ARQUITECTURA DE COMPUTADORAS (E1213)
CIRCUITOS DIGITALES Y MICROPROCESADORES (E0213)
Curso 2022

ACTIVIDADES PRÁCTICAS PARA EL MÓDULO 3
La arquitectura AVR programada en alto nivel

Entrega obligatoria del Proyecto Final en Moodle

Objetivos del Módulo

Programación de microcontroladores en lenguajes de alto nivel (sin perder de vista lo que sucede en bajo nivel). La arquitectura AVR programada en C utilizando la biblioteca estándar en el entorno Atmel Studio. Implementación de proyectos utilizando el kit de desarrollo, incluyendo módulos y periféricos avanzados. Gestión de proyectos y la utilización de bibliotecas estándar y de terceros. La importancia de codificar en forma comprensible y reutilizable.

Requisitos

Conocimientos básicos de la biblioteca estándar de C (libc). Los estándar ANSI y POSIX. Comprensión de los conceptos y correcta utilización de la terminología en castellano y en inglés: rutinas, bibliotecas, archivos cabecera (incluyen prototipos, tipo de datos y macros); entrada y salida estándar; el compilador, linker y depurador; portabilidad.
Tener en cuenta que en este módulo se utilizarán conceptos introducidos en la materia Programación (E0201/E1201). Si bien no se propone una guía para estos temas, se recomienda refrescar los contenidos de dicha asignatura.
¿Cuáles son los puntos importantes de cualquier lenguaje de alto nivel? a) sintaxis, b) palabras reservadas y estructura de un programa (el "vocabulario"), c) tipos de datos y operadores, d) funciones.
Buenas prácticas para la escritura de código. Recuerde que un lenguaje de programación de alto nivel por sí mismo no garantiza que el código sea comprensible y reutilizable. Para ello es necesario realizar un esfuerzo adicional de diseño y documentación, lo cual será evaluado en este módulo.

Entrando en calor

a) En la práctica anterior se realizó el compilado "a mano" de algunos códigos en C simples, incluyendo las estructuras elementales (repetición, bifurcación y llamado a subrutina). Verificar los resultados utilizando Atmel Studio. Comparar lo que haría usted con lo que hace efectivamente el compilador. Desensamblar algunos resultados. Ver el Anexo A.
b) Analice el siguiente problema. Se desea implementar un buffer circular[1] en memoria para almacenar las últimas N conversiones del ADC. ¿Cómo lo haría en lenguaje ensamblador y cómo lo haría en C? ¿Cómo puede garantizar que el compilador entendió lo que Ud. quiso decir? Si N fuera menor que 16 ¿el buffer se podría implementar con registros? ¿Se obtendría alguna ventaja?
c) La organización de un proyecto en C para un microcontrolador. Compruebe la estructura de un proyecto en Atmel Studio. Verifique cómo incorporar bibliotecas de funciones en los proyectos. ¿Cuáles son las que provee Atmel Studio? ¿Qué otras fuentes de bibliotecas existen? ¿Cómo se escribe una biblioteca propia de funciones? Compruebe el procedimiento para incorporar una sección en lenguaje ensamblador en un proyecto en C. Ver el tutorial disponible en Moodle sobre cómo iniciar correctamente un proyecto en C en Atmel Studio y cómo redireccionar la salida estándar a una consola serie.

[1] https://es.wikipedia.org/wiki/Buffer_circular

d) Teniendo en cuenta que el repertorio de instrucciones de AVR no contempla la representación en punto flotante, no incluye la operación de división y no tiene definida por defecto una salida estándar, considere la implementación del siguiente código. ¿Podría estimar la cantidad de instrucciones AVR que resultarían?

```
#include <stdio.h>

int main(void) {
   float pi = 3.141592;

   pi = pi / 3.0;
   printf("pi/3 = %f\n", pi);
}
```

e) Verificar las opciones de optimización disponibles en el compilador de Atmel Studio. Compruebe el efecto que tienen sobre el código generado, inspeccionando el reporte de salida del compilador.

Problema 1: El algoritmo de ordenamiento

Programar en C el algoritmo de ordenamiento de vectores implementado en las prácticas 1 y 2. A esta altura Ud. ya dispone de tres soluciones al mismo problema. Repetir las medidas y comparar con los resultados anteriores. Conclusiones.

