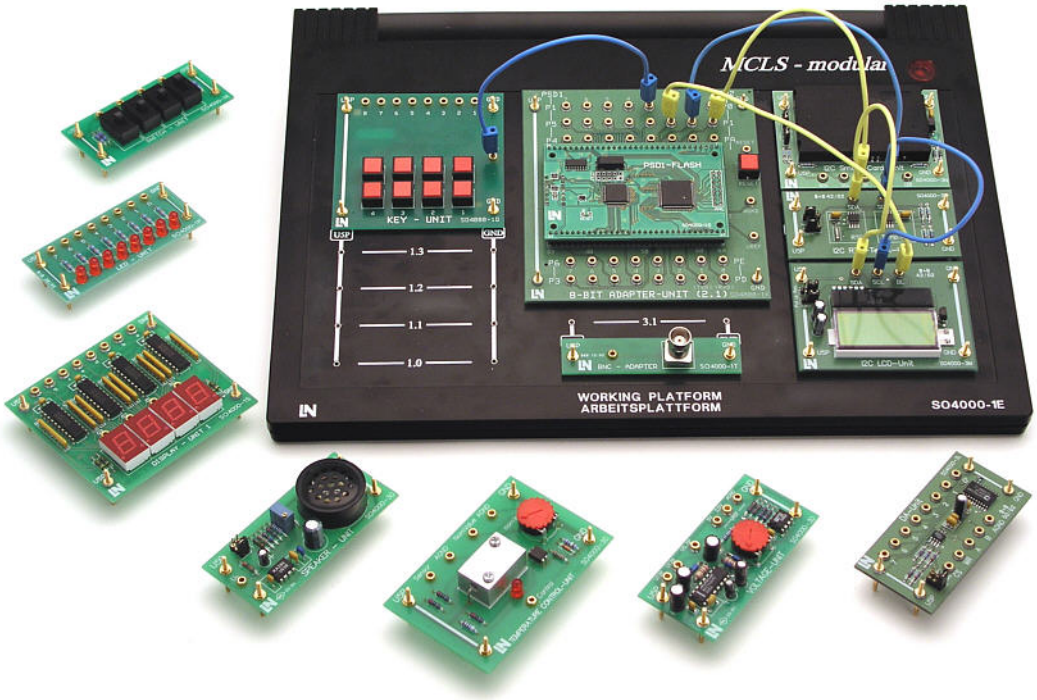
Programación
C de microcontroladores (C515C)

CMC 5



Guía de ejercicios para el estudiante

SH5004-1E

3ra. edición

Autor: equipo de autores ACMC Mittweida

Lucas-Nülle Lehr- und Messgeräte GmbH
Siemensstraße 2 • D-50170 Kerpen

Índice

Objetivo de los ensayos

1. Los aprendices / estudiantes adquieren conocimientos relativos a la estructura, al funcionamiento y a la programación C de un moderno sistema de microcontrolador.
2. Los aprendices / estudiantes se familiarizan con la estructura del MC y con la periferia “on-chip” seleccionada. Aprenden a emplear de forma óptima los componentes del MC.
3. Los aprendices / estudiantes se familiarizan con la programación C y la utilización de un sistema de desarrollo moderno.

Sinóptico de ensayos

Bloque de ensayos CMC 5-1

- Estructura de base de un programa C
- Utilización de diferentes estructuras de control
- Manipulaciones de bits
- Empleo de puertos y pins de puerto.

Bloque de ensayos CMC 5-2

- Utilización de interrupciones
- Interrupciones externas
- Interrupciones por temporizador
- Timer on chip como temporizador
- Timer on chip como contador
- Generación PWM

Bloque de ensayos CMC 5-3

- Bus I²C
- Activación de la unidad de LCD de I²C mediante bibliotecas de funciones
- Utilización del convertidor AD on chip
- VIsualización de valores medidos en el display

Bloque de ensayos CMC 5-4

- Activación del sensor de temperatura LM75
- Acceso indexado a un campo de datos
- Cálculos mediante operadores estándar
- Inserción de un reloj de tiempo real y su programación
- Posibilidades de acceso a una estructura

Bloque de ensayos CMC 5-5
(Trabajo de proyecto)

- Realización de una aplicación compleja
- Inserción y programación de una tarjeta chip I²C

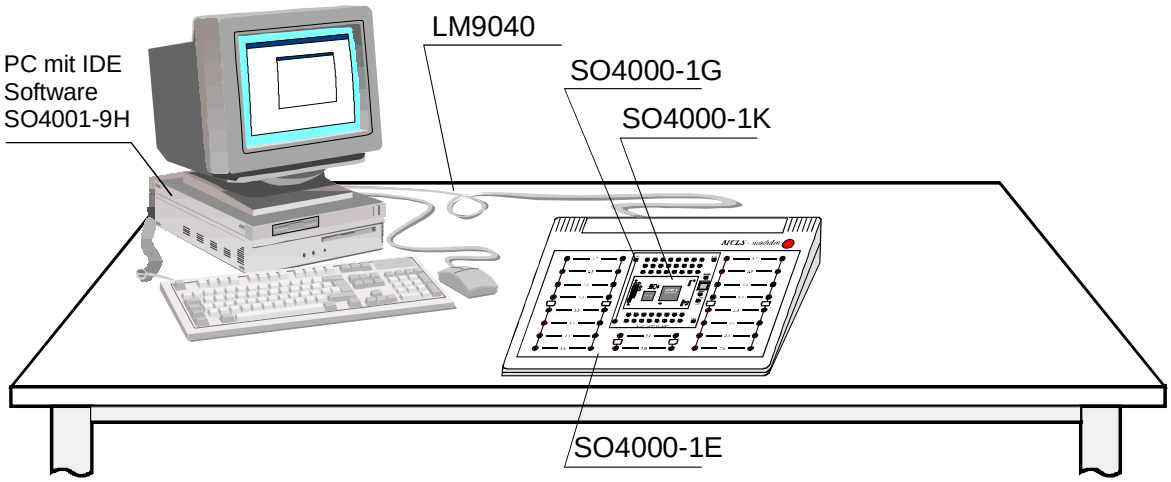


Figura 1: Instalación en el puesto de trabajo

Instalación de aparatos

Para una realización del ensayo efectiva y sin errores se representan a continuación y por separado:

- Tabla con todos los módulos, aparatos, cables de medición y accesorios necesarios.
- La instalación de los aparatos en la mesa de trabajo.

Recomendamos exponer los manuales de instrucciones de los paneles de aprendizaje empleados en la realización de los ensayos. Representan un valioso complemento a los respectivos ejercicios.
Los puntos de medición, los interruptores y los elementos de ajuste en los módulos se explican individualmente. Los manuales de instrucciones también incluyen indicaciones de aplicación fundamentales.

Componentes y aparatos necesarios

Denominación	Cant.	N.º pedido
Plataforma con módulo de alimentación de tensión +5V	1	SO4000-1E
Fuente de alimentación AC de enchufe 90...230V 45...65Hz, DC 9V 630mA	1	SO4000-1F
Software IDE para MCLS (D, GB, F, E)	1	SO4001-9H
Controlador Flash PSD1 C515C	1	SO4000-1G
Unidad de adaptador de 8 bit	1	SO4000-1K
Unidad LED (8 bit)	1	SO4000-1P
Unidad de teclas (8 teclas)	1	SO4000-1Q
Unidad de interruptores (4 interr.)	1	SO4000-1R
Adaptador BNC	1	SO4000-1T
Unidad del usuario para instalar circuitos de ensayo propios	1	SO4000-1U
Unidad de controlador de bus	1	SO4000-1V
Unidad indicadora 1 (7 segmentos)	1	SO4000-1S
Unidad de tensión	1	SO4000-3D
Unidad de altavoz	1	SO4000-3G
Unidad DA	1	SO4000-3L
Unidad de LCD I^2C	1	SO4000-3M
Unidad de Tarjeta inteligente I^2C	1	SO4000-3N
Unidad de memoria I^2C	10	SO4000-3O
Unidad RTC-Temp. I^2C	1	SO4000-3P
Cable de interfaz serial 9/9 polos	1	LM9040
Cables medición 2mm, 15cm, azul	10	SO5126-5K
Cables medición 2mm, 15cm, amar.	10	SO5126-5M
CMC 5 Programación C de microcontroladores (C515C)	1	SH5019-1E

Introducción y condiciones básicas

Los lenguajes de programación de alto nivel ofrecen, en el desarrollo de programas para microcontroladores, una serie de ventajas. Las razones esenciales para su empleo son el más alto nivel de abstracción, la transportabilidad relativamente sencilla del código a diferentes sistemas de destino, así como la gestión del software en la que varios diseñadores tramitan tareas parciales de un gran proyecto. Además, el desarrollo del código se encuentra más cerca del problema a solucionar, siendo su densidad y con ello la velocidad de realización no tan alta como en una solución de Ensamblador. La utilización de un lenguaje de alto nivel hace suponer que el código de máquina generado por un Compilador de lenguaje de alto nivel es en el 10% al 30% mayor en comparación con el Ensamblador, con lo cual se necesita más memoria de programa en la unidad de destino.

El lenguaje de programación de alto nivel más frecuentemente utilizado en relación con los microcontroladores es C, porque por una parte este lenguaje está muy divulgado en otros sistemas y por otra, C se aproxima más al hardware. Ello se muestra por ejemplo en la posibilidad de manipulación de bits y también en el empleo de campos de datos y punteros. Las posibilidades de estructuración de programas son muy amplias y pueden ser convertidas de forma muy eficiente en códigos de máquina por el Compilador. Hacia arriba, C es compatible con C++ porque muchos Compiladores generan códigos de máquina de ambas variantes. Ello permite un diseño de programa orientado en el objeto, lo que sin embargo no es objeto de las siguientes consideraciones.

El presente programa de ensayos permite una familiarización orientada en la práctica con el lenguaje de programación C, especialmente con C para *Sistemas Integrados* con microcontroladores. Como sistema de destino concreto para las soluciones de programa a realizar en los ejercicios se emplea el *MCLS-modular®* con el módulo *FLASH PSD1* (www.mcls-modular.de). El hardware de este sistema de entrenamiento dispone de numerosos componentes de aplicaciones de microcontrolador típicas (unidad indicadora LCD, teclas de funciones, sensores, actores, interfaces, ...).

El presente manual no está pensado primordialmente como obra de consulta para C estándar. Sin embargo ofrece, en el punto 2, una introducción resumida y un sinóptico de C. Durante el procesamiento del programa de ensayo, esta parte se puede utilizar para consultas rápidas. Sin embargo, se recomienda la utilización adicional de literatura apropiada sobre C (ver indicaciones de literatura en el punto 7). Las particularidades que caracterizan la programación C para Sistemas Integrados, forman parte del punto 3. Como herramienta potente para el desarrollo de software se emplea el ***S**mall **D**evice **C** **C**ompiler SDCC* bajo *Licencia Pública General* (*General Public License*). El funcionamiento y la aplicación de esta herramienta se describe detalladamente en el punto 4. Después de una presentación y descripción funcional de las bibliotecas de funciones (punto 5) necesarias para el hardware de destino, se presenta, en el punto 6, el programa de entrenamiento para los ensayos a realizar independientemente y se procede a explicar en detalle las distintas lecciones.

¡La combinación de un hardware de destino específico, de herramientas de software óptimas y de objetivos didácticos en el programa de ejercicios permite unos estudios autodidácticos eficientes!

¿Qué se necesita para estudiar a fondo las lecciones?

1. Como hardware de destino un *MCLS-modular*®

   - La información técnica y la información de distribución se obtiene en la página Web www.mcls-modular.de.
   - La distribución se realiza por medio de la empresa *LUCAS-NÜLLE Lehr- und Messgeräte GmbH, Kerpen* (www.lucas-nuelle.de).

2. *SDCC* como Compilador C así como MCLS-IDE

   - Para la realización de los ejercicios se emplea el Compilador C de Licencia Pública General *SDCC*.
   - La información técnica sobre el Compilador se obtiene en la página Web http://sdcc.sourceforge.net/.

3. Literatura complementaria

- Manual de instrucciones del módulo FLASH PSD1
- Una obra de consulta para el lenguaje de programación C (p. ej Mittelbach, H.: Introducción en C, Editorial Fachbuchverlag Leipzig en la Editorial Carl Hanser Verlag, 2001)

Nota importante:

Un desarrollo eficiente y con éxito de programas C para *Sistemas Integrados* presupone, entre otras cosas, conocimientos orientados hacia la máquina del sistema de destino. Por lo tanto es recomendable, como preparación a este programa de entrenamiento, estudiarse a fondo seleccionados cursillos de Ensamblador del *MCLS-modular*®:

- Cursillo CMC1: Introducción en la técnica de microcontrolador (80C535 / C515C)

- Cursillo CMC3: Programación de periferia on chip (80C535 / C515C)

Introducción resumida en C

Tipos de datos

En el lenguaje de programación C se distingue entre 4 tipos de datos base con los que se puede determinar el margen de valores de variables, así como de funciones. Hay que observar que estos tipos de datos base **dependen**, en parte, **del ordenador**. El tipo de datos *int*, por ejemplo, puede comprender 2 ó 4 bytes. A continuación se representan los márgenes de valores de los tipos de datos según el estándar ANSI.

Tipo de datos	Tamaño de byte	Designación	Margen de valores
char	1	Carácter ASCII	-128 … 127
int	2	Número entero	-32768 … 32767
float	4	Nº de punto flotante	-10 ^38 ... 10 ^38
double	8	Nº de punto flotante	-10 ^308 ... 10 ^308

Tabla 1: Tipos de datos

Al anteponer propiedades, los tipos de datos base ***char*** e ***int*** pueden ser calificados. Las propiedades posibles son **signed**, **unsigned**, **long** y **short**.

Tipo de datos	Tamaño de byte	Designación	Margen de valores
unsigned char	1	Carácter ASCII	0 … 255
unsigned (short) int	2	Número entero	0 … 65535
unsigned long int	4	Número entero	0 ... 4 294 967 295
signed long int	4	Número entero	- 2 147 483 648 ... 2 147 483 647

Tabla 2: Ejemplos para tipos de datos *char* e *int* con propiedades

Variables

Las variables son áreas de memoria de tamaño determinado y contenido variable. Para una variable se debe definir un nombre. El tamaño queda definido por el tipo de datos.

Forma general de un acuerdo de variable:

clase de memoria tipo identificador$_1$, ..., *identificador*$_n$ **;**

Ejemplos:

```
unsigned char signo;        /* una variable de 8 bits sin signo */
int i;                      /* una variable de número entero con signo */
signed int h;               /* una variable de número entero con signo */
float z;                    /* una variable del tipo de número de punto flotante */
```

Las variables se pueden declarar de forma **global** para la utilización en el programa entero o de forma **local** dentro de una función. Global significa que el Compilador asigna a la variable una dirección fija en la memoria, mientras que para variables locales solamente se utiliza la memoria de datos durante la ejecución de la función.

Definiciones de variables adicionales

En la declaración de variables, se le puede comunicar al Compilador el cómo debe tratar una variable. Esto es posible mediante las palabras clave ***const, volatile, register*** y ***static***.

const

Con esta palabra clave se genera una constante de una variable.

volatile

La palabra clave comunica al Compilador que la variable se puede modificar también fuera del control del programa, p. ej. en rutinas de servicio de interrupción o en partes de programa del Ensamblador. Se realiza un acceso directa a la memoria de una variable *volatile*.

register

A través de esta palabra clave, el Compilador intenta asociar a la variable un registro de la máquina de destino. Permite al programador realizar optimaciones del tiempo de ejecución del programa. La instrucción se ignora si no quedan registros libres en el destino.

static

Si, dentro de una función, está definida una variable local como *static*, su valor se mantiene también después de la finalización de esta función.

Funciones

Normalmente, un programa C está dividido en diferentes funciones. Una función se compone, de un modo general, de un encabezamiento y de un cuerpo.

```
Tipo de datos Nombre de función (parámetro)      // Encabezamiento de función
{                                                // Comienzo cuerpo de función

    Tipo de datos Variable;                      // Variable loc. para valor de retorno

    Expresión1;                                  // Contenido del cuerpo de función
    Expresión2;
    ...
    return Variable;                             // Valor de retorno de la función

}                                                // Fin cuerpo de función
```

Después del *nombre de la función* es forzosamente necesario un paréntesis doble, en el cual se pueden definir los parámetros de transferencia. El cuerpo de la función está caracterizado por las llaves entre las cuales están anotadas las instrucciones de la función.

Antes de finalizar el cuerpo de la función figura una instrucción de *return* con una variable de retorno.

¡El tipo de datos de esta variable de retorno determina al *tipo de datos* de toda la función!

Si la función no debe suministrar un valor, existe un tipo de datos adicional *void* que comunica al Compilador que se trata de una función sin tipo que en otros lenguajes de programación también se llama proceso.

Dentro de un programa se le debe indicar al Compilador exactamente **una** de las funciones como función principal. Con la función ***main*** empieza la ejecución del programa C realizándose las llamadas a todas las demás funciones desde esta función principal.

Antes de la función ***main*** todas las subfunciones empleadas en el programa deberán declararse mediante el prototipado. Un prototipo de una función tiene la misión de suministrarle al Compilador información adicional para convertir parámetros de forma automática para funciones y para generar mensajes de error o avisos en caso de datos no convertibles.

Ejemplo:

```
// Prototipos de funciones ------------------------------------
void func1(int)          // Prototipo Función1

int func2(void);         // Prototipo Función2

// Función principal ------------------------------------------
void main(void)          // Función principal
{
        int a;           // Variable loc.

        func1(a);        //  Llamada a la función1
        a = func2();     //  Llamada a la función2
}

// Contenidos de funciones ------------------------------------
void func1(int b)        // Función1
{
        Expresión;
}

int func2(void)          // Función2
{
        int c;           // Variable para valor de retorno

        Expresión;
        return c;
}
```

Operadores

Existen diferentes operadores para modificar datos. Se distingue entre operadores binarios y unarios. Los operadores unarios necesitan un operando, mientras que los operadores binarios precisan dos operandos. Las siguientes tablas ofrecen un resumen de los operandos utilizables.

Signo	Significado
+	Adición (Más binario)
-	Sustracción (Menos binario)
*	Multiplicación
/	División
%	Módulo (resto de número entero de una división)
<<	Arrastre a la izquierda
>>	Arrastre a la derecha
&	AND (Y) bit a bit
|	OR (O) bit a bit
^	XOR (O exclusivo) bit a bit
&&	AND lógico
||	OR lógico
=	Asignación
*=	Producto de asignación
/=	Cociente de asignación
%=	Módulo de asignación
+=	Suma de asignación
-=	Diferencia de asignación
<<=	Asignación arrastre a la izquierda
>>=	Asignación arrastre a la derecha
&=	Asignación AND bit a bit
|=	Asignacion OR bit a bit
^=	Asignación XOR bit a bit
<	inferior a
>	superior a
<=	inferior igual a
>=	superior igual a
==	Igualdad
!=	Desigualdad
.	Selección de elemento directa
->	Selección de elemento indirecta mediante puntero
,	Operador de coma para resumir expresiones

Tabla 3: Sinóptico de operadores binarios

Signo	Significado
&	Operador de dirección
!	Negación lógica
*	Indirección
++	Incremento
--	Decremento
~	Complemento bit a bit
-	Menos unario
+	Más unario

Tabla 4: Sinóptico de operadores unarios

Orden de operadores

Los operadores de la tabla 5 están posicionados según su jerarquía desde el rango más alto (arriba) hasta el rango más bajo (abajo). Todos los operadores que se encuentran en una línea tienen la misma jerarquía y, cuando se presentan al mismo tiempo en una expresión, son evaluados por el Compilador en un orden cualquiera.

Jerarquía	Operador	Agrupación (asociatividad)
alta	() [] -> .	de izquierda
	! ~ ++ -- - (*typ*) * & sizeof	de derecha
	* / %	de izquierda
	+ -	de izquierda
	<< >>	de izquierda
	< <= > >=	de izquierda
	== !=	de izquierda
	&	de izquierda
	^	de izquierda
	|	de izquierda
	&&	de izquierda
	||	de izquierda
	?:	de derecha
	= += -= +=	de derecha
baja	,	de izquierda

Tabla 5: Sinóptico de la jerarquía de operadores

Nota

Si se desea ejecutar primero una operación de jerarquía baja, esta expresión se debe marcar mediante parentesis.

En parte se emplean los mismos signos de operador para diferentes operadores:

Signo de operador	Interpretación 1	Interpretación 2
()	Paréntesis	type cast
*	Declaración de puntero	Multiplicación
-	Signo	Sustracción
&	Dirección de ...	AND (Y) bit a bit

Tabla 6: Operadores seleccionados y sus diferentes significados

Asignaciones

El operador = permite asignar valores a variables y a constantes.

```
a = b;        // a “a“ se asigna el valor de b
x = 72;       // a “x“ se asigna el valor 72
```

Operadores de aritmética

Para las operaciones fundamentales se dispone de los operadores +, -, *, /. Adicionalmente existe un operador de módulo *%* , con el cual se puede determinar el resto de una división, así como la negación aritmética – como operador unario. No hay signo para números positivos.

```
a = b * c;          // El valor de a resulta de la multiplicación de b con c
c = b % 10;         // El valor de c resulta de una operación de módulo
```

¡El operador de módulo no se puede aplicar a los tipos de datos *float* y *double*!

Operadores de incremento y decremento

En C hay un operador de incremento ++ y un operador de decremento --. Estos operadores unarios pueden figurar **delante** o también **detrás** del operando. Con ello se consigue un tratamiento diferente del operando.

Ejemplos:

```
++i;                // primero se incrementa el operando i,
                    // y luego se sigue empleando

i--;                // primero se emplea el operando i,
                    // y luego se decrementa
```

Comparaciones y operaciones lógicas

Los operadores *inferior a* (<), *inferior igual a* (<=), *superior a* (>), *superior igual a* (>=) e *igual a* (==), *no igual a* (!=) están disponibles para **comparaciones**. VERDAD o FALSO se decide por medio del valor numérico.

Para **operaciones lógicas** de operandos se emplean los operadores *negación lógica* (!), *Y* (&&), así como *O* (||).

Nota:

La ejecución de la operación lógica se realiza en expresiones de izquierda a derecha hasta que se obtenga un resultado inequívoco. Por esta razón la condición de mayor prioridad debería posicionarse al principio de la expresión.

Manipulaciones de bits

Para las manipulaciones de bits se pueden emplear los operadores *Complemento* (~), *Desplazamiento a la izquierda* (<<), *Desplazamiento a la derecha* (>>), *AND (Y) bit a bit* (&), *XOR (O exclusivo) bit a bit* (^), así como *OR (O) bit a bit* (|):

Constante de *operador* variable

Variable de *operador* variable

¡Las manipulaciones de bits **no** pueden aplicarse a los tipos de datos *float* y *long*!

Operadores compuestos

Para la escritura abreviada de asignaciones pueden utilizarse en C operadores compuestos. Se debe observar la siguiente estructura:

Expresión1 *Operando* = Expresión2

Como *operandos* se pueden emplear +, -, *, /, %, <<, >>, &, ^, |.

Ejemplo: *x* **+=** *20;* *// equivalente a x = x + 20*

El operador de coma

Con el operador de coma se combinan varias expresiones. Se puede emplear, por ejemplo, en la declaración de variables para colocarlas una al lado de la otra, separadas por una coma.

int a,b,c;

Tipos de datos estructurados

Campos

Un campo es una combinación de variables del mismo tipo de datos. El primer elemento de campo posee el índice *0*. A continuación se representa la estructura general de un campo:

Tipo de campo Nombre de campo[tamaño de campo];

Ejemplo:

Campo int[5]={7,9,10,0,0}; ó *Campo int.[]={7,9,10,0,0};*

Con esta declaración se reserva espacio de memoria para un campo del tipo *Integer* y con un número de elementos de 5 por el Compilador. Ambas inicializaciones son posibles, determinándose en la segunda variante el tamaño de campo con la inicialización de los elementos de campo.

Elemento	0	1	2	3	4
Contenido	7	9	10	0	0

Figura 2: Campo del tipo Integer y contenido de los elementos de campo

A los distintos elementos de campo [0] .. [4] se les puede asignar un valor mediante acceso indexado.

Campo[4] = 0x0fa1; *// Al elemento de campo 4 se le asigna el valor 4001*

x = Campo[4]; *// A x se asigna el contenido del elemento de campo 4*

Estructuras

Las estructuras permiten combinar objetos de diferente tipo. Como componentes de una estructura son posibles variables, campos y también estructuras. Para la declaración se debe emplear la palabra clave *struct*:

```
struct Nombre
{
        Objeto1;
        Objeto2;
};
```

Nota:

La declaración de la estructura determina sus correspondientes componentes y no reserva espacio en memoria. Por lo tanto se debe definir adicionalmente y como mínimo una variable de tipo estructura y con ello el espacio en memoria:

```
struct Nombre Variable;
```

¡Esta ***variable*** contiene primero los objetos *Objeto1* y *Objeto2*!

También es posible realizar la declaración de la estructura y la definición de las correspondientes variables en un paso. La siguiente inicialización conduce al mismo resultado:

```
struct Nombre
{
        Objeto1;
        Objeto2;
}Variable;
```

Al acceder a distintos elementos dentro de la estructura se emplea el operador de punto .:

```
struct date                 // Declaración de la estructura “date“
{
   unsigned char day[7];    // Objeto “day”
   unsigned char month;     // Objeto “month”
   unsigned int year;       // Objeto “year”
}today;                     // Definición de una variable de tipo estructura

unsigned char i;            // Variable del tipo Character
unsigned int j;             // Variable del tipo Integer
…

today.day[0]=1;             // Elemento de campo 0 en elemento de
                            // estructura day = 1
today.month=12;             // Elemento de estructura month = 12
i=today.month;              // Asignar valor del elemento de estructura i
j=today.year;               // Asignar valor del elemento de estructura j
```

Puntero

Para acceder de forma indirecta a áreas de memoria, por ejemplo variables, campos o estructuras, se emplean punteros. ¡Estos punteros o pointer no contienen el valor de una localidad de memoria, sino solamente su dirección! La ventaja principal radica en la mayor velocidad de acceso a localidades de memoria porque no se trabaja con el contenido.

Declaración de puntero

Para la declaración de un puntero se debe utiliza el operador *. Adicionalmente hay que observar qué tipo de datos se deben señalar. A continuación se dan algunos ejemplos para declaraciones de puntero:

```
int     *número;        // Puntero en Integer
char    *caracter;      // Puntero en Character
float   *p              // Puntero en Float
```

Nota:

Después de la declaración solamente se ha creado una localidad de memoria para el puntero. ¡La dirección que señala el puntero no está definida! Por este motivo hay que transferir una dirección definida al puntero!

Operadores de puntero

El operador *&* (operador de referenciación) devuelve la dirección del operando:

```
int     x;              // Variable del tipo Integer (Operando)
int     *p;             // Puntero en Integer

p = &x;                 //  p recibe la dirección de x
```

Con el operador * (operador de dereferenciación) se accede al contenido del operando señalado:

```
int     x=10, y;        // Variables del tipo Integer
int     *p;             // Puntero en Integer

p = &x;                 //  p recibe la dirección de x
y = *p;                 // y recibe el valor señalado por p (x)
*p += 1;                // equivalente a  x = x+1, porque p señala a x
```

Punteros y campos

El acceso a las direcciones y al contenido de campos se realiza mediante un puntero que está asociado al mismo tipo de datos del campo. Puesto que el nombre de un campo se puede interpretar como puntero, no es necesaria una referenciación, o sea la transferencia de la dirección inicial del campo con el operador *&* al puntero con el que se desea operar. Por esta razón la dirección del campo de datos se puede entregar directamente a un puntero correspondiente. Sin embargo, el contenido de un elemento de campo direccionado solamente puede leerse y modificarse, es decir referenciarse, con el operador *.

Ejemplo:

```
números int[5];      // Campo del tipo Integer con 5 elementos
int *p;              // Puntero en Integer
p=números;           // p apunta al 1º elemento → números[0]
*p=10;               // Elemento  números[0] recibe el valor 10
```

Análogamente a esta asignación de valores del primer elemento también es posible al acceso indexado:

```
números[0]=10;       // Elemento números[0] recibe el valor 10
```

La expresión *Elemento de campo[i]* corresponde a la expresión **(puntero+i)*, debiendo ser *i* un número entero. La adición del número entero a la variable de puntero es interpretada por el Compilador como un múltiplo del tamaño del tipo de datos al que apunta el puntero. En el ejemplo el puntero *p* apunta al elemento de campo Integer *0*. Incrementando el puntero, éste apunta después al elemento de campo Integer con el índice *1*.

```
números int[5];      // Campo del tipo Integer con 5 elementos
int *p;              // Puntero en Integer
p=números;           // p apunta al 1º elemento de campo → números[0]
p++;                 // p apunta al 2º elemento de campo → números[1]
```

Punteros y cadenas de caracteres

Para la generación de cadenas de caracteres o Strings existen dos posibilidades:

```
char string[]={"Texto"};
```

o:

```
char *string="Texto";
```

En la primera variante se genera un **campo del tipo Character** con 5 caracteres y en la segunda variante se declara un **puntero del tipo Character** que apunta a la primera dirección de un área de 5 caracteres. Las cadenas de caracteres se archivan internamente de la siguiente manera:

T	e	x	t	\0

El Compilador añade a las cadenas puestas entre comillas un *\0* como indicativo de final. Por esta razón, los *strings* también se designan como cadenas de caracteres terminadas en cero.

La segunda variante ofrece adicionalmente la posibilidad de poder direccionar, con un campo de puntero, cadenas de caracteres de diferentes longitudes.

```
char *día[]=    {       "Lunes",
                        "Martes",
                        "Miércoles",
                        "Jueves",
                        "Viernes"
                };
```

Para el procesamiento de cadenas de caracteres, el Compilador pone a disposición funciones de biblioteca que se pueden emplear tras la inclusión del archivo de encabezamiento *string.h* en el archivo fuente. En la siguiente tabla se representan funciones de biblioteca seleccionadas para el procesamiento de cadenas de caracteres:

Función	Descripción
char *strcpy(char *destination,const char *source)	copia `source` a `destination` incluyendo el `\0 final`. Se devuelve `destination`
char *strncpy(char *destination,const char *source,size_t n)	copia como máximo `n` caracteres del string `source` zu `destination` y devuelve `destination` terminando, como es habitual, con `\0`
char *strcat (char *destination,const char *source)	cuelga `source` a `destination` y devuelve `destination` .
`char *strncat(`char *destination,const char *source,size_t n)	cuelga como máximo `n` caracteres de `source` a `destination`, termina con `\0` y devuelve `s`.
`int strcmp(`const char *a,const char *b)	compara `a` y `b`. El valor de devolución es 0, si los dos strings son idénticos, negativo si a<b y positivo si a>b, entendiéndose el < y > en el sentido léxico, es decir se compara letra a letra y se comprueba cuál figura primero en el alfabeto teniendo prioridad las mayúsculas ante las minúsculas.
`int strncmp(`const char *a,const char *b,size_t n)	En lo esencial como antes, pero se comparan como máximo `n` caracteres.
`char *strchr(`const char *string,int c)	devuelve puntero, para la primera presentación de `c,` a `string`, o `CERO` si no se presenta.
`char *strstr(`const char *s,const char *t)	devuelve puntero para la primera presentación del string `t` a `s` ó `CERO` si no se presenta.
`size_t strlen(`const char *str)	devuelve la longitud del string `str` sin \0

Tabla 7: Funciones de biblioteca seleccionadas de string.h

Control de programa y sentencias de control

Para estructurar un problema de forma sinóptica, se deben emplear sentencias de control para el control del programa. Estas sentencias están divididas en selección y bucles.

