C161-Starterkit

Peter Pramberger

Mit diesem Artikel soll eine Einführung in die Programmierung des C161-Starterkits geboten werden. Zu diesem Zweck wird die Mikrocontroller-Platine mit Hardware zur Eingabe und Ausgabe versehen. Anschließend werden dann diese Erweiterungen in einem Beispielprogramm verwendet.

1 Hardware

1.1 Starterkit

Das C161-Starterkit beinhaltet neben dem C161O-Mikrocontroller von Siemens 256 KByte Flash-ROM und 64 KByte RAM. Beide Speicher können auf jeweils 1 MByte ausgebaut werden. Die Platine wird in der Standardausstattung mit einer stabilisierten 5V-Gleichspannung versorgt. Über eine optional erhältliche Erweiterung ist dann auch der direkte Anschluß einer unstabilisierten Spannung von 8–12V möglich. Weiters befinden sich auf der Platine die Anschlüsse für die beiden seriellen Schnittstellen. Das serielle Verbindungskabel befindet sich leider nicht im Lieferumfang, kann aber mit der mitgelieferten Anleitung leicht selbst hergestellt werden. Über eine 152-polige Steckerleiste sind alle relevanten Anschlüsse des Mikrocontrollers zugänglich. Dadurch ist ein einfacher Anschluß von externer Hardware möglich.

Im Paket ist jede Menge an Dokumentation zum Starterkit und zum Mikrocontroller enthalten. Daneben finden sich noch zwei CD-ROMs, eine mit den aktuellen Datenblättern und Handbüchern zu den Siemens-Mikrocontrollern und eine zweite mit den Programmen zur Verwendung der Platine und den C-Compilern. Diese sind zwar nur eingeschränkte Versionen, trotzdem lassen sich damit kleinere Programme realisieren.

1.2 C161O

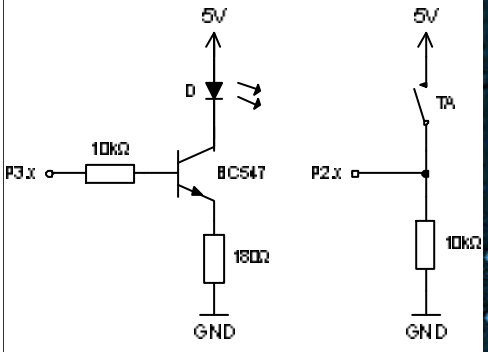
Der C161O ist einer von vier Derivaten der von Siemens entwickelten C161-Mikrocontrollerfamilie. Diese Mikrocontrollerfamilie ist eine preisgünstige Variante der C165/C163-Mikrocontrollerfamilie.

Der C161O arbeitet mit 125 ns Befehlszyklus zeit bei 16 MHz und besitzt eine vierstufige Pipeline. Dank dieser kann er bis auf wenige Ausnahmen jeden Befehl in der zuvorgenannten Zeit ausführen. Weiters kann der C161O bis zu vier MByte Speicheradressieren und beinhaltet zwei KByte interes RAM. Neben insgesamt 20 Interruptquellen (davon sieben externe Quellen), die auf 16 verschiedene Prioritätsebenen programmiert werden können, verfügt der C161O über acht PEC-Kanäle, die ähnlich einem DMA-Transfer beim PC CPU-unabhängige Speichertransfers ermöglichen. An Peripherie verfügt er über fünf Timer, eine asynchrone (bis zu 500 KBaud), eine synchrone serielle Schnittstelle (bis zu 2 MBaud) und 63 Portleitungen.