Problema 2: Un proyecto en alto nivel utilizando módulos periféricos

Utilizando un timer del microcontrolador generar una interrupción periódica cada 1 segundo. La rutina de interrupción debe realizar la lectura del canal del ADC que tiene conectado el sensor interno de temperatura. Una vez finalizada la lectura imprimir el resultado formateado en grados centígrados en la salida estándar (redireccionada al puerto serie) y poner el controlador en modo de bajo consumo hasta la próxima interrupción.
Sobre cómo leer el sensor de temperatura interno, ver el proyecto: https://microchipdeveloper.com/8avr:avradc.
Sobre cómo manejar interrupciones periódicas y modos de bajo consumo, ver: https://microchipdeveloper.com/8avr:low-power-example.

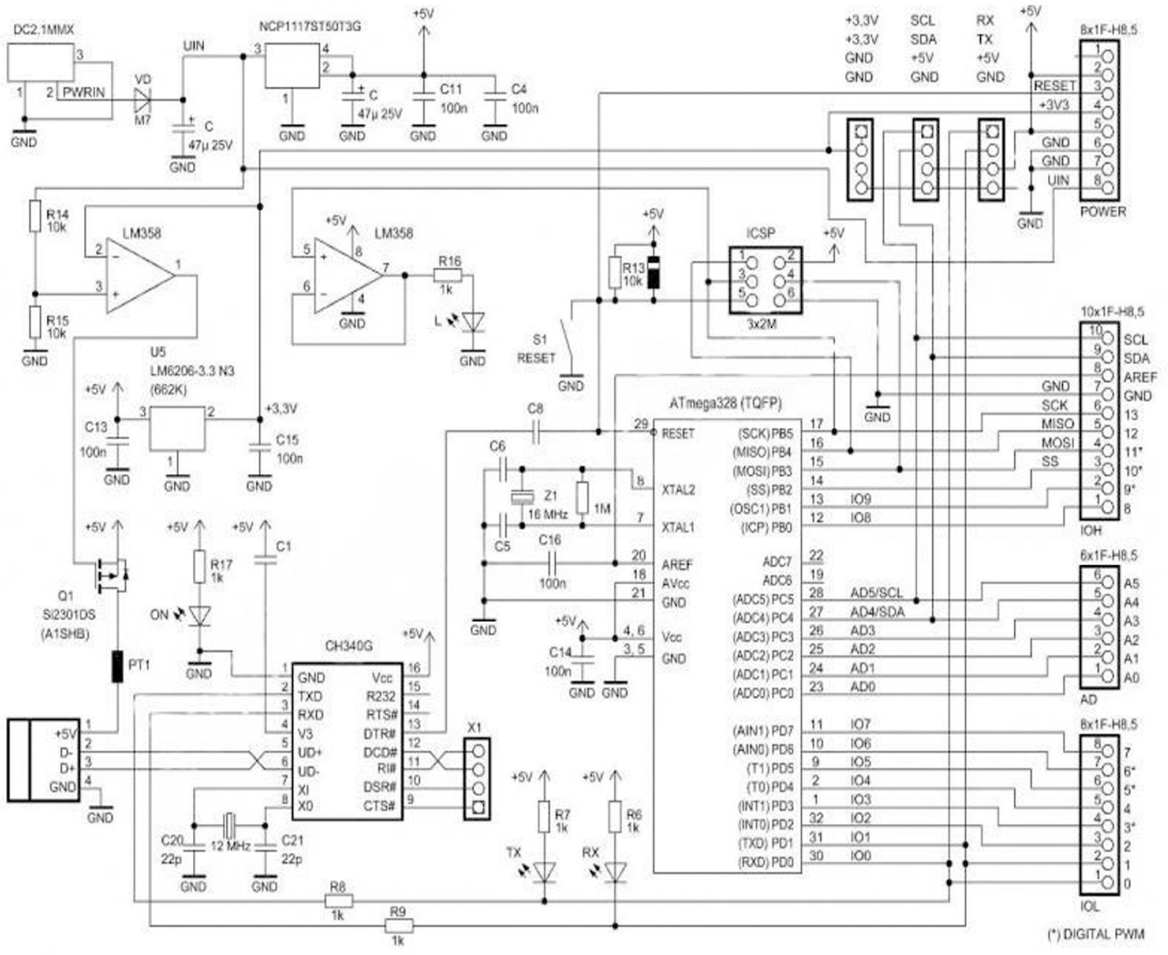
Proyecto Final

Los alumnos deberán llevar a cabo un proyecto propio, optando por una de las siguientes opciones:
Opción 1: AVR 100% virtual. Simulación del kit y los periféricos en Proteus. Esta opción se reserva para quienes, por algún motivo particular, no hayan tenido acceso a un kit físico durante la cursada.
Opción 2: Kit AVR con periféricos mínimos. Consola serie, leds, switches y resistencias. Dada la escasez de recursos hay que ser creativo con la propuesta. Involucra simulaciones e implementación física.
Opción 3: Kit AVR con periféricos avanzados. Sensores y actuadores externos. Involucra simulaciones e implementación física.
En general todas las opciones son de realización y entrega individual. Si el grado de complejidad del proyecto lo amerita, puede ser realizado en grupos de hasta 2 alumnos. Todas las propuestas deben ser aprobadas por los ayudantes previo a su implementación.
El proyecto se completa con una breve presentación y la entrega de un informe escrito. El informe deberá incluir una descripción precisa del problema, una justificación de la estrategia utilizada para la resolución y un detalle de la implementación. Deberán adjuntarse los resultados y mediciones que se consideren necesarios para mostrar el correcto funcionamiento y la performance del conjunto.

Importante