En la selección, el flujo del programa se ramifica dependiendo del grado de verdad de una condición.

El bucle también se denomina como repetición. Mediante una condición se decide si se continua con el programa o si se vuelve a entrar en el bucle.

La sentencia *if*

La sentencia *if* sirve para la ramificación del programa por medio de la selección de una ó varias condiciones. Esta(s) condición(es) puede(n) ser una expresión aritmética o lógica.

La sentencia se representa de la siguiente manera:

```
if (condición) instrucción x;
else instrucción y;
```

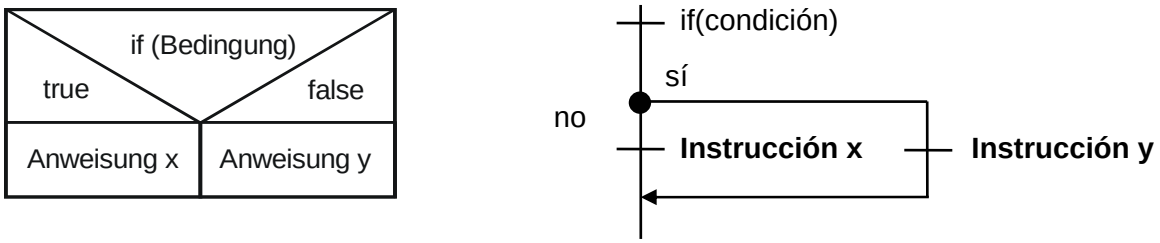


Figura 3: Estructograma if y diagrama de flujo

Si en la ramificación de *if* o de *else* se deben ejecutar varias instrucciones, éstas se deben agrupar con llaves en un bloque.

```
if(condición)
{
        Instrucción1;
        Instrucción x;
}
else
{
Instrucción2;
        Instrucióny;
}
```

La ramificación de *else* es opcional y puede suprimirse dependiendo del desarrollo del programa. Entonces no se ejecuta un tratamiento alternativo si no se cumple la condición.

La sentencia *switch*

La sentencia *switch* representa una ampliación de la sentencia *if*. Mediante *switch* se puede realizar una selección múltiple. Como ejemplo se nombra la ramificación de programa mediante una variable que puede adoptar varios valores. La sentencia se representa de la siguiente manera:

```
switch (expresión de prueba)
{
        case const. Expresión1:     Instrucción x;
                                    ...

        case const. Expresión x:    Instrucción y;
                                    ...
}
```

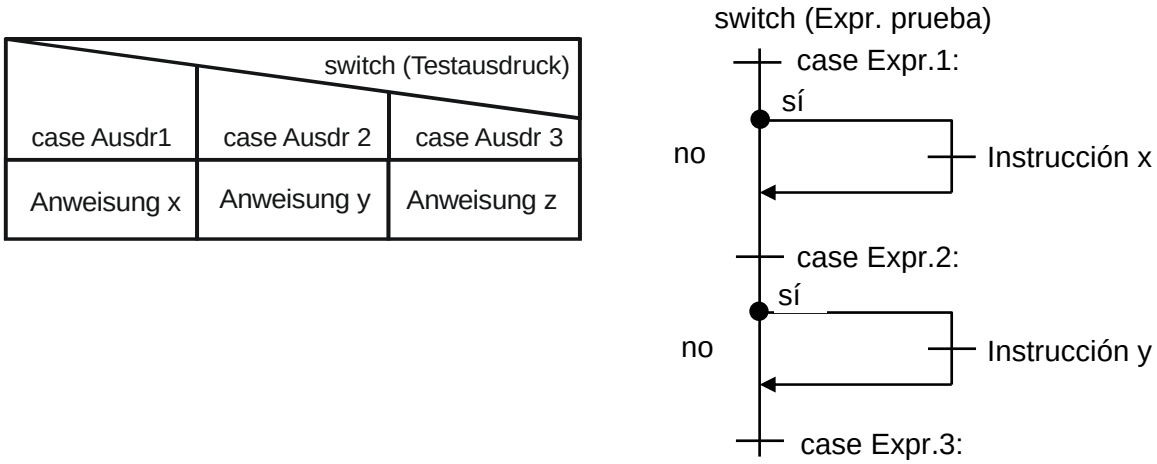


Figura 4: Estructograma switch y diagrama de flujo

El estado de la expresión de prueba se compara dentro del bloque de* switch *de arriba a abajo con cada expresión constante. Si hay igualdad, se realiza el procesamiento de las instrucciones correspondientes a la ramificación case correspondiente.

Nota:

¡La ejecución de la sentencia *switch* no finaliza automáticamente tras haber encontrado una expresión de comparación! ¡Para ello hay que aplicar la instrucción *break* después de cada ramificación de *case*!

```
switch (Expresión de prueba)
{
        case const. Expresión1:     Instrucción x;
                                    break;
        case const. Expresión x:    Instrucción y;
                                    break;
        default:                            Instrucción z;
                                    break;
}
```

La ramificación *default* es opcional, realizándose el procesamiento de las instrucciones contenidas en ella solamente si no ha habido coincidencia con las expresiones constantes. Por lo tanto debería introducirse al final del bloque *switch*.

La sentencia *while* / *do-while*

Las sentencias *while* y *do-while* permiten la repetición de una ó varias instrucciones mediante una ó varias condiciones.

La sentencia *while* es un bucle evaluado al principio porque primero se comprueba la condición y luego se procesa el bloque de instrucciones. A continuación se representa la síntaxis de la sentencia *while*:

```
while (condición)
{
        Instrucción x;
        ...
}
```

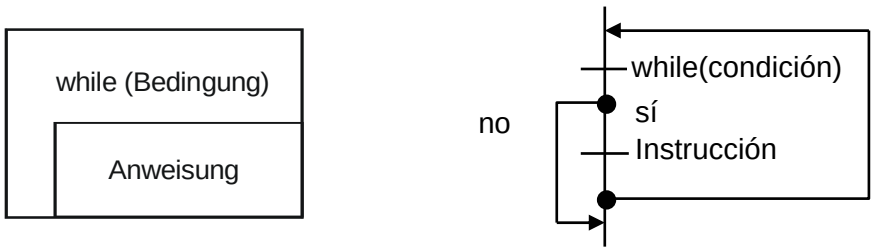


Figura 5: Estructograma while y diagrama de flujo

La sentencia *do-while* es un bucle evaluado al final porque primero se recorre el bloque de instrucciones y luego se comprueba la condicón. El bucle presenta la siguiente síntaxis:

```
do
{
        Instrucción x;
        ...
}
while(Condición);
```

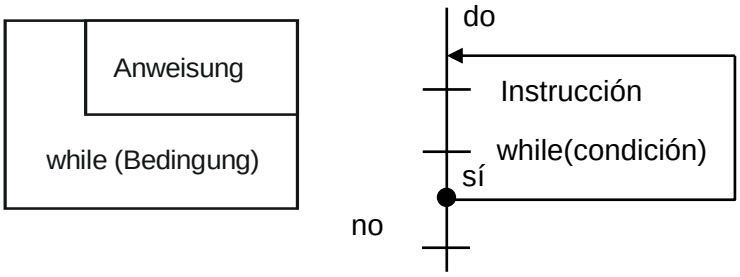


Figura 6: Estructograma do-while y diagrama de flujo

Nota:

¡La línea *while(condición);* de la sentencia *while* evaluada al final siempre se debe terminar con un punto y coma!

La sentencia *for*

La sentencia *for* permite la realización de bucles de conteo. Es un bucle evaluado al principio porque la condición de bucle siempre se realiza antes de ejecutar las instrucciones en el cuerpo del bucle. El bucle posee la siguiente síntaxis:

```
for (Expresión1 ; Expresión2 ; Expresión3)
{
        Instrucción x;
        Instrucción y;
        ...
}
```

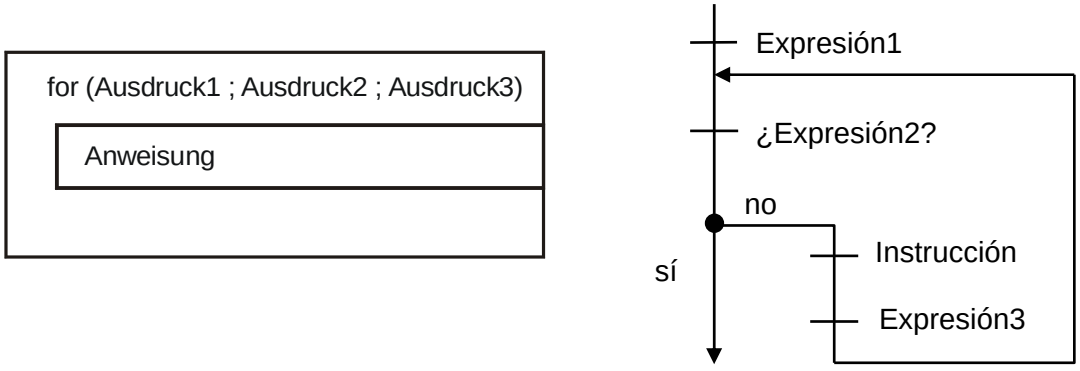


Figura 7: Estructograma for y diagrama de flujo

Al entrar en el bucle *for* se realiza, en primer lugar, una inicialización única (*Expresión1*), por ejemplo una asignación del valor de una variable contadora. A continuación se realiza la comprobación de la condición de bucle (*Expresión2)*. Después del procesamiento de las instrucciones en el cuerpo de bucle se ejecuta una última instrucción de bucle (*Expresión3*). En el siguiente paso se realiza una nueva comprobación de la condición de bucle (*Expresión2*).

Ejemplo:

```
for(i = 0; i  < 7; i ++)
{
        Instrucción;
        ...
}
```

Instrucciones de preprocesador

Antes de la traducción del código fuente C se llama al preprocesor. Éste inserta los archivos de encabezamiento y define los símbolos, así como los macros. Las instrucciones de preprocesador están marcadas con una almohadilla (#) antepuesta y no se terminan con el punto y coma. Se pueden emplear las siguientes instrucciones de preprocesador:

Instrucción	Significado
#include	Inserta los códigos de programa de un archivo de encabezamiento
#define	Define símbolos, macros
#undef	Anula la definición de macro y de símbolo
#if	Ramificación como función de una expresión
#elif	Otra alternativa para #if
#ifdef	Ramificación como función de una constante de preprocesador
#ifndef	Ramificación como función de una constante de preprocesador
#else	Alternativa para #if, #ifdef, #ifndef
#endif	Terminación para #if, #ifdef, #ifndef
#error	Mensaje de error
#pragma	Acción específica del fabricante

Figura 8: Instrucciones de preprocesador

La instrucción *#include* se emplea con la mayor frecuencia. Mediante esta instrucción es posible la estructuración y con ello la división del código de programa en varios archivos. Los llamados archivos de encabezamiento deberían contener de forma exclusiva las funciones de biblioteca. Existen bibliotecas estándar prefabricadas que también pueden ser específicas del Compilador. Como ejemplo se nombran funciones matemáticas en *math.h* o también funciones estándar de entrada y salida como *printf()* y *scanf()* del archivo de encabezamiento *stdio.h*. Adicionalmente es posible la creación y la inserción de archivos de encabezamiento propios, como por ejemplo para funciones para la activación de periferia externa.

Nota:

La instrucción *#include* se puede llamar de dos maneras diferentes:

- *#include <header.h>*

En este método, el archivo de encabezamiento se encuentra entre <antilambda> y se busca en el directorio estándar*...\inc* del Compilador. Se emplea casi siempre para las bibliotecas de funciones prefabricadas.

- *#include "header.h"*

Con esta llamada se buscan archivos de encabezamiento – el nombre está entre comillas – en el actual directorio de proyectos. Esta forma se emplea para insertar las bibliotecas de funciones propias.

Biblioteca ANSI

El lenguaje de programación C fue estandarizado en 1988 por el "American National Standard Institute" (ANSI). En general, la definición de lenguaje resultante se denomina estándar ANSI o ANSI-C.
La ventaja de la estandarización reside en la portabilidad de programas de un sistema a otro sistema de destino. A pesar de la estandarización, C posee algunos elementos de lenguaje dependientes de la implementación que pueden mostrar un comportamiento diferente con respecto a diferentes compiladores.
Un ejemplo para dependencias de implementación son los márgenes de valores de variables. Algunos compiladores procesan una variable del tipo Integer con un tamaño de 16 bits, otros de 32 bits. Gracias a la limitación a un margen de valores mínimo garantizado por el estándar ANSI se pueden reducir los problemas de transporte.
Las funciones definidas en el estándar ANSI representan un complemento adecuado del volumen original del lenguaje C. En la tabla 2.9-1 se indican archivos de encabezamiento seleccionados del estándar ANSI y su contenido de funciones con ejemplos. La descripción completa de las funciones C estándar se encuentra en la ayuda del entorno de desarrollo.

Archivo de encabezamiento	**Contenido de funciones**	**Ejemplos de funciones**

stdio.h	Funciones de entrada y salida	getchar();
		putchar();
		printf();
		scanf();

ctype.h	Funciones para la clasificación de caracteres	isalnum();
		isalpha();
		isascii();
		iscntrl();

string.h	Funciones para el procesamiento de cadenas de caracteres	strcpy(); strcmp();

math.h	Funciones matemáticas adicionales	sqrt();
		log();
		sin();
		cos();
		tan();

stdlib.h	Funciones auxiliares	atoi();
		atol();
		atof();
		itoa();

Tabla 8:Archivos de encabezamiento seleccionados de la biblioteca ANSI

Programación C para Sistemas Integrados

La programación C para *Sistemas Integrados (Embedded Systems)* con microcontroladores (MC) presenta algunas particularidades en comparación con la programación C para Estaciones de trabajo (Workstations). Éstas resultan por una parte de la multitud de posibles aplicaciones con MC y por otra de la necesidad de poder acceder directamente al hardware implementado del MC (memorias, puertos, temporizadores, ...). También el número muy grande y siempre en aumento de diferentes familias de MC ha dado lugar a gran número de compiladores C en el mercado, debiendo los compiladores ser adaptables a los diferentes derivados de una familia MC.

Otra diferencia esencial reside en el hecho de que el desarrollo de programas para un *Sistema Integrado* se realiza normalmente para un nuevo diseño de hardware. Allí se debe partir de fallos tanto en el hardware como en el software. También cabe la posibilidad de que un compilador nuevo presente problemas, de forma que la puesta en servicio y la optimación de un sistema nuevamente desarrollado se convierte en un desafío para el diseñador. Unos buenos conocimientos del hardware y software, así como una metódica meditada respecto a la puesta en servicio y a la depuración constituyen las condiciones previas para un trabajo con éxito.

En la programación C para *Sistemas Integrados* son necesarias adaptaciones o ampliaciones específicas del microcontrolador frente al estándar C ANSI. Estos cambios limitan el transporte del programa para diferentes sistemas de destino, pero resultan imprescindibles. Especialmente en caso de recursos limitados del MC, estas ampliaciones de lenguaje resultan indispensables para poder programar ahorrando en recursos. En detalle, ello se refiere a las siguientes áreas de problema:

- Localización de datos y códigos
- Apoyo del procesamiento de bits individuales y utilización de variables de bits
- Variables en registros
- Tipos de direccionamiento especiales
- Modelos de memoria específicos
- Control de sistema a nivel de interrupción

Ejemplos concretos para estas adaptaciones (ampliaciones C ANSI) son los siguientes:

- Tipos de datos adicionales, p.ej. *sfrb*
- Clases de memoria adicionales para variables, p. ej. *tiny*
- Especificación de funciones, p. ej. *interrupt*
- Directivas de control para conectar y desconectar ampliaciones de lenguaje

Las características esenciales de los compiladores C para MC son sobre todo el apoyo confortable del procesamiento de bits individuales, la utilidad especial de las diferentes zonas de memoria on chip y el apoyo de interrupciones de hardware.

Los fabricantes de MC actuales apoyan mediante características de diseño especiales la posibilidad de generar códigos compactos por medio del compilador C. Como ejemplo mencionamos la familia AVR de ATMEL que a ese respecto dispone de características de arquitectura óptimas. Así, el Core dispone de más de 32 registros de trabajo universalmente utilizables pudiéndose emplear cada registro como fuente o como destino de datos. Ello

permite ahorrar comandos de transporte después de la ejecución de operaciones lógicas e aritméticas. Puesto que todos los registros de trabajo también están representados en el área de direcciones lineales, se puede realizar un acceso conforme a C a estos registros por medio de punteros. Estas y otras características del AVR-Core permiten códigos compactos y con ello también rápidos. Estas ventajas no se dan de esta forma en el 8051-Core.

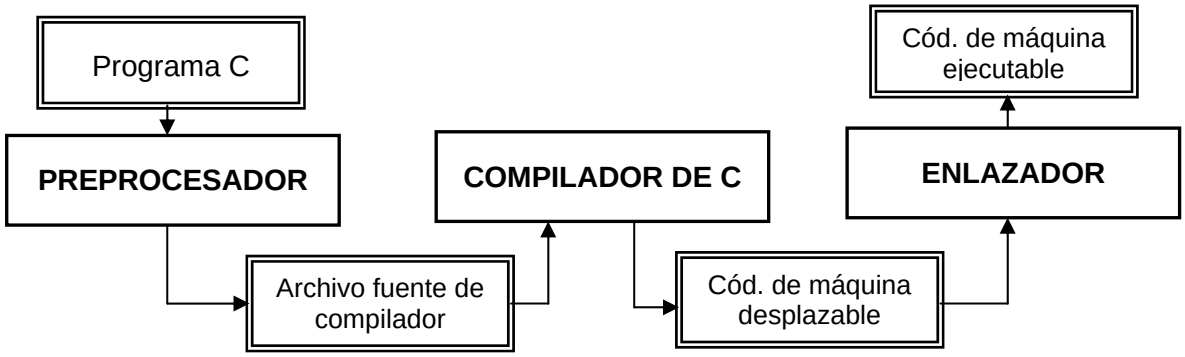


Figura 9: Principio de desarrollo del proceso de compilación

El principio de desarrollo de un proceso de compilación mostrado en la figura 8 se da tanto en la programación C para Estaciones de Trabajo como para *Sistemas Integrados*. Sin embargo, aparte de las características de diferenciación ya mencionadas se añaden, en la fase de generación de códigos por el enlazador, otras particularidades en la programación C para MC:

1. Insertar un *código startup*

Antes de iniciarse la función main(), se ejecuta el llamado programa startup. Con la ayuda de este programa de Ensamblador se realizan las inicializaciones básicas necesarias de la pila (stack), de la memoria de trabajo, de los registros, las variables y de la periferia on chip necesaria. Un programa de Ensamblador Startup se suministra por el fabricante del Compilador y puede adaptarse a requisitos especiales. Normalmente, el módulo Startup está enlazado con los módulos C.

2. Inserción de funciones de biblioteca específicas del MC

Para el acceso efectivo a la periferia on chip del MC se encuentran preparadas funciones especiales o macros. También existen, en parte, funciones equivalentes a la biblioteca ANCI-C, p. ej. para el acceso a las cadenas de caracteres. Para que estas funciones se puedan utilizar, se deben insertar los correspondientes archivos de encabezamiento por medio de la directiva *#include*. Los archivos con las bibliotecas de funciones se insertan por el preprocesador; el enlazado de las funciones utilizadas con el programa ejecutable se realiza más tarde por el Enlazador.

Para obtener más detalles respecto a las funciones de biblioteca implementadas, consulte *libdoc.txt* en el directorio de instalación *..\SDCC\doc* del Compilador SDCC. De gran ayuda para la activación de hardware externo son funciones de biblioteca especialmente preparadas, como por ejemplo displays de LCD, interfaces y sensores que se explican más detalladamente en el capítulo H.

El entorno de trabajo y el Compilador de C

Sinópsis

***Para la programación de los ensayos de este manual, se utiliza el lenguaje de alto nivel C, junto con el Compilador Small* Device C Compiler SDCC *que está protegido por licencia pública general. Esta herramienta está orientada hacia líneas de comandos y comprende Compilador de C, Ensamblador y Enlazador para diferentes microcontroladores de 8 bits, p. ej. derivados de las familias 8051, Z80, así como Dallas DS80C390. El microcontrolador estándar es, en este caso, la familia 8051. El Compilador está insertado en el IDE del* MCLS-modular®.**

El proceso de compilación se puede iniciar con diferentes opciones. En el IDE del* MCLS-modular® *se dan los siguientes ajustes:

- ***--debug*** → ***Generar información de depuración***
- ***--verbose*** → ***Visualizar las acciones ejecutadas***
- ***--model-large*** → ***Seleccionar el modelo de memoria del destino utilizado (memoria externa)***
- ***--xram-loc 0x5000*** → ***Memoria de trabajo externa desde la dirección 0x5000***
- ***--code-size 0x4fff*** → ***Tamaño de código máximo 0x4fff***

El usuario dispone de otros parámetros de línea de comando que sin embargo no son relevantes para el siguiente programa de ensayos.

Creación de proyecto en el MCLS-IDE

El IDE del* MCLS-modular® *reúne, en una superficie de usuario, las herramientas de desarrollo necesarias para la creación del programa. Se encuentran integrados los siguientes componentes:

- Editor de texto fuente
- Compilador de C, Ensamblador y Enlazador
- Depurador
- Administrador de archivos

Un programa de aplicación es descargado al sistema de destino mediante un software de depuración incluido en el IDE.

Secuencia para crear un proyecto:

1. Crear un nuevo proyecto en el MCLS-IDE
2. Crear e insertar el archivo fuente principal en el proyecto
3. Editar y compilar el texto fuente C propio

Una vez finalizada la instalación del software, IDE se iniciará desde el correspondiente ícono en el escritorio o desde el menú de inicio de Windows *Inicio* → *Programas* → *MCLS-modular*. Las instrucciones de instalacióin resumidas se pueden consultar en la página de ayuda del Website del *MCLS-modular* bajo www.mcls-modular.de y también en la copia incluida en la CD de instalación.
Una parte muy importante la constituye el administrador de módulos que le permite la integración de actualizaciones o de módulos adicionales. Estos módulos adicionales son ficheros especialmente comprimidos en formato CAB. Después de la instalación, estos ficheros se archivan en la carpeta "Módulos". Nuevos módulos adicionales pueden descargarse de Internet en la opción Downloads. Una vez descargados, estos módulos solamente deben copiarse al directorio "Módulos". Nunca abra o descomprima estos módulos mediante un doble click.
Con el nuevo inicio de la aplicación IDE, el administrador de módulos reconoce independientemente la presencia de un nuevo módulo y le ofrecerá la integración.

Nota:
Con el módulo Flash PSD1 se pusieron a disposición dos nuevos pérfiles integrados en forma de módulos adicionales. En el supuesto de que no estuviesen disponibles en el administador de módulos, se deberán copiar desde las fuentes arriba indicadas al directorio "Módulos". Esto se refiere al módulo "PSD1-C515C-AS" con todas las herramientas y el perfil integrado del mismo nombre para la programación del Ensamblador del C515C y al módulo "PSD1-C515C-SDCC" con todas las herramientas y el perfil integrado del mismo nombre para la programación C del C515C.

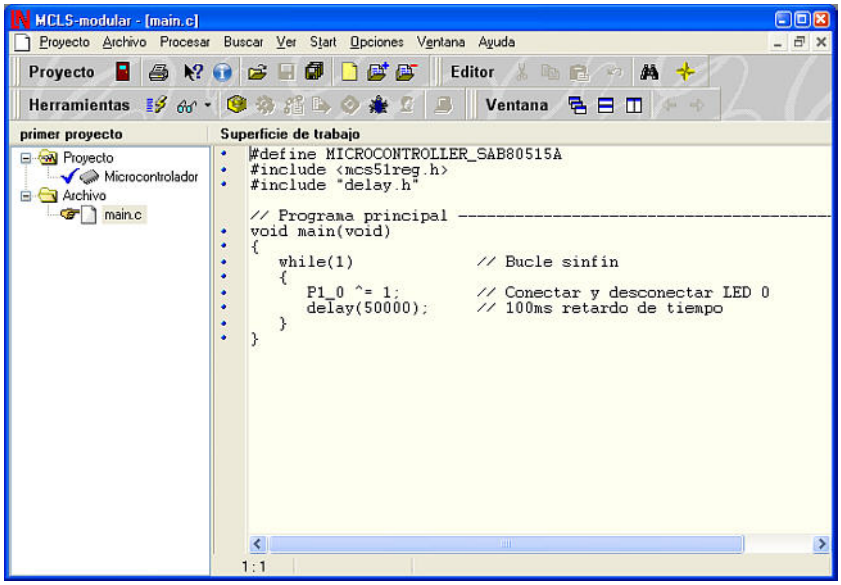


Figura 10: IDE del *MCLS-modular*®

El IDE está dividido en tres áreas. A la izquierda se encuentra la ventana del *Navegador*, a la derecha la ventana para la visualización del editor del texto fuente. Por medio de la barra de menús y los botones visualizados se pueden iniciar las diferentes funciones del entorno de desarrollo.

Crear un proyecto nuevo

Primero se debe iniciar el IDE del MCLS-modular® (*Inicio → Programas → MCLS-modular*). En la selección de perfiles para el microcontrolador se debe seleccionar el perfil integrado *PSD1-C515C-SDCC*.

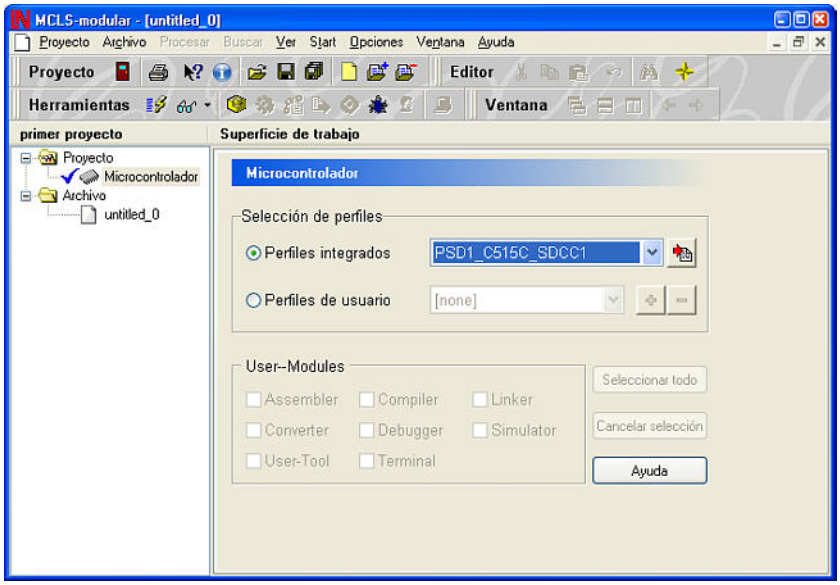


Figura 11: Nuevo proyecto con selección de perfiles

A continuación, el proyecto se debe guardar en una carpeta opcional. Crear e insertar el archivo fuente principal

Para generar un programa ejecutable, se debe crear un archivo fuente principal para insertarlo en el proyecto por medio de la opción de menú *Archivo → Archivo nuevo*.

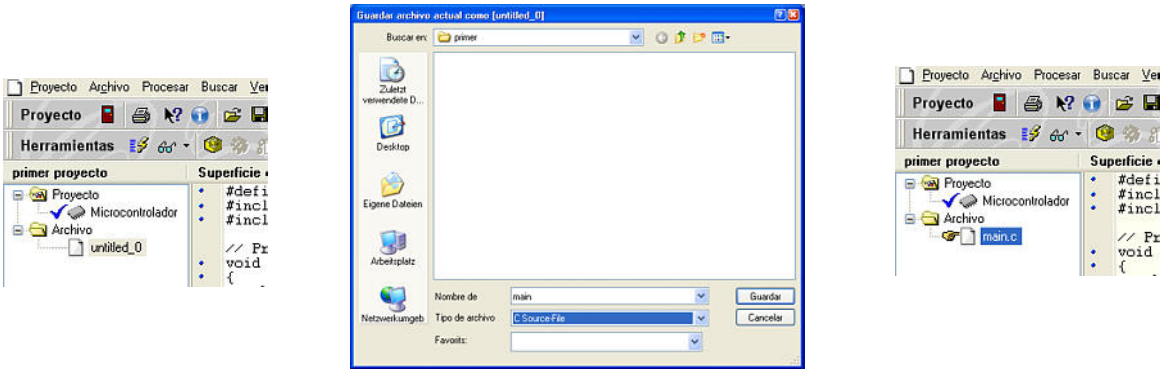


Figura 12: Guardar el archivo fuente principal

Este archivo creado debe guardarse con la terminación *.c* en el directorio de proyectos y colocarse como *archivo principal (mainfile)* en la ventana de navegación pulsando la tecla derecha del ratón.

Nota:
¡Al guardar el archivo fuente, es imprescindible ajustar ***C-Source-File*** en la ventana Tipo de archivo! En otro caso podría darse el caso de que se guarden incorrectamente terminaciones de archivos lo que impide el proceso de compilación.

Editar y compilar el texto fuente C propio

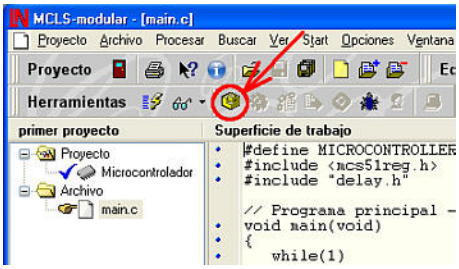


Figura 13: Inicio del Compilador

Dentro del archivo fuente principal, en el ejemplo *main.c*, es posible editar la propia ejecución del programa. Una vez finalizada la creación del texto fuente, se hace necesaria la traducción del mismo. A ese fin se seleccionará el botón *Compilador*.
Una vez finalizado el proceso de compilación, pueden mostrarse en la ventana del navegador *Mensajes* → *Compilador* los errores o las indicaciones de aviso del proceso de traducción.
Una vez eliminados los errores y después de la nueva llamada al Compilador se genera un archivo *.ihx ejecutable.

