 1)*. Například při nastavení ***AutoReload*** registru na hodnotu 54 (0x36), dojde k přetečení vždy po 55 inkrementacích.

Následující obrázek ukazuje průběh časovače s nastavením zmíněným v předchozích odstavcích. Čítací registr se inkrementuje každou čtvrtou periodu (hodnota předděličky je rovna 3) vstupního signálu a po jeho hodnotě 54 (0x36) dojde k jeho přetečení (hodnota ***AutoRelaod*** registru je rovna 0x36).



Inkrementace čítacího registru (Prescaler = 3; AutoReload registr = 0x36) [1]

Z předchozího je zřejmé, že jsou k dispozici tři nástroje, jimiž lze měnit periodu přetečení časovače. Prvním je volba zdroje signálu, druhým je nastavení předděličky a posledním je nastavení ***AutoReload*** registru.

Následující rovnice vyjadřuje vztah mezi kmitočtem přetečení časovače a nastavením jeho registrů:

$$f_{přetečení} = \frac{f_{pulzů}}{(Předdělička + 1) * (AutoReload + 1)} \tag{7.1}$$

Za předpokladu kmitočtu pulzů 220 kHz a nastavení časovače dle předchozího obrázku, bude docházet k přetečení s kmitočtem 1 kHz (1 ms).

$$f_{přetečení} = \frac{f_{pulzů}}{(3 + 1) * (54 + 1)} = \frac{220000Hz}{220} = 1000Hz \tag{7.2}$$

Přetečení časovače může být impulzem pro mnoho událostí, jako je například přerušení, spuštění převodu analogově digitální převodníku nebo přenosu dat pomocí ***DMA***.

Účel záchytného/porovnávacího registru (***Capture/Compare***) **(5)** bude popsán v následujících kapitolách.

7.1.1 Základní režimy časovače

Mezi čtyři nejčastěji používané režimy časovače patří jeho použití pouze jako časovače (bez speciálního režimu), režim ***Input Capture, Output Compare a PWM***.

V následujících kapitolách bude popsáno, jak jsou tyto režimy realizovány v STM32.

7.1.1.1 Časovač

Funkce časovače bez zvláštního režimu byla již částečně popsána v teorii. U takového časovače nastavujeme, kromě zdroje hodin a předděličky, pouze ***AutoReload*** registr, který nesmí být v momentě spuštění časovače nulový. Čítací registr časovače se inkrementuje až do hodnoty shodné s tou v ***AutoReload*** registru a poté dojde k přetečení, tedy vynulování čítacího registru, a vše se opakuje.

Časovač může čítat vzestupně s nulováním, sestupně vždy od maxima, nebo tzv. centralizovaně, kdy začne čítat vzestupně do maxima, a poté sestupně do nuly.

K čemu je tedy takový časovač dobrý? Jak již bylo zmíněno, s přetečením je možné spojit mnoho událostí, jako je například přerušení, v jehož obsluze můžeme provést jistou část programu, dále můžeme spustit převod analogově digitálního či digitálně analogového převodníku, případně můžeme začít přenos dat pomocí ***DMA***.

7.1.1.2 Output compare (OC)

Output Compare je první režim, který využívá Záchytný/Porovnávací registr (***Capture/Compare***) **(5)**.

Časovač se chová zcela shodně s předchozím režimem. Nyní však pomocí porovnávacího registru můžeme generovat ještě jednu událost v libovolném čase mezi (re)startem časovače a jeho přetečením. Touto událostí je nejčastěji přerušení, které má jiný příznak než přerušení způsobené přetečením, lze je tedy v obsluze přerušení rozeznat. Hodnotu porovnávacího registru je možno měnit kdykoliv za běhu časovače.

Registr časovače se tedy inkrementuje s každým hodinovým pulzem a v momentě, kdy je jeho hodnota shodná s hodnotou v porovnávacím registru, reaguje nastavenou událostí, zatímco se registr časovače dále inkrementuje.

