Generación de Estímulos con *Keil* uVision 4

Versión 3.0

Sistemas Electrónicos Digitales

Universidad de Alcalá
Departamento de Electrónica

1 Introducción al documento

Este documento pretende mostrar la forma de generar estímulos externos al microcontrolador dentro del simulador de Keil uVusion4. De esta manera se pueden simular la entrada de señales externas al microcontrolador para medir tiempos.

2 Generación de Estímulos

El entorno *Keil* uVision 4 dispone de un módulo de "Generación de Estímulos" que permite simular el comportamiento de sistemas reales externos al microcontrolador. Con la generación de estímulos se pueden simular, por ejemplo, la llegada de señales externas a los pines del microcntrolador.

Un "**fichero de estímulos**" contiene unas "**funciones de estímulos**" que se escriben siguiendo la sintaxis de Lenguaje C y que **son ejecutadas por el entorno de simulación** (No por el µC), en paralelo con la simulación del programa que se ejecuta en el microcontrolador.

En la Figura 1 se obseva que el simulador por un lado ejecuta el fichero de estímulos cuando así es requerido, y por otro simula el programa de usuario.

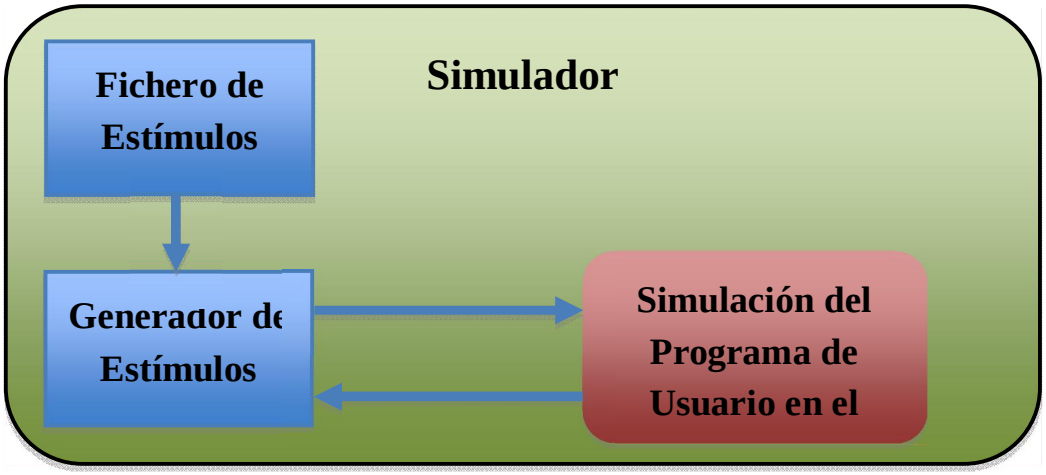


Fig.1 Relaciones entre el simulador y el generador de estímulos

A continuación se presenta el código de dos funciones de estímulos iguales (*periodica_1* y *periodica_2*) a las que se pasa como argumentos: el puerto, el pin en el que se aplica la señal, la frecuencia de la señal y el ciclo de trabajo (en tanto por uno):

- *periodica_1(2, 13, 1000,0.5)* à simula la entrada por P2.13 de una señal de frecuencia 1000 Hz con un ciclo de trabajo del 50%.
- *periodica_2(2, 12, 200, 0.1)* à simula la entrada por P2.12 de una señal de frecuencia 200 Hz con un ciclo de trabajo del 10%.
  NOTA: *La existencia de las dos funciones de estímulos idénticas es debido a que internamente al tener un while(1), no podríamos lanzar su ejecución en paralelo para generar de forma simultánea 2 señales periódicas distintas con una única función.*

La función *swatch()* realiza una espera igual al tiempo que se introduce como parámetro en segundos. Se trata de una función interna para hacer uso en el editor de estímulos.

La forma de hacer referencia a los pines externos GPIO2 desde el simulador es con la etiqueta **PORT2** y de forma similar al resto de los puertos. No es válido utilizar FIO2PIN, pues las funciones de estímulos no ejecutan código del µC.

En el ejemplo también se muestra cómo se pueden definir funciones que sean utilizadas desde diferentes señales o incluso desde la línea de comandos.