Programar el destino

Figura 14: Llamar al Depurador

El Compilador genera en el proceso de traducción un archivo ihx que contiene el código de programa ejecutable y que puede cargarse en el RAM externo del módulo FLASH PSD1. A ese fin se debe llamar, por medio del botón *Inicio Depurador*, al software de depuración.

El archivo ejecutable se transmite, con el inicio del software, automáticamente al RAM del módulo FLASH PSD1.

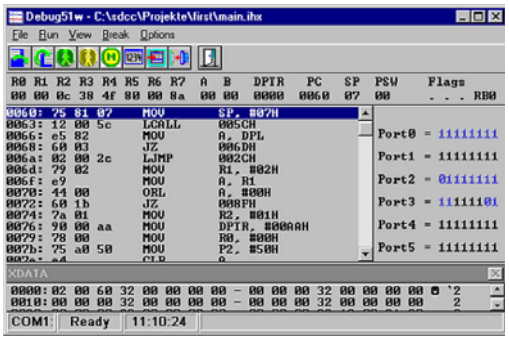


Figura 15: Depurador del MCLS-IDE

A continuación es posible la depuración del texto fuente. Con la ayuda de diferentes comandos de operación puede efectuarse la puesta en servicio o el ensayo del funcionamiento del programa. A ese fin se deberán emplear los siguientes comandos:

- Paso individual (Single Step) ***F7***
- Paso de proceso (Single Proc) ***F8***
- Run ***F9***
- Reset ***CTRL + F3***
- Act./Desact. punto interrupc. ***CTRL-B/ CTRL-E***

La activación o la desactivación de un punto de interrupción (Breakpoint) en la posición actual del cursor en la ventana del texto fuente es posible por medio del comando ***CTRL-B/ CTRL-E***. Con el comando ***F9*** (Run) se detiene la ejecución del programa en este punto.

Bibliotecas de funciones

Para los ensayos prácticos se ponen a disposición las siguientes bibliotecas de funciones (archivos de encabezamiento):

delay.h	→	Función para la generación de retardos de tiempo de software
intbcd.h	→	Funciones para convertir una variable del tipo de datos *unsigned int* en dígitos de BCD individuales
7seg.h	→	Funciones para controlar la UNIDAD INDICADORA 1 de 4 dígitos
iic.h	→	Funciones para controlar aparatos de I²C por medio del software
lcd.h	→	Funciones para controlar la UNIDAD de LCD de I^2C
rtc.h	→	Funciones para controlar el reloj de tiempo real de I²C DS1307 (UNIDAD RTC-Temp. de I^2C)
lm75.h	→	Funciones para controlar el sensor de temperatura de I²C LM75 (UNIDAD RTC-Temp. de I^2C)
dau.h	→	Generar una frecuencia triangular mediante la UNIDAD DA

delay.h

Función	Descripción	Transferencia de datos
delay	Generar retardos de tiempo	*unsigned int delaytime*

Tabla 9: Función *delay*

Descripción de la función

La función *void delay(unsigned int delaytime)* crea un bucle de retardo de tiempo de forma equivalente a los ejercicios de Ensamblador CMC1 y CMC3. A la función está asociada la variable de transferencia *delaytime* del tipo de datos *unsigned integer*, que está insertada en la función como variable contadora de 16 bits y que se utiliza para generar el retardo de tiempo.

Ejemplo para la llamada a la función:

```
delay (500);                    // Retardo de tiempo 1ms
```

Notas:
La ejecución de la función es realizada con el Ensamblador en línea porque ello permite una estimación más exacta del retardo de tiempo del software. La variable de transferencia es dividida por el Compilador en una parte baja de 8 bits y una parte alta de 8 bits. La parte baja está en el registro *dpl*, y la parte alta en el registro *dph* del 8051-Core. Con ello, el valor transferido se puede utilizar posteriormente en la sección del Ensamblador de la función de la siguiente manera:

```
Z00:   DJNZ  dpl,Z00      ; decrementa dpl, si no igual a 0 salta a la marca Z00
       DJNZ  dph,Z00      ; decrementa dph, si no igual a 0 salta a la marca Z00
```

Una estimación del tiempo de ejecución para la sección del Ensamblador puede realizarse con la siguiente fórmula:

*Retardo de tiempo = Valor transferido * 2µs*

Puesto que el Compilador inserta comandos adicionales para la ejecución de la función *delay(...);*, la estimación del tiempo de ejecución solamente tiene sentido para valores transferidos en el margen de entre 250 y 65535.
En la siguiente tabla se indican retardos de tiempo típicos:

Valor transferido *delaytime*	Retardo de tiempo resultante
250	0,5ms
500	1ms
750	1,5ms
1000	2ms
2500	5ms
5000	10ms
7500	15ms
10000	20ms
25000	50ms
50000	100ms

Tabla 10: Retardos de tiempo típicos

intbcd.h

Función	Descripción	Transferencia de datos
intbcd	Convertir un Integer en variables individuales	*unsigned int help*

Tabla 11: Función *intbcd*

Descripción de la función

La función *void intbcd(unsigned int help)* divide la variable transferida del tipo *unsigned integer* en distintas variables del tipo *unsigned character*. Estas variables para unidades, decenas, centenares, miles, así como decenas de miles están declaradas en las bibliotecas como variables globales. Una vez finalizada la función, los valores de las variables pueden seguir procesándose en otras secciones del programa.

Ejemplo para la llamada a la función:

```
unsigned int x = 45987;          // Variable con el valor decimal 45987

intbcd(x);                       // División de la variable x
```

Variable	Valor de resultado decimal	Valor de resultado hexadecimal
bcd1	7	0x07
bcd10	8	0x08
bcd100	9	0x09
bcd1000	5	0x05
bcd10000	4	0x04

Tabla 12: Resultados en variables

7seg.h para la UNIDAD INDICADORA 1

Función	Descripción	Transferencia de datos
seg_init	Inicialización de la UNIDAD INDICADORA 1	sin parámetros
seg_out	Visualización de un número transferido en un lugar definido de la indicadora 0 .. 3	*unsigned char out* *unsigned char seg*

Tabla 13: Funciones para UNIDAD INDICADORA 1

Descripción de la función

Función *void seg_init(void)*

Con la llamada a la función *seg_init* se apagan los cuatro dígitos de indicación de la UNIDAD INDICADORA 1.

Ejemplo para llamada a la función:

```
seg_init();          // borra los 4 dígitos de la indicadora
```

Función *void seg_out(unsigned char out,unsigned char seg)*

Con la función *seg_out* se puede visualizar un número entre 0 .. 9 en el dígito de 7 segmentos transferido 0 .. 3 en la UNIDAD INDICADORA 1. Los parámetros de transferencia son las variables *out* y *seg* del tipo *unsigned char*.

Para la variable *seg* se deben ajustar los siguientes valores:

0 → *Dígito de indicadora 0*
1 → *Dígito de indicadora 1*
2 → *Dígito de indicadora 2*
3 → *Dígito de indicadora 3*

La variable *out* puede asignarse con valores entre 1 y 9 decimal. Adicionalmente es posible borrar el dígito de 7 segmentos transferido 0 .. 3 con el valor de transferencia *blank*.

Ejemplos para la llamada a la función:

```
seg_out(9,0);          // visualiza el número 9 en el dígito de indicadora 0

seg_out(blank,3);      // borra el dígito de indicadora 3

unsigned char x = 4;   // Variable con el valor 4
seg_out(x,2);          // visualiza la variable x en el dígito de indicadora 2
```

iic.h para el interfaz de I²C del software

***Las funciones representadas en la siguiente tabla forman la base para otras bibliotecas para la comunicación del módulo FLASH PSD1 con componentes de I²C. Por parte del hardware están definidos el* Pin de puerto 5.0 = SDA *y el* Pin de puerto 5.1 = SCL.**

Función	Descripción	Transferencia de datos
IIC_INIT	Inicialización del Puerto 5.0 y 5.1	sin parámetros
IIC_START	Condición de inicio de I²C	sin parámetros
IIC_STOP	Condición de parada de I²C	sin parámetros
IIC_SEND	Enviar 1 byte al esclavo	*unsigned char buffer*
IIC_RECEIVE	Recibir 1 byte	Valor de retorno *unsigned char receive*
IIC_ACK	Comprobar la condición ACK (SDA=0)	sin parámetros
IIC_ACK_SEND	Enviar Ackknowledge	sin parámetros
IIC_NO_ACK_SEND	Enviar NO-Ackknowledge	sin parámetros

Tabla 14: Funciones para la comunicación I²C

Descripción de la función

Función *void IIC_SEND(unsigned char sendbyte)*

El byte a enviar se transfiere a la función *IIC_SEND()* con la variable *buffer* del tipo *unsigned char* .
Ejemplos para llamadas a la función:

```
code unsigned char adr_lcd = 0x74;      // dirección de I²C constante del LCD

IIC_SEND(adr_lcd);                      // direccionar LCD

IIC_SEND(0x40);                         // enviar un byte de datos
```

Función *unsigned char IIC_RECEIVE(void)*

Para recibir un byte de datos se llama a la función *IIC*_RECEIVE. Esta función es del tipo *unsigned char* porque suministra un valor de retorno *receive* del tipo *unsigned char*.

Ejemplo para la llamada a la función:

```
unsigned char Variable;         // variable del tipo unsigned char

Variable = IIC_RECEIVE();       // recibir un byte de un esclavo direccionado
                                // guardar el byte recibido en "Variable"
```

Función *void IIC_ACK_SEND(void) / void IIC_NO_ACK_SEND(void)*

Al recibir un byte de datos, éste se debe confirmar con una señal Acknowledge o NO-Acknowledge. Estas secuencias se realizan con la llamada a la respectiva función. Las funciones no necesitan parámetros de transferencia y no crean valores de retorno.

Ejemplos para llamadas a la función:

```
Variable = IIC_REC();        // recibir un byte de un esclavo direccionado
IIC_ACK_SEND();              // confirmar con Acknowledge

Variable = IIC_REC();        // recibir un byte de un esclavo direccionado
IIC_NO_ACK_SEND();           // confirmar con NO-Acknowledge
```

lcd.h para la unidad LCD de I2C

Función	Descripción	Transferencia de datos
LCD_INIT	Inicialización del LCD	sin parámetros
LCD_gotoXY	Ajustar línea y columna del cursor LCD	*unsigned char line/column*
LCD_SEND_STRING	Visualizar cadena de caracteres en LCD	*unsigned char *ptr_char*

Tabla 15: Funciones de la unidad indicadora LCD

Descripción de la función

Función *void LCD_INIT(void)*
Con la llamada a la función *LCD_INIT* se inicializa la unidad indicadora LCD sin cursor y se pone en su estado inicial. En la dirección 0x00 del CG-RAM se archiva el carácter ° para la salida de temperatura. Mediante cambios dentro de la función se pueden realizar otros ajustes según la hoja de datos (*LCD_s_7123.pdf*). No es necesario transferir parámetros a la función y ésta no envía un valor de retorno.

Función *void LCD_gotoXY(unsigned char línea, unsigned char columna)*

La función LCD_gotoXY permite posicionar el cursor en un lugar definido en la unidad indicadora LCD de I²C. La dirección del cursor se compone de la dirección de inicio de la línea del LCD y del número del carácter.

Los parámetros de transferencia son, en consecuencia, las variables *line* y *column* del tipo *unsigned char*. Para la variable *line* se deben utilizar los siguientes valores:

1 = Línea de LCD 1 → Dirección de LCD 0x00
2 = Línea de LCD 2 → Dirección de LCD 0x20
3 = Línea de LCD 3 → Dirección de LCD 0x40

A la variable *column* se pueden asignar valores entre 1 y 12 decimal.

Ejemplo para una llamada a la función:

```
LCD_gotoXY(2,3);          // pone cursor en línea 2 carácter 3
```

Función *void LCD_SENDSTRING(unsigned char *ptr_char)*

La función *LCD_SENDSTRING(unsigned char *ptr_char)* se puede utilizar para la visualización de cadenas de caracteres terminadas en cero en la unidad indicadora LCD. Antes de llamar a la función de visualización, se debe transferir la dirección de inicio de una cadena de caracteres.

Ejemplo de una llamada a la función:

```
code unsigned char text[]={"ACMC"};     // Cadena de caracteres constante

LCD_SENDSTRING(text);                   // Llamada a la función para visualizar el
                                        // contenido de "text"
```

rtc.h para la unidad RTC-Temp. de I2C

En esta biblioteca se encuentran implementadas todas las funciones necesarias para la inicialización y la lectura de los registros de RTC, así como para la visualización de los datos de RTC en la unidad indicadora LCD de I2C. La siguiente tabla contiene el listado de las funciones disponibles:

Función	**Descripción**	**Transferencia de datos**
RTC_INIT	Inicialización e inicio de RTC	sin parámetros
SEC_IN	Leer registro de segundos de RTC	Valor de retorno *unsigned char rtc_data*
SEC_OUT	Visualizar segundos en LCD	*unsigned char data_out*
MIN_IN	Leer registro de minutos de RTC	Valor de retorno *unsigned char rtc_data*
MIN_OUT	Visualizar minutos en LCD	*unsigned char data_out*
STD_IN	Leer registro de horas de RTC	Valor de retorno *unsigned char rtc_data*
STD_OUT	Visualizar horas en LCD	*unsigned char data_out*
TAG_IN	Leer registro de días de RTC	Valor de retorno *unsigned char rtc_data*
TAG_OUT	Visualizar día en LCD	*unsigned char data_out*
DATE_IN	Leer registro de fechas de RTC	Valor de retorno *unsigned char rtc_data*
DATE_OUT	Visualizar fecha en LCD	*unsigned char data_out*
MONTH_IN	Leer registro de meses de RTC	Valor de retorno *unsigned char rtc_data*
MONTH_OUT	Visualizar mes en LCD	*unsigned char data_out*
YEAR_IN	Leer registro de años de RTC	Valor de retorno *unsigned char rtc_data*
YEAR_OUT	Visualizar año en LCD	*unsigned char data_out*

Tabla 16: Funciones para RTC DS1307

Descripción de la función

Función *void RTC_INIT(void)*

La inicialización de RTC DS1307 se hace necesaria después de cambiar de batería de apoyo. El siguiente proceso se encuentra implementado en la función:

IIC_SEND(0x00);	// Segundos = 00, reloj conect.	(registro RTC 0x00)
IIC_SEND(0x00);	// Minutos = 00	(registro RTC 0x01)
IIC_SEND(0x00);	// Horas = 00, modo de 24h	(registro RTC 0x02)
IIC_SEND(0x02);	// Día de la semana = 2	(registro RTC 0x03)
IIC_SEND(0x01);	// Días = 01	(registro RTC 0x04)
IIC_SEND(0x01);	// Meses = 01	(registro RTC 0x05)
IIC_SEND(0x02);	// Años = 02	(registro RTC 0x06)
IIC_SEND(0x93);	// Respuesta conectada	(registro RTC 0x07)

Figura 16: Proceso de la inicialización_ RTC (→ DS1307.pdf)

Funciones para leer las disintas áreas de memoria de RTC DS1307

Para leer las distintas secciones de información de los registros de RTC se implementó una función respectivamente. Estas funciones son del tipo *unsigned char* porque envian un valor de retorno *rtc_data* del tipo *unsigned char*.

Funciones para la visualización compatible con la LCD de I²C de la información de RTC

Para poder visualizar los datos de RTC en la unidad indicadora de LCD, éstos deben ser convertidos en un formato compatible con la LCD. A ese fin se pueden utilizar las funciones de visualización disponibles. Las funciones son del tipo *void* porque no se envía un valor de retorno. Para la transferencia de datos se utiliza una variable *data_out* del tipo *unsigned char*. Ésta se convierte según decenas y unidades y se visualiza en la unidad indicadora LCD. Una particularidad presenta la función para la visualización del día de la semana, porque dentro de la RTC, a estos días están asignados valores numéricos entre 1 .. 7 = Lunes .. Domingo. Para la visualización de la abreviatura del día se incorpora un campo de texto en el Flash del microcontrolador y un correspondiente puntero.

Ejemplos para llamadas a la función:

```
unsigned char minute;          // Variable para rtc

min     = MIN_IN();            // Minutos de RTC
MIN_OUT(minute);               // Minutos hacia LCD
```

Las respectivas posiciones de los datos de tiempo en la LCD están fijadas mediante la función *LCD_gotoXY* dentro de la correspondiente función de indicación. El usuario puede modificarlas. A ese fin se deben introducir, en la correspondiente llamada a la función *LCD_gotoXY*, otros valores para *line* y *column*.

lm75.h para la unidad RTC-Temp. de I2C

Función	Descripción	Transferencia de datos
TEMP_LCD	Convertir byte de temperatura en código LCD	*unsigned char t2* Valores de retorno en variables globales *temperatue1* y *temperature10*
TEMP_IN	Entrada del 1º byte de temperatura del LM75	Valor de retorno *tdata*
TEMP_OUT	Conversión y visualización del valor transferido	*unsigned char t1*

Tabla 17: Funciones para el sensor de temperatura LM75

Descripción de la función

Función *void TEMP_LCD(unsigned char t2)*

La función convierte una variable del tipo *unsigned char* en dígitos de unidades y decenas en variables individuales codificadas en BCD. Como valor se debe transferir una variable del tipo *unsigned char* en el margen de 0..99. Los resultados figuran en las variables *temperature1* y *temperature10*.

Ejemplo para la llamada a la función:

unsigned char temperature = 23;

TEMP_LCD(temperature); // Variable se divide en unidades / decenas de BCD

→ Valores de resultado: *temperature1 = 3*
temperature10 = 2

Función *unsigned char TEMP_IN(void)*

Con esta función se visualiza el primer byte de temperatura, es decir el valor de temperatura completo, del sensor LM75 de I²C. Es del tipo *unsigned char* porque devuelve el contenido del registro de datos TWI *TWDR*.

Ejemplo para la llamada a la función:

Variable = TEMP_IN(); // El valor de la función es asignado a la variable

Función *void TEMP_OUT(unsigned char t1)*

Se debe transferir un valor de salida del tipo *unsigned char* a la función. Se convierte, dentro de la función, en un formato compatible con LCD y se visualiza en la actual posición del cursor.

Ejemplo para la llamada a la función:

```
LCD_gotoXY(2,3);          // Poner cursor de LCD
TEMP_OUT(Variable);       // Visualizar valor de temperatura de TEMP_IN()
                          // en LCD
```

iiccard.h para la Unidad de Tarjeta Inteligente de I2C

Función	Descripción	Transferencia de datos
IIC_CARD_WR	Describir la dirección de la tarjeta chip de I²C ***cadr*** con el valor ***cdata***	*unsigned char cadr* *unsigned char cdata*

Tabla 18: Función para la tarjeta chip I²C

Descripción de la función

Función *void IIC_CARD_WR(unsigned char cadr,unsigned char cdata)*

La función incorpora la completa secuencia funcional para describir una dirección de memoria de una tarjeta chip de I²C. A la función se deben transferir como parámetros de transferencia la dirección de la locación de memoria en la tarjeta chip de I²C y el correspondiente valor. Ambas variables de transferencia son del tipo de datos *unsigned char*. Este tipo de datos es suficiente porque la tarjeta chip empleada en el sistema *MCLS-modular*® incluye una memoria con 256 direcciones, cada una de una anchura de 8 bits.

Ejemplo para la llamada a la función:

```
IIC_ CARD_WR(0x00,0xfe);  // en la dirección de memoria de tarjeta chip 0x00 se
                          // archiva el valor 0xfe
```

En el archivo de encabezamiento está definida la dirección del bus de I²C de la tarjeta chip. La dirección de intercambio de datos está puesta en *write*, de forma que resulta, para la dirección del bus, el valor *adr_card = 0xa0;*.

dau.h para la Unidad DA

Función	Descripción	Transferencia de datos
Impuls_out	Generación de un impulso triangular	sin parámetros

Tabla 19: Función para la Unidad DA

Descripción de la función

Función *void impuls_out(void)*

Con la llamada a esta función se genera un impulso en forma de triángulo con Δt = 560µs por la unidad DA. Si esta función se utiliza en un bucle de repetición, la generación de una frecuencia triangular se realiza en la salida A de la unidad DA con filtro paso bajo y en la salida B de la unidad DA sin filtro paso bajo.

CMC 5-1 Bloque de ensayos 1

Estructura base de un programa C, utilización de sentencias de control, utilización de puertos y pins de puerto

El primer programa C

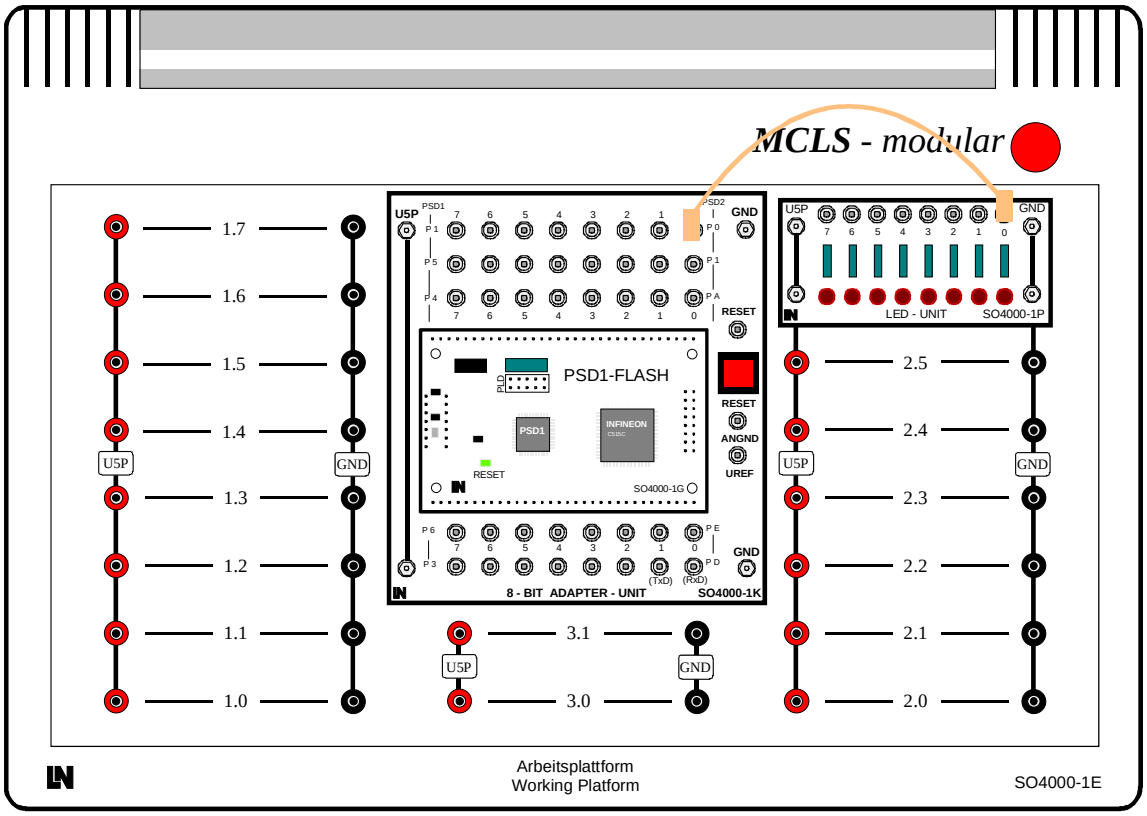


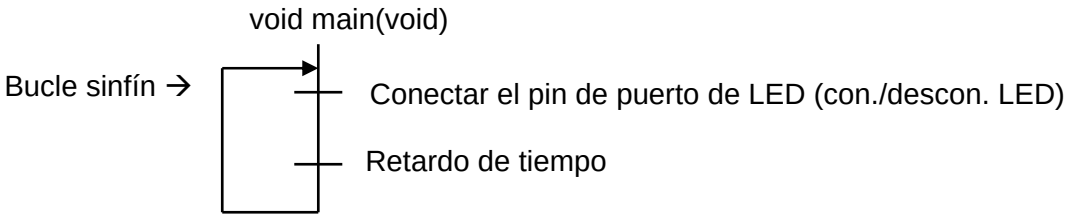
Fig. 101: Instalación de aparatos del ensayo CMC 5-1.1

Realice los siguientes primeros pasos de trabajo:

1. ¡Inicie el programa MCLS-modular en el PC!
2. ¡Conecte el Puerto1.0 de la UNIDAD de ADAPTADOR DE 8 BITS (módulo FLASH PSD1) al LED0 de la UNIDAD de LED!
3. ¡Conecte la plataforma de trabajo al interfaz serial del PC y conecte la fuente de alimentación de enchufe externa!

Explicación del ensayo:

En el primer ensayo se deberá compilar un programa sencillo que haga parpadear el díodo luminoso LED0 de la UNIDAD de LEDs. A ese fin se programará la siguiente secuencia en un bucle sinfín dentro de la función *main*:



Como bucles sinfín se pueden emplear dos diferentes secuencias de control. Para la sentencia *while* se debe poner una condición que siempre sea válida.

```
while(1)          o:          for(;;)
{
{
      ...                 ...
}
}
```

La segunda posibilidad reside en el empleo de una sentencia *for*, en la cual la condición de bucle se deja en blanco.

Adicionalmente se debe transferir al Compilador el tipo y el archivo de definición del microcontrolador empleado para que se puedan utilizar los nombres simbólicos de la periferia on chip. Después de la indicación del MC mediante instrucción *#define* se debe insertar el archivo de encabezamiento "*mcs51reg.h*" con el comando *#include* en el archivo fuente principal.

```
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
```

El cambio de nivel en un pin de puerto se realiza con la ayuda de la manipulación de bits estándar ^= (→Sección E).

La generación del retardo de tiempo puede realizarse empleando los bucles de conteo del archivo de encabezamiento *delay.h* (→ sección H):

```
delay(10000);
```

La función de retardo se elegirá en conformidad con el margen de conteo deseado de la variable de transferencia, es decir *delay (..);* para el margen de milisegundos con transferencia de un valor contado en tamaño *integer* según la tabla 10.

Nota:
En el módulo FLASH PSD1 se emplea un C515C de Infineon. Este controlador posee, en comparación con el C515A, adicionalmente un interfaz CAN que sin embargo no se utiliza en los bloques de ensayos. La definición del tipo de microcontrolador 515A es suficiente porque en el archivo de encabezamiento disponible "mcs51reg.h" del SDCC están registradas todas las definiciones necesarias para direcciones de registros de funciones especiales, etc. para la realización de los ensayos. Si se desea utilizar el interfaz CAN del C515C, se debe insertar adicionalmente el archivo de encabezamiento "regc515c.h".

Ejercicios de programación:

- ¡Abra un nuevo proyecto, crée un archivo de texto fuente e insértelo como archivo de texto fuente principal en el proyecto (→Sección G)!
- ¡Defina el microcontrolador e inserte las bibliotecas de funciones *mcs51reg.h* y *delay.h* (→Sección H)!
- ¡Compile un programa que conecte y desconecte de forma retardada un díodo luminoso de la UNIDAD de LEDs utilizando la manipulación de bits (→Sección E)!
- ¡Utilice en el bucle sinfín del programa principal un retardo de tiempo apropiado del archivo de encabezamiento *delay.h* (→ Sección F)!