Antes de conectar periféricos al kit de desarrollo se recomienda analizar en detalle el circuito esquemático del kit. Asegúrese de comprender todas las partes, principalmente la estrategia de alimentaciones de 3.3V, 5V y externa. Verifique el circuito de clock y de reset. Calcule la máxima corriente que puede entregar cada port y la máxima corriente total. Verifique los niveles lógicos. Considere la posible reutilización de los leds RX y TX. Analice la mejor estrategia para agregar pulsadores y leds adicionales.

Compruebe el funcionamiento y compatibilidad del circuito integrado CH340G. Ver https://www.prometec.net/ch340g/. Asegúrese de comprender la diferencia entre la grabación por medio del bootloader y del ICSP.



Protocolo obligatorio de entrega

*a) Nombrar los archivos con el formato **NNNNN_P_E**, donde NNNNN es el número de alumno, P es el número de práctica (en este caso 3) y E es el número de ejercicio (para el proyecto utilizar 0).*

*b) El proyecto debe acompañarse con un informe técnico nombrado **NNNNN_3_0.pdf**.*

*c) Juntar todos los archivos en un único comprimido llamado **NNNNN_3.zip** y subirlo en el formulario en Moodle. Verificar fecha límite de entrega.*

No utilice directorios dentro del archivo comprimido.
No incluya archivos que no respeten el formato.

Anexo A: Un último ejercicio resuelto en bajo nivel

Parte 1

Analizar un programa mínimo que conmuta un port de salida a máxima velocidad. En assembler, en C y en Processing (Arduino IDE). Inspeccionar los .hex generados. Desensamblar y asegurarse de entender lo que hacen el ensamblador y el compilador.

ODA: https://onlinedisassembler.com/odaweb/
HEX: https://es.wikipedia.org/wiki/HEX_(Intel)

Si se observa el port de salida con un osciloscopio, la forma de onda no es cuadrada. ¿Por qué? ¿Cuál sería la máxima frecuencia que puede obtenerse con AVR funcionando a máxima velocidad?

Atmel Studio en Assembler	`sbi DDRB,5`
	
	`LOOP:`
	`  cbi PORTB,5`
	`  sbi PORTB,5`
	`  rjmp LOOP`
		Inspeccionando la salida HEX:
		
		`:02000002 0000FC`
		`:08000000 259A2D982D9AFDCFE1`
		`:00000001FF`
		
		El archivo .hex tiene 59 bytes = 8 bytes de instrucciones
		
		En GUI Atmel Studio también se verifica, en la dirección de programa 0x0000:
		25 9a 2d 98 2d 9a fd cf ff ff ff ff
		
		259A 0010 0101 1001 1010
		2D98 0010 1101 1001 1000
		2D9A 0010 1101 1001 1010
		FDCF 1111 1101 1100 1111
		
		Desensamblado con ODA
		`.data:00000000 25 9a  sbi  0x04, 5`
		`.data:00000002 2d 98  cbi  0x05, 5`
		`.data:00000004 2d 9a  sbi  0x05, 5`
		`.data:00000006 fd cf  rjmp .-6`
		`  ;  0x00000002`

