DSV7L

Hinweise für die Softwareentwicklung

Die Software für den Analog Devices DSP ADSP2181 wird mit Hilfe eines IDE (Integrated Desktop Environment) entwickelt. Dieses hat den Vorteil, dass sie dem Programierer in einer einheitlichen Umgebung alle Tools wie Editor, Compiler, Assembler, Linker, Simulator usw. zur Verfügung stellt.

Für jede Übungen ist es notwendig mit „**Project->New“** jeweils ein Projekt in einem eigenen Ordner anzulegen, in dem alle Dateien, die für das lauffähige DSP-Programm notwendig sind, verwaltet werden. Dem Projekt sollte bei der Erstellung ein eindeutiger Name zugewiesen werden. In das Projekt können vorhanden Dateien über **„Project->Add to Project“** eingebunden werden. Für ein Projekt können unter „**Project->Project Options“** Parameter angegeben werden, die das Verarbeiten der Dateien und das Erstellen der binären Ausgabedatei beeinflussen. Zu nächst werden jedoch die Standardeinstellungen des Projektes verwendet.

Für ein lauffähiges Programm wird eine Programmdatei benötigt, die den aktuellen Quellcode beinhaltet, sowie eine LDF-Datei (Linker Description File). Der Quellcode soll in Assembler geschrieben werden und muß daher in einer Datei mit der Endung *.asm gespeichert werden. Die LDF-Datei beschreibt den internen Speicheraufbau des DSPs und gibt dem Linker die notwendigen Informationen um den Quellcode und die Daten an die richtigen physikalischen Adressen zu legen. Zusätzlich wird noch eine Definitionsdatei benötigt, die häufig verwendete Werte global festlegt. Für das erste Programm werden also folgende Dateien benötigt.

- ADSP-2181.ldf
- uebung1.asm
- Ports.h

Diese sind als Template auf dem Rechner abgelegt und müssen in das Projekt eingefügt werden.

Bitte kommentiert den Quellcode in Englisch!

Wenn der Quellcode für die jeweilige Übung programmiert wurde und das Programm im Simulator lauffähig ist, dann kann die binäre Ausgangsdatei erstellt werden. Dazu müssen folgende Schritte vorgenommen werden. In den Projektoptionen muß auf der Karte **„Project“** der Wert **„Type“** auf **„Loader/Splitter file“** geändert werden. Dadurch wird ein Dateiformat erzeugt, das sich mit Hilfe des Programmes PEPSI in den EPROM-Simulator laden läßt. Für das Laden muß in der Karte **„Post Built“** die Zeile **„c:\command.com /c pepsi debug\uebung1.bnm“** eingegeben werden.

Um das Programm anschließend zu starten, muß auf dem DSP ein Reset ausgelöst werden.

Dann sollte hoffentlich alles funktionieren!

;-)

Im Anhang findet Ihr die vorgegeben Templates für die, die lieber tippen wollen!