Tímto způsobem můžeme například v obsluze přerušení ***OC*** rozsvítit LED, a zhasnout ji v obsluze přerušení přetečení. Doba přetečení nám udává frekvenci blikání, doba ***OC*** její střídu (poměr doby zhasnuté a rozsvícené LED). Samozřejmě je mnohem snazší pro tuto funkci použít režim ***PWM***, kterému se věnuje jiná podkapitola.

Jelikož se předpokládá, že režim ***OC*** bude použit především pro řízení výstupních napěťových úrovní pinů, jsou některé z nich v alternativním režimu hardwarově spojeny s blokem časovače **(6)**. Časovač je řídí (přepnutí do vysoké, nízké nebo opačné napěťové úrovně) bez nutnosti vyvolání přerušení, a tedy zásahu do programu. Máte tedy naprostou jistotu, že ke změně úrovně dojde vždy ve stejný čas, a ani například přerušení s vyšší prioritou nezmění interval řízení pinu.

7.1.1.3 Input Capture (IC)

Režim ***Input Capture*** také využívá ke své funkci Záchytný/Porovnávací registr (***Capture/Compare***) **(5)**.

Na rozdíl od ***OC*** režimu, jehož funkce byla řízena časovačem a vyvolávala událost na výstupním pinu, je ***IC*** režim řízen vstupním pinem v alternativním režimu, jenž spustí událost časovače.

Časovač se opět chová stejně jako volně běžící, reaguje však na událost vstupního pinu. **(7)** Vstupní pin je připojen k detektoru náběžné/sestupné hrany. Hrana způsobí okamžitý automatický přesun hodnoty čítacího registru časovače do záchytného registru. Běh časovače tím není nijak ovlivněn, pokračuje dál ve svých inkrementacích. Program má však možnost si do příchodu další události kdykoliv hodnotu registru vyčíst, v jaký čas k události došlo.

S událostí ***IC*** je možné spojit vyvolání přerušení. Běžné použití tohoto režimu tedy vypadá tak, že spuštěný časovač inkrementuje s každým hodinovým pulzem. V momentě události na vstupním pinu se okamžitě přesune hodnota čítacího registru do záchytného a vyvolá se přerušení. V přerušení může být hodnota vyčtena ze záchytného registru a spočítána doba, která uběhla od spuštění časovače do chvíle události na vstupním pinu. Při další iteraci procesu je možné spočítat přesnou dobu mezi dvěma událostmi (případně kmitočet periodického vstupního signálu), jelikož událost na vstupním pinu nikterak neovlivňuje běh časovače.

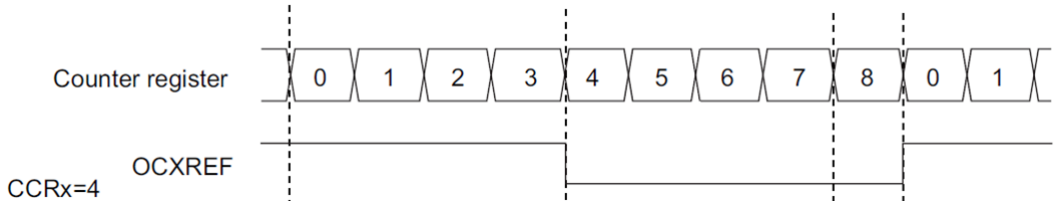
Na obrázku s blokovým diagramem je zřejmé, že jeden vstupní pin může řídit dva kanály časovače a naopak dva piny mohou řídit jeden kanál **(7)**. První případ může být využit pro elegantní řešení měření střídy vstupního periodického signálu.

7.1.1.4 PWM - pulzně šířková modulace

Režim ***PWM*** se v mnohém podobá režimu ***OC***.

Hlavní rozdíl mezi těmito režimy spočívá v tom, že řízení výstupního pinu neproběhne pouze při shodě čítacího a porovnávacího registru, ale také při přetečení časovače. Při obou událostech dojde k přepnutí pinu do opačné napěťové úrovně.

Následující obrázek vykresluje napěťové úrovně výstupního pinu v momentě, kdy má časovač v ***PWM*** režimu nastaven ***AutoReload***registr na hodnotu 8 a porovnávací registr na hodnotu 4.



