Sistemas Embebidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2017

Laboratorio N^{o} 2
E/S y Dispositivos Periféricos Integrados

Fecha límite de presentación: 25/09/2017. 17:00hs
Fecha de evaluación: 29/09/2017.

Objetivo: Los objetivos del laboratorio son:

- Manejo de E/S digital mediante la utilización de un display LCD [6, 7].
- Uso de interrupciones y temporizadores.
- Adquisición de datos utilizando el conversor analógico/digital (ADC) del ATmega328P.
- Implementación de drivers.

Desarrollo: El laboratorio deberá realizarse en comisiones de no más de 2 alumnos.

Actividad 1: Librería LyquidCrystal, LCD1602 Keypad Shield y PWM

1. Abrir el ejemplo "LCD1602KeyShieldTest" [4] provisto por la cátedra. Examinar el código del ejemplo y familiarizarse con el mismo. Prestar atención al manejo de los pulsadores realizados mediante un único canal de ADC, y al uso de la función `analogRead` [1]. ¿Qué ventaja presenta este esquema de conexión, en relación a conectar pulsadores de manera directa a los puertos de E/S?

2. Construir el proyecto, descargar la imagen al microcontrolador y probarlo.

3. Examinar la documentación de la función `analogWrite` [2] y determinar de qué manera es utilizada para controlar el brillo del backlight del display LCD. ¿Cómo es posible generar una señal analógica mediante un pin de salida digital?

4. Experimentar con diversos valores de brillo para el *backlight*,modificando el código para cada caso.

Actividad 2: Interrupciones externas y Timers

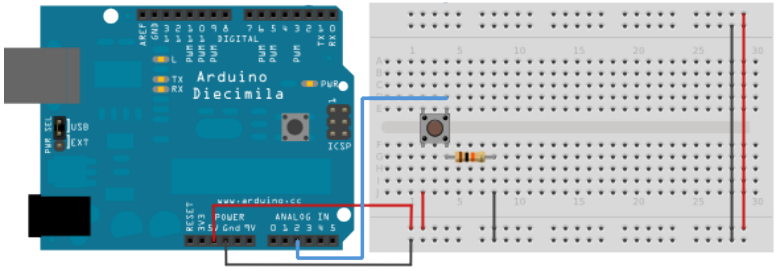
1. Analizar el tutorial [5] para utilizar interrupciones externas directamente mediante las librerías AVR, desde Arduino.

2. Analizar el capítulo *External Interrupts* del datasheet del ATmega328P [3].

3. Analizar el tutorial [8] para utilizar timers directamente mediante las librerías AVR, desde Arduino.

4. Desarrollar un programa que haciendo uso de las *Pin ChangeInterrupts* del ATmega328P, utilice un pulsador para implementar un cronómetro digital. Para el mismo, se deberá utilizar el circuito

Figura 1: Circuito Arduino y pulsador a A2



de conexionado del pulsador del laboratorio anterior, modificando la conexión del pulsador al pin 2 por el pin A2, como se muestra en la figura 1.

El cronómetro debe presentar, a través del *display* LCD 16x2 minutos, segundos y centésimas de segundo. Al accionar el pulsador, debe iniciar un conteo ascendente, el cual debe detenerse con una segunda presión del botón. Una tercera señal del pulsador debe resetear el cronómetro a cero.

La medición del tiempo debe realizarse mediante uno de los *timers* provistos por el microcontrolador, estableciendo una base temporal adecuada y realizando la configuración para utilizarlo mediante interrupciones.

El pin al que se conecta el pulsador debe ser configurado para generar una *Pin Change Interrupt*. Esta interrupción indica que el pulsador ha sido presionado, reemplazando a la lectura de puertos que se utilizó en el Laboratorio 1. La configuración debe realizarse utilizando la biblioteca de AVR.

Actividad 3: Adquisición de datos - Medición de luminosidad

1. Implementar el circuito indicado en la figura 2.

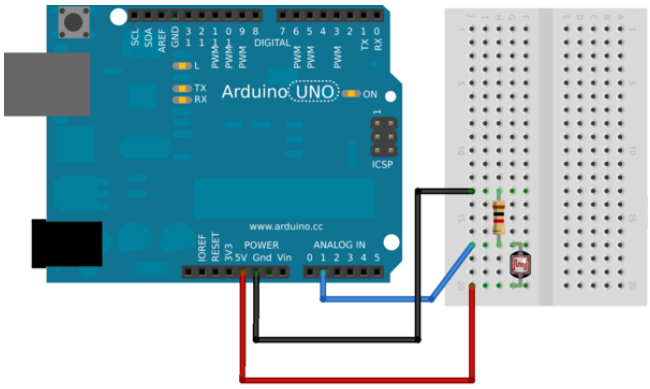


Figura 2: Circuito Arduino y resistencia LDR

2. En base al valor de voltaje entregado por el sensor y al voltaje de referencia utilizado por el ADC, determine la función para convertir cada valor digitalizado al valor correspondiente de luminosidad (en Lux).