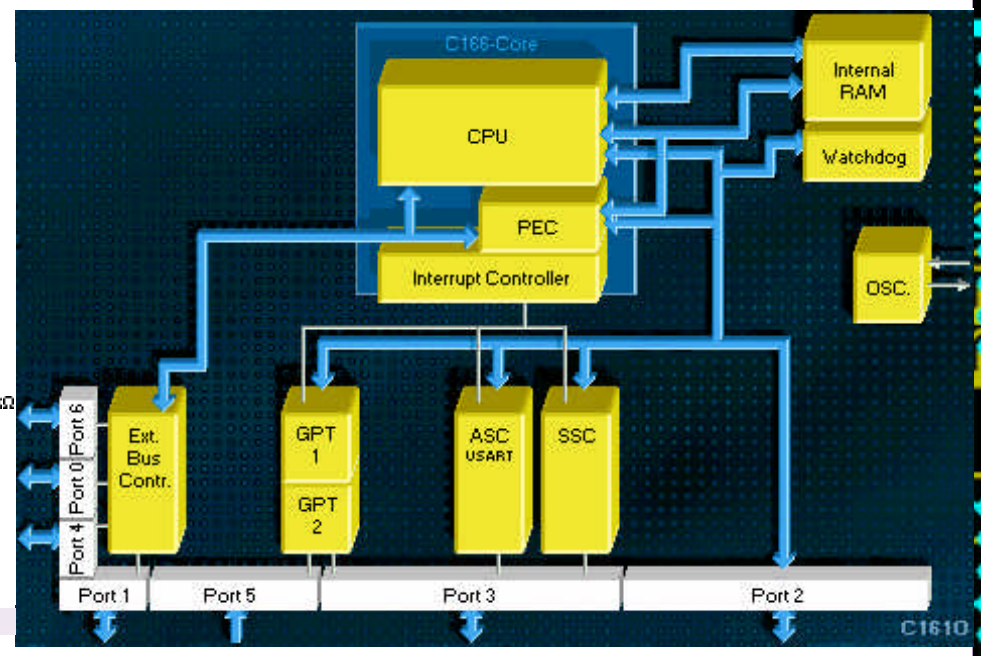
1.3 Erweiterungen

Wenn man das Paket auspackt und die Platine vor sich liegen hat, kann man damit zunächst noch nicht viel anfangen. Lediglich die serielle Schnittstelle steht für erste Programmierversuche zur Verfügung. Daher ist es sinnvoll, die Mikrocontroller-Platine mit externer Hardware zur Eingabe (in der Form von Tastern) und zur Ausgabe (in der Form von LEDs) zu erweitern.

Auf der Mikrocontroller-Platine sind die Pins P2.9 bis P2.15 und die Pins P3.2 bis P3.13 frei verfügbar. Die Pins 2.x werden auch als Triggereingänge für die externen



Die angewendete Interface-Schaltung



Interrupts benutzt. Dadurch bietet sich Port 2 für den Anschluß der Taster an. Dadurch hat man immer noch die Wahl, ob die externen Tasten über Interrupts oder durch Polling eingelesen werden. Die LEDs werden an Port 3 angeschlossen.

Für die meisten Anwendungen müßten vier bis sechs LEDs und zwei bis drei Taster vollkommen ausreichend sein. Für das nachfolgende Beispielprogramm sind mindestens vier LEDs und zwei Taster notwendig.

Die Schaltungen können entweder direkt auf der Mikrocontroller-Platine oder auf einer eigenen Platine, die dann über ein Flachbandkabel mit der Mikrocontroller-Platine verbunden wird, angeschlossen werden.

2 Programmierung

Für die Programmierung des C161-Starterkits wird der auf der CD-ROM mitgelieferte C166-Compiler von Keil verwendet, da er die geringeren Einschränkungen und den größeren Funktionsumfang aufweist. Dabei wird vorausgesetzt, daß der Compiler bereits installiert ist.

2.1 Lauflicht

Als Beispielprogramm wurde ein Lauflicht gewählt, welches über zwei externe Taster gesteuert werden kann. Der erste Taster ermöglicht eine Auswahl aus fünf verschiedenen Lichtfolgen, während der zweite Taster zur Umschaltung der Laufrichtung dient. Die Ausgabe erfolgt an fünf LEDs, die an Port 3 angeschlossen sind. Die zwei Taster sind an Port 2 angeschlossen und aktivieren den jeweiligen externen Interrupt.

