Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

ADSP-2181 EZ-Kit Lite IDMA to PC Printer Port Interface

Contributed by HS, last change 11-Apr-2002

Introduction:

The ADSP-218x family can be used either in host or full memory interface configuration, ADSP-2181 and ADSP-2183 offer both interfaces at the same time. This host port interface allows through the IDMA port easy read / write access to the internal memory of the DSP. With little additional software on the DSP even DSP system registers can be accessed or external overlays be booted.

Application Idea:

The IDMA interface is an easy to handle asynchronous 16bit wide interface and requires only four strobe lines to operate and synchronize it to the DSP clock. Due to IDMA long read and write capabilities, almost every host can access this interface.

Often DSP software is developed on a PC platform using VisualDSP. Code verification is usually done with the simulator. Even todayís PCs do not match the performance of the DSP, so regression testing by driving test vector to an easy to use hardware platform is desired. With a little external circuitry, which can be easily fitted to an ADDS-2181-EZ-Lite, application code can be booted into to DSP and test vectors driven from a bi-directional PC parallel port (Printer Port).

Hardware Design:

The IDMA interface of the ADSP-218x family requires just few data and strobe lines to operate the interface. To realize the interface, the four strobe lines and 16 data lines must be connected to the PC printer port by a port expander. An i8255 offers three freely programmable data ports, which can be switched between input and output. A fourth internal register controls the setting of the device. To access the port expander, the 8 data lines of the PC printer port are connected to the I/O data port. The 8255 has two additional address line to select the internal ports and requires besides read or write a chip select. As the PC printer port has only eight data lines and four output strobes, a little glue logic (exnor of /WR and /RD) is required to realize the interface for /CS. The i8255 has a nasty trait of pulling all signals low when the direction of any the data ports PA, PB or PC is changed. To avoid troubles at the IDMA side, all four strobe lines (IAL, /IS, /IRD and /IWR) have therefore an additional inverter placed. After these inverters, PA of the i8255 is used to drive the IDMA strobe lines, while PB and PC are used as 16bit IAD data port. A complete schematic of a 5V PC to DSP interface is presented in this document. To make the interface compatible with other 3.3V ADSP-218x components, level transceivers like 74LVT245 have to be put in between PA, PB and PC and switched accordingly.

Software Design:

To ease the application and to provide a generic example of routines, a driver for PC systems allowing direct I/O access to PC system resources is provided.

To access the IDMA port, first the i8255 must be set up correctly from the reset state. Afterwards, a write to the PA port allows to drive the required strobe signals. Further writes to the PB and PC port will latch the data to be transmitted to the ADSP-218x. A write sequence will look like:

1	PA: set IAL
2	PB, PC: drive address
3	PA: clear IAL
4	PA: set write strobes /IS, /IWR
5	PB, PC; drive data
6	PA: set write strobe IS; IWR

Table 1: typical write sequence

To drive multiple data words into the ADSP-218x through IDMA interface, it is just necessary to repeat steps 4 to 6.

A read sequence looks like the write sequence, with a little difference: before data is read in step 5, the PB and PC port direction must be switched between step 3 and 4 to an input and must be changed again past step 6.

First Time Preparations:

By default, the ADDS-2181-EZ-Kit-Lite is configured for BDMA booting. To allow IDMA and host booting, the BMODE pin (pin 35 on the ADSP-2181 package) must be lifted from ground and connected the VCC. Following figure shows the change. If just IDMA data access without boot capabilities is desired, there is no need to change.

Figure1: BMODE pin change

Booting from the EPROM is still supported, as a BDMA bootloader, which can be downloaded through the IDMA interface, is supplied.

Additionally to this, the connector must be soldered into P2 of the EZ-Kit Lite, as shown in figure 2.

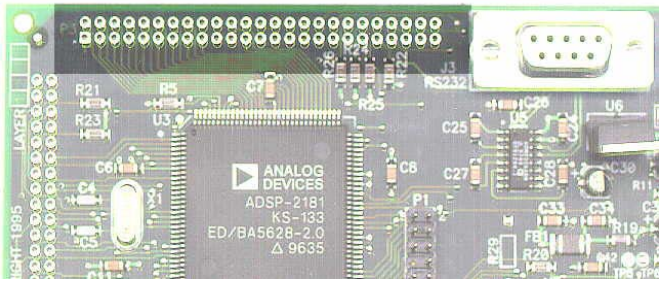

Figure 2: location of the IDMA interface connector.

Having performed these two changes, the interface is operational, just connect a straight 25pin connector cable between the PC and the EZ-IDMA adaptor card. Please make sure at that point, that your PC printer port is bidirectional to prevent damage to the PC when reading back data from the IDMA port.

The supplied routines can added to a PC program allowing easy regression testing.

API functions:

A list of provided API functions is given below. The compete sources are attached to this document.