Fichero **ESTIMULOS.INI**

```
// Función que pone a nivel alto un pin de entrada de un puerto
//
func void SetPin (int puerto, int pin)
{
  switch(puerto) {
    case 0:
      PORT0|=(1<<pin);                    // Pone P0.pin a 1
           break;
    case 1:
      PORT1|=(1<<pin);                    // Pone P1.pin a 1
           break;
    case 2:
      PORT2|=(1<<pin);                    // Pone P2.pin a 1
           break;
  }
}

// Función que pone a nivel bajo un pin de entrada de un puerto
//
func void ClrPin (int puerto, int pin)
{
  switch(puerto) {
    case 0:
      PORT0&=~(1<<pin);                   // Pone P0.pin a 0
           break;
    case 1:
      PORT1&=~(1<<pin);                   // Pone P1.pin a 0
           break;
    case 2:
      PORT2&=~(1<<pin);                   // Pone P2.pin a 0
           break;
  }
}

// Señal periódica de frecuencia "frecuencia" y ciclo de trabajo "ciclo" que se introduce
// por el pin "pin" del puerto "puerto"
//
signal  void periodica_1 (int puerto, int pin, float frecuencia, float ciclo)
{
  while(1){
    SetPin(puerto, pin);              // Pone Puerto.pin a 1
    swatch(ciclo*(1/frecuencia));     // Espera ciclo*T segundos
    ClrPin(puerto, pin);              // Pone Puerto.pin a 0
    swatch((1-ciclo)/frecuencia);     // Espera (1-ciclo)*T segundos
  }
}

// Señal periódica de frecuencia "frecuencia" y ciclo de trabajo "ciclo" que se introduce
// por el pin "pin" del puerto "puerto"
//
signal  void periodica_2 (int puerto, int pin, float frecuencia, float ciclo)
{
  while(1){
    SetPin(puerto, pin);              // Pone Puerto.pin a 1
    swatch(ciclo*(1/frecuencia));     // Espera ciclo*T segundos
    ClrPin(puerto, pin);              // Pone Puerto.pin a 0
    swatch((1-ciclo)*(1/frecuencia)); // Espera (1-ciclo)*T segundos
  }
}
```

Este fichero debe editarse con el Editor de Funciones que se encuentra en el menú Debug → Function Editor. Se escribe la función y se compila.

Una vez compilado sin errores, es necesario que el simulador se entere de su existencia. Para ello es necesario, en la línea de comandos del compilador poner:

```
>INCLUDE ESTIMULOS.INI
```

Y para ejecutarlo hay que poner en la misma línea de comandos:

- Para generar una señal periódica de 1kHz en la entrada del pin P2.13 de 50% de ciclo de trabajo

```
>periodica_1(2,13,1000,0.5)[enter]
```

- Para generar una señal periódica de 200Hz en la entrada del pin P2.12 de 10% de ciclo de trabajo.

```
>periodica_2(2,12,200,0.10) [enter]
```

Para desactivar las funciones de estímulos, puesto que internamente tienen un while(1):

```
>SIGNAL KILL periodica_1 [enter]
>SIGNAL KILL periodica_2 [enter]
```

Es importante hacer notar que para modificar el pin x del puerto 2 como **estímulo externo**, es necesario usar la etiqueta **PORT2.x**. Lo mismo pasa con los otros puertos.

3 Ejemplo de uso de funciones de estímulos externos

El ejemplo que se muestra a continuación conmuta un LED conectado al pin P1.29 con un flanco de bajada en la entrada de interrupción externa del LPC1768 EINT3 y conmuta un LED conectado a P1.28 con EINT2.