Für die Weiterschaltung der LEDs wird Timer 3 eingesetzt. Er läuft im Timer-Mode mit etwa 50 ms Intervall. Bei einem Überlauf wird das entsprechende Flag in der Interrupt-Service-Routine gesetzt und der Timer nachgeladen. Für die Taster werden die externen Interrupts 1 und 2 verwendet. Bei einem Tastendruck wird das entsprechende Flag in der zugehörigen Interrupt-Service-Routine gesetzt. Die Verarbeitung der Flags erfolgt dann im Hauptprogramm. Je nach Betätigung der Taste 1 wird die Variable `Program` erhöht und dadurch eine andere Lichtfolge gewählt. Je nach Betätigung der Taste 2 wird die Variable `LED_Direction` auf 0 oder 1 gesetzt und damit die Laufrichtung umgeschaltet.

Mit jedem Timerüberlauf wird die Lichtfolge um eine LED weitergeschaltet, und zwar mit den Funktionen `_irol_` bzw. `_iror_`. Diese Funktionen verschieben den Inhalt von Port 3 um jeweils eine Stelle nach links bzw. rechts. Zur Tastenentprellung werden einfach die externen Interrupts für eine Dauer von sieben Timerüberläufen gesperrt.

```
// ***************************************************************************
// * Header:    LLICHT.H                                                     *
// ***************************************************************************
// * Funktion: Definitionen für LLICHT.C                                     *
// ***************************************************************************
// * Autor:     Pramberger Peter                                             *
// ***************************************************************************

// ------- Definitionen -------
#define ulong unsigned long
#define uword unsigned int
#define ubyte unsigned char
#define bool bit

#define EXTERNAL_INT2  CC10IE
#define EXTERNAL_INT3  CC11IE
#define TIMER3_INT     T3IE
#define TIMER_3        T3

#define CC10INT  0x1A
#define CC11INT  0x1B
#define T3INT    0x23

// ---------- Makros ----------
#define GT1_LoadTmr(TimerNr,Value) TimerNr = Value
#define INT_EnableInterrupt(IntName)  IntName = 1
#define INT_DisableInterrupt(IntName) IntName = 0
#define IO_WritePort(Port,Data) Port = Data

// ---------- Prototypen ----------
void GT1_Init (void);
void INT_Init (void);
void IO_Init (void);

// ------- Projekt-Includes -------
#include <reg161.h>
#include <intrins.h>
```

2.2 Erstellung des lauffähigen Programms

2.2.1 Startup-Code

Zunächst muß die Datei `START161.A66` von der CD-ROM in das Verzeichnis des Beispielprogramms kopiert werden. Sie enthält den angepaßten Startcode für die Mikrocontroller-Platine und befindet sich auf der CD-ROM im Verzeichnis

```
R:\CDROM\3RDTOOLS\PHYTEC\FLASH166\
STARTUP.
```

Anschließend muß sie mit folgendem Aufruf übersetzt werden. Dabei muß die Angabe des Speichermodells unbedingt mit dem Speichermodell des Programms übereinstimmen, in diesem Fall `SMALL`.

```
A166 START161.A66 MOD167 SET(SMALL)
```

Der Assembler erzeugt daraus die Datei `START161.OBJ`, die später zum fertigen Programm dazugelinkt werden muß.

2.2.2 Kompilieren

Anschließend wird das Beispielprogramm mit folgendem Aufruf kompiliert. Auch hier muß unbedingt das verwendete Speichermodell angegeben werden.

```
C166 LLICHT.C MOD167 SMALL
```

Der Compiler erzeugt die Datei `LLICHT.OBJ`, die dann zum Linken verwendet wird, und die Datei `LLICHT.LST`, die Informationen zum übersetzten Programm enthält.

