Taller del Tema 6: Temporizadores/Contadores

Ángel Grover Pérez Muñoz, Manuel M. Nieto Rodríguez

Noviembre de 2024

El objetivo de este taller es poner en práctica ejemplos de uso de los temporizadores/contadores embebidos en el microcontrolador ATMega32U4. Se implementarán las funcionalidades para medir tiempos (reloj y espera relativa) y generar ondas mediante modulación de ancho de impulsos (PWM).

Actividad 1: Reloj

El objetivo es implementar un reloj global con precisión de milisegundos. De forma similar a la función de Arduino `uint32_t millis()`, vamos a implementar una función `uint32_t clock_ms()` que devuelva el tiempo transcurrido como valor de retorno.

Esto require de la configuración de un timer en modo periódico que actualice un contador software global, cuyo valor permitirá inferir el tiempo transcurrido. Para ello programaremos el Timer 1 con la siguiente configuración:

- Modo CTC: Esto nos permitirá fijar un valor de tope distinto del valor máximo (registro `ICR1`).
- No es necesario activar ningún canal de salida (bits `WGM1[0:3]`).
- Se debe activar el biestable de máscara de interrupción cuando el contador alcance el tope (bit `ICIE1`).
- La frecuencia para alcanzar el tope debe ser de 1 KHz. Para ello:
  - Tope: `ICR1`=15999
  - Prescaler: `N`=1

Actividad 2: Espera relativa y absoluta

El objetivo es implementar funciones que permitan pausar la ejecución del programa. Esto require usar la función `clock_ms()` de la "Actividad 1" ya que devuelve el tiempo actual.

a. Espera relativa: El objetivo es implementar una función que permita pausar la ejecución del programa *durante una franja de tiempo*. De forma equivalente a la función `void delay(uint32_t ms)` de Arduino, la función a implementar será `void delay_ms(uint32_t ms)` y realizará una espera activa durante los milisegundos especificados en el parámetro `ms`.

b. Espera absoluta: El objetivo es implementar una función que permita pausar la ejecución del programa *hasta un tiempo determinado, pasado como parámetro*. La función a implementar será `void delay_until_ms(uint32_t end_ms)` y realizará una espera activa hasta el instante de tiempo `end_ms`.

Actividad 3: PWM en modo "fast"

La modulación por ancho de impulso, o Pulse Width Modulation (PWM), es un mecanismo que permite emitir señales analógicas. Esta señal está compuesta por un tren de pulsos cuyo ciclo de trabajo (duty-cycle en inglés) permite emitir un valor medio distinto del valor alto (5V) o bajo (0V).

El objetivo de esta actividad es emitir una señal analógica mediante PWM en modo "fast" a través del Timer 1.

Considere lo siguiente:

- Modo "Fast PWM" con `OCR1A` como valor de tope.
- Señal PWM emitida en los canales B y C.
- Los valores de `OCR1B` y `OCR1C` se corresponden con el ciclo de trabajo de las ondas en los canales B y C.

Se plantea realizar lo siguiente:

- Implementar una función `void set_pwm_duty(int ch, float dc)` que configure el ciclo de trabajo `dc` en un canal de salida `ch`.
- Generar ondas con las frecuencias 50 Hz y 10 kHz.
- Conectar un LED a la salida de los canales para ver los cambios en su luminosidad.

Actividad 4: PWM en modo "phase & frequency correct"

El objetivo de esta actividad es emitir una onda PWM en modo "fase y frecuencia correcta". Esto permitirá ejemplificar la diferencia entre los distintos modos PWM disponibles. Para ello, también se empleará el Timer 1.

Considere lo siguiente:

- Modo "Phase & Frequency Correct PWM" con `OCR1A` como valor de tope.
- Señal PWM emitida en los canales B y C.
- Los valores de `OCR1B` y `OCR1C` se corresponden con el ciclo de trabajo de las ondas en los canales B y C.

Se plantea realizar las mismas funcionalidades que en la "Actividad 3":

- Implementar una función `void set_pwm_duty(int ch, float dc)` que configure el ciclo de trabajo `dc` en un canal de salida `ch`.
- Generar ondas con las frecuencias 50 Hz y 10 kHz.
- Conectar un LED a la salida de los canales para ver los cambios en su luminosidad.