SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Dokumentácia k Úlohám z cvičenia 5 z predmetu VRS

**Bratislava 2016 Bc. Matej Vargovčík**

# Vypracovanie úloh

## Prerušenie pre AD prevodník

Úlohou bolo prerobiť úlohu z cvičenia 4 tak, aby AD prevodník fungoval pomocou prerušenia a tým pádom neblokoval hlavný cyklus čakaním na spracovanie analógovej hodnoty. Konfigurácia AD prevodníka sa vykonáva štandardným spôsobom:

Spustenie GPIOA a ADC periférií:

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOA, *ENABLE*);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, *ENABLE*);

Nastavenie ADC prevodníka - tu bolo potrebné znížiť frekvenciu vzorkovania prevodníka, aby prerušenia úplne nevyblokovali celý procesor:

ADC\_InitTypeDef ADC\_InitStructure;

/\* Enable the HSI oscillator \*/

RCC\_HSICmd(*ENABLE*);

/\* Check that HSI oscillator is ready \*/

**while**(RCC\_GetFlagStatus(RCC\_FLAG\_HSIRDY) == *RESET*);

/\* Initialize ADC structure \*/

ADC\_StructInit(&ADC\_InitStructure);

/\* ADC1 configuration \*/

ADC\_InitStructure.ADC\_Resolution = ADC\_Resolution\_12b;

ADC\_InitStructure.ADC\_ContinuousConvMode = *ENABLE*;

ADC\_InitStructure.ADC\_ExternalTrigConvEdge = ADC\_ExternalTrigConvEdge\_None;

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;

ADC\_InitStructure.ADC\_NbrOfConversion = 1;

ADC\_Init(ADC1, &ADC\_InitStructure);

/\* ADCx regular channel8 configuration \*/

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_4, 1, ADC\_SampleTime\_384Cycles);

ADC\_Cmd(ADC1, *ENABLE*);

**while** (ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_ADONS) == *RESET*);

ADC\_SoftwareStartConv(ADC1);

ADC\_ITConfig(ADC1, ADC\_IT\_EOC, *ENABLE*);

Nastavenie GPIOA portu pre AD prevodník a LED diódu:

GPIO\_InitTypeDef GPIO\_InitStructure;

/\* Configure ADCx Channel 2 as analog input \*/

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_AN*;

GPIO\_InitStructure.GPIO\_Speed = *GPIO\_Speed\_40MHz*;

GPIO\_InitStructure.GPIO\_OType = *GPIO\_OType\_PP*;

GPIO\_InitStructure.GPIO\_PuPd = *GPIO\_PuPd\_NOPULL* ;

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_4;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_OUT*;

GPIO\_InitStructure.GPIO\_PuPd = *GPIO\_PuPd\_UP*;

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_5;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

Novým prvkom bola inicializácia prerušenia. Najprv bolo treba nastaviť NVIC na 16 preemption prerušení, čo sa vykonalo zaradením do NVIC\_PriorityGroup\_4:

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_4);

Potom nasledovalo samotné nastavenie prerušenia a jeho priorít:

NVIC\_InitTypeDef NVIC\_InitStructure;

NVIC\_InitStructure.NVIC\_IRQChannel = *ADC1\_IRQn*; //zoznam prerušení nájdete v súbore stm32l1xx.h

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 1;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = *ENABLE*;

NVIC\_Init(&NVIC\_InitStructure);

Keď je prerušenie nakonfigurované, môžeme pridať kód, ktorý bude vykonávať (prečítanie hodnoty AD prevodníka do globálnej premennej AD\_value):

**void** **ADC1\_IRQHandler**() {

**if** (ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC) == *SET*)

AD\_value = ADC1->DR;

}

Hlavný cyklus programu (všetko okrem čítania hodnoty z prevodníka) je taký istý ako v cvičení 4:

**int** delays[] = {2,4,9,16,25};

**int** mode = 4;

**int** lastButtonPressed = 4;

**while** (1)

{

GPIOA->ODR ^= (uint32\_t)(0b01<<5);

**for** (**int** i=0; i<delays[mode]; i++) {

// led blinking mode changing

**int** buttonPressed = getPressedButton();

**if** (lastButtonPressed != 4 && buttonPressed == 4) {

mode = lastButtonPressed;

}

lastButtonPressed = buttonPressed;

Delay(20);

}

}

**int** **getPressedButton**() {

**const** **int** buttonValues[] = {2500, 3000, 3550, 3800};

**int** buttonPressed = 0;

**for** (buttonPressed = 0; buttonPressed < 4; buttonPressed++) {

**if** (buttonValues[buttonPressed] > AD\_value)

**break**;

}

**return** buttonPressed;

}

## Sériová komunikácia

Úlohou bolo posielať hodnotu načítanú v AD prevodníku pomocou sériovej linky a zároveň čítať príkazy z linky. Konfigurácia UART komunikácie sa vykonáva nasledovne:

Poznámka: keďže procesor podporuje viacero UART rozhraní pre rôzne účely, používal som v tomto cvičení makrá pre číslo UARTu, a to USARTi namiesto USART1 alebo USART2. Pre rôzne UARTy sa používajú odlišné APB, preto číslo APB som označil písmenom j. USART2 používa APB1, USART1 používa APB2. V programe som používal USART2 vhodný pre sériovú komunikáciu cez ST Link, teda i = 2, j = 1.