El proyecto que contendría este programa también debería contener los ficheros *startup_LPC17xx.s* y *system_LPC17xx.c*

```
// *****************************************************
//  File: PruebaEstimulosPrincipal.c
//
//  Programa que:
//    - Con un flanco de bajada en EINT3 (P2.13) conmuta P1.29
//    - Con un flanco de bajada en EINT2 (P2.12) conmuta P1.28
// *****************************************************

#include <LPC17xx.H>

// Definiciones de los pines de salida
#define LED_1  (1<<28)
#define LED_2  (1<<29)
```

```
//Funcion de interrupcion EINT2

void EINT2_IRQHandler(void)
{

  LPC_SC->EXTINT = 1 << 2; // Borrar el flag de la EINT3 --> EXTINT.2
  LPC_GPIO1->FIOPIN ^= LED_1; // Conmuta el LED  --> GPIO1 FIOPIN  P1.28
}

//Funcion de interrupcion EINT3

void EINT3_IRQHandler(void)
{
 LPC_SC->EXTINT = 1 << 3;  // Borrar el falg de la EINT3 --> EXTINT.3
 LPC_GPIO1->FIOPIN ^= LED_2; // Conmutar el LED  --> GPIO1 FIOPIN  P1.29
}


// Funcion de inicialización

void Config(void)
{
  // Configuracion de los pines P2.12 y P2.13 como EINT2 y EINT3
  //       --> PINSEL4.24 y PINSEL4.25
  //       --> PINSEL4.26 y PINSEL4.27
  LPC_PINCON->PINSEL4 |= (1 << (12*2)) | (1 << (13*2));

  // Configura los pines P1.28 y P1.29 como salida --> GPIO1 FIODIR.28 y FIODIR.29
  LPC_GPIO1->FIODIR |= LED_1 | LED_2;

  // Configuracion activa por flanco de bajada       --> EXTMODE.2 y EXTMODE.3
  LPC_SC->EXTMODE |= (1<< 2) | (1 << 3);

  // Configuramos las prioridades de EINT 2 y EINT3
  NVIC_SetPriority(EINT2_IRQn, 0x02);
  NVIC_SetPriority(EINT3_IRQn, 0x01);
  NVIC_SetPriorityGrouping(2); // 32 niveles pre-emptive (sin subprioridad)


  // Habilitar interrupciones EINT2 y EINT3
  NVIC_EnableIRQ(EINT2_IRQn);
  NVIC_EnableIRQ(EINT3_IRQn);
}



// Programa principal
void main(void)
{
  //Funcion de inicializacion
  Config();

  // Programa principal
  while (1);
}
```

En la Fig. 1 se muestra el resultado de la simulación del programa tras activar los estímulos *periodica_1()* y *periodica_2()* con los parámetros visto anteriormente .

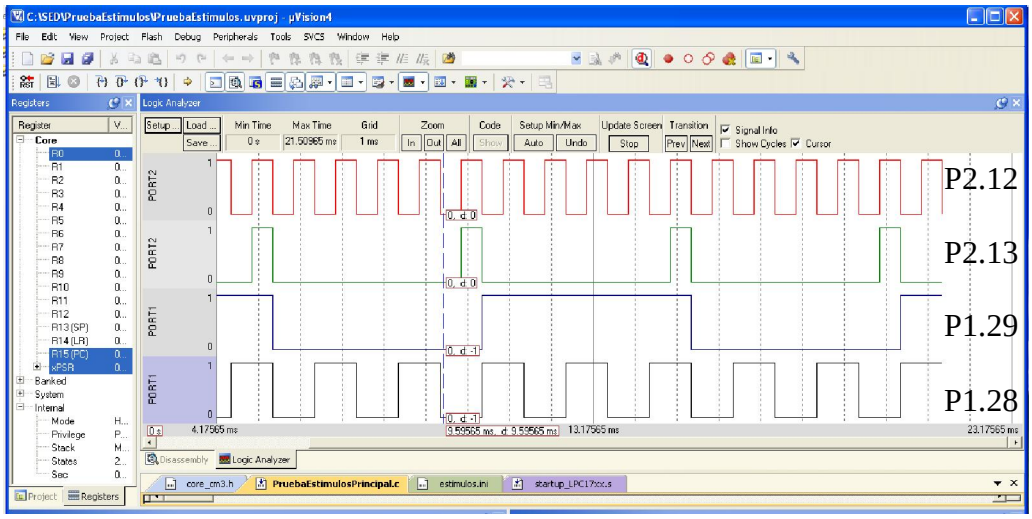


Fig 1: Simulación que visualiza las señales de estímulos y las salidas

4 Utilizando “Botones”

Se puede asociar un botón a una función de estímulos (o a un comando del simulador) introduciendo, en la ventana Command, el comando

