STM32 – Programování

Petr Novák (novakpe@fel.cvut.cz) / 12.02.2019

Klíčové slovo: volatile

Pokud se nějaká proměnná nastavuje například v přerušení a poté testuje v hlavní smyčce, tak je potřeba před její definici naspat klíčové slovo „volatile“, které zajistí její uložení / vyzvednutí z RAM při každém jejím nastavení / použití. Jinak při zapnutí nějakého vyššího stupně optimalizace (režim release) může vzniknout situace, kde bude hodnota této proměnné vyzvednuta a vložena do registru a tam bude neustále dokola testována tedy její poslední vyzvednutá hodnota z RAM a tudíž její nově zapsaná hodnota do RAM například z přerušení již nebude vyzvednuta. Program se v podstatě „kousne“ při čekání na požadovanou hodnotu, která však může být již v RAM z přerušení zapsána, ale nebude nikdy vyzvednuta do testování. Registr bude obsahovat stále původní vyzvednutou hodnotu z RAM ještě před její změnou z přerušení. Pozor na tuto skutečnost.

uint8_t / uint16_t / uint32_t

Při definování proměnných nemá až tak velký význam minimalizovat obsazení RAM pokud je RAM dostatečně velká. Pokud je požadavek na skutečně pouze 8bit hodnotu (například pro využití přetečení bitů), tak je tedy vhodné použít „uint8_t“. Pokud se však použije název uvedený u „typedef enum“ (popsáno dále), tak je vždy automaticky použito uint32_t, protože ARM je nativně 32bitový (některé operace mohou být mnohem rychlejší s 32bit hodnotou než 8bit hodnotou).

Podmíněný překlad

Pro podmíněný překlad používat (pokud jsou použity globálně, tak v „GlobalDef.h“):

```
// jde o verzi 2
#define VERZE2
```

Poté v hlavičkovém / zdrojovém souboru použít:

```
#if defined(VERZE1) || defined(VERZE2)
	// přeloží se pouze pro verzi 1 nebo 2
#else
	// přeloží se pro všechny ostatní verze
#endif
```

Lze použít i:

```
#if !defined  (VERZE1)
	// pro vše mimo verze 1
#endif
```

Konstanty / hodnoty, které lze nastavit / upravit

Zejména v hlavičkovém souboru (pokud jsou použity globálně, tak v „GlobalDef.h“) vytvořit definice pomocí:

```
#define HODNOTA číslo
#define HODNOTA_MOJE 5
```

Poté v hlavičkovém / zdrojovém souboru použít:

```
#if HODNOTA == 1
        // pro hodnotu 1
#else
        // pro všechny ostatní hodnoty
#endif

#if HODNOTA == HODNOTA_MOJE
        // …
#endif
```

Zajištění výchozí hodnoty

Pokud se ve zdrojovém souboru využívá nějaká definice / konstanta, která by měla být definována například v “GlobalDef.h“, ale může nabývat nějaké výchozí hodnoty i pokud není definována, tak lze použít:

```
// pokud není vůbec definována „COMMUN_TIME_TO_SEND_NEXT“
#if !defined  (COMMUN_TIME_TO_SEND_NEXT)
  // tak se definuje s výchozí hodnotou „100“
  #define COMMUN_TIME_TO_SEND_NEXT 100
#endif
```

Konstanty jako výčet / enum

Pokud se používají seznamy čísel, jako soubor konstant lze pro přehlednost použít:

```
typedef enum
{
   EnErrorNone = 0,
   EnErrorBattery = 1,
   EnErrorBLE = 10,
   EnErrorUpdate = 15,
} EnumErrors;
```

Potom se používá:

```
EnumErrors error = EnErrorNone;
If (error == EnErrorBattery) …
```

Pokud se proměnná zobrazí v „debug“ okně, tak se místo čísel zobrazují tyto přehlednější názvy. Samozřejmě lze použít i „uint32_t error = EnErrorNone“, tak se ale v „debug“ okně nezobrazujíc hodnoty pomocí textů, ale pouze čísel.

Využití bitových příznaků

Bitové příznaky je vhodné využít pro kompresy dat. Do jedné 32bit hodnoty lze takto vložit až 32 příznaků a to následovně:

```
typedef enum
{
   EnTestSideNone = 0,
   EnTestSideLeft = 1,
   EnTestSideRight = 2,
   // lze vytvořit i složený příznak
  EnTestSideBoth = EnTestSideLeft || EnTestSideRight,
} EnumTestSide;

// výchozí stav
EnumTestSide side = EnTestSideNone;
// nastavení příznaku
side |= EnTestSideLeft;
// nulování příznaku
side &= ~EnTestSideLeft;
// test příznaku
If ((side & EnTestSideLeft) != EnTestSideNone) …
```