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   cmc5-11: Programa de parpadeo                    */
/*    Autor:    ACMC/hpo                                         */
/*    Fecha:    06/04                                            */
/*    Software: SDCC                                             */
/*    Hardware: Flash PSD1                                       */
/*    Nota:     UNIDAD LEDs                                      */
/*         LED 0 -> Puerto1.0                                    */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"

// Programa principal ---------------------------------------------
void main(void)
{









}
```

Luz de desplazamiento sucesivo

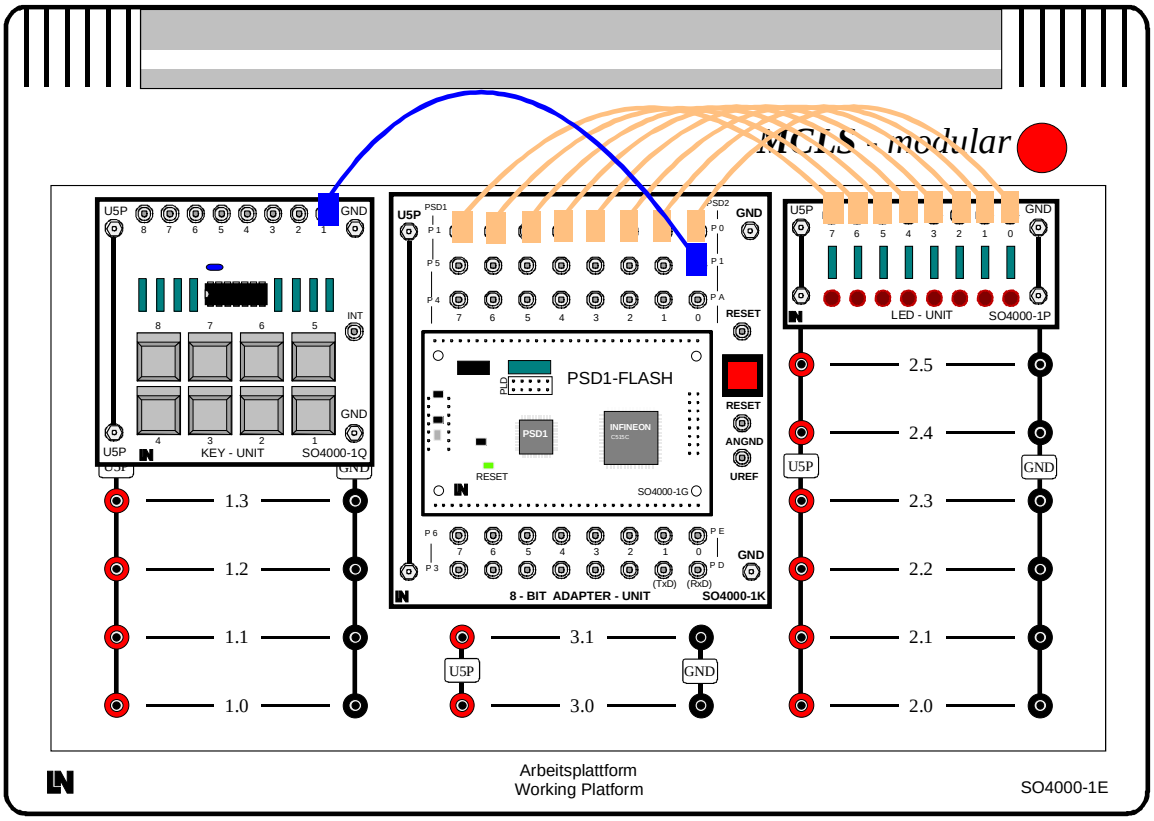


Fig.: 102: Instalación de aparatos del ensayo CMC 5-1.2

Explicación del ensayo:

En el siguiente ensayo se generará una luz de desplazamiento sucesivo con control de dirección en la UNIDAD de LEDs. Para la manipulación de bits (→ Sección E, Tabla 4), se encuentran disponibles en el Compilador los operadores estándar <<=, >>=, |, &, así como ^. Adicionalmente se pueden utilizar secuencias especiales para la generación efectiva de códigos por el Compilador y con ello para el acceso a los comandos del Ensamblador *swap*, *rrc*, *rl* y *rr* del 8051-Core. Estas secuencias pueden aprovecharse para la rotación o el desplazamiento de bits individuales dentro de una variable. La siguiente tabla muestra ejemplos:

Sintaxis de C	Comando del Ensamblador	Explicación
i >>= 4;	swap …	Cambia mitades de byte
i > >= 9;	rrc …	Desplaza 9 posiciones a la dcha. mediante Carry
i = ((i < < 1) | (i > > 7));	rl …	Desplaza una posición a la izda. y el MSB en el LSB → Rotación
i = ((i > > 7) | (i < < 1));	rr …	Desplaza una posición a la dcha. y el LSB en el MSB → Rotación

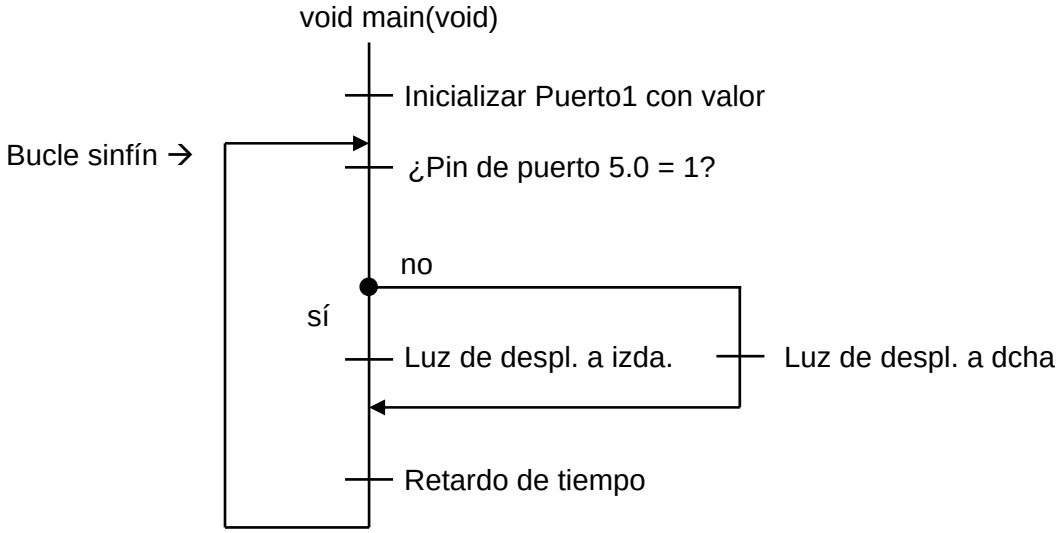
Tabla 101: Ejemplos para manipulaciones de bits

Para el control de dirección el programa debe incluir una evaluación de una tecla de la unidad de TECLAS que está conectada a un pin de puerto. A ese fin se puede emplear una sentencia de control *if* con ramificación *else* (→ Sección E). La evaluación del estado del pin de puerto es posible mediante una consulta directa.
Ejemplo de código de la consulta directa del estado de la tecla en el pin de puerto P5.0):

```
if(P5_0==1) Acción1;
else Acción2;
```

Ejercicios de programación:

- ¡Abra un nuevo proyecto, crée un archivo de texto fuente e insértelo como archivo de texto fuente principal en el proyecto (→Sección G)!
- ¡Defina el microcontrolador e inserte las bibliotecas de funciones *mcs51reg.h* y *delay.h* (→Sección H)!
- ¡Compile, según el siguiente programa de flujo, un programa que genere una luz de desplazamiento sucesivo con dirección controlada por medio del puerto 1 del microcontrolador en la la UNIDAD de LEDs!



- ¡Utilice las secuencias indicadas en la tabla para desplazar o rotar las posiciones de bit dentro de una variable!

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   cmc5-12: Luz de desplazamiento sucesivo           */
/*    Autor:    ACMC/hpo                                          */
/*    Fecha:    06/04                                             */
/*    Software: SDCC                                              */
/*    Hardware: Flash PSD1                                        */
/*    Nota:     UNIDAD LED 0 ... 7 en Puerto 1.0 ... 1.7          */
/*              UNIDAD TECLAS Tecla 1 en Puerto 5.0               */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"


// Programa principal-------------------------------------------------
void main(void)
{









}                               // end main
```

Contando accionamientos de tecla mediante Polling (modo de espera)

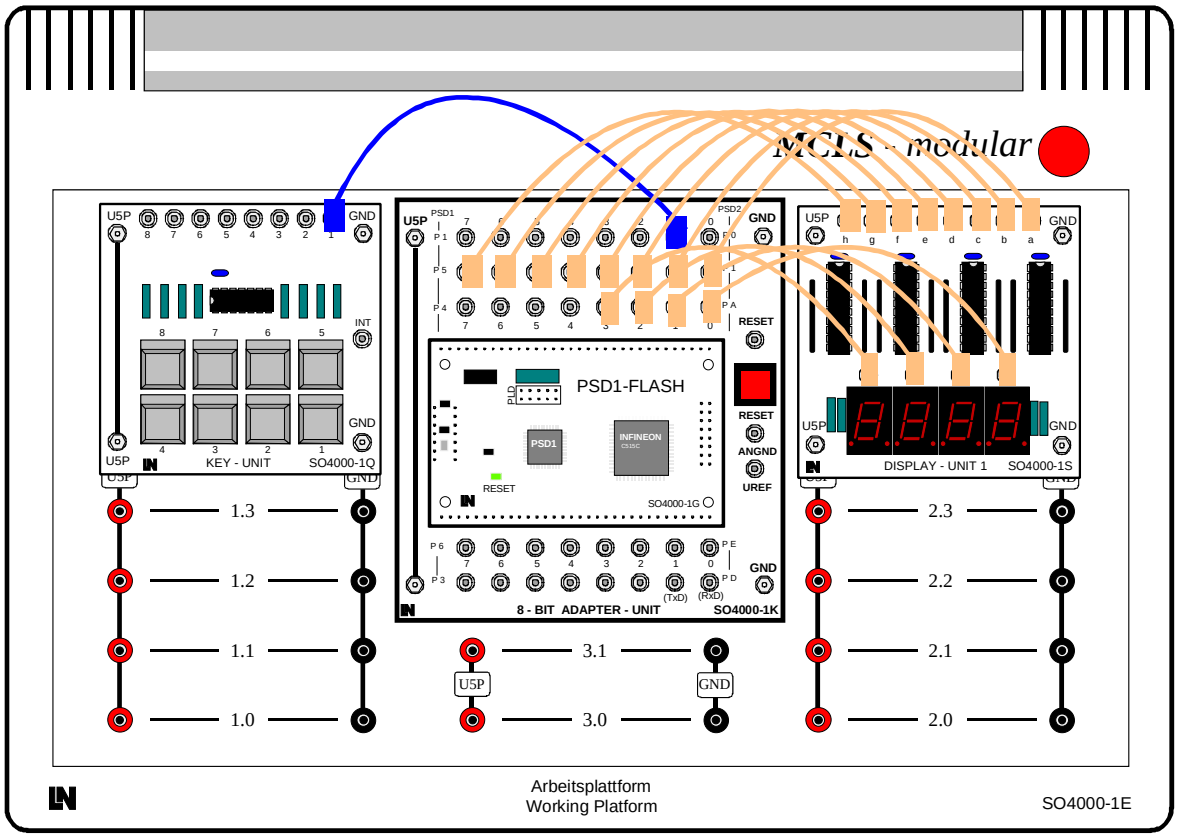
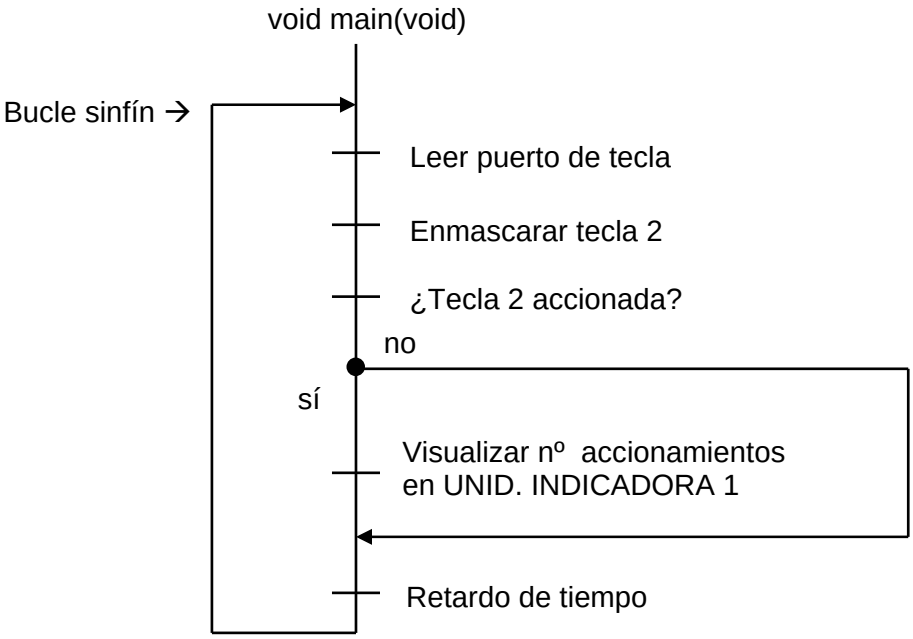


Fig. 103: Instalación de aparatos del ensayo CMC 5-1.3

Explicación del ensayo:

En este ensayo se compilará un programa que lee las teclas de la UNIDAD de TECLAS de forma retardada y en modo Polling, y que, después de pulsar la tecla 2 visualiza el número de accionamientos de teclas en la UNIDAD INDICADORA 1. Para detectar la tecla pulsada, en este ensayo se utilizará una operación de enmascaramiento. El diagrama de flujo del programa se indica a continuación:



La lectura del puerto al que están conectadas las teclas se puede realizar con una asignación sencilla:

```
unsigned char in;          // Variable
in = Px;                   // Leer puerto de tecla
```

A continuación se enmascarán los pins de puerto para el posterior procesamiento. A ese fin se ofrece una operación lógica con una constante:

```
in = in  &  0x01;      ó      in &= 0x01;    // para tecla 1 de la UNIDAD de TECLAS
```

La variable *in* puede utilizarse en una selección *if* o *switch* como condición o como expresión de ensayo:

```
if (in == 0x...)...      ó:      switch (in)
                                 {
                                        case 0x…:
                                        ...
                                 }
```

Como alternativa, C ofrece la posibilidad de combinar la expresión:

```
if (Px &= 0x...)...      ó:      switch (Px &= 0x...)
                                 {
                                        case 0x…:
                                        ...
                                 }
```

En este caso, el resultado es una evaluación directa del grado de verdad.

Si se pulsa la tecla 2, se debe incrementar una variable contadora. El valor de esta variable contadora se muestra por medio de las bibliotecas de funciones *intbcd.h* y *7seg.h* como número de 4 dígitos. Para el retardo de tiempo puede utilizarse el mismo bucle de conteo que en *VK5-1.1*.

Ejercicios de programación:

- ¡Abra un proyecto nuevo!
- ¡Lea las explicaciones de las bibliotecas de funciones *intbcd.h* y *7seg.h* en la sección H!
- ¡Compile un programa que lea el estado de los pins de puerto de las teclas separados por un retardo de tiempo y que al accionar la tecla 2 visualice el número de accionamientos en la UNIDAD INDICADORA!
- ¡Utilice las secuencias indicadas en las explicaciones!

Ejemplo de solución:

```
/*********************************************************************/
/* Título: cmc5-13: Contar accionamientos de tecla mediante Polling */
/* Autor:   ACMC/hpo                                                 */
/* Fecha:   06/04                                                    */
/* Software: SDCC                                                    */
/* Hardware: Flash PSD1                                              */
/* Nota:    UNIDAD INDICADORA 1                                      */
/*     Puerto5    -> Puerto de datos (segmentos a - h)               */
/*     Puerto 4.0 -> D0 (dígito 0)                                   */
/*     Puerto 4.1 -> D1 (dígito 1)                                   */
/*     Puerto 4.2 -> D2 (dígito 2)                                   */
/*     Puerto 4.3 -> D3 (dígito 3)                                   */
/*     UNIDAD DE TECLAS                                              */
/*     Tecla 1...8 -> Puerto 1.0 ... 1.7                             */
/*********************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"
#include "7seg.h"
#include "intbcd.h"

// Programa principal ------------------------------------------------
void main(void)
{









}
```

Preguntas de control para el bloque de ensayos CMC 5-1

¿Qué misión tiene la función *main* dentro de un programa C?

¿Cómo se pueden realizar los bucles sinfín?

¿De qué sirve mcs51reg.h?

¿Cómo se introducen los estados de puerto?

¿Cómo se puede comprobar el nivel lógico en el Puerto 1.7?

¿Cómo se establece el nivel lógico *alto* en el pin de puerto P5.2?

CMC 5-2 Bloque de ensayos 2

Utilización de interrupciones, interrupción externa, interrupción por Timer, generación de PWM

Contar y visualizar accionamientos de teclas con interrupción externa

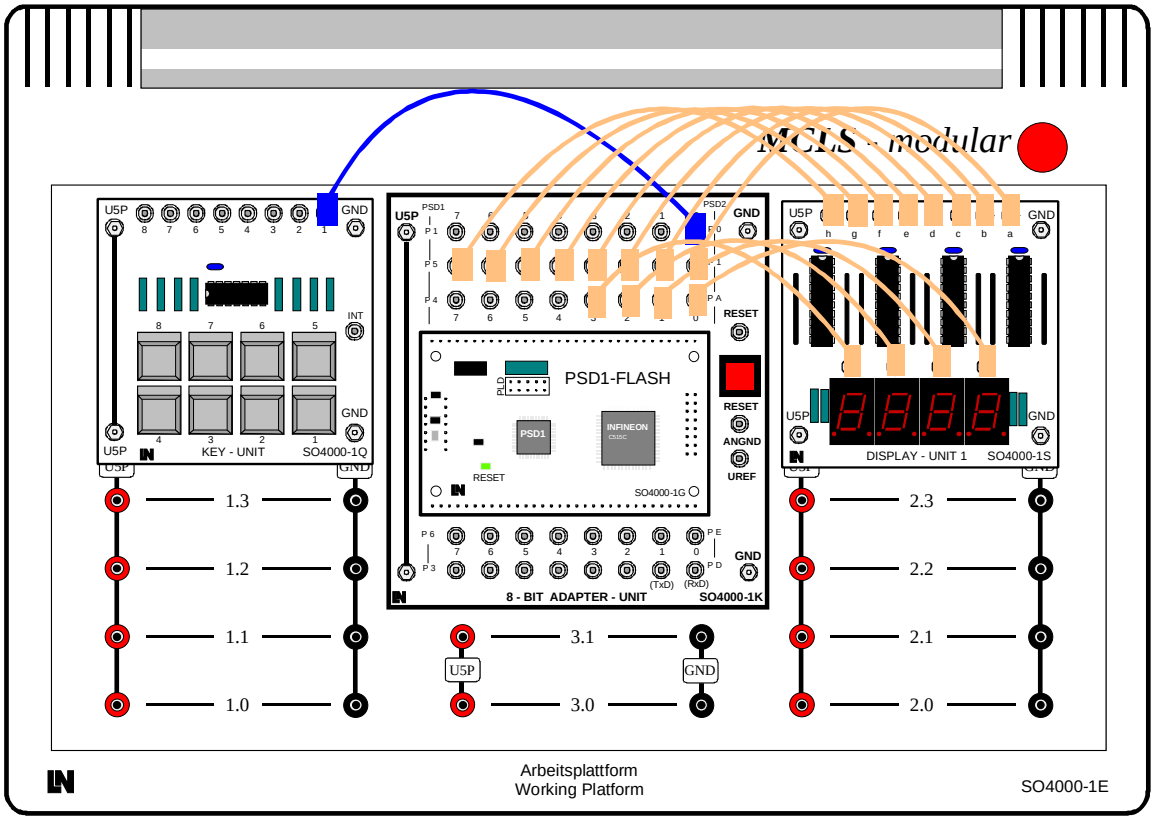


Fig. 201: Instalación de aparatos del ensayo CMC 5-2.1

Explicaciones del ensayo:

El ensayo muestra el principio de la utilización de interrupciones con el ejemplo de la interrupción3 externa. En el ensayo se activará mediante la tecla 1 de la UNIDAD DE TECLAS. Esta última se conectará al pin de puerto P1.0 del microcontrolador.

Para la utilización de una interrupción es recomendable la siguiente forma de proceder:

1. ***Configurar e inicializar la interrupción en el programa principal utilizando el manual de instrucciones del módulo FLASH PSD1.***

   La interrupción3 se activará poniendo el bit 2 (***EX3***) en el registro de funciones especiales ***IEN1***. Para las inicializaciones se empleará el operador de asignación =:

   EX3 =1; *// Activación de la interrupción3 externa*

2. ***Compilar una rutina de servicio de interrupción y tratar la interrupción con el software***

   Las rutinas de servicios de interrupción poseen, para el Compilador SDCC empleado, la siguiente estructura:

   void ex3_isr (void) interrupt IEX3_VECTOR using 1
   { … }

   Con la palabra clave ***interrupt*** se comunica al Compilador que la función es una rutina de servicio de interrupción (ISR). La siguiente indicación ***IEX3_VECTOR*** especifica al vector de interrupción que está definido en el archivo de encabezamiento *mcs51reg.h* por medio de una instrucción *#define*. La indicación *using 1* determina el banco de registro a emplear.

   Dentro del cuerpo de la función de la rutina de servicio de interrupción (llaves) se deberá introducir el tratamiento de interrupción.

3. ***Conexión global de todas las interrupciones registradas en el programa principal***

   Finalmente se deben conectar de forma global todas las interrupciones. A ese fin se deberá poner el bit 7 (***EA***) en el registro de funciones especiales ***IEN0***:

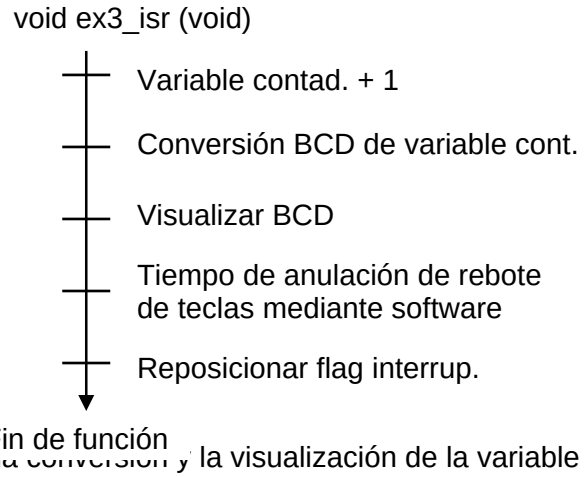
   EA = 1; *// Activación de todas las interrupciones*

Notas:
La interrupción externa3 puede ser activada por un flanco descendente o ascendente en el pin de puerto P1.0. El flanco que se debe utilizar para activar la rutina de servicio de interrupción se puede configurar con el bit de control *I3FR* en el registro de funciones especiales *T2CON*. Partiendo de la inicialización del *Puerto1* tras un RESET de MC con 0xff y el estado del interruptor */S1*, sin tarjeta chip insertada en la unidad de Tarjeta Inteligente de I2C, se configurará la interrupción externa3 en estado inicial para que sea activada por el flanco descendente (*I3FR=0*). Este flanco se genera cuando se inserta una tarjeta chip. Al retirar la tarjeta de la unidad de Tarjeta Inteligente de I2C, se abre el interruptor */S1* generando un flanco ascendente. Por consiguiente, antes de retirar la tarjeta chip, se debe tener en cuenta la reacción de la interrupción externa3 respecto a este flanco (*I3FR=1*).

Puesto que las **teclas** son componentes mecánicos, al accionarlas se produce un **rebote** de los contactos en el rango de milisegundos, lo que puede activar varias interrupciones y dar lugar a una ejecución errónea del programa. La solución de este problema reside en la inserción de un tiempo de anulación de rebote. Adicionalmente se deberá reposicionar el Edge-Flag de la interrupción3 antes de finalizar la rutina de servicio de interrupción para impedir la activación de interrupciones múltiples.

Ejercicio de programación:

- ¡Abra un proyecto nuevo!
- ¡Configure la interrupción externa3 (→ manual de instrucciones para el módulo FLASH PSD1)!
- ¡Cuente, en la rutina de servicio de interrupción ex3_isr, los accionamientos de la tecla 1 de la UNIDAD DE TECLAS! ¡Visualice el número de accionamientos de tecla como valor de 4 dígitos en la UNIDAD INDICADORA 1! Implemente en la ISR la siguiente secuencia:



- ¡Utilice, para la conversión y la visualización de la variable contadora, las bibliotecas de funciones *intbcd.h* y *7seg.h*!

Ejemplo de solución:

```
/*********************************************************************/
/*  Título:   cmc5-21: Contar y visualizar accionamientos de teclas */
/*      mediante interrupción externa3                               */
/*  Autor:    ACMC/hpo                                               */
/*  Fecha:    06/04                                                  */
/*  Software: SDCC                                                   */
/*  Hardware: Flash PSD1                                             */
/*  Nota:     UNIDAD DE LED                                          */
/*      LED 0 -> Puerto1.0                                           */
/*  Hardware: Flash PSD1                                             */
/*  Nota:     UNIDAD INDICADORA 1                                    */
/*                                                                   */
/*      Puerto5    -> Puerto de datos (segmentos a - h)              */
/*      Puerto 4.0 -> D0 (dígito 0)                                  */
/*      Puerto 4.1 -> D1 (dígito 1)                                  */
/*      Puerto 4.2 -> D2 (dígito 2)                                  */
/*      Puerto 4.3 -> D3 (dígito 3)                                  */
/*                                                                   */
/*      UNIDAD DE TECLAS                                             */
/*                                                                   */
/*      Tecla 1 -> Puerto 1.0 (INT3 ext.)                            */
/*                                                                   */
/*********************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"
#include "7seg.h"
#include "intbcd.h"
```

```
// Variables globales ----------------------------------------


// Prototipos de funciones -----------------------------------------
void ex3_isr (void) interrupt IEX3_VECTOR using 1;
void init(void);

// Programa principal ----------------------------------------------
void main(void)
{









}  // end main

// ISR Interrupción 3 externa --------------------------------------
void ex3_isr (void) interrupt IEX3_VECTOR using 1
{









}  // end ex3_isr

// Inicializaciones de MC ------------------------------------------
void init(void)
{



}  // end init
```

Emisión de sonido mediante interrupciones cíclicas del Timer2 como temporizador

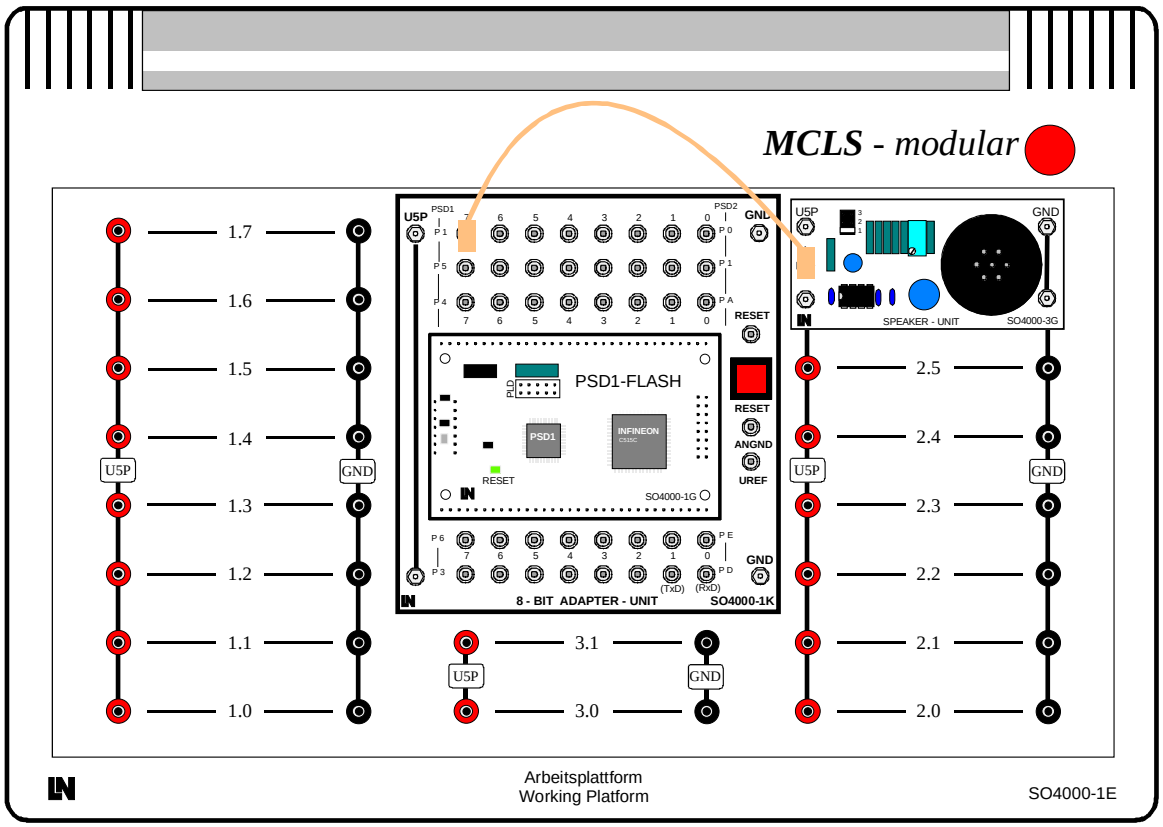


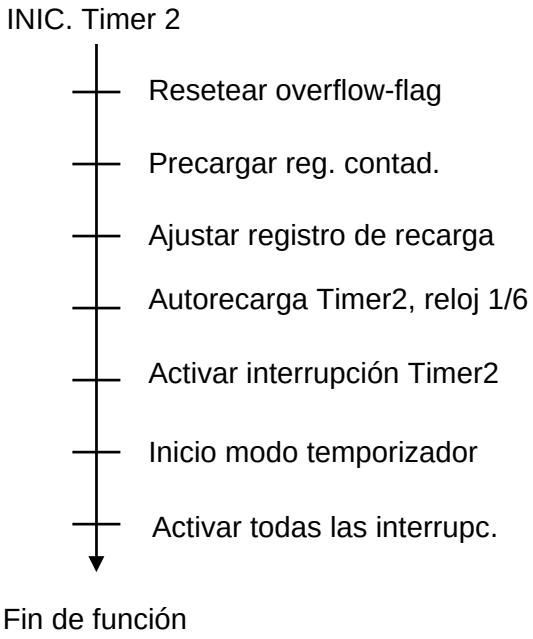
Fig. 202: Instalación de aparatos del ensayo CMC 5-2.2

Explicación del ensayo:

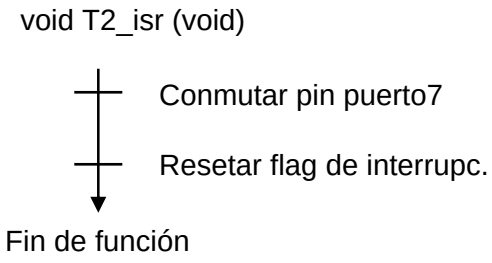
El ensayo incluye la configuración del Timer2 como temporizador en modo de desbordamiento (overflow) con recarga automática para una frecuencia fija (→ Ensayo de Ensamblador CMC 3-02). En la rutina de servicio de interrupción activada por el desbordamiento, un pin de puerto (P1.7) al que se encuentra conectada la unidad de altavoces, se deberá conectar y desconectar durante medio periodo respectivamente.

Ejercicios de programación:

- ¡Abra un proyecto nuevo!
- ¡Configure el Timer2 como temporizador en modo de desbordamiento con autorecarga para una frecuencia de 1kHz! ¡Utilice la siguiente secuencia (→ Manual de instrucciones del módulo FLASH PSD1)!



- ¡Emita la frecuencia generada por medio del pin de puerto P1.7 utilizando la rutina de servicio de interrupción del Timer2 Overflow como señal en la unidad de altavoces!



- ¡Utilice para el conmutado del pin de puerto una manipulación de bits adecuada!