Atmel Studio en C	`int main(void) {`
	`  DDRB |= 0x20;`
	`  while(1) {`
	`  PORTB |= 0x20;`
	`  PORTB &= 0xDF;`
	`  }`
	`}`
		0C9434000C943E000C943E000C943E00
		0C943E000C943E000C943E000C943E00
		0C943E000C943E000C943E000C943E00
		0C943E000C943E000C943E000C943E00
		0C943E000C943E000C943E000C943E00
		0C943E000C943E000C943E000C943E00
		0C943E000C943E0011241FBECFEFD8E0
		DEBFCDBF0E9440000C9444000C940000
		259A2D9A2D98FDCFF894FFCF
		
		El archivo .hex tiene 410 bytes
		
		En el HEX de salida están las 4 inst (BIEN!). Además están inicializados todos los vectores de interrupción y la pila. Ver en detalle

Arduino IDE	`void setup() {`
	`  pinMode(LED_BUILTIN, OUTPUT);`
	`}`
	
	`void loop() {`
	`  digitalWrite(LED_BUILTIN, HIGH);`
	`  digitalWrite(LED_BUILTIN, LOW);`
	`}`
		Salida del compilador: El Sketch usa 734 bytes (2%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
		Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 2039 bytes para las variables locales. El máximo es 2048 bytes.
		
		El archivo .hex tiene 2079 bytes

Interesante: Why I'm switching over from the awesome Arduino IDE to Atmel Studio
https://www.youtube.com/watch?v=648Tx5N9Zoc

Parte 2

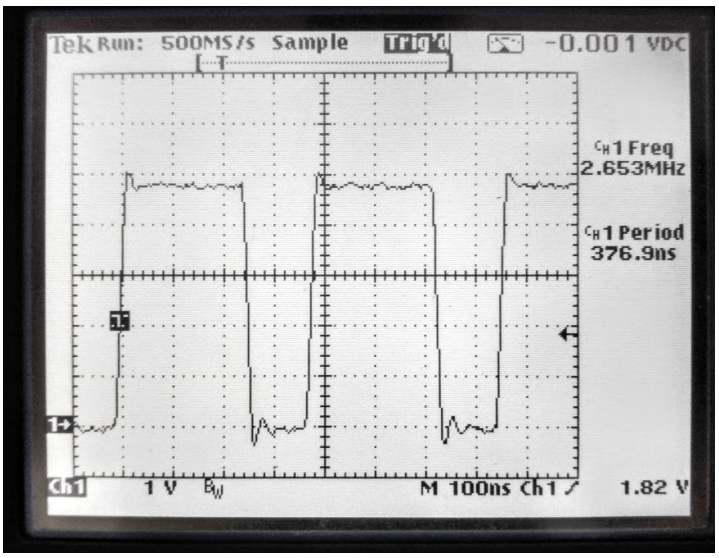
Extendiendo el ejemplo anterior, suponga que desea generar una onda **cuadrada** de máxima frecuencia, encendiendo y apagando el bit 0 del PortB, sin alterar el resto del estado del port. Compare el programa resultante si se utilizan instrucciones que permiten direccionamiento de bits (sbi y cbi) frente a la opción de no utilizarlas, tratando a los ports como bytes. Si el procesador opera a 16 MHz, ¿cuál es la máxima frecuencia de la onda cuadrada en cada caso? ¿Es realmente cuadrada?

T=1/16MHz=62.5ns

// Byte addressing
 	
in r16,DDRB
ori r16, 0x01;
out DDRB,r16
in r16,PORTB
 	
LOOP:
 	
ori r16, 0x01
out PORTB,r16
nop
nop
andi r16, 0xFE
out PORTB,r16
 	
rjmp LOOP
	// Bit addressing
	
	sbi DDRB,0
	
	LOOP:
	
	cbi PORTB,0
	nop
	nop
	sbi PORTB,0
	
	rjmp LOOP
		// Byte addressing 2
		
		in r16,DDRB
		ori r16,0x01;
		out DDRB,r16
		in r16,PORTB
		ldi r17,0x01
		
		LOOP:
		
