Algunos ejercicios sobre Arduino

- Guía Arduino -

Laboratorio de Electrónica – Departamento de Física – FCEyN – UBA

Algunos de estos ejercicios están basados en los capítulos 13 y 18 del libro 'Arduino Cookbook'.

Registros de puertos y operaciones BITWISE

1. La forma mas sencilla de setear o leer el estado de un pin del arduino es usando las funciones *digitalWrite* o *digitalRead*. Sin embargo, los distintos pines están agrupados en *puertos*. Por ejemplo, los pines 0-7 del Arduino Uno forman un puerto de 8 bits (el puerto D). Los estados de estos 8 pines estan determinados por una variable de registro de 8 bits (es decir 1 byte), donde cada bit lógico corresponde a un pin. Pueden leer sobre los registros asociados a los distintos puertos acá. Habiendo configurado los pines 0-7 como salida (usando la funcion *pinMode* o el registro DDRD), modifique el valor del registro PORTD y chequee el valor del salida de cada pin. (Observación: si habilitan la comunicación serie, entonces los pines 0 y 1 no se pueden usar como entrada o salida).

2. Se desea setear el estado de dos pines de un puerto *simultaneamente* (el estado de los otros 6 pines es irrelevante). ¿Puede hacerse esto utilizando la función *digitalWrite*? ¿Y de alguna otra forma?

3. Para cambiar el valor de algunos bits de un registro sin afectar los otros, o para leer el valor de un bit particular, suelen usarse *máscaras binarias* o *bitmasks*. Por ejemplo, la operación $X = X|4$ setea en 1 el tercer bit de la variable X independientemente de su valor anterior (sin afectar el resto de los bits), ya que el operador | implementa la compuerta OR a nivel de bits y el valor 4 se representa como B00000100 en binario. Otros operadores a nivel de bits son &, $\wedge$, y $\sim$; ¿cómo funcionan?. Repita el ejercicio anterior sin afectar el estado de los bits que no se desea controlar.

Generación de señales

4. La forma más sencilla de generar una señal cuadrada es prender y apagar un determinado pin de salida cada intervalos regulares de tiempo. Hacer eso usando las funciones *digitalWrite*, *delay* y/o *delayMicroseconds*. Estudie la señal de salida con un osciloscopio.

5. Repita el ejercicio anterior utilizando la función *noInterrupts* y estudie si se nota algún cambio en la señal de salida (para esto conviene usar delays de unos pocos microsegundos). Las interrupciones son muy útiles (como vemos en la siguiente sección) pero pueden ser un problema en casos donde el control temporal dentro de la secuencia principal es crítico.

6. Otra forma de generar señales cuadradas es usar el hardware de Pulse Width Modulation (PWM) integrado en el Arduino. Estudie esta manera usando la función *analogWrite*.

7. Las frecuencias configuradas por default para los PWM son bajas (490 Hz o 976 Hz en el Arduino Uno, dependiendo de los pines). Estas frecuencias se pueden cambiar manipulando algunos timers o contadores internos del Arduino. Explore esta opción utilizando los registros TCCR0B, TCCR1B y TCCR2B, o alternativamente la librería Timer1.

Interrupciones

8. Supongamos que queremos realizar alguna acción cuando un determinado pin de entrada cambia su estado. Una forma de hacer esto es chequear constantemente el estado de ese pin en la secuencia o programa principal. Una forma alternativa es usando interrupciones de hardware. Estudie esta segunda manera utilizando la función *attachInterrupt*.

9. Supongamos que queremos realizar una determinada tarea cada intervalos regulares de tiempo. Esto se puede hacer en la secuencia principal con las funciones *delay* o *delayMicroseconds*. Una manera alternativa es utilizando contadores externos y una interrupción que se genera cada vez que estos contadores llegan a cero o se resetean. Estudie esta alternativa utilizando la libreria MsTimer2.

Comunicación

10. El Arduino Uno tiene un único puerto tipo UART para comunicación serie, que utiliza los pines 0 y 1. Además, este puerto está acoplado a la conexión USB, de forma que puede comunicarse directamente con una computadora. El uso más común de este puerto es para enviar información de diagnóstico o sensores desde el arduino a la computadora (seguramente ya lo hicieron a travez de la función *Serial.print*), pero también puede usarse en el otro sentido. Use la función *Serial.read* para enviar comandos desde la computadora hacia el Arduino. Escriba un programa que funcione como un 'echo', es decir que espere recibir palabras y luego las repita en la salida.

11. El Arduino Uno soporta dos protocolos de comunicación serie adicionales, I2C y SPI, que son utilizados para comunicación entre distintos módulos de un sistema. Hay muchos chips con funciones específicas (por ejemplo relojes, pantallas, acelerómetros, etc) cuya interfaz con el mundo exterior es también a travez de esos protocolos. En el laboratorio hay disponibles algunos de estos chips. Elija alguno y realice los siguientes puntos:

    - Intente interrogar al chip usando directamente las instrucciones de las librerias Wire o SPI de Arduino, según corresponda.
    - Encuentre si hay alguna librería para Arduino desarrollada para la comunicación con el chip que haya elegido.
    - Si la librería es de código abierto, trate de ver cómo los protocolos I2C o SPI son utilizados de forma subyacente.
    - Busque la hoja de datos del chip e investigue si todas las funcionalidades ofrecidas se pueden acceder a travez de la librería.

    Algunos módulos de productos comerciales utilizan directamente los protocolos I2C o SPI. Un ejemplo es uno de los controles de la Nintendo Wii estudiado en la seccion 13.2 del libro Arduino Cookbook.

12. Dos placas Arduino pueden comunicarse entre sí a travez del protocolo I2C. Por ejemplo, una placa puede ocuparse del control y lectura de algún sensor, para luego transmitir los datos obtenidos a otra placa que se encarga de procesar los datos e integrarlos con otros. Implemente alguna versión de esta idea.