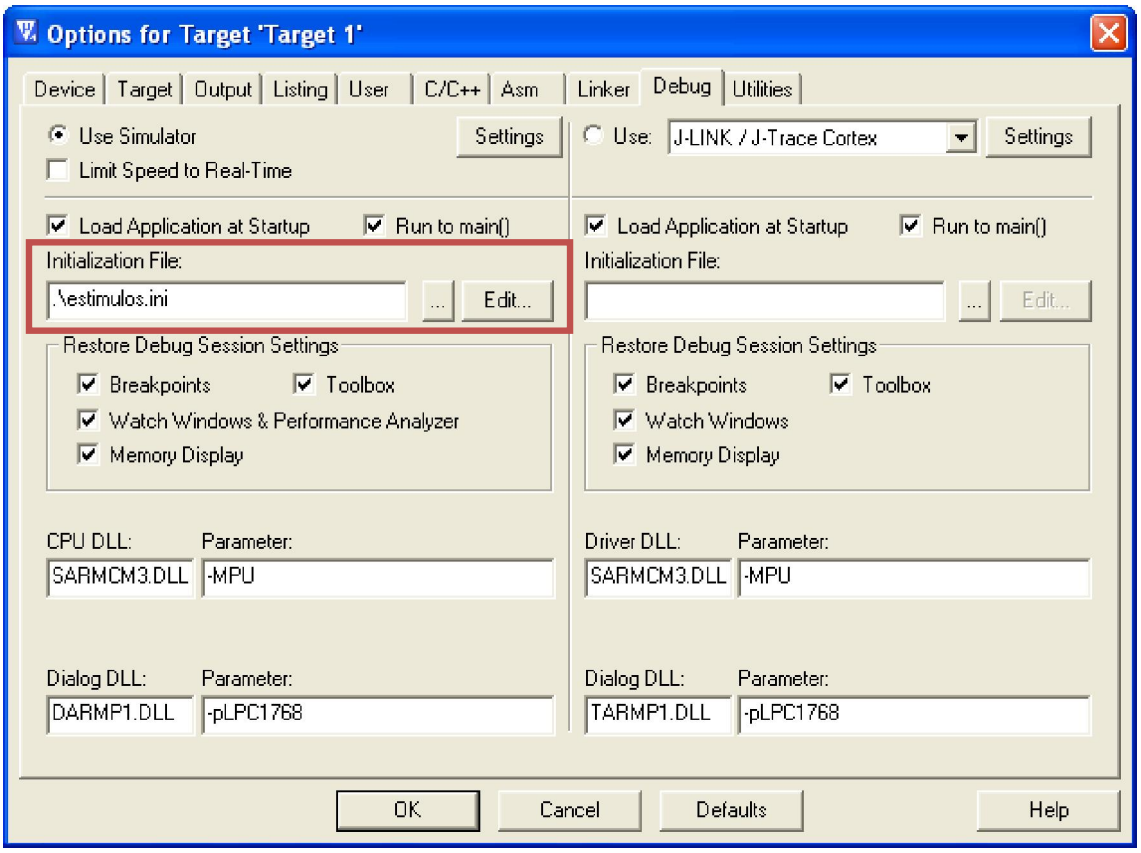
```
>DEFINE BUTTON “<button label>”,”command”
>DEFINE BUTTON “<button label>”,”nombre de la funcion”
```

Por ejemplo, se puede añadir al fichero “ESTIMULOS.INI” los siguientes comandos para tener acceso a los estímulos del apartado anterior (Si no aparecen pulsar en ):

```
DEFINE BUTTON "1KHz  (50%) por P2.13 (ON)" , "periodica_1(2,13,1000,0.5)"
DEFINE BUTTON "1KHz  (50%) por P2.13 (OFF)", "SIGNAL KILL periodica_1"
DEFINE BUTTON "200Hz (10%) por P2.12 (ON)" , "periodica_2(2,12,200,0.1)   "
DEFINE BUTTON "200Hz (10%) por P2.12 (OFF)", "SIGNAL KILL periodica_2"
```

Nota 1: Si la ventana no se refresca durante la ejecución del programa, es necesario activar: View à Periodic Window Update

Nota 2: El fichero .INI se puede cargar automáticamente al entrar en la simulación si así se configura en las opciones del proyecto (Debug).


Options for Target 'Target 1'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Simulator
Settings
Limit Speed to Real-Time
Use:
J-LINK / J-Trace Cortex
Settings
Load Application at Startup
Run to main()
Initialization File:
.\estimulos.ini
Edit...
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Memory Display
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows
Memory Display
CPU DLL:
Parameter:
SARMCM3.DLL
-MPU
Driver DLL:
Parameter:
SARMCM3.DLL
-MPU
Dialog DLL:
Parameter:
DARMP1.DLL
-pLPC1768
Dialog DLL:
Parameter:
TARMP1.DLL
-pLPC1768
OK
Cancel
Defaults
Help