   Para la conversión tenga en cuenta la estructura del circuito presentado en la figura 2, y la tabla 1 (la cual se deberá interpolar linealmente) que mapea valores de la resistencia R_1 del LDR a valores de luminosidad medidos en Lux:

Tabla 1: Relación entre luminosidad y resistencia en LDR

R_1	Lux
92	< 1
41	≈ 1
24	≈ 3
16	≈ 6
10	≈ 10
7	≈ 15
5	≈ 35
3	≈ 80
1	> 100

3. Escribir un programa que utilizando la función `analogRead`, tome la información provista por el ADC en el canal 1, la procese utilizando la ecuación desarrollada en el ítem 2, y muestre el valor de lux medido, a través del LCD montado en el *shield*.

4. Implementar un *driver* de ADC, que provea la siguiente interfaz:

   - `int adc_init(adc_cfg *cfg)`: Función que inicializa el *driver* de ADC y acepta como parámetro una estructura de tipo `adc_cfg`.

     La estructura `adc_cfg` tiene que tener un campo que selecciona el canal a configurar, y una función de callback que será invocada desde `adc_loop()` sólo cuando se haya completado la conversión de un valor. Se puede añadir cualquier otro campo que se considere necesario.

     Esta función inicializará el ADC del ATMega328P en el modo de operación más conveniente para la aplicación (ver [3]) y asociará el canal seleccionado con su callback cada vez que sea invocada, retornando 1 si la configuración resultase exitosa, o 0 en caso de producirse un fallo en la inicialización o configuración.
   - `void adc_loop()`: Esta función es llamada desde el bucle de ejecución del programa principal. Ejecuta las funciones de *callback* correspondientes, sólo cuando se hayan obtenido valores de conversión.

   El *driver* debe estar escrito en archivos independientes al programa principal, con la implementación escrita en un archivo ċ/ċpp, y su interfaz pública declarada en un archivo *header* (.h).

   Para obtener el resultado de la conversión del ADC, el *driver* debe valerse de la interrupción provista por el periférico.

5. Implementar nuevamente el inciso 3, reemplazando el uso de `analogRead` por el *driver* implementado en el inciso 4.

6. ¿Por qué motivo no es recomendable que la función de *callback* sea invocada desde el código de la interrupción?

Actividad 4: *Driver* de teclado

Se requiere implementar un *driver* no bloqueante de teclado, utilizando interrupciones y sin retardos por software que incluya las teclas del *LCD Keypad Shield* y la posibilidad de conectar pulsadores externos adicionales en los pines A2, A3, A4 y/o A5 de la placa Arduino Uno. Debe utilizar el *driver* de ADC implementado en la **Actividad 3**, y poseer la siguiente interfaz:

- `void key_down_callback(void (*handler)(), int tecla)`: Asocia la función `handler` al evento de presionar la tecla `tecla`.
- `void key_up_callback(void (*handler)(), int tecla)`: Asocia la función `handler` al evento de soltar la tecla `tecla`.

donde `tecla` puede tomar uno de los siguientes valores:

- `TECLA_UP = 0` *//botón up del LCD Keypad Shield*
- `TECLA_DOWN = 1` *//botón down del LCD Keypad Shield*
- `TECLA_LEFT = 2` *//botón left del LCD Keypad Shield*
- `TECLA_RIGHT = 3` *//botón right del LCD Keypad Shield*
- `TECLA_SELECT = 4` *//botón select del LCD Keypad Shield*
- `BOTON_A2 = 5` *//botón externo conectado al pin A2 de Arduino Uno*
- `BOTON_A3 = 6` *//botón externo conectado al pin A3 de Arduino Uno*
- `BOTON_A4 = 7` *//botón externo conectado al pin A4 de Arduino Uno*
- `BOTON_A5 = 8` *//botón externo conectado al pin A5 de Arduino Uno*

1. Analizar el tutorial [9] para utilizar interrupciones directamente mediante las librerías AVR, desde Arduino.
2. Identificar las tareas e ISRs del sistema. Especificar la lógica de cada tarea e ISR mediante diagramas de flujo.
3. Determinar en base a la información disponible en [3] y a los requerimientos del proyecto, el modo más conveniente para configurar el ADC. Justifique su elección.
4. Implementar el driver descripto anteriormente, el cual deberá hacer uso del driver de ADC implementado en la Actividad 3 con las modificaciones que sean necesarias para soportar mediciones en más de un canal, y de las *Pin Change Interrupts* del ATmega328P, según el tipo de conexionado de cada pulsador considerado. Se pueden seguir los pasos descriptos en [10].
5. Para completar el sistema, se debe implementar una aplicación que, utilizando el driver de teclado implementado muestre por el display LCD los eventos ocurridos sobre los pulsadores (i.e. si se trata de un evento de key_up o key_down, y sobre qué tecla).
6. Descargar la imagen modificada al microcontrolador y probar el sistema.
7. Analizar de qué manera se soluciona el problema de *debouncing* en el driver implementado.