Průběh PWM signálu (porovnávací registr = 4; AutoReload registr = 8) [1]

Z obrázku je zřejmé, že hodnota ***Autoreload registru*** určuje kmitočet výstupního signálu a hodnota porovnávacího registru jeho střídu (poměr času signálu ve vysoké úrovni vůči době v nízké). Obrázek znázorňuje řežim ***PWM1***.Režim ***PWM2*** se liší pouze v opačné polaritě signálu.

Hodnotu porovnávacího registru je možné, stejně jako u ***OC*** režimu, měnit kdykoliv za běhu časovače. Stejně tak mohou obě události tohoto režimu vyvolat přerušení.

Tímto způsobem lze velice jednoduše vytvořit např. výstupní hodinový signál pro jiné externí periferie, ale především je takto možné různou rychlostí řídit motory, nastavovat jas LED, nebo, po připojení ke správnému hardwarovému filtru, generovat analogový signál.

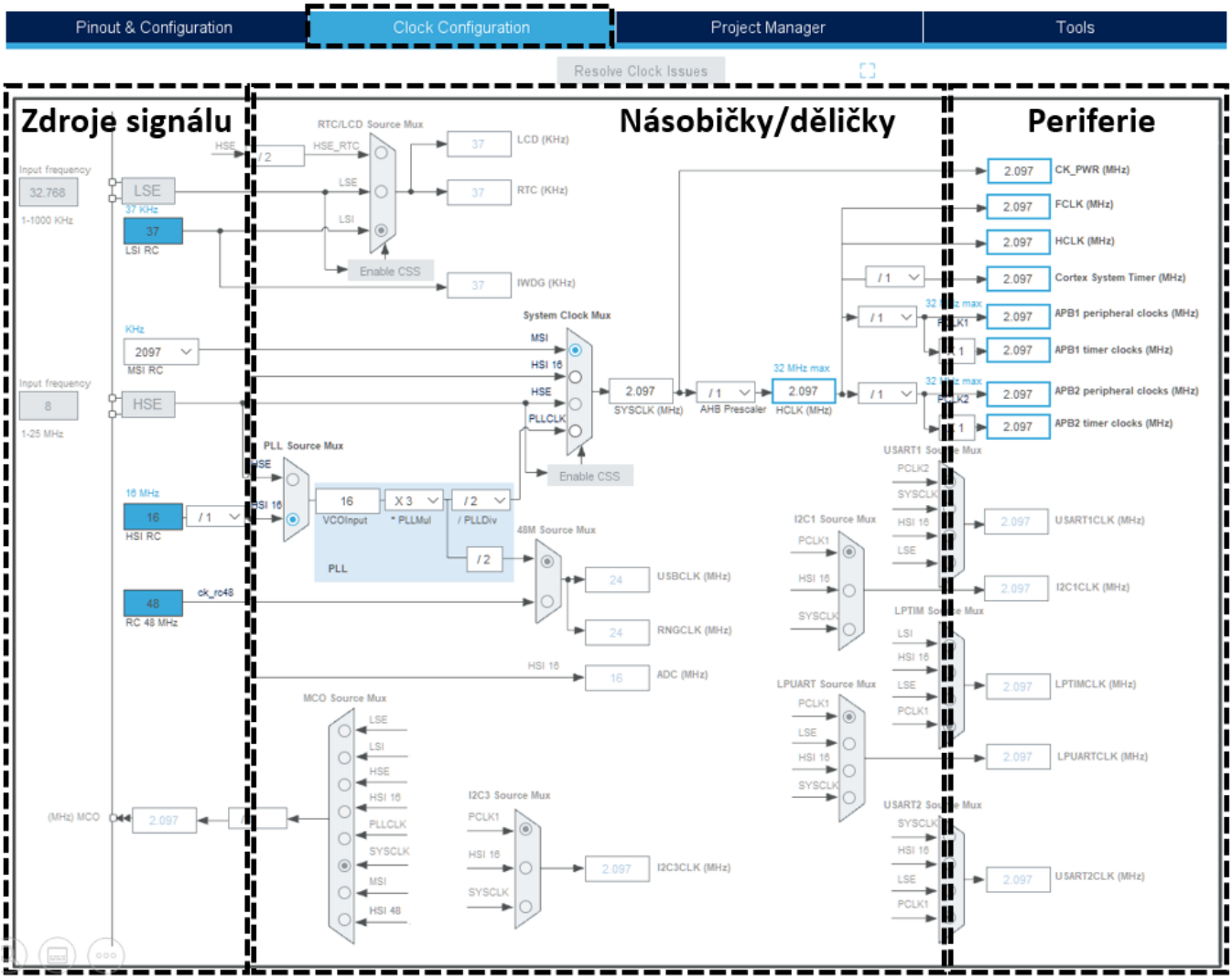
7.2 Praktické ukázky

7.2.1 Blikání pomocí časovače s přerušením