Ejemplo de solución:

```
/*************************************************************/
/*    Título:   cmc5-22: Emisión de sonido con Timer2        */
/*              Overflow Interrupt autoreload (1kHz)         */
/*    Autor:    ACMC/hpo                                     */
/*    Fecha:    06/04                                        */
/*    Software: SDCC                                         */
/*    Hardware: Flash PSD1                                   */
/*           Unidad de altavoces  Uin -> P1.7                */
/*************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>

// Prototipos de funciones ----------------------------------------


// Programa principal ---------------------------------------------
void main(void)
{


}  // end main

// Timer 2 ISR ----------------------------------------------------







// Inicialización -------------------------------------------------
```

Regulación de la iluminación de fondo de la LCD mediante generación de PWM

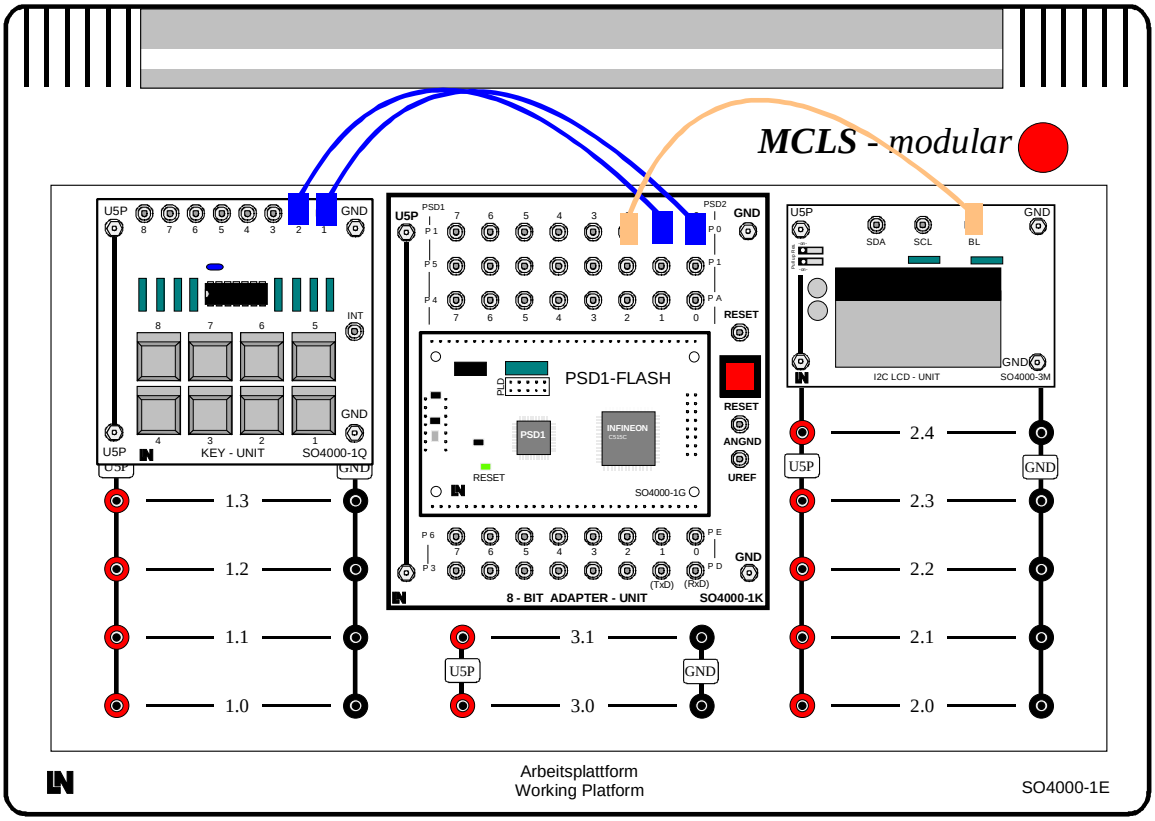


Fig. 203: Instalación de aparatos del ensayo CMC 5-2.3

Explicación del ensayo:

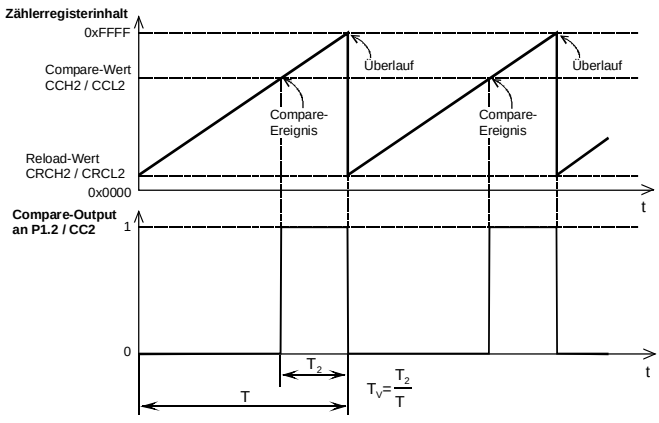


Figura 204: Timer 2 en modo de Comparación 0

En el modo *Compare Mode 0*, el conteo del Timer 2 se realiza desde un valor de recarga establecido hasta el valor máximo 0xFFFF con recarga automática del registro de conteo. Al alcanzar el valor de comparación del registro CCH2/CCL2 y el desbordamiento del registro de conteo de TH2/TL2, el pin de puerto P1.2, al que está conectada la iluminación de fondo de la unidad LCD de I2C, se conecta y se desconecta.

Aumentando o reduciendo el valor de comparación, puede modificarse la relación de exploración dentro de un periodo de la señal (→ *Programación del temporizador CMC 3-02*). En este ensayo, la modificación del valor se realizará en las rutinas de servicio de interrupción de las interrupciones externas 3 y 4 donde están conectadas las teclas 1 y 2 de la UNIDAD de TECLAS.

Nota:
El valor para el evento de comparación tiene una anchura de 16 bits. Está dividido entre los dos registros de 8 bits CCH2 (parte alta) y CCL2 (parte baja). Ambos registros solamente se pueden inicializar por separado con valores. Esto significa que la variable contadora de 16 bits (tipo de datos Integer) se debe dividir en dos valores de 8 bits cada uno (tipo de datos Unsigned Character) para la parte baja y la parte alta. En el lenguaje de programación C, para las variables existe la posibilidad de asignar tipos de datos fuente de gran ocupación a tipos de datos de destino de pequeña ocupación. Este procedimiento se llama Casting.

Ejemplo:

El valor de una variable del tipo *unsigned int a* se debe asignar a una variable del tipo *unsigned char b*, siendo asignada solamente la parte baja de *a*, de 8 bits de anchura, a la variable *b* del tipo *unsigned char* e ignorándose la parte alta. En la programación, el tipo de datos de destino deseado precede entre paréntesis el tipo de datos fuente.

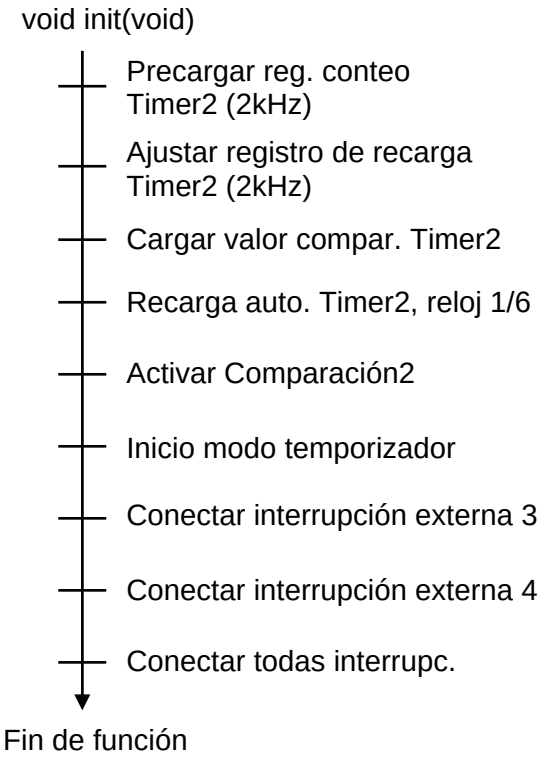
```
unsigned int a = 0xfe4d;      // Variable del tipo unsigned int
unsigned char b = 0x00;       // Variable del tipo unsigned char

b = (unsigned char) a;        // la parte baja de a es asignada a b
                              // → unsigned int a es asignado a unsigned char
```

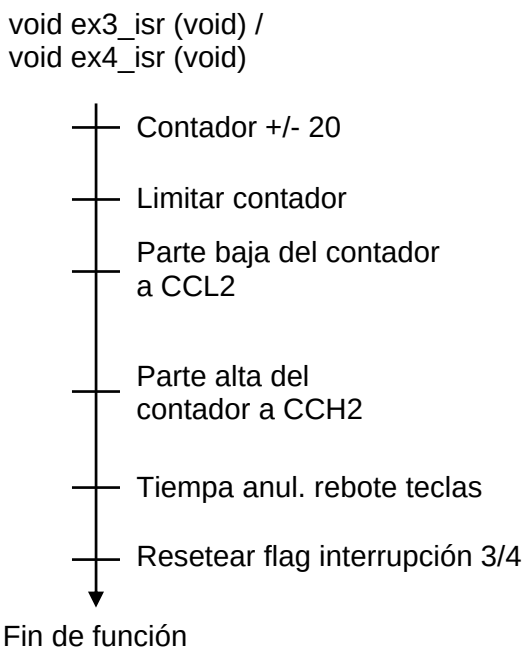
Este Casting puede emplearse para la inicialización de los registros de comparación CCH2/CCL2 dentro de las rutinas de servicio de interrupción de las interrupciones externas 3 y 4:

```
unsigned int out =0x fe 01;     // Valor de ejemplo para “out”
CCL2=(char)out;                 // Parte baja de 8bits de "out" a     CCL2 => 0x01
out = ((out << 8) | (out >> 8)); // Cambiar parte baja con parte alta,  out => 0x01fe
CCH2=(char)out;                 // Parte alta de 8bits de "out" a     CCH2 = 0xfe
out = ((out << 8) | (out >> 8)); // Cambiar parte alta con parte baja, out => 0xfe01
```

Ejercicios de programación:

- ¡Configure el Timer 2 como temporizador en modo de desbordamiento con autorecarga para una frecuencia de 2kHz!
- ¡Active el modo de comparación 0 del Timer 2 y las interrupciones externas 3 y 4!
- ¡Escriba una rutina de servicio de interrupción para las interrupciones externas 3 y 4! En la ISR de la interrupción externa3, el valor de comparación para la PWM se debe incrementar en 20 pasos y en la ISR de la interrupción externa4 se deberá reducir en 20 pasos. ¡Preste atención a que el valor de comparación se sitúe dentro del margen de conteo del temporizador! ¡Utilice para la configuración el manual de instrucciones del módulo Flash PSD1 y los siguientes diagramas de flujo!


void init(void)
Precargar reg. conteo Timer2 (2kHz)
Ajustar registro de recarga Timer2 (2kHz)
Cargar valor compar. Timer2
Recarga auto. Timer2, reloj 1/6
Activar Comparación2
Inicio modo temporizador
Conectar interrupción externa 3
Conectar interrupción externa 4
Conectar todas interrupc.
Fin de función


void ex3_isr (void) /
void ex4_isr (void)
Contador +/- 20
Limitar contador
Parte baja del contador a CCL2
Parte alta del contador a CCH2
Tiempa anul. rebote teclas
Resetear flag interrupción 3/4
Fin de función

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   cmc5-23: PWM para regular LED HG de LCD          */
/*    Autor:    ACMC/hpo                                         */
/*    Fecha:    07/04                                            */
/*    Software: SDCC                                             */
/*    Hardware: Flash PSD1                                       */
/*    Note:     UNIDAD LCD I2C                                   */
/*        BL -> Puerto1.2                                        */
/*        UNIDAD TECLAS   Tecla 1 -> P1.0 (Interrup3 ext.)       */
/*             Tecla 2 -> P1.1 (Interrup4 ext.)                  */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"

unsigned int out;

// Prototipos de funciones ----------------------------------------
```

Medición de la frecuencia con Timer0 como contador, Timer2 como base de tiempo y la unidad DA para generar una frecuencia fijada

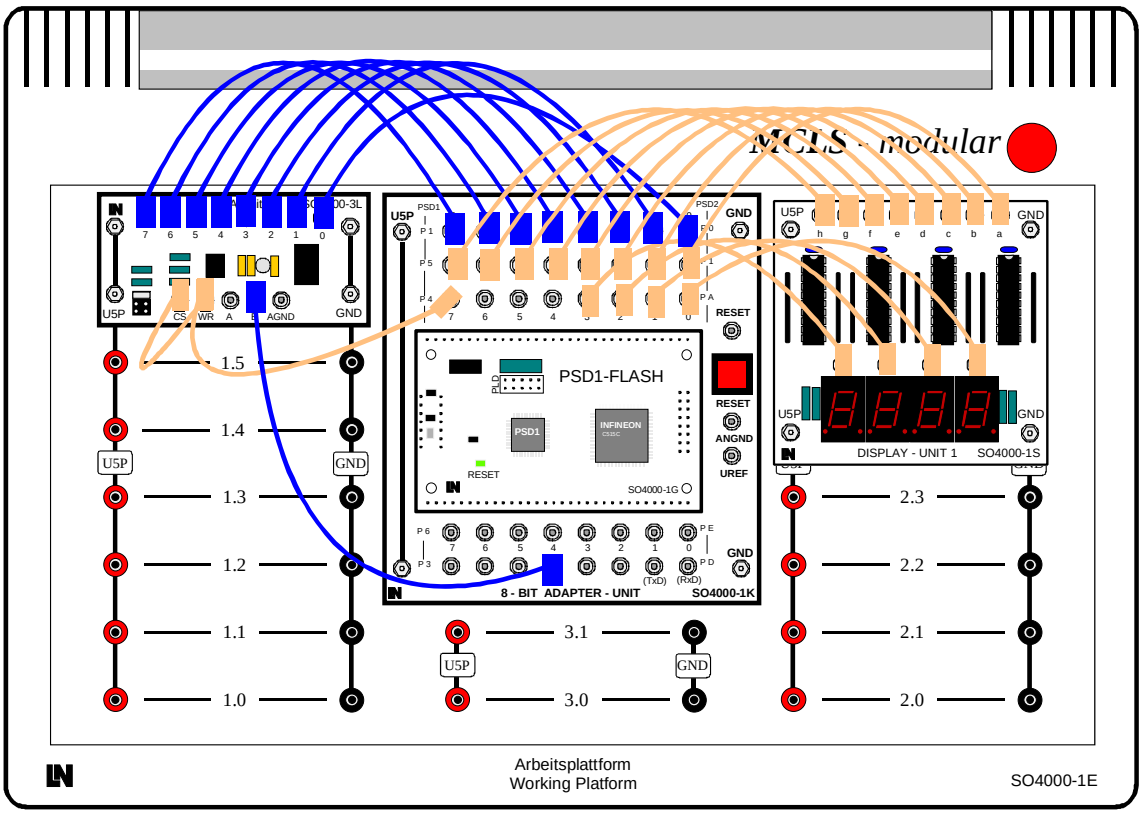


Fig. 205: Instalación de aparatos del ensayo CMC 5-2.4

Explicación del ensayo:

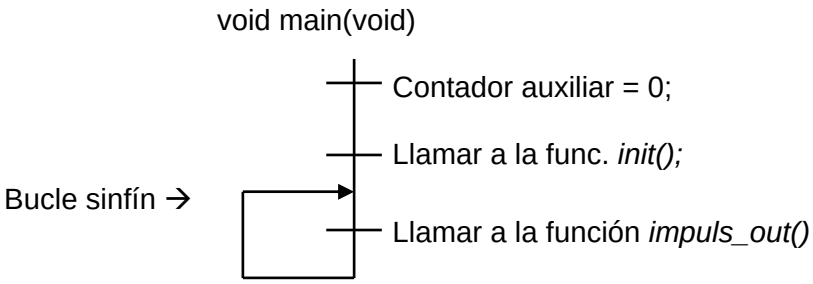
Con el microcontrolador y la unidad DA se genera, mediante la función *void freq_out(void)* del archivo de encabezamiento *dau.h,* por medio del software una frecuencia fija (aprox. 1765 Hz) que está disponible en la salida B de la unidad DA.

La base para la medición de la frecuencia la forman el Timer0 y el Timer 2. Mientras que con el Timer 2 se suministra la base de tiempo de 1s (función de temporizador), con Timer0 y su entrada de conteo (pin de puerto P3.4) se cuentan los impulsos que llegan (función de contador). Los impulsos registrados en el registro de conteo por el Timer0 dentro de un segundo se ofrecen como valor binario de 16 bits y deben convertirse en dígitos de BCD para la UNIDAD INDICADORA 1. La función para la conversión y visualización se encuentra en el archivo de encabezamiento *intbcd.h*.

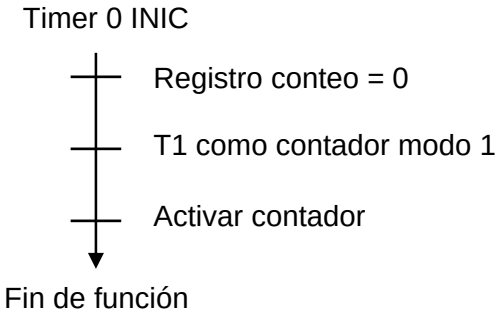
El Timer 2 se deberá configurar para la generación de un tiempo de ciclo de 1s. Estos 1000ms pueden realizarse suministrando una base de tiempo de 50ms con el Timer 2 en modo de desbordamiento con autorecarga, así como con una variable auxiliar dentro de la rutina de servicio de interrupción del Timer 2.

Ejercicios de programación:

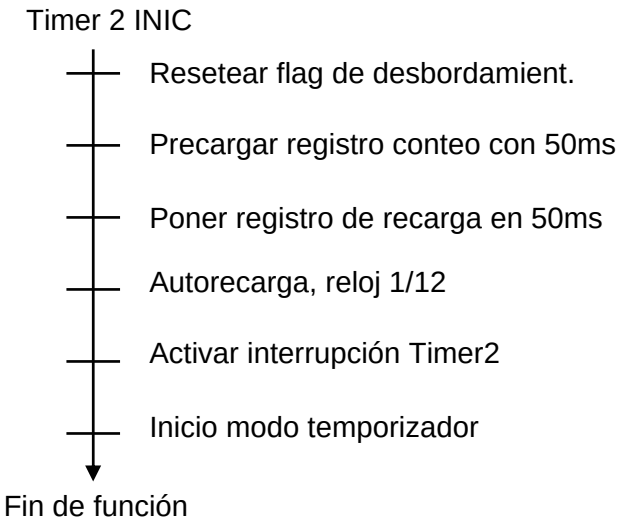
- ¡Abra un proyecto nuevo!
- ¡Genere en el bucle sinfín del programa principal una frecuencia triangular mediante la función *void impuls_out(void)* y la unidad DA!



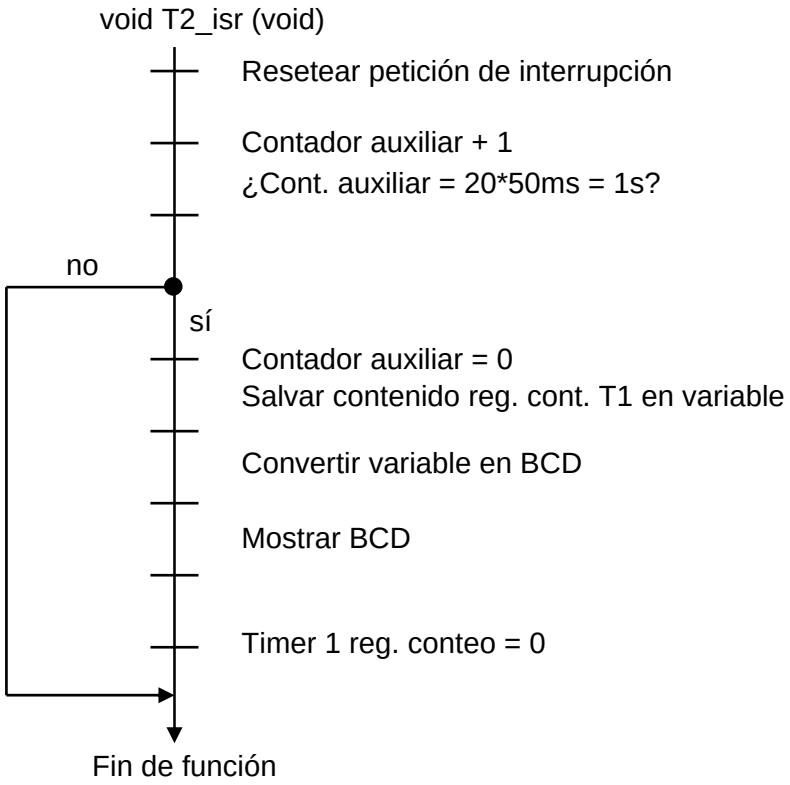
- ¡Configure el Timer0 como contador de impulsos de 16bits para la frecuencia suministrada por la unidad DA (→manual de instrucciones para el módulo FLASH PSD1)!



- ¡Configure el Timer 2 como temporizador en modo de desbordamiento con autorecarga para un tiempo de 50ms! ¡Utilice el manual de instrucciones del módulo FLASH PSD1!



- ¡Realice dentro de la rutina de servicio de interrupción del Timer2 un tiempo de ciclo de 1s! ¡Utilice una sentencia de control apropiada!



- ¡Convierta, transcurrido un segundo, el valor de conteo del Timer 0 en dígitos BCD individuales (→ int2ebcd.h) y visualícelos en la UNIDAD INDICADORA 1 como valor de 4 dígitos (→ 7seg.h)!

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   cmc5-24 Medición de frecuencia con UDA como entrada */
/*    Autor:    ACMC/hpo                                           */
/*    Fecha:    06/04                                              */
/*    Software: SDCC                                               */
/*    Hardware: Flash PSD1                                         */
/*    Nota:     UNIDAD INDICADORA 1                                */
/*                                                                 */
/*        Puerto5    -> Puerto de datos (segmentos a - h)          */
/*        Puerto 4.0   -> D0 (dígito 0)                            */
/*        Puerto 4.1   -> D1 (dígito 1)                            */
/*        Puerto 4.2   -> D2 (dígito 2)                            */
/*        Puerto 4.3   -> D3 (dígito 3)                            */
/*                                                                 */
/*        UNIDAD DA    -> genera frecuencia triangular             */
/*                                                                 */
/*        Datos        -> Puerto 1                                 */
/*        /CS y /WR    -> P4.7                                     */
/*        B            -> P3.4                                     */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "7seg.h"
#include "intbcd.h"
#include "dau.h"

// Variables globales ---------------------------------------


// Prototipos de funciones ----------------------------------




// Programa principal ---------------------------------------




// Inicializaciones de MC -----------------------------------
```

```
// Timer 2 ISR ------------------------------------------------
```

Preguntas de control para el bloque de ensayos CMC 5-2

¿Cuál es el propósito de la declaración de prototipos de funciones?

¿Qué registros de funciones especiales se deben inicializar para la utilización de la interrupción externa3?

¿Por qué, al pulsar una tecla dentro de una operación de interrupción, se debe resetear el bit de petición de interrupción de la interrupción externa3 sólo después de un tiempo de anulación de rebote?

¡Comente los componentes de la siguiente descripción de función!
void ex3_isr(void) interrupt IEX_3 VECTOR using1

¿Qué se entiende por Casting en la programación C?

¿Cómo se puede generar un tiempo de ciclo de 1 segundo utilizando el Timer 2 en modo de desbordamiento?

CMC 5-3 Bloque de ensayos 3

Bus I²C, control de la LCD, utilización del ADC (convertidor AD)

Principio de funcionamiento del bus I²C

El bus I²C (Inter Integrated Circuit Bus) fue desarrollado por Philips para la conexión en serie de circuitos integrados en una placa o en un aparato. Para la conexión de los componentes se emplean dos líneas, la línea de datos seriales SDA y la línea de reloj serial SCL. Ambas líneas están conectadas por medio de una resistencia de *pull up* a Vcc (R1 y R2 en la figura). Cada aparato dispone de una dirección de bus de 7 bits propia y puede trabajar, dependiendo de su función, como emisor o como receptor. Ello permite conectar hasta 128 aparatos diferentes a un bus. Como mínimo un usuario de bus debe trabajar como maestro. Facilita el reloj serial y es responsable del control del intercambio de datos en el bus.

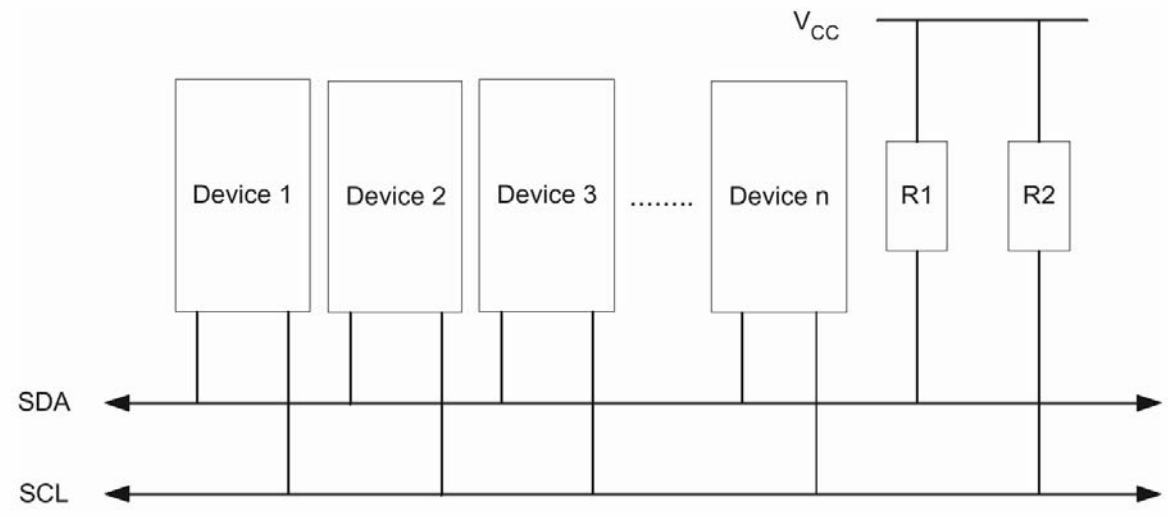


Figura 301: Configuración del hardware de un sistema I²C

La transmisión de datos entre maestro y esclavo empieza con la generación de la secuencia de inicio por parte del maestro en el sistema I²C. El maestro pone la línea de datos en Bajo (Low) mientras que la línea de reloj permanece en Alto (High). Con el siguiente reloj de la línea SCL se direcciona, con los siguientes 8 bits, un esclavo. Como primer bit se envía el Bit Más Significante de la dirección. Los primeros siete bits son la dirección del esclavo en el sistema. El octavo bit ajusta la dirección de la transmisión de datos. Un Bajo (Low) del bit de dirección 8 significa el envío de datos por parte del maestro al esclavo, un Alto (High) del bit de dirección 8 significa la recepción de datos por el maestro desde el esclavo. Con el siguiente noveno reloj, el esclavo direccionado envía una señal de confirmación al maestro poniendo la línea de datos en Bajo (Low).

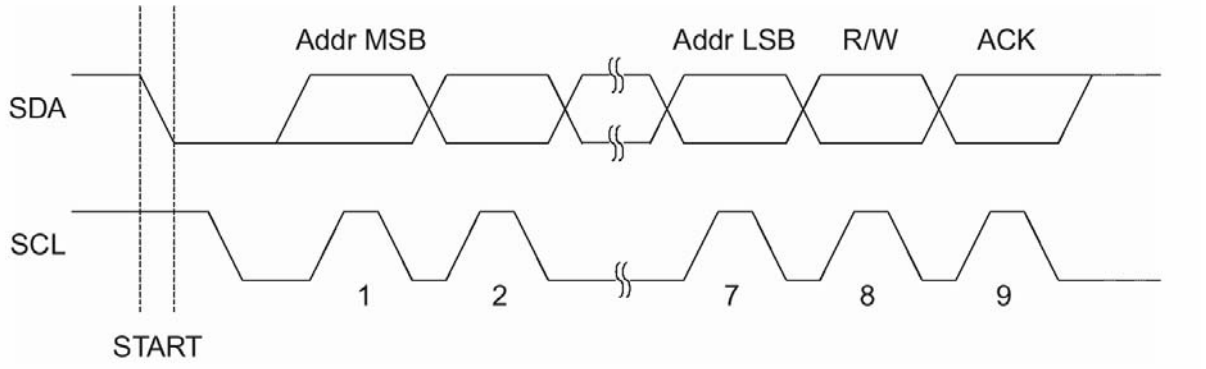


Figura 302: Secuencia de inicio y direccionamiento de esclavo

Con el siguiente reloj en la línea SCL del maestro en el sistema I²C se inicia el intercambio de datos, byte a byte, según la dirección de transmisión establecida en el byte de dirección. El componente, maestro o esclavo, que ha recibido correctamente un byte de datos, envía como noveno bit una señal de confirmación. A continuación puede enviarse un nuevo byte de datos o generarse una secuencia de generar. Esta secuencia de parada se genera por un Alto (High) de la línea de reloj y un cambio de nivel de la línea de datos de Bajo (Low) a Alto (High).

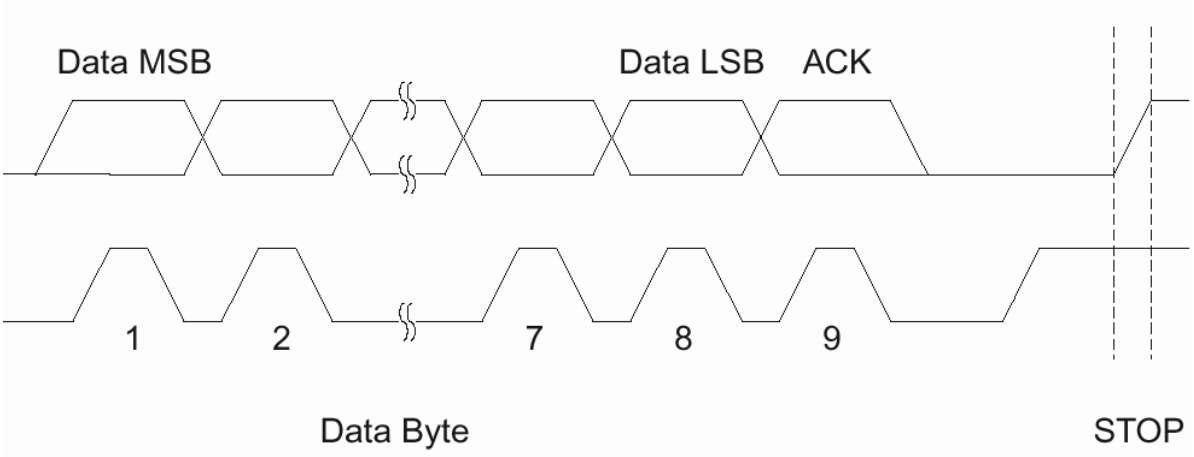


Figura 303:Transporte de datos y secuencia de parada

La siguiente figura muestra el principio de la secuencia cuando un maestro I²C recibe dos bytes de datos de un esclavo I²C.

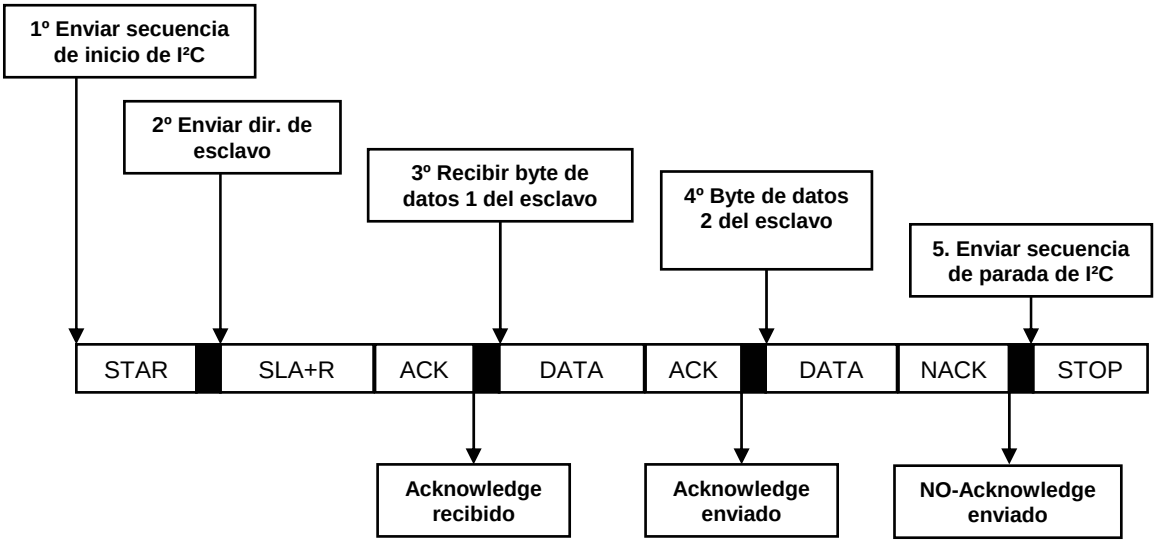


Figura 304: Secuencia funcional de una activación de esclavo

Puesto que el microcontrolador del módulo FLASH PSD1 no dispone de hardware para soportar un interfaz de I²C, las funciones de control para el protocolo de I²C están realizadas mediante el software en la biblioteca de funciones *iic.h*. Las funciones están especialmente adaptadas al hardware empleado. Ello permite utilizar el módulo FLASH PSD1 como maestro único en un sistema de bus I²C.