2.2.3 Linken

Beim Aufruf des Linkers müssen diesem nicht nur die zu linkenden Module, sondern auch die zu verwendenden Speicherbereiche angegeben werden, also wo ein bestimmter Datentyp oder Code abgelegt werden muß. Da diese Angaben zu lang für eine direkte Befehlszeileneingabe sind, muß man eine Kommandodatei schreiben, die dann dem Linker als Parameter übergeben wird. In diesem Beispiel sieht die se Datei (`LLICHT.LNK`) folgendermaßen aus:

```
LLICHT.OBJ,
START161.OBJ
TO LLICHT.ABS
CL(NCODE(000000H-00FFFFH),
FCODE(010000H-037FFFH),
NCONST(000000H-003FFFH),
FCONST(004000H-037FFFH),
HCONST(004000H-037FFFH),
IDATA(00F600H-00FDFFH),
IDATA0(00F600H-00FDFFH),
NDATA(100000H-103FFFH),
NDATA0(100000H-103FFFH),
FDATA(104000H-10FFFFH),
FDATA0(104000H-10FFFFH),
HDATA(104000H-10FFFFH),
HDATA0(104000H-10FFFFH),
BDATA(00FD00H-00FDFFH),
BDATA0(00FD00H-00FDFFH),
SDATA(00C000H-00FFFFH),
```

```
// ********************************************************************************************
// * Programm: LLICHT.C                                                                       *
// ********************************************************************************************
// * Funktion: Lauflicht mit fünf verschiedenen Programmen und umschaltbarer Laufrichtung.    *
// ********************************************************************************************
// * Autor:    Pramberger Peter                                                               *
// ********************************************************************************************

// ------- Projekt-Includes -------
#include „LLICHT.H“

// ------- Globale Variablen -------
bool TMR_Overflow = 0;          // Überlauf des Timers aufgetreten
bool INT_ExtInt2 = 0;           // Externer Interrupt 2 aufgetreten
bool INT_ExtInt3 = 0;           // Externer Interrupt 3 aufgetreten
ubyte idata Zaehler = 0;       // Zählvariable zur Tastenentprellung

// ------- Unterprogramme -------
void INT_Init (void)            // Initialisiert das Interrupt-System
{
   EXICON = 0x0050;             // ---- Ext. Interrupt 2 Trigger ----
   DP2   &= 0xF3FF;             // Steigende Flanke an Pin P2.10
                                // ---- Ext. Interrupt 3 Trigger ----
                                // Steigende Flanke an Pin P2.11
   CC10IC = 0x0048;             // ------ Externer Interrupt 2 ------
                                // Interrupt-Prioritätslevel (ILVL) = 2
                                // Interrupt-Gruppenlevel (GLVL) = 0
   CC11IC = 0x0044;             // ------ Externer Interrupt 3 ------
                                // Interrupt-Prioritätslevel (ILVL) = 1
                                // Interrupt-Gruppenlevel (GLVL) = 0
}

void IO_Init (void)             // Initialisiert Port 3
{
   P3    = 0x0000;              // ------ Port 3 Datenregister ------
                                // Alternative Funktion von P3.2 deaktiviert
                                // Status von P3.2 ist Low-Level
                                // Alternative Funktion von P3.3 deaktiviert
                                // Status von P3.3 ist Low-Level
                                // Alternative Funktion von P3.4 deaktiviert
                                // Status von P3.4 ist Low-Level
                                // Alternative Funktion von P3.5 deaktiviert
                                // Status von P3.5 ist Low-Level
                                // Alternative Funktion von P3.6 deaktiviert
                                // Status von P3.6 ist Low-Level
   ODP3 = 0x0000;               // ---- Port 3 Open-Drain-Register ----
                                // Pin P3.2 auf Push/Pull-Modus setzen
                                // Pin P3.3 auf Push/Pull-Modus setzen
                                // Pin P3.4 auf Push/Pull-Modus setzen
                                // Pin P3.5 auf Push/Pull-Modus setzen
                                // Pin P3.6 auf Push/Pull-Modus setzen
   DP3   = 0x007C;              // ---- Port 3 Richtungsregister ----
                                // Pin P3.2 als Ausgang konfigurieren
                                // Pin P3.3 als Ausgang konfigurieren
                                // Pin P3.4 als Ausgang konfigurieren
                                // Pin P3.5 als Ausgang konfigurieren
                                // Pin P3.6 als Ausgang konfigurieren
}

void TMR_Init (void)            // Initialisiert Timer 3
{
   T3CON = 0x0003;              // ---- Timer 3 Control Register ----
                                // Timer arbeitet im Timer-Modus
                                // Prescaler-Faktor ist 64
                                // Zählrichtung: aufwärts
                                // Ext. Zählrichtungsumschaltung deaktiviert
   T3 = 0x3FFF;                 // Timer Register laden
   T3IC = 0x004C;               // ---- Timer 3 Interrupt Register ----
                                // Interrupt aktivieren
                                // Interrupt-Prioritätslevel (ILVL) = 3
                                // Interrupt-Gruppenlevel (GLVL) = 0
   T3R = 1;                     // Timer aktivieren
}

void Project_Init (void)
{
   IO_Init();                   // Initialisiert Port 3
   TMR_Init();                  // Initialisiert Timer 3
   INT_Init();                  // Initialisiert das Interrupt-System

   INT_EnableInterrupt (TIMER3_INT);      // Aktiviert Timer 3 Interrupt
   INT_EnableInterrupt (EXTERNAL_INT2);   // Aktiviert den ext. Interrupt 2
   INT_EnableInterrupt (EXTERNAL_INT3);   // Aktiviert den ext. Interrupt 3

   IEN = 1;                     // Setzt das Global-Enable-Bit
}

// ---- Interrupt-Service-Routinen ----
void INT_IsrExt2 (void) interrupt CC10INT  // ISR für externen Interrupt 2
{
```