V prvním příkladu, který využívá časovače v základním režimu, si ukážeme, jak nastavit volně běžící časovač tak, aby každou sekundu vyvolal přerušení. Poté upravíme vygenerovaný program takovým způsobem, aby došlo ke spuštění časovače v režimu přerušení. V obsluze přerušení změníme logickou úroveň výstupního pinu, čímž dojde k zhasnutí či rozsvícení LED nacházející se na ***Nucleo*** desce s přesně definovaným intervalem.

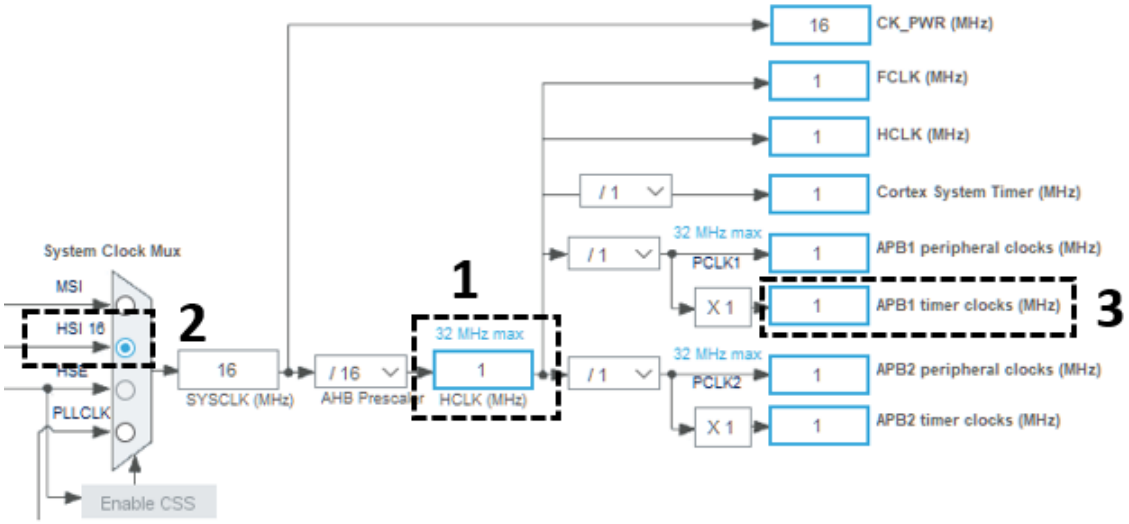
7.2.1.1 Nastavení hodinového signálu

V tomto příkladu se budeme poprvé zajímat o kartu nastavení hodin – ***Clock Configuration***. Po jejím otevření uvidíme interaktivní diagram. V pravé části diagramu vidíme rozličné zdroje hodinového signálu, které mohou pohánět procesor a periferie. V prostřední části můžeme nastavením násobiček a děliček kmitočtu ovlivnit výsledný kmitočet zdroje signálu. V pravé části se nachází periferie kontroleru, do nichž hodinový signál vstupuje.