Para obtener más información consulte la especificación de I²C de la empresa Philips, así como el libro *Computerschnittstellen und Bussysteme (Interfaces de ordenador y sistemas de bus)* de *K. Dembowski* , 2ª edición 2001 revisada y ampliada de la editorial Hüthig (ISBN 3-7785-2782-7).

Visualización de caracteres en la indicadora LCD de I²C

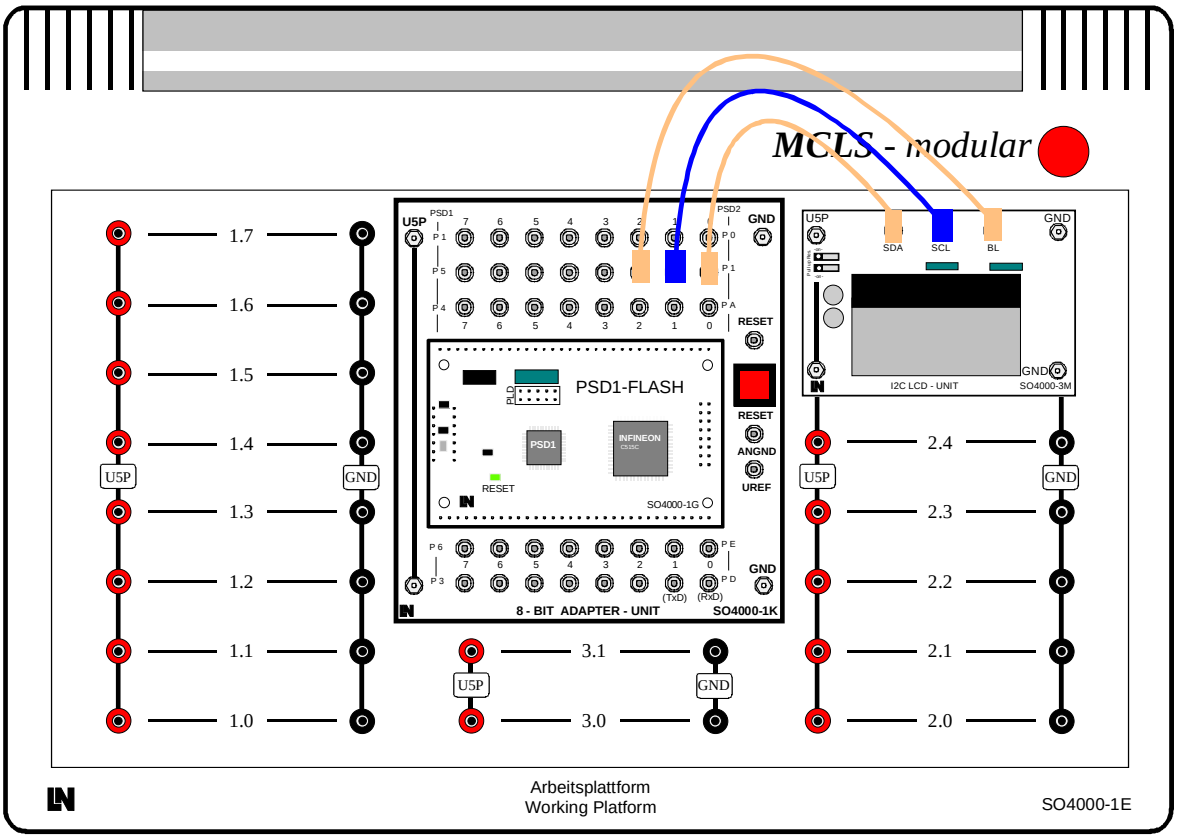


Fig. 305: Instalación de aparatos del ensayo CMC 5-3.1

Explicación del ensayo:

El siguiente ensayo se centra en la utilización de funciones preparadas. La base la forman las diferentes posibilidades de la visualización de caracteres en la indicadora LCD de I²C. Puesto que la LCD se opera por medio de un interfaz I²C con la dirección de I²C *0111010Xb* , hay que incluir los archivos de encabezamiento *iic.h* y *lcd.h* en el archivo fuente principal.

Antes del primer uso,la LCD debe inicializarse según el orden siguiente:

Secuencia	Función indicadora	Configuración posible
1	***no operation***	
2	***function set***	Modo transmisión 4 líneas / 8 bits
3	***display control***	Indicador ON y cursor OFF
4	***entry mode set***	Indicación congelada / Incremento cursor
5	***clear display***	Cursor en dirección 0

Tabla 301:Secuencia de la inicialización del LCD

Instruction	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		Exe-cute Time (max.)
NOP	0	0	0	0	0	0	0	0	no Operation	0
Clear Display	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter	165ms
Return Home	0	0	0	0	0	0	1	0	Sets DDRAM address 0 in counter. Also returns shifted display to original position. DDRAM contents remain unchanged	165ms
Entry Mode Set	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40µs
Display Control	0	0	0	0	1	D	C	B	Sets entire display on/off (D), cursor on/off (C) and blink of cursor position character (B)	40µs
Cursor dis-play/shift	0	0	0	1	S/C	R/L	0	0	Moves cursor and shifts display without changing DDRAM contents	40µs
Function set	0	0	1	DL	N	M	G	0	Sets interface data length (DL), number of display lines (N,M), and voltage generator control (G)	40µs
Set CGRAM Address	0	1	address CGRAM						Sets CGRAM address.	40µs
Set DDRAM Address	1	address DDRAM							Sets DDRAM address.	40µs

Tabla 302: Comandos de control para indicadora LCD EA 7123-I²C

Bit	0	1
I/D	Decrement	increment
S	display freeze	display shift
D	display off	display on
C	cursor off	cursor on
B	character at cursor position does not blink	character at cursor position blinks
S/C	cursor move	display shift
R/L	left shift	right shift
DL	4 bits	8 bits
G	voltage generator: $V_{LCD} = V_O$	voltage generator: $V_{LCD} = V_O - 0{,}8V_{DD}$
N, (M=0)	1 line x 24 characters	2 lines x 24 characters
N, (M=1)	Reserved	4 lines x 12 characters

Tabla 303: Definición de bits de control

El archivo de encabezamiento *lcd.h* contiene la función *LCD_INIT();*. Esta función está basada en las funciones de control indicadas en la tabla 19 para la LCD y solamente debe llamarse una sola vez en el programa total. Tras la llamada a esta función, el cursor de la LCD se encuentra en la posición *Línea 0*, *Carácter 0* (esquina superior izquierda del LCD).

La indicadora LCD puede procesar directamente los códigos ASCII. Adicionalmente es posible visualizar caracteres especiales que también están archivados en una tabla de caracteres contenida en el controlador de la LCD.

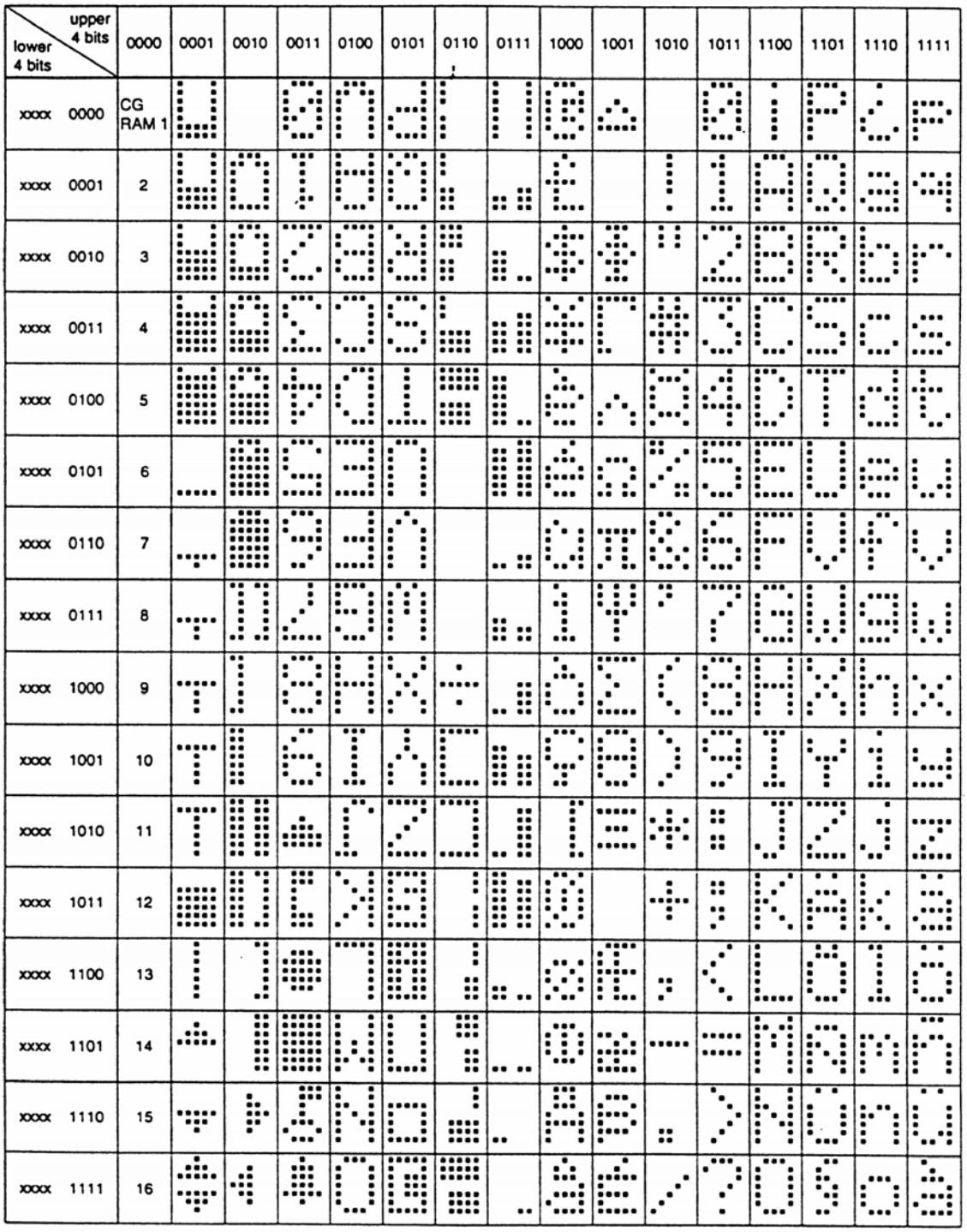
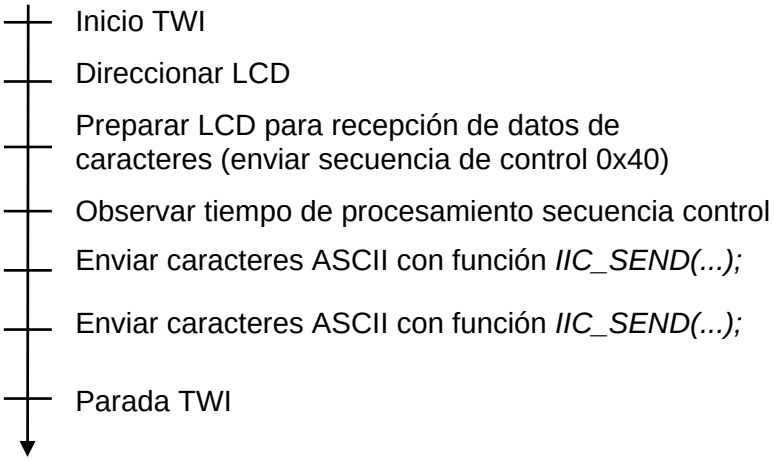


Figura 306: Tabla interna de caracteres de la LCD de I²C

Para la visualización de caracteres ASCII y especiales se debe observar la siguiente secuencia:



Nota:

En la tabla de caracteres de la LCD, el código ASCII está archivado por una parte para un posicionamiento normal de la LCD y por otra para un posicionamiento girado en 180°, de forma que la indicadora se pueda emplear en ambas posiciones de montaje. De la posición de montaje realizada de la indicadora en la unidad LCD de I2C resulta la necesidad de una adaptación del carácter a visualizar. Poniendo el MSB (operador OR con 0x80) en el byte de datos a enviar, se le asigna a cada carácter ASCII la dirección correcta en la tabla del controlador de la LCD.

Ejemplo de programa: *IIC_SEND('A'|0x80);* *// Visualización carácter ASCII A*

Puesto que la indicadora dispone de 3 líneas con 12 caracteres cada una (columnas), el archivo de encabezamiento *lcd.h* contiene la función *LCD_gotoXY(unsigned char line,unsigned char column);* para posicionar el cursor de la indicadora. Para cada línea de indicadora está definida una dirección de inicio.

Línea de indicadora	Dirección de inicio
1	0x00
2	0x20
3	0x40

Tabla 304: Direcciones de inicio de las líneas de indicadora

A esta dirección de inicio se suma en la función *LCD_gotoXY(..);* el número de columna transferido (posición de carácter) y se envía a la LCD. Para obtener más información consulte la sección H del presente programa de ensayo.

Ejercicios de programación:

- ¡Abra un nuevo proyecto!
- ¡Copie los archivos de encabezamiento *iic.h* y *lcd.h* al directorio de proyecto!
- ¡Incluya los archivos de encabezamiento *iic.h* y *lcd.h* en el archivo fuente principal!
- ¡Tras la llamada a las funciones de inicialización para el interfaz I²C y la LCD utilizando las funciones disponibles en *iic.h* y *lcd.h*, visualice caracteres de texto individuales!
- ¡Modifique las posiciones de visualización de los caracteres ASCII por medio de la función *LCD_gotoXY();*

Ejemplo de solución:

```
/*********************************************************************/
/*    cmc5-31: Programa para visualizar caracteres ASCII en LCD I2C */
/*    Autor:      ACMC/hpo                                           */
/*    Fecha:      07/04                                              */
/*    Software:   SDCC                                               */
/*    Hardware:   Flash PSD1                                         */
/*       Unidad LCD de I2C                                           */
/*       BL  -> P5.2                                                 */
/*       SCL -> P5.1                                                 */
/*       SDA -> P5.0                                                 */
/*********************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>

#include "iic.h"     // Funciones de software para bus I2C
#include "lcd.h"     // Funciones para la activación de la unidad LCD
                     // de I2C

// Programa principal ------------------------------------------------
void main(void)
{
   // INIC para I2C, LCD
   IIC_INIT();
   LCD_INIT();
      LCD_gotoXY(2,1);      // Línea2, Carácter1
      IIC_START();
      IIC_SEND(adr_lcd);
      IIC_SEND(0x40);
      wait_IIC(5);
   IIC_SEND('A'|0x80);
   IIC_SEND('C'|0x80);
   IIC_SEND('M'|0x80);
   IIC_SEND('C'|0x80);
      IIC_STOP();

      for(;;)              // Bucle sinfín
      {

   }  // end for
}  // end main
```

Visualización de cadenas de caracteres en la LCD

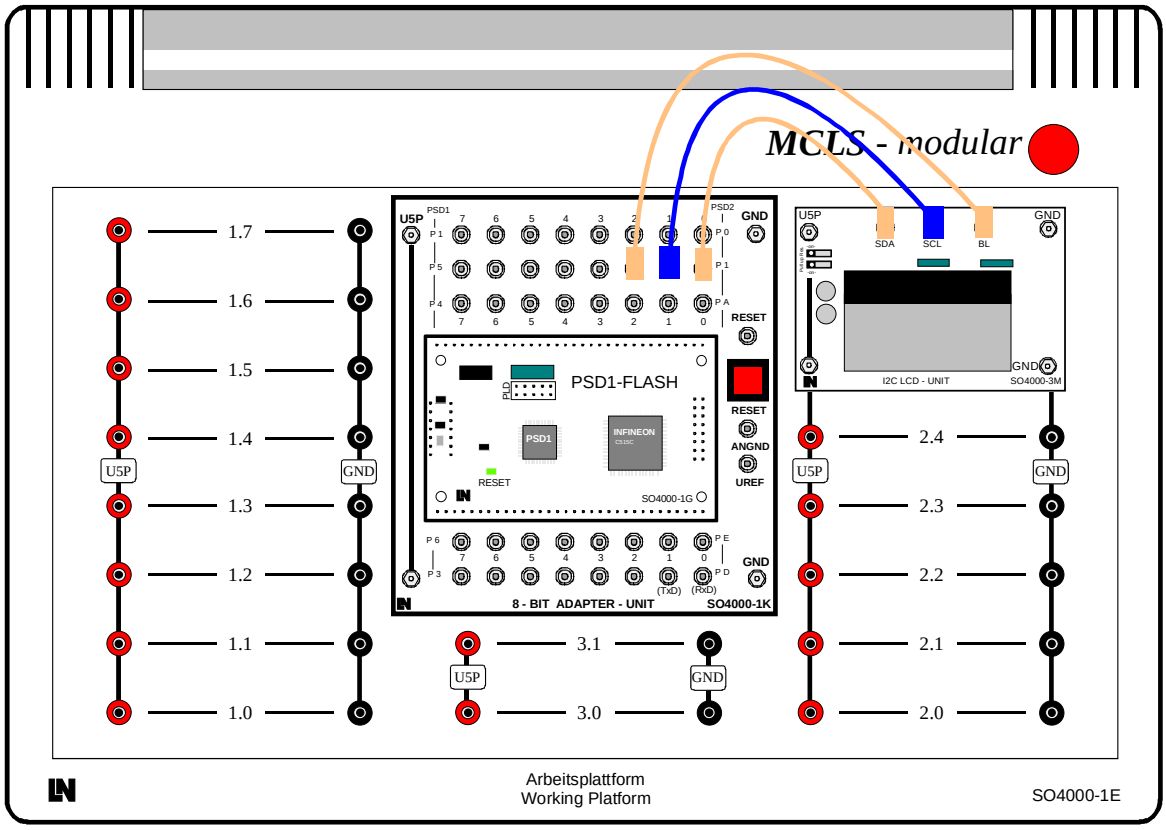


Fig. 307: Instalación de aparatos del ensayo CMC 5-3.2

Explicación del ensayo:

En el archivo de encabezamiento *lcd.h* se encuentra implementada una función que permite la visualización de cadenas de caracteres con terminación en cero (→ sección H). A esta función *LCD_SENDSTRING(unsigned char *ptr_char);* se debe transferir la dirección de inicio de una cadena terminada en cero.

Ejemplo para la declaración de una cadena de caracteres terminada en cero en la memoria de códigos:

```
code  unsigned char String[] = {"Texto"};     // Cadena de caracteres
```

La palabra clave *code* es específica del Compilador y ordena al Enlazador de guardar la cadena como cadena de caracteres constantes en el área de memoria de códigos.

Ejemplo para la llamada a la función para visualizar una cadena de caracteres terminada en cero:

```
LCD_SENDSTRING(String);          // Llamada a la función de visualización
```

La función *LCD_SENDSTRING()* puede ser precedida de una llamada a la función *LCD_gotoXY()* para el posicionamiento de la cadena de caracteres terminada en cero en la LCD.

Ejercicios de programación:

- ¡Estudie el texto fuente de la función *LCD_SENDSTRING(unsigned char *ptr_char);*!
- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h* y *lcd.h* al directorio de proyectos!
- ¡Incluya los archivos de encabezamiento *iic.h* y *lcd.h* en el archivo fuente principal!
- ¡Inicialice una cadena de caracteres terminada en cero en el área de códigos del módulo FLASH_PSD1 y visualícela mediante la función *LCD_gotoXY(...)* en la indicadora!

Ejemplo de solución:

```
/*****************************************************************/
/*    cmc5-32:    Programa para la visualización de una cadena    */
/*                de texto en LCD de I2C-LCD                      */
/*    Autor:      ACMC/hpo                                        */
/*    Fecha:      07/04                                           */
/*    Software:   SDCC                                            */
/*    Hardware:   Flash PSD1                                      */
/*       Unidad LCD I2C                                           */
/*       BL  -> P5.2                                              */
/*       SCL -> P5.1                                              */
/*       SDA -> P5.0                                              */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>

#include "iic.h"     // Funciones de software para bus I2C
#include "lcd.h"     // Funciones para la activación de la unidad LCD
                     // de I2C

 // Campos de texto para visualización en LCD
const unsigned char TEXT[]={"MCLS-modular"};

// Programa principal ------------------------------------------
void main(void)









}  // end main
```

Visualización de un texto progresivo en la LCD

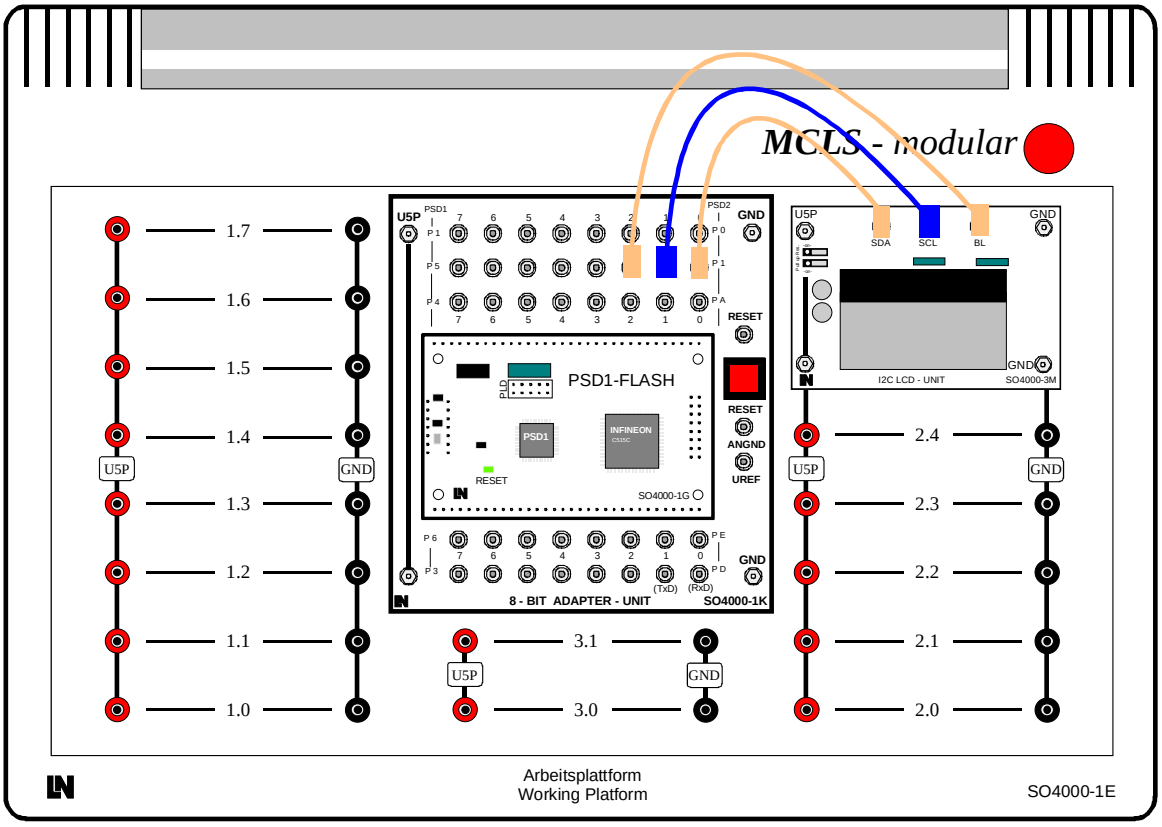


Fig. 308: Instalación de aparatos del ensayo CMC 5-3.3

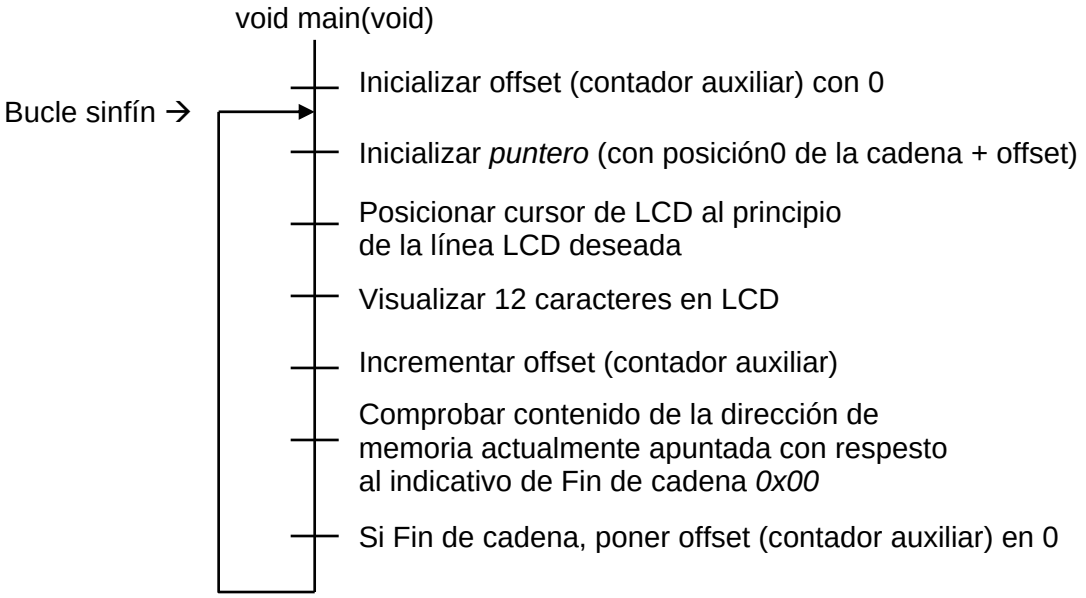
Explicación del ensayo:

La función *LCD_SENDSTRING(unsigned char *ptr_char);* solamente está apropiada para la visualización de cadenas de caracteres terminadas en cero que contienen hasta 12 caracteres de texto. Si hay que visualizar cadenas de caracteres más largas, esto se puede realizar mediante un texto progresivo. El tamaño de una cadena individual entonces solamente está limitado, en teoría, por el tamaño de la memoria de códigos disponible. El problema puede solucionarse de forma efectiva utilizando un puntero.

Ejemplo para la declaración de una cadena de caracteres en el área de memoria de códigos:

```
code  unsigned char cadena de caracteres [] = {"      Texto"};   // Cadena de caracteres con
                                                                 // 12 espacios

unsigned char code  *puntero = cadena de caracteres;          // Puntero en cadena
```

Tras las inicializaciones necesarias para la periferia, hay que implementar en el programa principal la siguiente secuencia de programa:



Ejercicios de programación:

- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h* y *lcd.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!
- ¡Inicialice una cadena de caracteres y visualícela de forma retardada en la línea 2 de la indicadora de la LCD como texto progresivo de la derecha a la izquierda! ¡Utilice el diagrama de flujo del programa y emplée sentencias de control apropiadas!

Ejemplo de solución:

```
/*****************************************************************/
/*    cmc5-33: Programa para la visualización de una cadena      */
/*             de texto progresiva en LCD de I2C                 */
/*    Autor:      ACMC/hpo                                       */
/*    Fecha:      07/04                                          */
/*    Software:   SDCC                                           */
/*    Hardware:   Flash PSD1                                     */
/*       Unidad LCD de I2C                                       */
/*       BL  -> P5.2                                             */
/*       SCL -> P5.1                                             */
/*       SDA -> P5.0                                             */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515
#include <mcs51reg.h>
#include "delay.h"
#include "iic.h"  // Funciones de software para bus I2C
#include "lcd.h"  // Funciones para la activación de la unidad LCD
                  // de I2C

// Campo de texto para display LCD
code unsigned char TEXT[]={"            MCLS-modular -> created in
Germany by LN and ACMC            "};
unsigned char code *ptr_menu;

// Programa principal ---------------------------------------------
```

Medición y visualización continua de una tensión análoga mediante el ADC

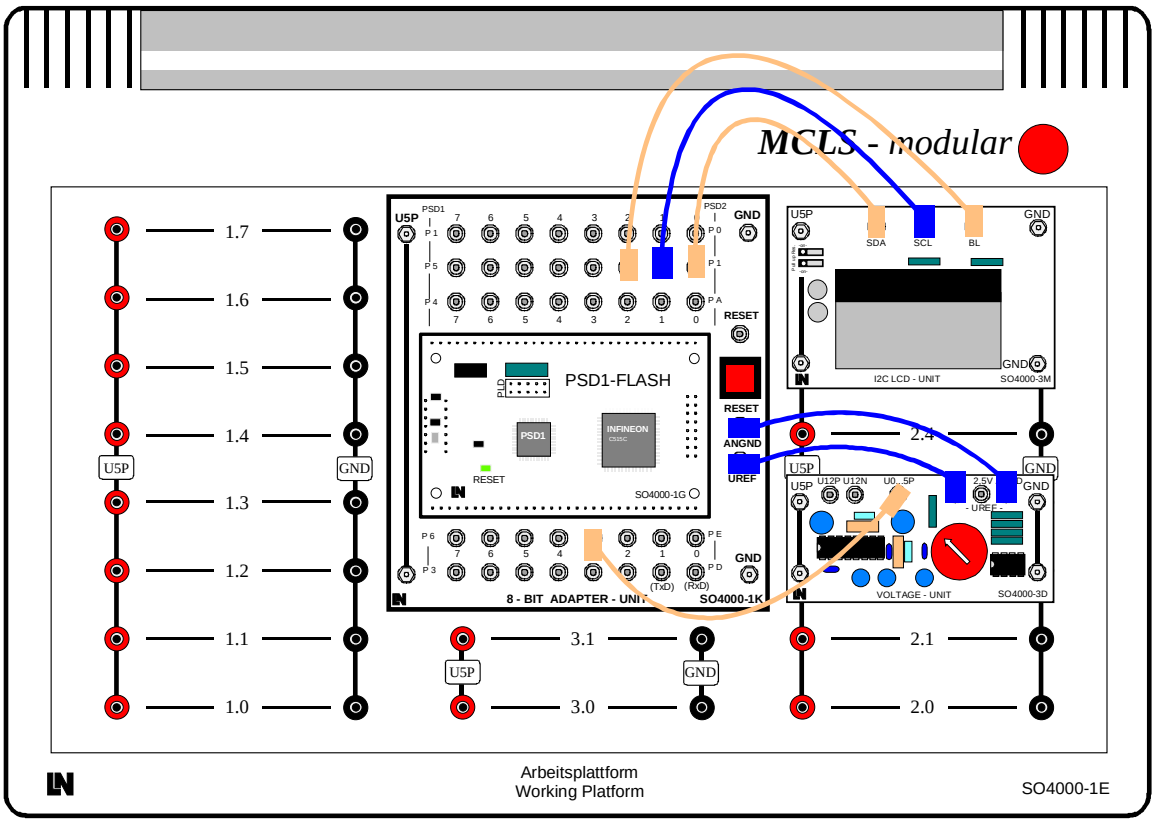


Fig. 309: Instalación de aparatos del ensayo CMC 5-3.4

Explicación del ensayo:

La tensión disponible en la salida de tensión continua variable de la unidad de tensión (*0 ... 5V*) deberá ser medida de forma continua por medio del Convertidor Análogo/Digital (ADC) integrado y visualizada en la indicadora utilizando la entrada análoga *AN3* del convertidor. La visualización del valor de tensión medido se realiza por medio de la unidad LCD de I²C. Para la inicialización del ADC se utilizará el registro de funciones especiales *ADCON0*.