Linker Description File (ADSP-2181_template.ldf)

```
ARCHITECTURE(ADSP-2181)

SEARCH_DIR( $ADI_DSP\218x\lib )

// Libraries from the command line are included in COMMAND_LINE_OBJECTS.
$OBJECTS = $COMMAND_LINE_OBJECTS;

// 2181 has 16K words of 24-bit internal Program RAM and 16K words of 16-bit
// internal Data RAM the commented seg_pmovly and seg_dmovly would be to map
// the external overlay pages seg_pmpage1,2 and seg_dmpage1,2 (these pages
// are unused in this default ldf; instead, all of DMOVLAY 0 space is in
// seg_data1 and PMOVLAY0 space is divided between seg_code and seg_data2.

MEMORY
{
  seg_inttab { TYPE(PM RAM) START(0x00000) END(0x0002f) WIDTH(24) }
  seg_code   { TYPE(PM RAM) START(0x00030) END(0x037ff) WIDTH(24) }
  seg_data2  { TYPE(PM RAM) START(0x03800) END(0x03fff) WIDTH(24) }

  seg_data1  { TYPE(DM RAM) START(0x00000) END(0x02fff) WIDTH(16) }
  seg_heap   { TYPE(DM RAM) START(0x03000) END(0x037ff) WIDTH(16) }
  seg_stack  { TYPE(DM RAM) START(0x03800) END(0x03fdf) WIDTH(16) }
}

PROCESSOR p0
{
  LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
  OUTPUT( $COMMAND_LINE_OUTPUT_FILE )

  SECTIONS
  {
    sec_inttab
    {
      INPUT_SECTIONS( $OBJECTS( interrupts ))
    } >seg_inttab

    sec_code
    {
      INPUT_SECTIONS( $OBJECTS(program) )
      INPUT_SECTIONS( $OBJECTS(irhandler) )
    } >seg_code

    sec_data1
    {
      INPUT_SECTIONS( $OBJECTS(data1) )
    } >seg_data1

    sec_data2
    {
      INPUT_SECTIONS( $OBJECTS(data2) )
    } >seg_data2

    sec_stack
    {
      ldf_stack_limit = .;
      ldf_stack_base = . + MEMORY_SIZEOF(seg_stack) - 1;
    } >seg_stack

    sec_heap
    {
      .heap = .;
          .heap_size = MEMORY_SIZEOF(seg_heap);
      .heap_end = . + MEMORY_SIZEOF(seg_heap) - 1;
    } >seg_heap

}
```

Source File (uebung_template.asm)

```
/*****************************************************************************
*       Description:
*
*       Author:                                 Date:
*****************************************************************************/

/* includes */
#include <def2181.h>

/* global defines */

/* global variables */
.section/dm data1

/*****************************************************************************
* Initializing interrupt vectors
*****************************************************************************/
.section/pm interrupts;
                JUMP start; nop; nop; nop;              /* Reset vector */
                nop; nop; nop; nop;                     /* IRQ2 */
                nop; nop; nop; nop;                     /* IRQL1 */
                nop; nop; nop; nop;                     /* IRQL0 */
                nop; nop; nop; nop;                     /* SPORT0 transmit */
                nop; nop; nop; nop;                     /* SPORT0 recieve */
                nop; nop; nop; nop;                     /* IRQE */
                nop; nop; nop; nop;                     /* BDMA */
                nop; nop; nop; nop;                     /* SPORT1 transmit */
                nop; nop; nop; nop;                     /* SPORT1 recieve */
                nop; nop; nop; nop;                     /* timer */

/*****************************************************************************
* main program
*****************************************************************************/
.section/pm program;

start:
                ar = b#0011011011011011;                // set value for wait state
                dm(Dm_Wait_Reg ) = ar;                  // write wait state register
wai:            jump wai;                               // idle loop - wait for interrupt

/*****************************************************************************
* Interrupt handler
*****************************************************************************/
.section/pm irhandler;
```

Definition File (ports_template.h)

```
/*binary input and output */
#define latch_in 0                  // 16-Bit input
#define latch_out 0                 // 16-Bit output

/*DACs */
#define dac1_lb 0x200               // DAC 1 low byte address
#define dac2_lb 0x201               // DAC 2 low byte address
#define dac3_lb 0x202               // DAC 3 low byte address
#define dac4_lb 0x203               // DAC 4 low byte address
#define dac1_hb 0x204               // DAC 1 high byte address
#define dac2_hb 0x205               // DAC 2 high byte address
#define dac3_hb 0x206               // DAC 3 high byte address
#define dac4_hb 0x207               // DAC 4 high byte address
#define dac1_up 0x208               // DAC 1 update address
#define dac2_up 0x209               // DAC 2 update address
#define dac3_up 0x20A               // DAC 3 update address
#define dac4_up 0x20B               // DAC 4 update address
#define dac_upa 0x20C               // DAC update all address

/*ADCs */
#define adc_1 0x400                 // ADC 1 address
#define adc_2 0x401                 // ADC 1 address
#define adc_3 0x402                 // ADC 1 address
#define adc_4 0x403                 // ADC 1 address
#define sc 0x000                    // start convert
```