Dabei bedeutet:

NCODE	Near Code
FCODE	Far Code
NCONST	Near Constant
FCONST	Far Constant
HCONST	Huge Constant
IDATA	Interner RAM (2 kByte)
IDATA0	Interner RAM (globale Variablen)
NDATA	Near Data
NDATA0	Near Data (globale Variablen)
FDATA	Far Data
FDATA0	Far Data (globale Variablen)
HDATA	Huge Data
HDATA0	Huge Data (globale Variablen)
BDATA	Bit-adressierbarer Speicherbereich (256 Byte)
BDATA0	Bit-adressierbarer Speicherbereich (globale Variablen)
SDATA	Systembereich

Diese Speicherbereiche gelten nur für die Programmierung eines Programms in das Flash-ROM der Mikrocontroller-Platine!

Der Linker wird mit folgender Syntax aufgerufen:

```
L166 @LLICHT.LNK
```

Er erzeugt die Datei LLICHT.ABS, die das fertige Programm darstellt. Optional kann noch die Datei LLICHT.M66 erzeugt werden, die Informationen über die Speicherbelegung und die verwendeten Funktionen enthält.

2.2.4 Konvertieren

Bevor das Programm jetzt in das Flash-ROM programmiert werden kann, muß es zunächst in das Intel-Hex-Format konvertiert werden. Dies erfolgt folgendermaßen:

```
OH166 LLICHT.ABS H167
```

Damit wird die Datei LLICHT.HEX erzeugt, die dann in das Flash-ROM programmiert werden kann.