7	6	5	4	3	2	1	0	
BD	CLK	ADEX	BSY	ADM	MX2	MX1	MX0	ADCON0

Figura 310: SFR ADCON0

La selección del canal AD se realiza por medio de los bits 0..2, y con el bit 3 se ajusta el tipo de conversión, o sea conversión sencilla (*ADM=0*) o continua (*ADM=1*). El bit 4 indica el estado del ADC.

El resultado de la conversión AD en el canal AD ajustado está archivado en los registros de funciones especiales ADDATH y ADDATL.

Bit No.	MSB							LSB	
	7	6	5	4	3	2	1	0	
$D9_H$	MSB .9	.8	.7	.6	.5	.4	.3	.2	ADDATH
DA_H	.1	LSB .0	–	–	–	–	–	–	ADDATL

Figura 311: Registro de resultados del ADC

Puesto que los registros de resultados del ADC solamente pueden leerse uno a uno, resulta conveniente la utilización de operaciones de manipulación de bits para guardar el resultado completo en una variable del tipo de datos *unsigned int* para su posterior utilización.

```
unsigned int ADin;          // Variable para resultado de 10bits
...
ADin = ADDATH;              // Leer 8 bits superiores del ADC
ADin <<= 2;                 // Desplazar 2 posiciones a la izquierda
ADin |= (ADDATL>>6);        // Leer 8 bits inferiores del ADC, desplazar 6
                               posiciones a la derecha, enlazar con variable
                               de resultado O
```

Tras esta secuencia, el resultado de 10bits está en la variable *ADin*.

Bit 1510	9	8	7	6	5	4	3	2	1	0	
-	-	MSB .9	.8	.7	.6	.5	.4	.3	.2	.1	LSB .0

Figura 312: Resultado de 10bits en Adin

Ejercicios de pogramación:

- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h* y *intbcd.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!
- ¡Inicialice el ADC en modo de conversión continua! ¡Consulte el manual de instrucciones del módulo FLASH PSD1!
- ¡Visualice el resultado de conversión de 10bits *ADDATH* y *ADDATL* en la línea 1 y el valor de tensión normalizado en la línea 2 de la unidad LCD de I²C! ¡Dibuje un diagrama de flujo del programa para el bucle sinfín del programa principal!
- ¡Utilice, para la visualización, las bibliotecas de funciones disponibles *iic.h*, *lcd.h* y *intbcd.h*!
- ¡Utilice operadores estándar para la normalización!

Ejemplo de solución:

```
/*********************************************************************/
/*    CMC5-34:   Conversión AD continua de una tensión análoga       */
/*    Autor:      ACMC/hpo                                           */
/*    Fecha:      07/04                                              */
/*    Software:   SDCC                                               */
/*    Hardware:   Flash PSD1                                         */
/*       Unidad LCD I2C                                              */
/*                                                                   */
/*       SCL -> P5.1                                                 */
/*       SDA -> P5.0                                                 */
/*                                                                   */
/*       UNIDAD DE TENSIÓN                                           */
/*                                                                   */
/*       V0..5P  -> P6.3 (entrada análoga AN3)                       */
/*       Uref 5V -> UREF                                             */
/*       GND     -> AGND                                             */
/*********************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>

#include "iic.h"    // Funciones de software para bus I2C
#include "lcd.h"    // Funciones para la activación de la unidad LCD
                    // de I2C
#include "intbcd.h"

// Variables globales
unsigned int ADin;

// Prototipos de funciones
void init(void);

// Programa principal ----------------------------------------------
void main(void)
{
   init();

      for(;;)                  // Bucle sinfín
      {

      if(BSY == 0)             // Espere mientras conversión en
                               // curso -> BSY = 1
      {
         ADin = ADDATH;        // Leer los 8 bits superiores del ADC
         ADin <<= 2;           // Desplazar 2 posiciones a la izquierda
         ADin |= (ADDATL>>6); // Leer los 2 bits inferiores del ADC,
                               // desplazar 6 posiciones a la derecha,
                               // enlazar con variable resultado
                               // O lógico

                               // Convertir variable resultado en
                               // dígitos BCD

                               // Visualizar en línea 1

         IIC_START();          // Comunicación TWI
         IIC_SEND(adr_lcd);    // Direccionar LCD
         IIC_SEND(0x40);       // Secuencia de control para LCD
         wait_IIC(5);
         IIC_SEND(0xb0|bcd1000);
         IIC_SEND(0xb0|bcd100);
         IIC_SEND(0xb0|bcd10);
```

```
            IIC_SEND(0xb0|bcd1);
            IIC_STOP();

            ADin = ((ADin*49)/10);  // Normalización de la variable
                                    // resultado
                                    // 5Volt / 1024 = 4,88mV => 49/10
                                    // Convertir variable resultado en
                                    // dígitos BCD

                                    // Visualizar en línea 2

            IIC_START();            // Comunicación TWI
            IIC_SEND(adr_lcd);      // Direccionar LCD
            IIC_SEND(0x40);         // Secuencia de control para LCD
            wait_IIC(5);
            IIC_SEND(0xb0|bcd1000);
            IIC_SEND(0x80|'.');
            IIC_SEND(0xb0|bcd100);
            IIC_SEND(0xb0|bcd10);
            IIC_SEND(0xb0|bcd1);
            IIC_SEND(0x80|'V');
            IIC_STOP();
         }  // end if
      }  // end for
   }  // end main

// Funciones ----------------------------------------------------------
void init(void)
{
   // INIC para I2C, LCD
   IIC_INIT();
   LCD_INIT();

   // Inicializar ADC
   ADCON0&=0xe0;
   ADCON0|=0x0b;
}  // end init
```

Medición y visualización cíclica de una tensión análoga por medio del ADC mediante interrupción de temporizador

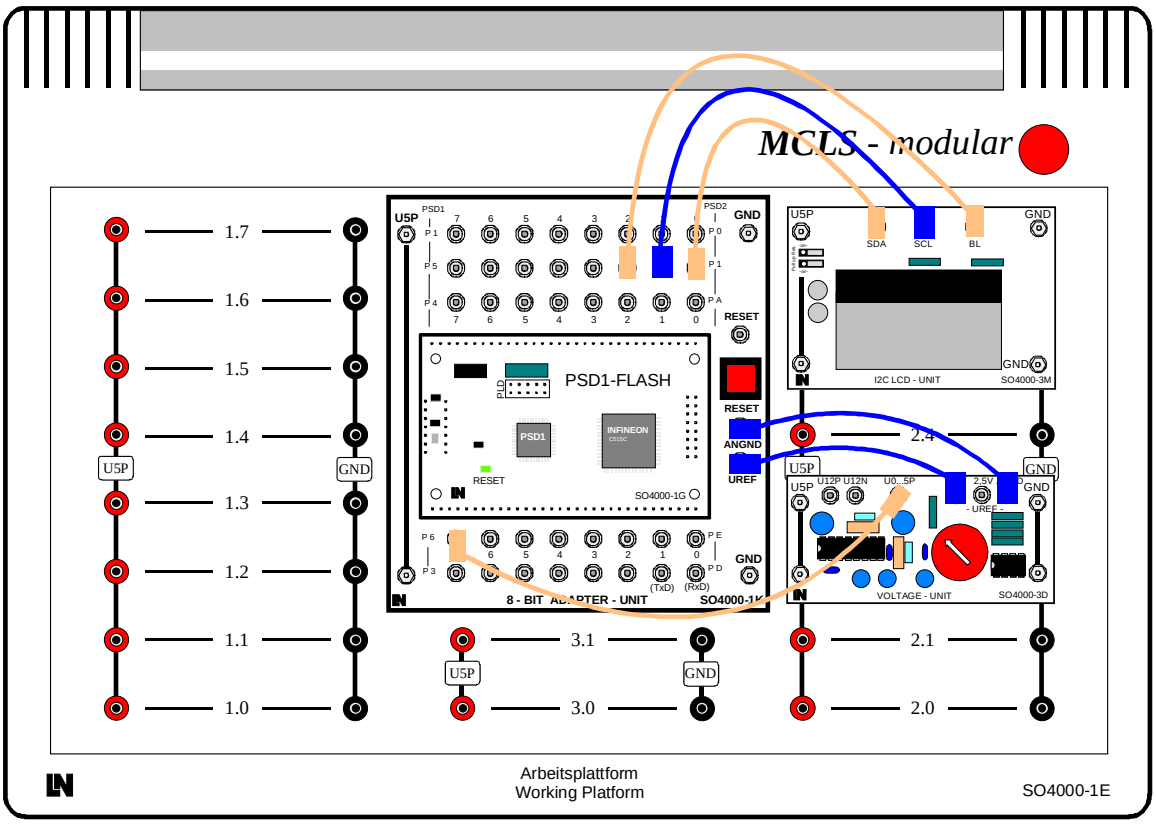


Fig. 312: Instalación de aparatos del ensayo CMC 5-3.5

Explicación del ensayo:

La tensión disponible en la salida de tensión continua variable de la unidad de tensión (0 ... 5V) deberá ser medida de forma cíclica por medio del Convertidor Análogo/Digital (ADC) integrado y visualizada en la indicadora utilizando la entrada análoga AN7 del convertidor. La visualización del valor de tensión medido se realiza por medio de la unidad LCD de I²C.

Para la generación del retardo de tiempo de exactamente 800ms se empleará el Timer 2 (→ ***CMC5-2***).

El ADC se configurará en modo de conversión sencilla (→ ***CMC5-3.4***).

Ejercicios de programación:

- ¡Dibuje un diagrama de flujo del programa para la rutina de servicio de interrupción del Timer 2!
- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h* y *intbcd.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!

- ¡Inicialice el Timer 2 como temporizador para 800 milisegundos (→ manual de instrucciones del módulo FLASH PSD1)!
- ¡Inicialice el ADC en modo de conversión sencilla! ¡Utilice el manual de instrucciones del módulo FLASH PSD1 (→***CMC5-3.4)***!
- ¡Inicie, después de cada intervalo de 800 ms, la conversión sencilla de la señal en la entrada análoga AN7!
- ¡Visualice el resultado de conversión de 10bits de *ADDATH* y *ADDATL* en la línea 1 y el valor de tensión normalizado en la línea 2 de la unidad LCD de I²C (→***CMC5-3.4)***!
- ¡Utilice, para la visualización, las bibliotecas de funciones disponibles *iic.h*, *lcd.h* y *intbcd.h*!
- ¡Utilice operaciones estándar para la normalización!

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   Conversión AD con Timer2                         */
/*              Overflow Interrupt autoreload (800ms)            */
/*    Autor:    ACMC/hpo                                         */
/*    Fecha:    07/04                                            */
/*    Software: SDCC                                             */
/*    Hardware: Flash PSD1                                       */
/*           UNIDAD de TENSIÖN  U0..5P -> P6.7 (entr. análoga AN7)*/
/*           Unidad LCD I2C SDA   -> P5.0                        */
/*                          SCL   -> P5.1                        */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>

// Includes ------------------------------------------------------
#include "iic.h"           // Funciones de software para bus I2C
#include "lcd.h"           // Funciones para la activación de la
                           // unidad LCD de I2C
#include "intbcd.h"        // Integer para BCD sencillo

// Variables globales --------------------------------------------
unsigned int ADin;
volatile unsigned int T2_count;

// Prototipos de funciones ---------------------------------------
void T2_isr (void) interrupt TF2_VECTOR using 1;
void init(void);
void wait(unsigned int);

// Programa principal --------------------------------------------




// Timer 2 ISR ---------------------------------------------------
```

```
// INIC de MC ----------------------------------------------------
void init(void)
{
   IIC_INIT();          // Inic I2C
   LCD_INIT();          // Inic LCD

   EAL   = 0;           // Bloquear todas las interrupciones
   // Timer 2 INIT
   TF2 = 0;             // Resetear flag de desbordamiento
   TH2   = 0x3c;        // Precargar registro
   TL2   = 0xaf;        // de conteo con 50ms
   CRCH  = 0x3c;        // Ajustar registro
   CRCL  = 0xaf;        // de recarga en 50ms
   T2CON &= 0x60;
   T2CON |= 0x10;       // Autorecarga, reloj 1/6
   ET2   = 1;           // Liberar interrupción Timer2
   T2I0  = 1;           // Inicio modo de temporizador

   // ADU INIT
   ADCON0&=0xe0;        // Conversión sencilla
   ADCON0|=0x07;        // AN7

   EAL   = 1;           // Liberar todas las interrupciones
}  // end init

// Retardo de tiempo ---------------------------------------------
void wait(unsigned int us)
{
   while(us) us--;
}
```

Preguntas de control del bloque de ensayos CMC5-3

¡Indique algunas características esenciales del bus I²C!

¿Qué dirección de bus I²C tiene la indicadora LCD de I²C?

¿Qué secuencia es necesaria par visualizar un carácter ASCII en la indicadora LCD de I²C?

¿Por qué se debe poner el MSB de cada carácter ASCII?

¿Cómo se puede declarar una cadena de caracteres terminada en cero en la memoria de códigos?

¿Con qué resolución trabaja el convertidor análogo-digital del C515C?

¿Cómo se puede realizar la conversión o normalización del valor medido de 10bits en un valor de tensión en síntaxis de C?

CMC 5-4 Bloque de ensayos 4

Aplicación del sensor de temperatura I²C LM75, utilización de campos de datos

Visualización de valores de temperatura del LM75

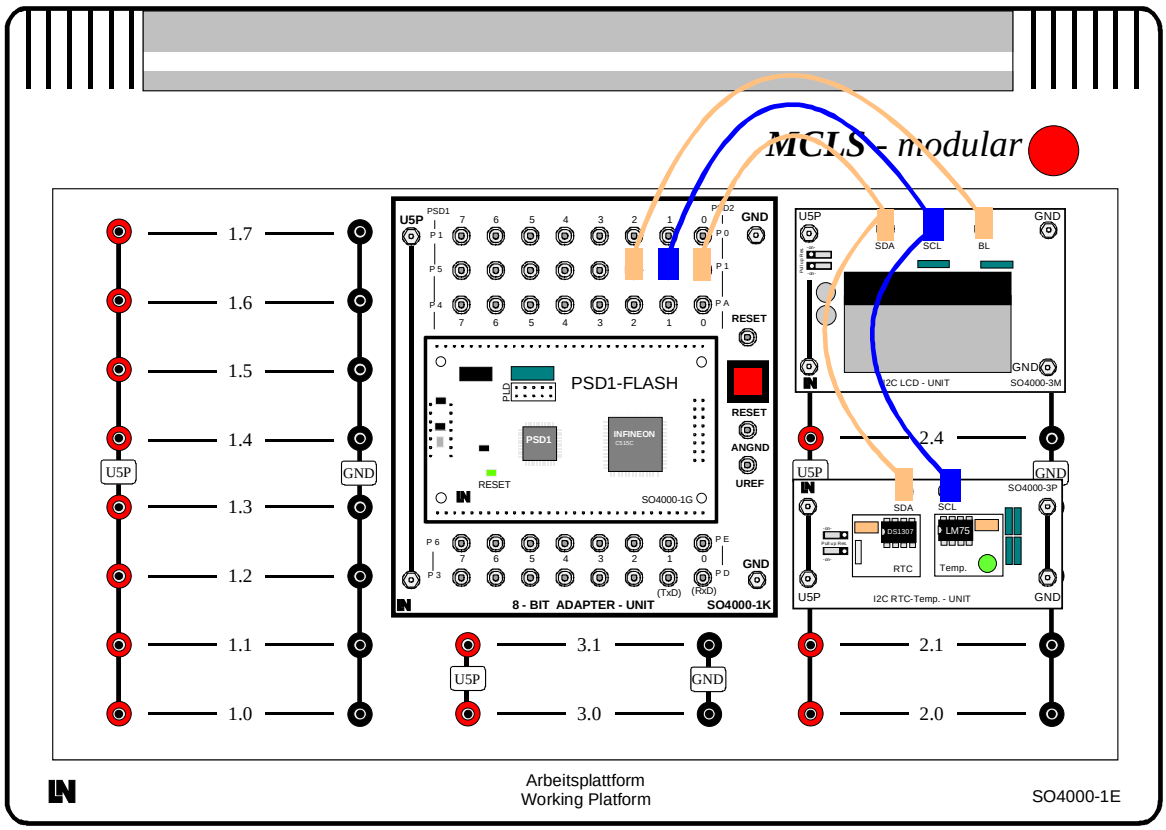


Fig. 401: Instalación de aparatos del ensayo CMC 5-4.1

Este ensayo ilustrará la activación del sensor de temperatura LM75, así como las posibilidades de procesamiento de los datos recibidos. El archivo de encabezamiento *lm75.h* contiene funciones para la lectura del valor de temperatura completo y para su visualización en la posición actual del cursor en el display del LCD de I²C.

Tras la generación de la secuencia de inicio de I²C, se activa el sensor de temperatura al anviar el microcontrolador el byte de dirección (1001 111X b) estando puesto el Bit0 en *read*. A ese fin se emplea el subprograma *IIC_SEND*. El LM75 envía una señal de confirmación (Acknowledge) y a continuación empieza a enviar el MSB del byte de temperatura. Con el subprograma *IIC_RECEIVE();* el microcontrolador recibe el byte de datos. Una vez finalizada con éxito la transmisión, el microcontrolador genera una señal de NO-confirmación (NO-Acknowledge). Puesto que la transmisión de datos finaliza en este punto, el microcontrolador envía la secuencia de parada de I²C al sensor. El siguiente gráfico representa la secuencia temporal de la recepción del byte de temperatura del sensor de temperatura LM75 por el microcontrolador mediante protocolo I²C.

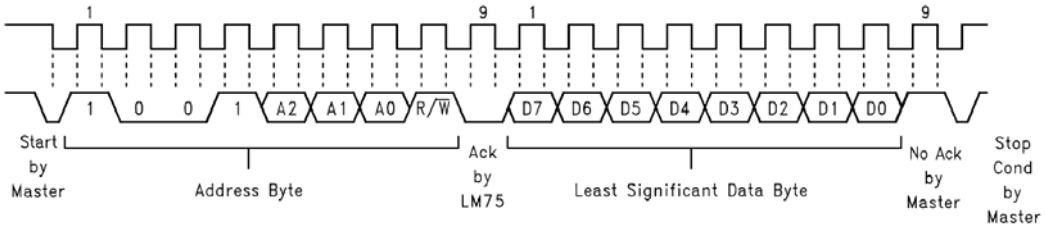


Figura 402: Transmisión de datos del LM75 al módulo FLASH PSD1

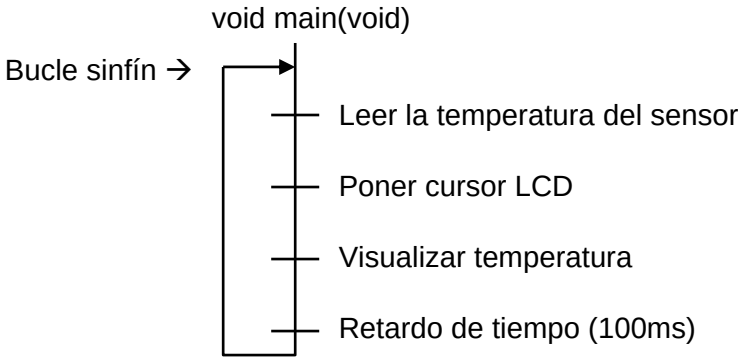
Puesto que los datos del LM75 se reciben en formato de complemento a dos, deben convertirse en un formato compatible con la indicadora LCD. El valor del byte de datos puede utilizarse directamente para una temperatura completa positiva. Se convierte dentro de la función *TEMP_OUT(unsigned char t1)* mediante la función *TEMP_LCD (unsigned char t2);* en variables codificadas en BCD para los dígitos de unidades y decenas para a continuación ser visualizado en la LCD de I²C.

Nota:

Entre la lectura de valores de temperatura sucesivos del LM75 se debe observar un tiempo de 100ms porque el convertidor análogo-digital interno del LM75 necesita este tiempo para poder suministrar el siguiente valor de temperatura. Si se realiza una nueva lectura del registro de temperatura dentro de los 100ms, se interrumpe la conversión AD interna en curso y se suministra el antiguo (anterior) valor de temperatura (→ *LM75.pdf*).

Ejercicio de programación:

- ¡Estudie las funciones del archivo de encabezamiento *lm75.h*!
- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h* y *lm75.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!
- ¡Utilizando las funciones disponibles y la secuencia de programa indicada para el bucle sinfín del programa principal, compile un programa que visualice de forma continua la temperatura actual en la línea 2 de la LCD de I²C!



Ejemplo de solución:

```
/*********************************************************************/
/* Título:   cmc5-41: Visualizar temperatura del LM75 en LCD de I2C */
/* Autor:    ACMC/hpo                                                */
/* Fecha:    07/04                                                   */
/* Software: SDCC                                                    */
/* Hardware: Flash PSD1 (C515)                                       */
/*       Unidad LCD I2C                                              */
/*       Unidad RTC-Temp. I2C                                        */
/* Nota: Unidades LCD y RTC-Temp. de I2C                             */
/*    SCL = P5.1                                                     */
/*    SDA = P5.0                                                     */
/*                                                                   */
/*********************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"

// Funciones de la periferia
#include "iic.h"           // Interface TWI
#include "lcd.h"           // LCD I²C
#include "lm75.h"          // Sensor de temperatura LM75

// Prototipos de funciones ---------------------------------------
void init(void);

// Programa principal --------------------------------------------
```

```
// MC-INIT -------------------------------------------------------------
void init(void)
{
   IIC_INIT();          // INIC de I2C
   LCD_INIT();          // INIC de LCD
}  // end init
```

Guardar varios valores de temperatura en un campo de datos, determinar valor máximo y valor mínimo

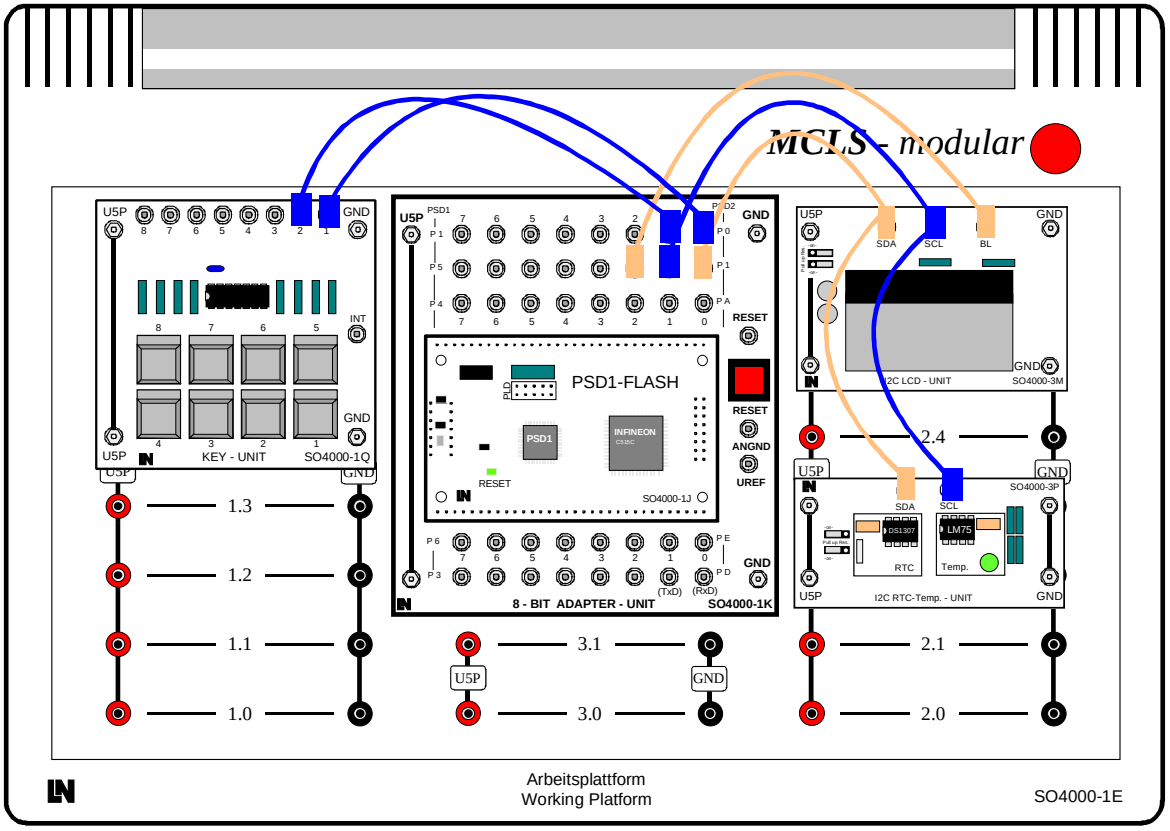


Fig. 403: Instalación de aparatos del ensayo CMC 5-4.2

Este ensayo mostrará la activación de I²C del sensor de temperatura LM75 así como las posibilidades de procesamiento de los datos recibidos. La compilación del programa se divide en varios pasos.

El primer paso consiste en la lectura de la temperatura completa de LM75 análogamente a *VK5-4.1*.

En el segundo paso se realiza la inserción de una evaluación de teclas en modo polling. A las teclas se deberá asignar la siguiente funcionalidad:

Tecla 1	→	Visualización de la temperatura máximo en la línea 2 de la LCD
Tecla 2	→	Visualización de la temperatura mínima en la línea 2 de la LCD

Para la evaluación de las teclas se puede utilizar una sentencia de control *switch*:

```
switch(P...)
{
        case 0x...:     break;
        …
}
```

Para la determinación de los valores de temperatura mínimo y máximo se deberán emplear sentencias de control, comparaciones y asignaciones apropiadas.

Ejemplo de programa para la determinación del valor máximo de la temperatura:

```
unsigned char temperature,tmin;          // Declaración de variable

 temperature = TEMP_IN();                // Introducción de temperatura de LM75

if(temperature < tmin) tmin = temperature;  // Comparación de variables
```

En el tercer paso del programa, un valor medio aritmético de 100 valores de temperatura con dos posiciones decimales deberá ser calculado y visualizado de forma continua en la LCD. Para la realización se puede utilizar un campo de datos. Puesto que el valor de temperatura de LM75 es un byte, el campo debería declararse con 100 elementos del tipo *unsigned char*. El acceso indexado a los distintos elementos de campo se puede realizar con una variable contadora.