Nastavení hodinového signálu je pro nás nyní důležité, protože je s ním spjata doba, za kterou časovač inkrementuje svůj čítací registr. Předpokládejme, že chceme, aby hodinový signál, který bude vcházet do časovače, měl kmitočet 1 MHz. Nejjednodušší způsob, jak toho docílit, je zapsáním hodnoty 1 do pole ***HCLK***

(1). Po potvrzení hodnoty se nás program zeptá, zda může použít jiný než právě používaný zdroj hodinového signálu. Po povolení dojde k automatickému výběru zdroje signálu **(2)** a nastavení násobiček a děliček tak, aby 1 MHz byl hlavním hodinovým kmitočtem všech periferií. Tedy včetně časovače **(3)**, který budeme v příkladu využívat.

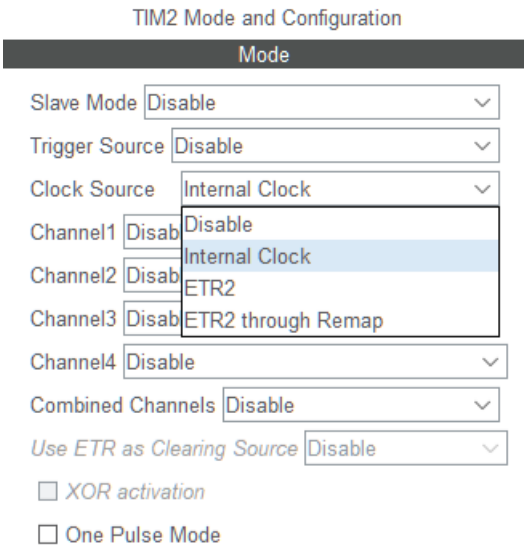


Tímto jsme hotovi s nastavením hodinového signálu pro časovač.

7.2.1.2 nastavení periferií

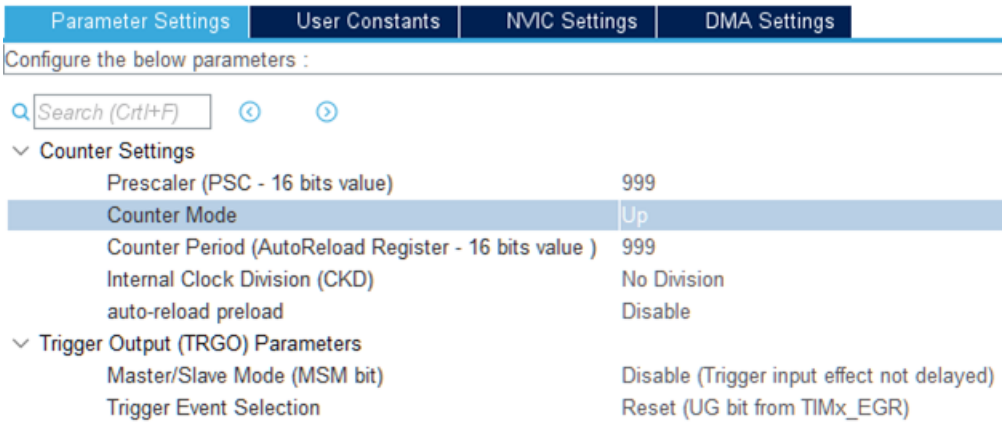
V dalším kroku nastavíme časovač tak, aby se jeho čítací registr inkrementoval každou milisekundu a jednou za sekundu vyvolal přerušení.

V kartě ***Pinout & Configuration*** v abecedním seznamu periferií klikneme na ***TIM2***. Zobrazí se nabídka nastavení časovače. V horní části nabídky – nabídky režimu - můžeme vybrat zdroje hodin a režimy jednotlivých kanálů časovače. Pro náš příklad, v němž budeme pracovat pouze v základním módu časovače, stačí zvolit vnitřní zdroj hodinového signálu – ***Internal clock*** – a zbytek nechat ve výchozím stavu.