Actividad 5: Contadores, Timers, PWM y E/S Digital

Implementar un sistema de medición de luminosidad. El sistema mantendrá registro de la luminosidad actual correspondiente al último valor obtenido mediante el sensor LDR), luminosidades máxima y mínima, y una luminosidad promedio calculado sobre las 200 muestras más recientes tomadas a lo largo de los últimos 12 segundos (para lo cual se espera que el sistema implemente un arreglo circular de tamaño suficiente).

La interfaz de salida se implementará con el display LCD de 16x2 del LCD Keypad Shield, por el cual se mostrará la información de luminosidad antes descripta. Se utlizarán 3 de los 5 pulsadores del *shield* (*Left*, *Right*, *Up* y *Down*) y el botón de la Actividad 2.

El sistema cuenta con los siguientes modos de operación:

- *Luminosidad actual (LA)*: Se muestra el valor en lux obtenido a través de la medición del LDR.
- *Luminosidad máxima y mínima (LM)*: Muestra la máxima y mínima luminosidad registrada en lux.
- *Luminosidad promedio (LP)*: Muestra la luminosidad promedio registrada.
- *Ajuste de dimmer (AD)*: En este modo, es posible ajustar la intensidad luminosa del backlight del display LCD usando los pulsadores *Up* y *Down* (20 %, 40 %, 60 %, 80 % o 100 % del brillo).Una vez alcanzado un valor extremo de brillo, el sistema ignorará la acción del pulsador correspondiente. El dimmer sobre el display deberá implementarse usando el generador PWM del microcontrolador.

En todos los casos debe mostrarse el modo en el que se encuentra el sistema. El sistema regresará automáticamente de cualquier modo a *LA* al no registrarse ninguna intervención del usuario sobre los pulsadores durante los últimos 3 segundos.

Los modos de operación y ajustes del sistema son seleccionables mediante los pulsadores del *shield* de acuerdo al siguiente detalle:

- *Up*: En *AD*, incrementa el nivel de brillo del display.
- *Down*: En *AD*, decrementa el nivel de brillo del display.
- *Left:* En *LA* cambia a *AD*, en *AD* a *LP*, en *LP* a *LM*, y finalmente en *LM* cambia a *LA*.
- *Right:* En *LA* cambia a *LM*, en *LM* a *LP*, en *LP* a *AD*, y finalmente en *AD* cambia a *LA*.
- *Botón externo*: Reinicia los valores registrados de luminosidad máxima y mínima, y vacía el arreglo de valores utilizados para calcular el promedio.

El sistema inicialmente deberá mostrar por el display datos del laboratorio, materia, cuatrimestre.Se iniciará en *LA*, con un nivel de brillo del backlight del display del 80 %. Para configurar la intensidad del backlight, puede utilizarse la función de Arduino `analogWrite`.

En base a la descripción del sistema dada previamente:

1. Identificar las tareas e ISRs del sistema e indicar, para cada una, los dispositivos asociados (display, pulsadores, timers, etc).
2. Especificar la lógica de cada tarea e ISR identificada mediante diagramas de flujo. Determinar los eventos que iniciarán la ejecución de cada una de las tareas identificadas.
3. Representar mediante un diagrama de transición de estados, los modos de operación del sistema y los eventos que ocasionan los cambios entre dichos modos.
4. Determinar las dependencias (sincronización y comunicación) entre tareas así como los datos compartidos entre ellas.
5. Integrar el sistema, construir y depurar el proyecto.
6. Descargar la imagen al microcontrolador y probar el sistema.

Referencias

[1] Analog Read. `http://arduino.cc/en/pmwiki.php?n=Reference/analogRead`.

[2] Analog Write. `http://arduino.cc/en/pmwiki.php?n=Reference/analogWrite`.

[3] Atmel AVR ATmega48A/48PA/88A/88PA/168A/168PA/328/328P Data Sheet.

[4] Ejemplo "LCD1602KeyShieldTest" disponible en la pÃ¡gina web de la materia.

[5] External Interrupts on the ATmega168/328. https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328.

[6] LCD1602 Keypad Shield Data Sheet.

[7] LCD1602 Keypad Shield Schematic.

[8] Microcontroller tutorial series: AVR and Arduino timer interrupts. http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/.

[9] We interrupt this program to bring you a tutorial on... Arduino interrupts. http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/.

[10] Michael Barr. *Programming Embedded Systems in C and C++*. O'Reilly, 1999.

[11] Elliot Williams. *Make: AVR Programming: Learning to Write Software for Hardware*. Maker Media, Inc, 2014.