```
unsigned char field_counter=0;          // Declaración/Inicialización de la variable
                                           contadora

middle_data[field_counter] = TEMP_IN();  // Guardar valor de temperatura del LM75
                                           en el campo

field_counter ++;                        // Contador de índices + 1
if(field_counter >99) field_counter =0;  // Limitación del contador de índices
```

Este tipo del acceso al campo también se denomina búfer de anillo, porque una vez alcanzado el último elemento de campo, el contador de índices se pone a cero y con ello el primer elemento de campo se sobreescribe con un valor nuevo.

Para el cálculo del valor medio aritmético de los 100 valores de temperatura con dos posiciones decimales pueden utilizarse los operadores estándar de C +, / y %.

Ejercicios de programación:

- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h* y *lm75.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!
- ¡Compile un programa con las siguientes funcionalidades!

Función de la tecla 1 de la UNIDAD de TECLAS:	Visualización temperatura máxima
Función de la tecla 2 de la UNIDAD de TECLAS:	Visualización temperatura mínima
sin tecla:	Visualización continua del valor medio aritmético de la temperatura con dos posiciones decimales que se forma de 100 valores

Ejemplo de solución:

```
/*********************************************************/
/* cmc5-42: Programa para visualizar la temperatura      */
/*          de LM75 en LCD de I2C                        */
/* (C) HPO 06/2004                                       */
/* Software:      SDCC  Versión 2.3.5 (Sep 29 2003)      */
/*          Debug51w for MCLS-modular PSD1-Unit          */
/*                                                       */
/* Hardware:   MCLS-modular con:                         */
/*             Flash PSD1 (C515)                         */
/*             Unidad LCD I2C                            */
/*             Unidad RTC-Temp I2C                       */
/*                                                       */
/* Conexiones de hardware en MCLS-modular                */
/*                                                       */
/*       UNIDAD DE TECLAS                                */
/*       Tecla1 -> P1.0                                  */
/*       Tecla2 -> P1.1                                  */
/*                                                       */
/* Unidad LCD I2C:  BL  = P5.2                           */
/*          SCL = P5.1                                   */
/*          SDA = P5.0                                   */
/*                                                       */
/* Unidad RTC-Temp I2C:SCL = P5.1                        */
/*          SDA = P5.0                                   */
/*********************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"

// Bibliotecas de funciones ------------------------------
#include "iic.h"                    // Interface TWI
#include "lcd.h"                    // LCD I²C
#include "lm75.h"                   // Sensor de temperatura

// Variables globales y campos ---------------------------

unsigned char middle_data[100];// Campo de datos para valores de
temperatura

unsigned char field_counter,tmax=0x00,tmin=0xff;
unsigned int middle=0,middle_rest=0;      // Variables auxiliares

const unsigned char min[]={"Min:  "};
const unsigned char max[]={"Max:  "};
const unsigned char mid[]={"MW "};

// Prototipos de funciones -------------------------------------------
void MiddleValue(void);
void MinMax(void);

// Programa principal ------------------------------------------------
```

```
// Funciones ----------------------------------------------------
void MiddleValue(void)
```

Inicialización de RTC DS1307, lectura y visualización de parámetros de tiempo

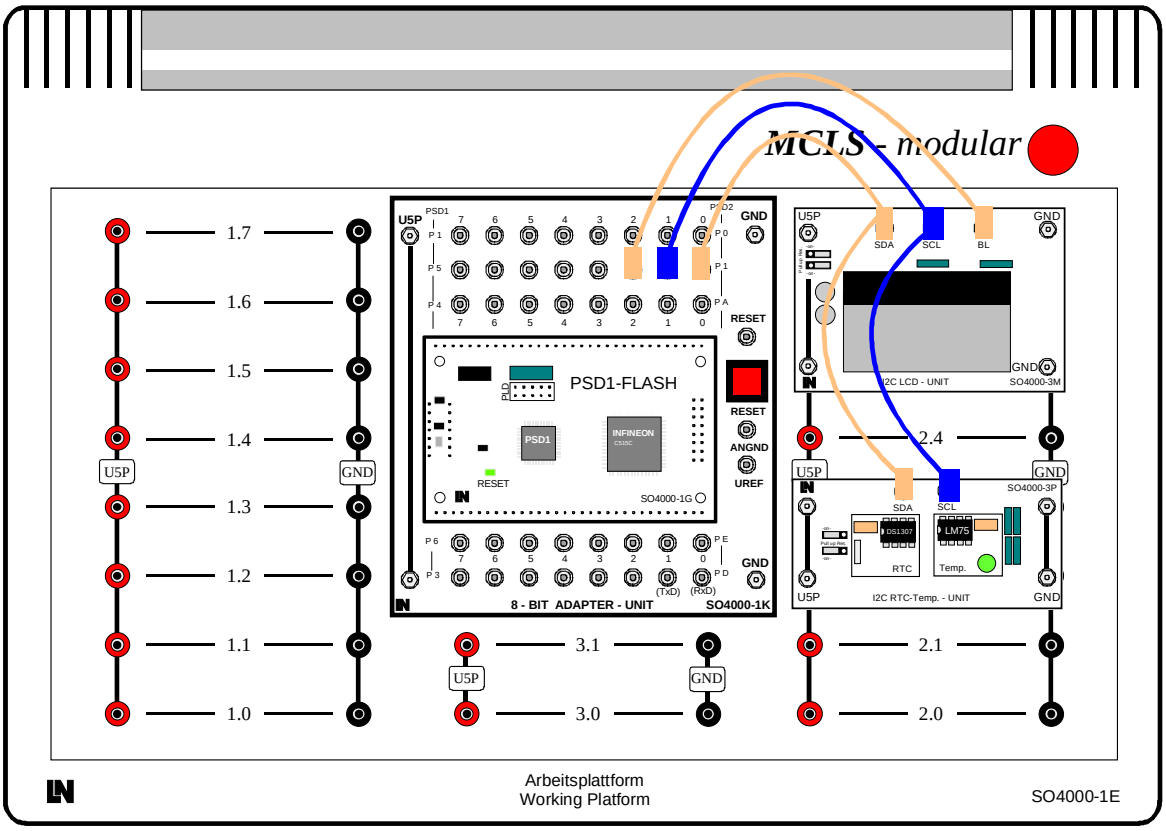


Fig. 404: Instalación de aparatos del ensayo CMC 5-4.3

El ensayo comprende el acceso a una estructura con varios objetos. La base la forma la activación de la unidad RTC DS1307. Las funciones para el intercambio de datos entre el microcontrolador y el reloj de tiempo real están contenidas en el archivo de encabezamiento *rtc.h* que se debe incluir en el archivo fuente principal.
El reloj de tiempo real DS1307 de DALLAS Semiconductor empleado en la unidad RTC-Temp. I2C es un reloj de baja potencia codificado en formato BCD con calendario y un RAM estático de 56 bytes que contiene información para segundos, minutos, horas, día de la semana, fecha, mes y año. La siguiente figura muestra la estructura interna del DS1307.

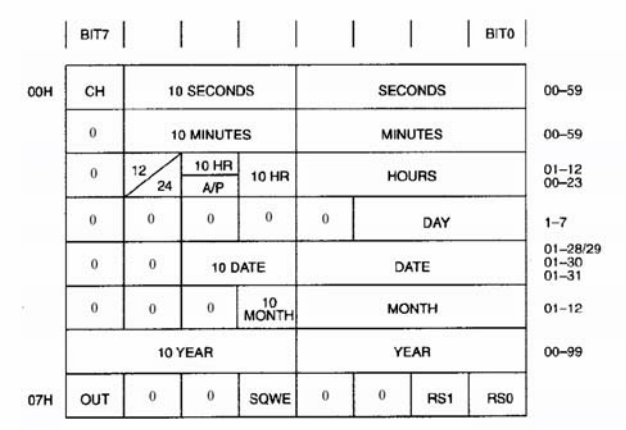


Figura 405: Estructura interna del DS1307

El reloj puede configurarse con el formato de 12 ó de 24 horas. Las transmisiones de datos desde y hacia el chip se realizan mediante protocolo I²C. El DS1307 trabaja como esclavo en el sistema I²C. La dirección de bus I²C ha sido ajustada por el fabricante a 1101000X binaria.

La recepción de datos por la RTC empieza con la generacón de la secuencia de inicio I²C por el microcontrolador. A continuación la RTC se direcciona con la dirección del intercambio de datos *write* y el puntero RTC interno es puesto en la dirección de memoria del registro deseado mediante la llamada al subprograma *IIC_SEND*. Con una nueva secuencia de inicio, así como el redireccionamiento de la RTC para la dirección de intercambio de datos *read,* el byte de datos de este registro RTC puede ser recibido por el microcontrolador con el subprograma *IIC_REC*. La comunicación finaliza con la secuencia de parada.

En la compilación del programa debe incluirse, en el archivo fuente principal, la biblioteca de funciones *rtc.h* que contiene las funciones de control del protocolo I²C (→ sección H). Además se deberá declarar una estructura con los distintos objetos para los datos del reloj de tiempo real, así como una variable estructura que contenga los objetos:

```
struct RTC
{
        unsigned char second;
        unsigned char minute;
        …
}rtc_ data;
```

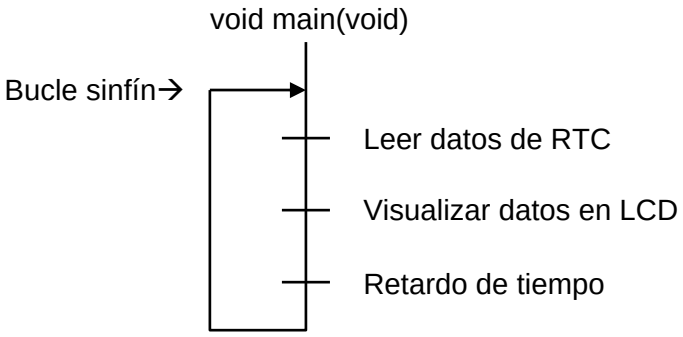
Con la ayuda del operador de punto se puede acceder a los distintos objetos en la variable estructura *rtc_data*. En el siguiente ejemplo de síntaxis se asigna al objeto de estructura *second* el valor de retorno de la función de biblioteca *SEC_IN();*:

```
rtc_data. second = SEC_IN();
```

Ejercicios de programación:

- ¡Abra un proyecto nuevo (→***VK5-1.1***)!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h*, así como *rtc.h* al directorio de proyectos e incluya los archivos en el archivo fuente principal!
- ¡Declare una estructura con la correspondiente variable estructura donde se puedan coordinar los datos del DS1307!

- ¡Utilice el siguiente diagrama de flujo para la compilación del programa!:



- ¡Lea mediante las funciones disponibles del archivo de encabezamiento *rtc.h* la información de RTC DS1307 y asigne los datos a los objetos de estructura declarados en la variable estructura declarada!
- ¡Visualice los cambios de los datos leídos de la RTC continuamente en la indicadora LCD!

Ejemplo de solución:

```
/*****************************************************************/
/*    Título:   cmc5-43: Leer RTC / Visual. parám. de tiempo   */
/*    Autor:    ACMC/hpo                                       */
/*    Fecha:    07/04                                          */
/*    Software: SDCC                                           */
/*    Hardware: Flash PSD1                                     */
/*    Nota:     UNIDAD LCD I2C                                 */
/*         Unidad RTC-Temp. I2C                                */
/*         Puerto5 -> P5_0 = SDA                               */
/*            P5_1 = SCL                                       */
/*****************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"
// Funciones de la periferia
#include "iic.h"              // Interfaz TWI
#include "lcd.h"              // LCD I²C
#include "rtc.h"              // RTC DS1307
// Prototipos de funciones ----------------------------------------
void init(void);
void data_in(void);
void data_out(void);

// Estructura para datos RTC con variable estructura ----------------
struct RTC
{
    unsigned char second;     // Objeto para segundos
    unsigned char minute;     // Objeto para minutos
    unsigned char hour;       // Objeto para horas
    unsigned char date;       // Objeto para días
    unsigned char month;      // Objeto para mes
    unsigned char year;       // Objeto para años
    unsigned char day;        // Objeto para días de la semana
}rtc_data;

// Programa principal ---------------------------------------------
void main(void)
```

```
{
   init();                    // Inicialización

   while(1)                   // Bucle sinfín
   {
      data_in();              // Leer datos de RTC

      data_out();             // Visualizar datos en LCD

      delay(50000);           // Retardo de tiempo
   }  // end while
}  // end main

// INIC de MC ---------------------------------------------------------
void init(void)
{
   IIC_INIT();                // INIC TWI
   LCD_INIT();                // INIC LCD
}  // end init

// Guardar datos en variable estructura -------------------------------
void data_in(void)
{
   rtc_data.second   =SEC_IN();  // Segundos de RTC
   rtc_data.minute   =MIN_IN();  // Minutos de RTC
   rtc_data.hour  =HOUR_IN();    // Horas de RTC
   rtc_data.date  =DATE_IN();    // Fecha de RTC
   rtc_data.month =MONTH_IN();   // Mes de RTC
   rtc_data.year  =YEAR_IN();    // Año de RTC
   rtc_data.day   =DAY_IN();     // Día de la semana de RTC
}  // end data_in

// Visualizar datos de variable estructura en LCD ---------------------
void data_out(void)
{









}  // end data_out
```

Preguntas de control, bloque de ensayos CMC 5-4

¿Qué formato de datos tienen los valores de temperatura medidos del LM75?

¿Cómo se puede determinar, en síntaxis C, el valor medio aritmético partiendo de n valores medidos?

¿Cómo se realiza, en síntaxis C, la declaración de un campo de datos con 10 elementos en tamaño de 8 bits?

¡Explique la diferencia entre campos y estructuras!

¿Qué función tiene el operador de punto en las estructuras?

¿En qué formato está guardada la información de los segundos dentro de la RTC DS1307?

CMC 5-5 Bloque de ensayos 5

Realización de una aplicación compleja (tarea de proyecto)

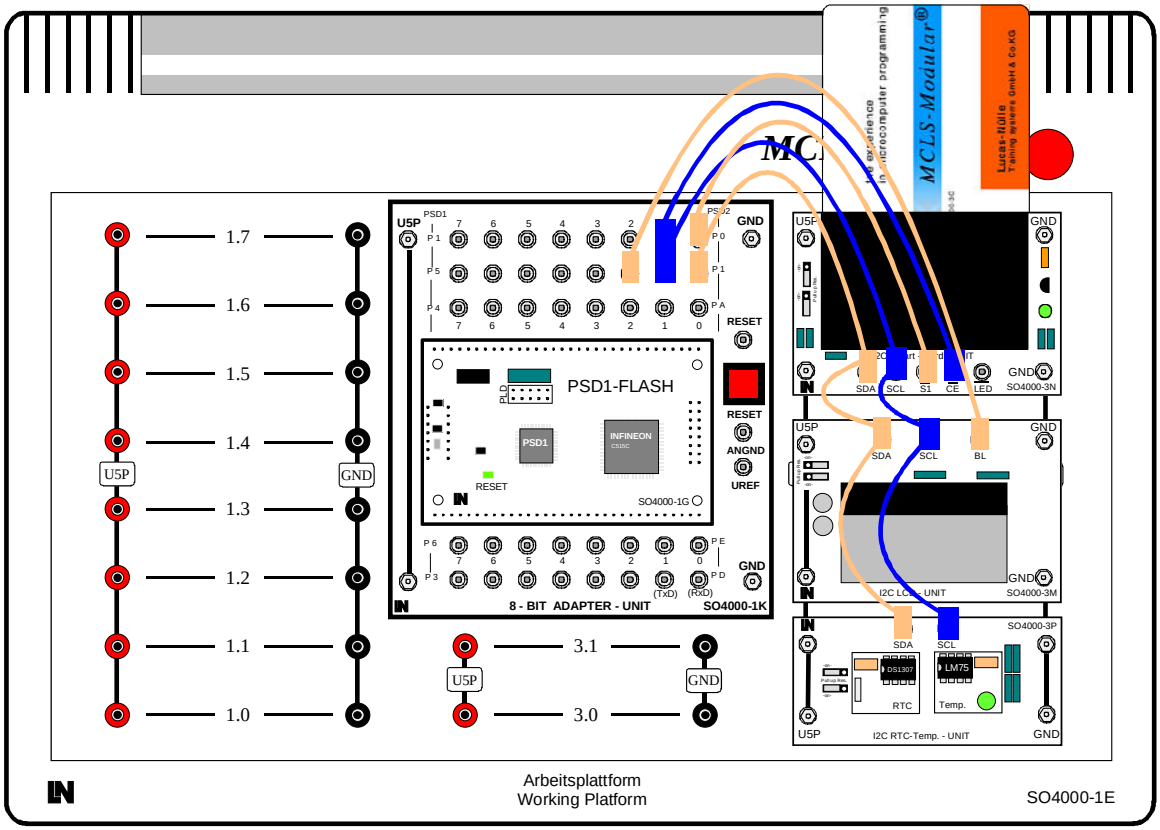


Fig. 501: Instalación de aparatos del ensayo CMC 5-5

En el último ensayo es un sistema de bus I²C que forma la base del hardware. El sistema se compone de un módulo FLASH PSD1 que actúa de maestro I²C, de la unidad RTC-Temp. I2C, de la unidad LCD I2C y de la unidad de Tarjeta Inteligente I2C con tarjeta chip I²C.

La tarjeta chip suministrada junto con la unidad de Tarjeta Inteligente I2C contiene un EEPROM que se puede cargar o que se puede leer, byte a byte, por medio del interfaz I²C. El tamaño de la memoria de tarjeta chip es de 256 bytes. La dirección del bus I²C ha sido fijada por parte del fabricante con 1010000Xb y está definida en el archivo de encabezamiento *iiccard.h*.

Para la comprobación de los valores en la tarjeta chip I²C se emplea el ChipDrive con el correspondiente software para el PC.

Explicación del ensayo:

El software de MC a realizar deberá registrar cíclicamente, a intervalos de 5 segundos que se generan por medio del Timer2, un valor del sensor de temperatura LM75 en una tarjeta chip I²C. El siguiente gráfico representa la estructura básica del programa:

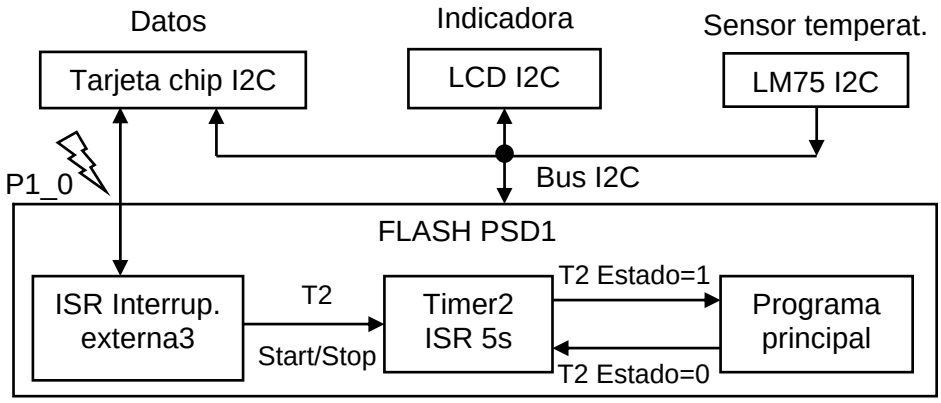


Figura 502: Estructura básica del ensayo complejo

Descripción del programa:
La interrupción externa3 que está conectada al interruptor /S1 de la unidad de Tarjeta Inteligente I2C, controla el Timer2. Con la tarjeta chip insertada, se inicia el Timer 2, y se detiene al retirarse la tarjeta. De este modo se garantiza que los valores de temperatura solamente se transmiten del LM75 cuando la tarjeta chip está insertada.

Después del inicio del Timer2, en la rutina de servicio de interrupción (cíclicamente tras 5 segundos), se pone un bit de estado que se debe evaluar en el bucle sinfín del programa principal. En el bucle sinfín del programa principal se consulta el estado de este bit de estado en modo Polling. Si el bit de estado está puesto, se lee el valor de temperatura actual del sensor LM75, se guarda este valor en la tarjeta chip y se visualiza en la LCD de I2C.

Las 256 locaciones de memoria para valores de 8 bits disponibles en la tarjeta chip utilizada se emplearán en su totalidad para guardar los valores de temperatura. A ese fin es conveniente emplear una variable contadora (cíclica) con el tipo de datos *character* sin límite.
Para la realización de la secuencia son necesarios los siguientes pasos:

void init(void)
- INIC I2C
- INIC LCD
- Contador aux. para ISR de Timer2 = 0
- Interr. externa3 = Flanco descendente
- Liberar interrupción externa3
- Precargar registro conteo con 50ms
- Poner registro recarga en 50ms
- Timer2 Modo 0: Autorecarga tras des- bordamiento, reloj 1/6
- Liberar interrupción Timer2
- Liberar todas las interr.

Fin de función

➢ **Inicializaciones de MC**

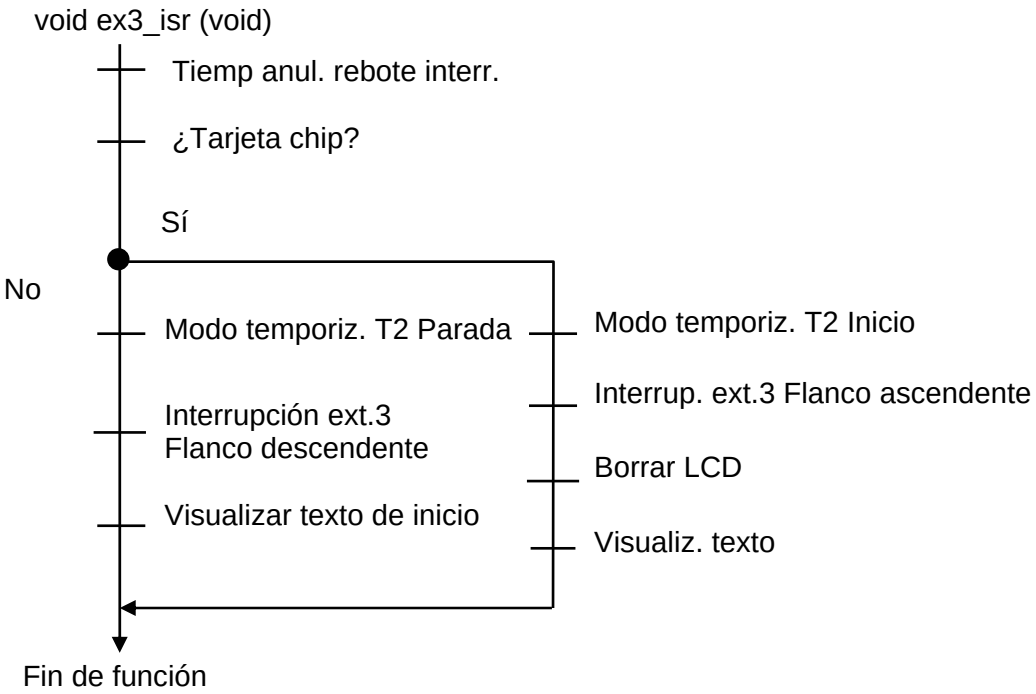
En esta parte del programa se deben inicalizar los componentes on chip y de periferia I²C necesarios.

➢ **ISR Interrupción externa3**

Por medio de la interrupción externa3 se realiza la vigilancia de la unidad de Tarjeta Inteligente I2C. Al insertar una tarjeta chip I2C, se inicia el Timer2 para generar la ventana de tiempo de 5 segundos.

La interrupción externa3 puede activarse mediante un flanco descendente o ascendente en el pin de puerto P1.0. El flanco que activa la rutina de servicio de interrupción puede configurarse mediante el bit de control *I3FR* en el registro de funciones especiales *T2CON*. Partiendo de la inicialización del *Puerto1* tras un *RESET de MC* con 0xff y el estado del interruptor */S1*, sin tarjeta chip insertada en la unidad de Tarjeta Inteligente I2C, la interrupción externa3 se debe configurar en estado inicial para que sea activada por un flanco descendente (*I3FR=0*). Al insertar la tarjeta chip se genera este flanco. Al retirar la tarjeta de la unidad de Tarjeta Inteligente I2C se abre el interruptor */S1* lo que da lugar a la generación de un flanco ascendente. Por lo tanto, antes de retirar la tarjeta chip, la interrupción externa3 debe ser ajustada para que sea activada por este flanco (*I3FR=1*).

El siguiente gráfico muestra la secuencia de la rutina de servicio de interrupción:



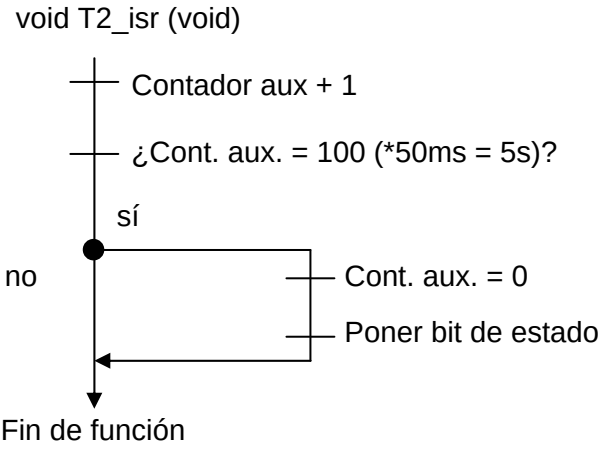
- **Timer 2 ISR**

La rutina de servicio de interrupción para el Timer2 en modo de desbordamiento con recarga automática suministra una ventana de tiempo de 5 segundos. Una vez transcurrido el tiempo hay que poner un bit de estado. Para la declaración de una variable en tamaño de bit se puede utilizar la siguiente definición en el SDCC*:*

```
bit T2ready=0;
```

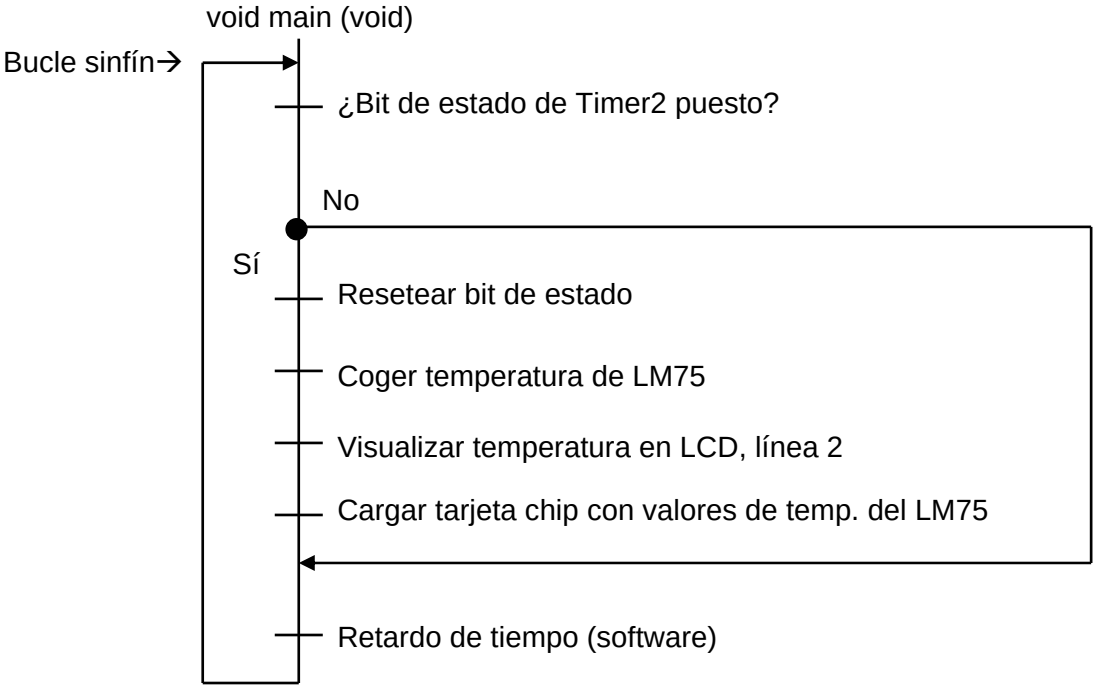
El Timer2 se inicia en la rutina de servicio de interrupción de la interrupción externa3 cuando se pone el bit *T2I0=1* y se detiene con el reseteo de *T2I0=0*. Este bit está localizado en el registro de funciones especiales *T2CON*.

El siguiente gráfico representa la rutina de servicio de interrupción:



➢ **Programa principal**

El bucle sinfín del programa principal evalua, con un retardo de tiempo, el estado del bit de estado del Timer2 en modo Polling. Cuando el bit de estado está puesto, se llama a las funciones útiles para la lectura del valor de temperatura del sensor LM75, para la visualización del valor convertido en la LCD de I²C y para la carga de la memoria de la tarjeta chip. El intercambio de datos entre los componentes del hardware se realiza mediante protocolo I²C y las funciones para el control del respectivo aparato I²C. Adicionalmente a los ensayos previamente realizados, el archivo de encabezamiento "*iiccard.h*" contiene una función para escribir en una locación de memoria definida en la tarjeta chip de I²C. El siguiente diagrama de flujo describe el programa principal como bucle sinfín:



Nota acerca del formato de datos de los valores medidos para guardarlos en la tarjeta chip:

Para un posterior procesamiento con un programa de hojas de cálculo en un PC (p. ej. MS-Excel) es conveniente guardar los valores medidos en la tarjeta chip en formato BCD comprimido. Se puede utilizar la siguiente secuencia de comandos:

```
unsigned char temperature,help;     // Variables para la siguiente secuencia

temperature = 0x17;                 // Valor de ejemplo 17 hex => 23 decimal
intbcd(temperature);                // Conversión bcd de temperature
help = 0x00;                        // Borrar contenido de help
help |= bcd10;                      // Archivar BCD de decenas en help
help  = ((help<<4)|(help>>4));      // Cambiar BCD de decenas en parte alta
                                   // de 4bits
help |= bcd1;                       // Enlazar help con BCD de unidad
```

→ Resultado: help = 0x23 → Formato BCD comprimido

Ejercicios de programación:

- ¡Abra un proyecto nuevo!
- ¡Copie los archivos de encabezamiento *iic.h*, *lcd.h, lm75.h, iiccard.h, intbcd.h* y *delay.h* al directorio de proyectos e inclúyalos con la instrucción *include* en el archivo fuente principal!
- ¡Inicialice la interrupción externa3, el Timer2, así como los componentes de I²C según las explicaciones del ensayo!
- ¡Realice, en el primer paso, el control del Timer 2 mediante la interrupción externa3! ¡Utilice los diagramas de flujo indicados en las explicaciones del ensayo!
- ¡Implemente, en un segundo paso, la secuencia representada del bucle sinfín utilizando las bibliotecas de funciones disponibles para la periferia de I²C!
- ¡Lea los datos guardados con la ayuda de un ChipDrive para tarjetas chip (opción recomendada para el bloque de ensayos) en el PC y crée, utilizando un programa de hojas de cálculo adecuado, un diagrama del curso de la temperatura en función del tiempo!

Ejemplo de solución

```
/*************************************************************/
/*    Título:     cmc5-5 Sistema I2C                         */
/*    Autor:      ACMC/hpo                                   */
/*    Fecha:      07/04                                      */
/*    Software:   SDCC                                       */
/*    Hardware:   Flash PSD1                                 */
/*                                                           */
/*          UNIDAD LCD I2C                                   */
/*       P5.0 -> SDA                                         */
/*          P5.1 -> SCL                                      */
/*                                                           */
/*          UNIDAD RTC-Temp. I2C                             */
/*       P5.0 -> SDA                                         */
/*          P5.1 -> SCL                                      */
/*                                                           */
/*          UNIDAD Tarjeta Inteligente I2C                   */
/*          P1.0 -> /S1                                      */
/*          P1.1 -> /CE                                      */
/*       P5.0 -> SDA                                         */
/*          P5.1 -> SCL                                      */
/*************************************************************/
#define MICROCONTROLLER_SAB80515A
#include <mcs51reg.h>
#include "delay.h"

// Bibliotecas de funciones ----------------------------------------
#include "iic.h"
#include "lcd.h"
#include "lm75.h"
#include "iiccard.h"
#include "intbcd.h"

// Variables globales ----------------------------------------------



 // Campos de texto para display de LCD



// Prototipos de funciones -----------------------------------------



// Programa principal ----------------------------------------------
```

```
// Timer 2 ISR -----------------------------------------------------
```

```
// ISR interrupción externa3 ---------------------------------------
```

```
// Inicializaciones de MC ----------------------------------------

// Función para cargar la tarjeta chip con valores de temperatura --
```

Preguntas de control, bloque de ensayos CMC 5-5

¿Cuántos valores medidos de temperatura de 8 bits se pueden guardar en la tarjeta chip empleada?

¿Por qué es necesario conmutar el flanco que activa la interrupción de la interrupción externa3?

¿Cuál es el método más sencillo para comprobar si los valores medidos de temperatura se han cargado correctamente en la tarjeta chip?

Nos gusta trabajar con y para usted.

Usted es nuestro mejor colaborador. En base a las experiencias que haya tenido con nuestros equipos, puede contribuir a mejorarlos dándonos sugerencias o avisándonos de cualquier error encontrado. Toda observación que nos haga será tratada con importancia y tomada en cuenta a la hora de actualizar nuestras guías de ejercicios.

¡Muchas gracias por su interés y su colaboración!

Asunto: Guía de ejercicios
Notas:

Fecha:

LUCAS-NÜLLE Lehr- und Messgeräte GmbH
Dirección: Siemensstr. 2 • D-50170 Kerpen (Sindorf) • Alemania
Apdo. postal: Postfach 11 40 • D-50140 Kerpen • Alemania
Telf.: + 49 / (0)2273 / 567-0 • Fax: +49 / (0)2273 / 567-30 • vertrieb@lucas-nuelle.com

Lucas-Nülle Lehr- und Meßgeräte GmbH

Siemensstraße 2 · D-50170 Kerpen-Sindorf
Telefon +49 2273 567-0 · Fax +49 2273 567-30

www.lucas-nuelle.de

LUCAS-NÜLLE