2.2.5 Flash-ROM

Um ein Programm in das Flash-ROM zu übertragen, muß Jumper JP4 geöffnet und Jumper JP9 in Stellung 1-2 gesetzt sein. Nach dem man ein serielles Kabel an P2 angeschlossen hat, wird das Programm FLASHT aufgerufen. Es befindet sich auf der CD-ROM im Verzeichnis

```
   INT_ExtInt2 = 1;              // Wenn externer Interrupt - Flag setzen
}

void INT_IsrExt3 (void) interrupt CC11INT  // ISR für externen Interrupt 3
{
   INT_ExtInt3 = 1;              // Wenn externer Interrupt - Flag setzen
}

void GT1_IsrTmr3 (void) interrupt T3INT    // ISR für Timer 3
{
   TMR_Overflow = 1;             // Wenn Timer über läuft - Flag setzen
   GT1_LoadTmr (TIMER_3, 0x3FFF);  // Timerregister nachladen
}

// ------ Hauptprogramm ------
void main (void)
{
   bool LED_Direction = 0;       // Umschaltung der Laufrichtung
   uword idata Temp = 0;         // Temporäre Variable
   ubyte idata Program = 0;      // Auswahl des Lauflicht-Programms

   Project_Init();               // Initialisieren der Hardware
   IO_WritePort (P3, 0x0F0F);    // Port 3 voreinstellen
   Temp = 0x0F0F;                // Variable voreinstellen
   LED_Direction = 1;            // Laufrichtung voreinstellen

   for (;;)                      // Endlos-Schleife
   {
      if (INT_ExtInt2 == 1)      // Wenn externer Interrupt 2 aufgetreten
      {
         INT_ExtInt2 = 0;
         if (Program 4)         // Wenn Program klein er vier ist -
         {
            Program ++;         // Variable erhöhen.
         }
         else                   // Wenn Program gleich vier ist -
         {
            Program = 0;        // Variable auf Null setzen.
         }
         switch (Program)       // Je nach Wert von Program verzweigen
         {
            case 0: IO_WritePort (P3, 0x0F0F);  // Programm 1
                    Temp = 0x0F0F;
                    break;
            case 1: IO_WritePort (P3, 0x1111);  // Programm 2
                    Temp = 0x1111;
                    break;
            case 2: IO_WritePort (P3, 0x3333);  // Programm 3
                    Temp = 0x3333;
                    break;
            case 3: IO_WritePort (P3, 0x5555);  // Programm 4
                    Temp = 0x5555;
                    break;
            case 4: IO_WritePort (P3, 0x7777);  // Programm 5
                    Temp = 0x7777;
                    break;
         }
      }
      if (INT_ExtInt3 == 1)      // Wenn externer Interrupt 2 aufgetreten
      {
         INT_ExtInt3 = 0;
         LED_Direction = LED_Direction ^ 1;  // Laufrichtung umschalten
      }
      if (TMR_Overflow == 1)    // Wenn Timer-Interrupt aufgetreten
      {
         TMR_Overflow = 0;
         Zaehler ++;            // Zähler zur Tastenentprellung erhöhen
         if (LED_Direction == 0) // Wenn Variable gleich Null ist
         {
            Temp = _iror_ (Temp, 1);  // Variableninhalt nach rechts rotieren
            IO_WritePort (P3, Temp);  // Variable auf Port 3 schreiben
         }
         else                   // Wenn Variable gleich Eins ist
         {
            Temp = _irol_ (Temp, 1);  // Variableninhalt nach links rotieren
            IO_WritePort (P3, Temp);  // Variable auf Port 3 schreiben
         }
      }
      if (Zaehler == 6)          // Wenn Variable gleich sechs ist -
      {  Zaehler = 0;
         INT_EnableInterrupt(EXTERNAL_INT2);  // Reaktiviert den ext. Int. 2
         INT_EnableInterrupt(EXTERNAL_INT3);  // Reaktiviert den ext. Int. 3
      }
      if (Zaehler 2)            // Wenn Variable klein er zwei ist -
      {  INT_DisableInterrupt (EXTERNAL_INT2);   // Deaktiviert ext. Int. 2
         INT_DisableInterrupt (EXTERNAL_INT3);   // Deaktiviert ext. Int. 3
      }
   }
}
```

R:\CDROM\3RDTOOLS\PHYTEC\FLASH166\FLASHT\161.

Zur schnelleren Verarbeitung empfiehlt es sich, eine höhere Baudrate zu verwenden, hier 38400 Baud:

FLASHT BR(38400)

Nachdem die Flash-Tools geladen wurden, erscheint ein Menü mit zahlreichen Optionen. Am einfachsten wird über Punkt 7 (Erase, Load and Software-Reset) der Inhalt des Flash-ROM gelöscht und danach über F2 der Dateiname des Programms (LLICHT.HEX) angegeben. Nach erfolgter Programmierung muß nur noch der Jum per JP9 in Stellung 2-3 gebracht und ein Reset ausgelöst werden und der Mikrocontroller arbeitet ab sofort mit dem neuen Programm.

Detail eines Bildes der CD "Picture Pool, Ed.3, August 1997, SIEMENS