		eor r16,r17
		out PORTB,r16
		
		rjmp LOOP

04 b1 01 60 04 b9 05 b1 01 60 05 b9 0e 7f 05 b9 fb cf	20 9a 28 98 28 9a fd cf	04 b1 01 60 04 b9 05 b1 11 e0 01 27 05 b9 fd cf
Las 4 inst 1c, salto 2c 011000 011000 Ton=2 Toff=4 Fmax=1/6T=2.66MHz Agrego dos nop Fc=1/8T=2MHz	Las 3 inst toman 2c 001111 001111 Ton=4 Toff=2 Fmax=1/6T=2.66MHz Agrego dos nop de 1c Fc=1/8T=2MHz	Las 2 inst 1c, salto 2c 100 011 100 011 Ton=4 Toff=4 Fmax=1/8T=2MHz Ya es cuadrada Fc=1/8T=2MHz

Si la captura de osciloscopio de la figura corresponde al programa de la segunda columna, ¿con qué frecuencia de reloj se está ejecutando?



Ahora veamos qué hacen los compiladores de lenguajes de alto nivel.
Verifique el funcionamiento del siguiente programa en C utilizando Atmel Studio

```
int main(void) {

  DDRB |= 0x01;

  while(1) {
       PORTB ^= 0x01;
  }

}
```

```
in r16,DDRB
ori r16,0x01;
out DDRB,r16

LOOP:
  ldi r17,0x01
  in r16,PORTB
  eor r16,r17
  out PORTB,r16
  rjmp LOOP
```

Midiendo con el osciloscopio resulta una cuadrada de 1.59 MHz.

Pruebe el siguiente sketch en Arduino IDE:

```
void setup() {
  DDRB = DDRB | 0x01;
}

void loop() {
  PORTB = PORTB ^ 0x01;
}
```

```
in r16,DDRB
ori r16,0x01;
out DDRB,r16

LOOP:
  ldi r17,0x01
  in r16,PORTB
  eor r16,r17
  out PORTB,r16
  jmp LOOP
```

Midiendo con el osciloscopio resulta una cuadrada de 1.137 MHz. Usa jmp en vez de rjmp! Y lee dentro del loop (ver equivalente en la columna derecha).

Si quisiera generar 4 ondas cuadradas desfasadas 90°, ¿es conveniente utilizar direccionamiento por bits? ¿cuál sería la máxima frecuencia posible? Reflexionar sobre las limitaciones de un programa secuencial frente a un diseño discreto o en FPGA de tecnología equivalente.

Anexo B: Herramientas de alto nivel

Subiendo un nivel más de abstracción (con cuidado)

El proyecto Arduino: https://www.arduino.cc/
Arduino programming language (basado en Wiring):
https://www.arduino.cc/reference/en/
"Wiring is an open-source programming framework for microcontrollers"
http://wiring.org.co/
Arduino Software (IDE), basado en Processing
https://www.arduino.cc/en/Main/Software
"Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts" https://processing.org/

Sistemas operativos de tiempo real (RTOS)

Mejor llamados "Embedded Programming Platforms". Un tema interesante para los que quieren seguir investigando en la capa de aplicación. Más en la materia optativa "Sistemas Embebidos". Cómo utilizar, en un microcontrolador, procesos, señales, semáforos, etc.
FreeRTOS https://www.freertos.org/
Simba https://simba-os.readthedocs.io/en/latest/index.html
Trampoline https://github.com/TrampolineRTOS/trampoline
DuinOS https://github.com/DuinOS/DuinOS

Beneficios de programar en el entorno de un RTOS:

- ➔ Threads scheduled by a priority based cooperative or preemptive scheduler
- ➔ Channels for inter-thread communication (Queue, Event)
- ➔ Timers
- ➔ Counting semaphores
- ➔ Device drivers (SPI, UART, …)
- ➔ A simple shell
- ➔ Logging
- ➔ Internet protocols (TCP, UDP, HTTP, …)
- ➔ File systems (FAT16, SPIFFS)

Desarrollando para AVR en CodeBlocks

La arquitectura AVR es muy popular. Es soportada por varios entornos integrados de desarrollo (IDE), como por ejemplo los populares Eclipse y CodeBlocks:
https://www.avrfreaks.net/sites/default/files/Howto%20Code_Blocks%20and%20AVR1_3.pdf