Spustenie UART periféria:

RCC\_APBjPeriphClockCmd(RCC\_APBjPeriph\_USARTi, *ENABLE*);

Nastavenie GPIOA pre UART (UART2 má TX na PA\_2 a RX na PA\_3):

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_AF*;

GPIO\_InitStructure.GPIO\_Speed = *GPIO\_Speed\_40MHz*;

GPIO\_InitStructure.GPIO\_OType = *GPIO\_OType\_PP*;

GPIO\_InitStructure.GPIO\_PuPd = *GPIO\_PuPd\_NOPULL*;

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_2 | GPIO\_Pin\_3;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource2, GPIO\_AF\_USARTi);

GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource3, GPIO\_AF\_USARTi);

Inicializácia sériovej komunikácie pre čítanie (Rx) a zápis (Tx):

USART\_InitTypeDef USART\_InitStructure;

USART\_InitStructure.USART\_BaudRate = 9600;

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;

USART\_Init(USARTi, &USART\_InitStructure);

USART\_ITConfig(USARTi, USART\_IT\_RXNE, *ENABLE*);

USART\_ITConfig(USARTi, USART\_IT\_TC, *ENABLE*);

USART\_Cmd(USARTi, *ENABLE*);

Inicializácia prerušenia (s menšou preemp prioritou ako ADC, tj. s vyššou hodnotou):

NVIC\_InitStructure.NVIC\_IRQChannel = USARTi\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 2;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = *ENABLE*;

NVIC\_Init(&NVIC\_InitStructure);

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_4);

Kód prerušenia pre prijímanie znaku je podobne jednoduchý ako čítanie ADC hodnoty:

**void** **USART2\_IRQHandler**() {

**if** (USART\_GetFlagStatus(USARTi, USART\_FLAG\_RXNE) == *SET*) {

USART\_input\_value = USART\_ReceiveData(USARTi);

USART\_ClearFlag(USARTi, USART\_FLAG\_RXNE);

}

Keďže som chcel, aby odosielanie bolo sofistikovanejšie a odosielalo celé reťazce bez akejkoľvek závislosti od hlavného cyklu, navrhol som program tak, že odosielaciemu sa pošle pointer na string, ktorý treba odoslať. Keď dôjde k prerušeniu, modul si zapamätá tento pointer ako odosielaný reťazec a začne ho po znakoch odosielať. Keď je odoslaný, modul zisťuje, či neprišiel nový reťazec na odoslanie a ak áno, tak ho začne odosielať. Tým sa docieli, že reťazec je vždy odoslaný celý a zároveň sa tým predchádza zahlteniu modulu, ak nestíha posielať, pretože ak počas posielania pribudne viacero reťazcov na odosielanie, tak sa vždy odošle iba posledný z nich.

Pozn.: V prípade, že modul nemá čo odosielať, posiela nuly (ak by neposlal nič, k ďalšiemu prerušeniu by nedošlo).

Zvyšok vyššie začatej metódy prerušenia:

**else** **if** (USART\_GetFlagStatus(USARTi, USART\_FLAG\_TXE) == *SET*) {

**if** (!\*actualCharacterToSend) {

**if** (stringBeingSent != stringToSend) {

stringBeingSent = stringToSend;

actualCharacterToSend = stringBeingSent;

}

}

USART\_SendData(USARTi, \*actualCharacterToSend);

**if** (\*actualCharacterToSend)

actualCharacterToSend++;

USART\_ClearFlag(USARTi, USART\_FLAG\_TC);

}

}

Pozn.: Keďže riadenie pamäti v týchto jednoduchých procesoroch nie je také jednoduché ako napr. v osobnom počítači, neodporúča sa počas behu programu pamäť alokovať alebo dealokovať (ak to nie je nevyhnutné). Našťastie na odosielanie postačia pre pamäť najviac tri reťazce - odosielaný, určený na odoslanie a pripravovaný. Keďže sa v prerušení nevytvárajú žiadne nové reťazce (pracuje sa iba s tými, ktoré boli vytvorené v hlavnom cykle) a operácia priradenia pointera je (predpokladám) atomická, tento prístup je "thread-safe". V pamäti sú teda vytvorené tri reťazce str1, str2 a str3 a pri "alokácii" nového pripravovaného reťazca sa použije ten z nich, ktorý je voľný (teda nie je ani odosielaný, ani určený na odoslanie).

**char** str1[50] = "";

**char** str2[50] = "";

**char** str3[50] = "";

**char** \*stringToSend;

**char** \*stringBeingSent;

**char** \*actualCharacterToSend;

**void** **initStrings**() {

stringToSend = str1;

stringBeingSent = str1;

actualCharacterToSend = str1;

}

**char** \***allocateString**() {

**char** \*str = str1;

**if** (str == stringToSend || str == stringBeingSent)

str = str2;

**if** (str == stringToSend || str == stringBeingSent)

str = str3;

**return** str;

}

Do hlavného cyklu programu bolo oproti úlohe 1 doplnené spracovanie prečítaného príkazu zo sériovej linky (príkazom 'm' sa prepína formát odosielania (volty / čísla)) a príprava nového reťazca na odosielanie:

// AD value displaying mode changing

**if** (USART\_input\_value == 'm') {

USART\_input\_value = 0;

displayMode = (displayMode+1)%2;

}

**char** \*str = allocateString();

fillWithDisplayedValue(str, displayMode);

stringToSend = str;

**void** **fillWithDisplayedValue**(**char** \*str, **int** displayMode) {

**if** (displayMode == 0) {

**float** value = AD\_value/(**float**)(4095)\*3.3f;

sprintf(str, "\r\n%d.%dV", (**int**)value, ((**int**)(value\*100))%100);

}

**else**

sprintf(str, "\r\n%d", (**int**)AD\_value);

}

Nastavenie sériového portu:

