VII UART w ADC

Contreras R Orlando[1,2*] and Garcia B Iker[1,2*†]

[3]Departamento de Sistemas Electrónicos, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al307630}@edu.uaa.mx;

†Ambos autores contribuyeron de forma equitativa a este trabajo.

Abstract

El propósito de esta práctica es implementar la lectura de un ADC para poder mostrarlo en el monitor serial mientras que funciona para encender un LED.

Keywords: ASM, ARM, Shift, Registros, UART, Punto Flotante, ADC, PWM

1 Introducción

Esta práctica se aprovecha del punto fijo para hacer la conversión en lugar de la tradicional regla de 3 a la hora de hacer conversiones. Asimismo esta práctica la mayor parte del código son configuraciones de los periféricos, por lo que el ciclo principal se ejecutan unas cuantas instrucciones de igual forma se mencionara el algoritmo en el pseudo código

2 Objetivos

- Configurar correctamente el ADC y PMW
- Hacer la conversión para el valor del ADC hacia PWM
- Imprimir el valor en el puerto serie

3 Pseudocódigo y Explicación del Código

A continuación, se presenta el pseudocódigo:

Función *dividiridigitos*:

```
    // Divide e imprime las decenas y unidades en UART
1   PUSH {LR} Cargar 10 en R10 UDIV R1, R2, R10 ;          // Decenas en R1
2   Llamar a Write_UART ;                                  // Imprimir decena
3   MUL R3, R1, R10 ;                                     // R3 = decenas * 10
4   SUB R1, R2, R3 ;                                       // Unidades en R1
5   Llamar a Write_UART ;                                  // Imprimir unidad
6   POP {PC}
```

Función *WriteUART*:

```
    // Envía un carácter por UART desde R1
7   PUSH {R2, LR} Cargar dirección de USART1_DR en R0 Comparar R1 con 10 Si
     R1 < 10 entonces llamar a sumar30 Escribir R1 en [R0] Cargar dirección de
     USART1_SR en R0
    // Esperar hasta que TXE esté listo
8   while bit 6 de [R0] no esté en 1 do
9       Leer [R0] en R2 Comprobar bit 6 con TST R2, #(1<<6)
10  POP {R2, PC}
```

GitHub Repository

Configuración del ADC (`Config_ADC`)

Esta rutina configura el módulo ADC para realizar conversiones de 12 bits (resolución por defecto), utilizando el canal 7 y un tiempo de muestreo de 15 ciclos. El proceso se realiza en los siguientes pasos:

1. **Encendido inicial del ADC:**
   - Se accede al registro `ADC_CR2`.
   - Se activa el bit `ADON` (bit 0) para encender el ADC por primera vez.
2. **Configuración del tiempo de muestreo:**
   - Se accede al registro `ADC_SMPR2`.
   - Se configura el campo correspondiente al canal 7 (bits [23:21]) con el valor `001`, equivalente a 15 ciclos de muestreo.
3. **Selección del canal de conversión:**
   - Se modifica el registro `ADC_SQR3`.
   - Se coloca el valor `7` en los bits [4:0], seleccionando el canal 7 como el primer (y único) canal de la secuencia de conversión.
4. **Activación de la conversión continua:**
   - Nuevamente se accede al registro `ADC_CR2`.
   - Se cargan los bits necesarios para:
     - `ADON = 1` (activar el ADC nuevamente),
     - `CONT = 1` (modo de conversión continua),
     - `SWSTART = 1` (opcional, iniciar conversión por software – línea comentada).

- Se utiliza el valor `0x40000003` para establecer los bits mencionados.

Configuración del UART (`Config_UART`)

Esta rutina inicializa el periférico UART1 para transmitir datos a una velocidad de 9600 baudios, utilizando una trama estándar con 1 bit de inicio, 8 bits de datos y 1 bit de parada.

1. **Configuración del Baud Rate:**
   - Se accede al registro `USART1_BRR`.
   - Se carga el valor `0x683`, correspondiente a 9600 baudios (asumiendo un reloj apropiado).
2. **Configuración de la trama y habilitación:**
   - Se accede al registro `USART1_CR1`.
   - Se cargan los bits:
     - `UE = 1` (bit 13): habilita el UART,
     - `TE = 1` y `RE = 1` (bits 3 y 2): habilita transmisión y recepción.
   - Se usa el valor `0x200C` para activar esta configuración de manera compacta.

Este es el código original en ensamblador que corresponde al pseudocódigo anterior:

```
PWMFULL         EQU 1600
        AREA myData, DATA, READWRITE

    AREA juve3dstudio,CODE,READONLY
        ENTRY
        EXPORT __main

__main
    BL      Config_RCC
    BL      Config_GPIO
        BL                  Config_TIM
    BL      Config_UART

loop
        ;leyendo el ADC
    LDR     R0, =ADC_DR
    LDR     R1, [R0]

    LDR     R5, =1599
    LDR     R2, =PWMFULL
    MUL     R2,R2,R1 ; 1600*2^12
    LSR     R2, #12; 1600
    CMP     R2,R5
    BEQ     es100

    LDR     R0, =TIM2_CCR1
    STR     R2, [R0]

    LSR     R2, #4
    MOV     R1,# 'P'
    BL      Write_UART
    MOV     R1,# 'W'
    BL      Write_UART
    MOV     R1,# 'M'
    BL      Write_UART
    MOV     R1,# ':'
    BL      Write_UART
    MOV     R1,# ' '
    BL      Write_UART
        BL      dividiridigitos

;    BL      Read_UART
fue100

    LDR     R1, =0x25
    BL      Write_UART
    LDR     R1, =0x0D
    BL      Write_UART
    LDR     R1, =0x0A
    BL      Write_UART

        B                   loop
es100
       MOV     R1,# 'P'
    BL      Write_UART
    MOV     R1,# 'W'
    BL      Write_UART
    MOV     R1,# 'M'
    BL      Write_UART
    MOV     R1,# ':'
    BL      Write_UART
    MOV     R1,# ' '
    BL      Write_UART

    LDR     R1,=0x01
    BL      Write_UART
    EOR     R1,R1
    BL      Write_UART
    BL      Write_UART

    b       fue100


dividiridigitos
    ; r2 Val

    PUSH {LR}
```

```
    ; Divide by 10 to get tens 74
    MOV     R10, #10

    UDIV    R1, R2, R10

    bl      Write_UART

    UDIV    R1, R2, R10
    MUL     R3, R1, R10
    SUB     R1, R2,R3
    bl      Write_UART

    POP {PC}



Write_UART
    PUSH{R2,LR}
    LDR    R0, =USART1_DR
    CMP    R1, #10
    BLLT    sumar30
    STR    R1, [R0]
    LDR    R0, =USART1_SR

writeCycle
    LDR   R2, [R0] ; Read status register
    TST   R2, #(1<<6) ; Check if TXE is set
    BEQ   writeCycle ; If not, wait

    POP{R2,PC}

sumar30
    PUSH{LR}
    ADD    R1,#0x30
    POP{PC}
```

```
Read_UART
    PUSH{LR}
    LDR    R0, =USART1_SR
readCycle
    LDR   R1, [R0] ; Read status register
    TST   R1, #(1<<5) ; Check if RXNE is set
    BEQ   readCycle ; If not, wait
    LDR   R0, =USART1_DR ; Read data register
    LDR   R1, [R0] ; Read received data

    POP{PC}
    Config_RCC
    LDR     R0,=RCC_AHB1      ; 0    00            0     0
    LDR     R1,[R0]           ;GPIOH RESERVE   GPIOE  GPIOD
    ORR     R1,#(0x3)        ; 0    00            0    0
    STR     R1,[R0]

        ; Activamos el CLK del TIM2 0x4000 0000

        ; TIM5  TIM4  TIM3  TIM2
        ; 0      0     0      1
    LDR     R0,=RCC_APB1
    LDR     R1,[R0]
    ORR     R1,#0x01
    STR     R1,[R0]

    ; Activamos el CLK del ADC 0x4001 2000
    LDR     R0,=RCC_APB2
    LDR     R1,[R0]
    ORR     R1,#(1<<8)|(1<<4) ; ADC1 y TIM2
    STR     R1,[R0]

    BX      LR
```

4 Conclusiones

La práctica nos pareció relativamente sencilla e interesante, esta práctica nos puede servir de base para leer sensores e imprimirlos en el monitor serial, inclusive podríamos usar ese mismo protocolo para un módulo Bluetooth abriéndonos las puertas a transmitir datos de una forma más rápida, efectiva.