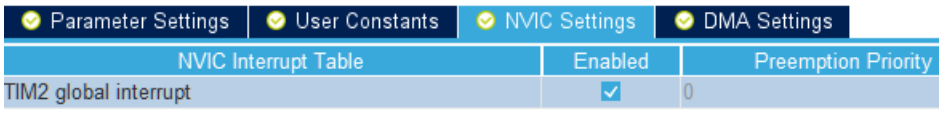


Ve spodní části nabídky – nabídky konfigurace - v kartě ***Parameter Settings*** vidíme další nastavení časovače. Naším cílem je, aby k přerušení vlivem přetečení časovače došlo jednou za sekundu. V tomto momentě

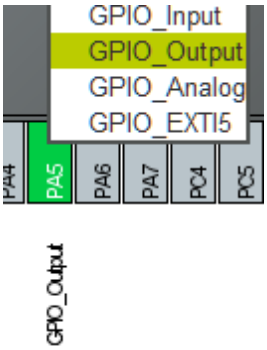
se čítací registr časovače inkrementuje s kmitočtem 1 MHz. Nastavením ***Prescaler*** registru na hodnotu 999 zajistíme, že se čítací registr inkrementuje pouze jednou za tisíc period hodinového signálu – k inkrementaci tedy dochází s kmitočtem 1 kHz. Změnou ***AutoReload*** registru můžeme nastavit, při jaké hodnotě čítacího registru dojde k přetečení časovače a vyvolání přerušení. Aby k přetečení došlo vždy po tisíci inkrementacích, nastavíme tento registr na hodnotu 999. S takovým nastavením bude docházet k přetečení s kmitočtem 1 Hz – jednou za sekundu. V této kartě již není třeba nic dalšího měnit.



Poslední věc, kterou je třeba u časovače nastavit, je povolení přerušení při přetečení časovače. To uděláme opět ve spodní části nabídky v kartě ***NVIC Settings***, kde zaškrtneme pole povolující globální přerušení časovače ***TIM2***.



Abychom mohli blikat LED na ***Nucleo*** desce, je stejně jako v předcházejících příkladech nutné nastavit pin ***PA5*** jako výstupní.



Nyní můžeme vygenerovat kód.

7.2.1.3 Tvorba programu

Po vygenerování kódu otevřeme soubor s hlavním programem, kde do místa pro uživatelský kód před nekonečnou smyčkou vložíme pouhý jeden příkaz, funkci z ***HAL*** knihovny, který spustí časovač v režimu s přerušením.

```
 99   /* Infinite loop */
100   /* USER CODE BEGIN WHILE */
101
102   // spusti casovac 2 v rezimu s prerusenim
103   HAL_TIM_Base_Start_IT(&htim2);
104
105   while (1)
106   {
107     /* USER CODE END WHILE */
108
109     /* USER CODE BEGIN 3 */
110   }
111   /* USER CODE END 3 */
112 }
```

To je vše, co je potřeba přidat do hlavního programu, a proto se můžeme přesunout do souboru obsahující obsluhy přerušení, jak tomu bylo v kapitole o přerušeních.

V tomto souboru najdeme obsluhu přerušení časovače. Vložíme do ní pouze jeden příkaz, kterým zajistíme změnu výstupní úrovně pinu, k němuž je připojena LED.

stm32l0xx_it.c

```
144   */
145 void TIM2_IRQHandler(void)
146 {
147   /* USER CODE BEGIN TIM2_IRQn 0 */
148
149   /* USER CODE END TIM2_IRQn 0 */
150   HAL_TIM_IRQHandler(&htim2);
151   /* USER CODE BEGIN TIM2_IRQn 1 */
152
153   // prepne vystupni uroven LED
154   HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
155
156   /* USER CODE END TIM2_IRQn 1 */
157 }
```

Po kompilaci projektu a nahrání do kontroleru uvidíme LED blikající s každou uběhlou sekundou.

Do obsluhy přerušení můžeme vložit ***breakpoint***, který běh programu každou sekundu pozastaví.

Na následujícím obrázku je vidět, že doba svitu je skutečně jedna sekunda. Nepřesnost je dána tolerancí vnitřního oscilátoru.