Reset_Board:	bring the i8255 in a defined state
Boot_2181:	download and start a DSP executable
Set_Address:	set a new IDMA address
Put_Data:	drive a dataword from the PC to 218x
Get_Data:	retrive a dataword from 218x
Get_Status:	query the state of FL1

File_open_read: open ASCII file for read

File_open_write: open ASCII file for write

Get_int_ff:	collect a 16bit integer word from file
Get_hex_ff:	collect a 16bit hex word from file
Get_char_ff:	collect single character from file
Find_port:	detect PC printer port

Boot_ext_OVLAYMGR: prepare for boot into external overlays

Boot_ext_OVLAY: download code / data to external overlay

References:

http://www.analog.com/dsp
ADDS-2181-EZ-Lite documentation
ADSP-2181 data sheet
Intel i8255 data sheet
VisualDSP Documentation

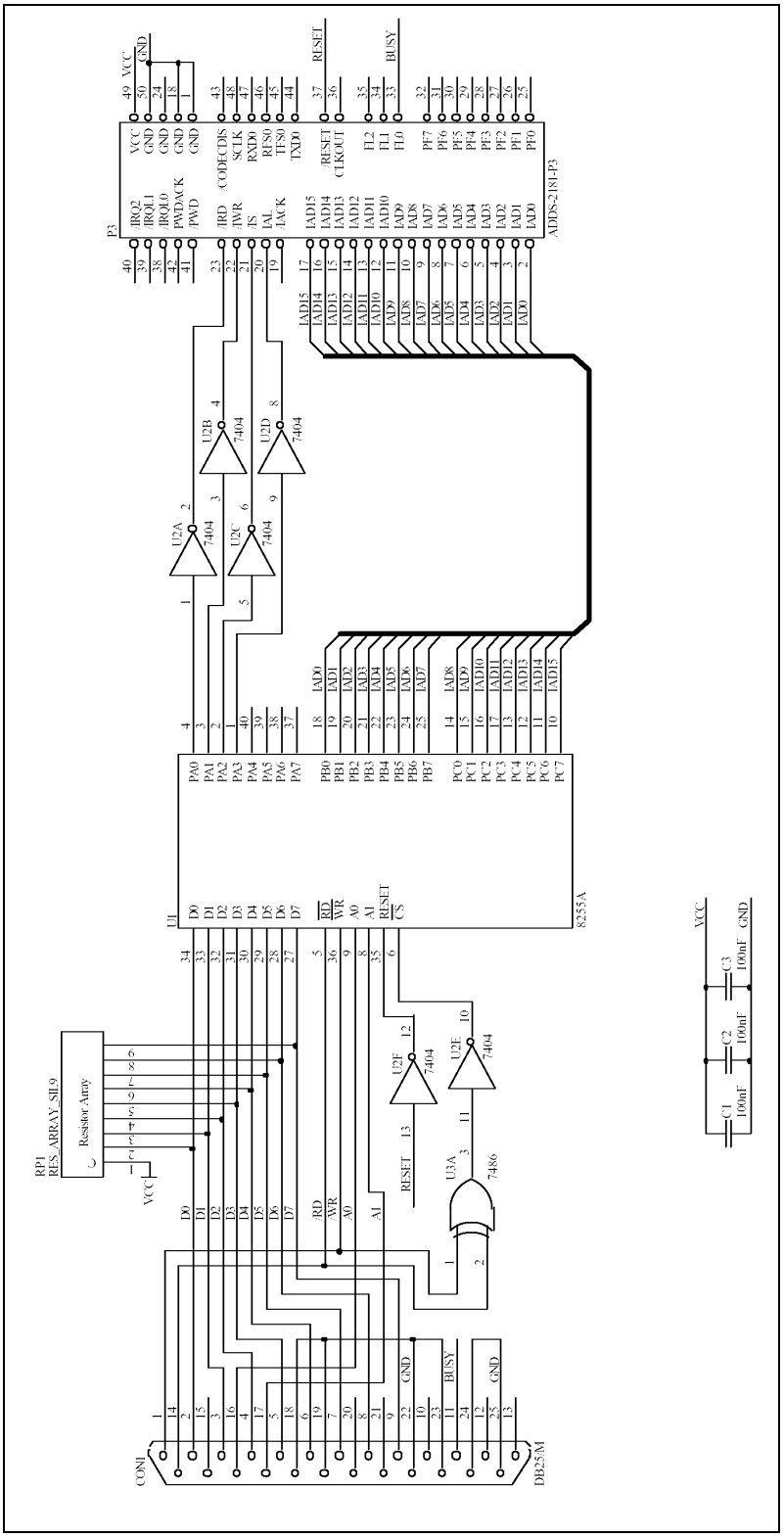

Figure 3: Schematics for the EZ-IDMA Interface

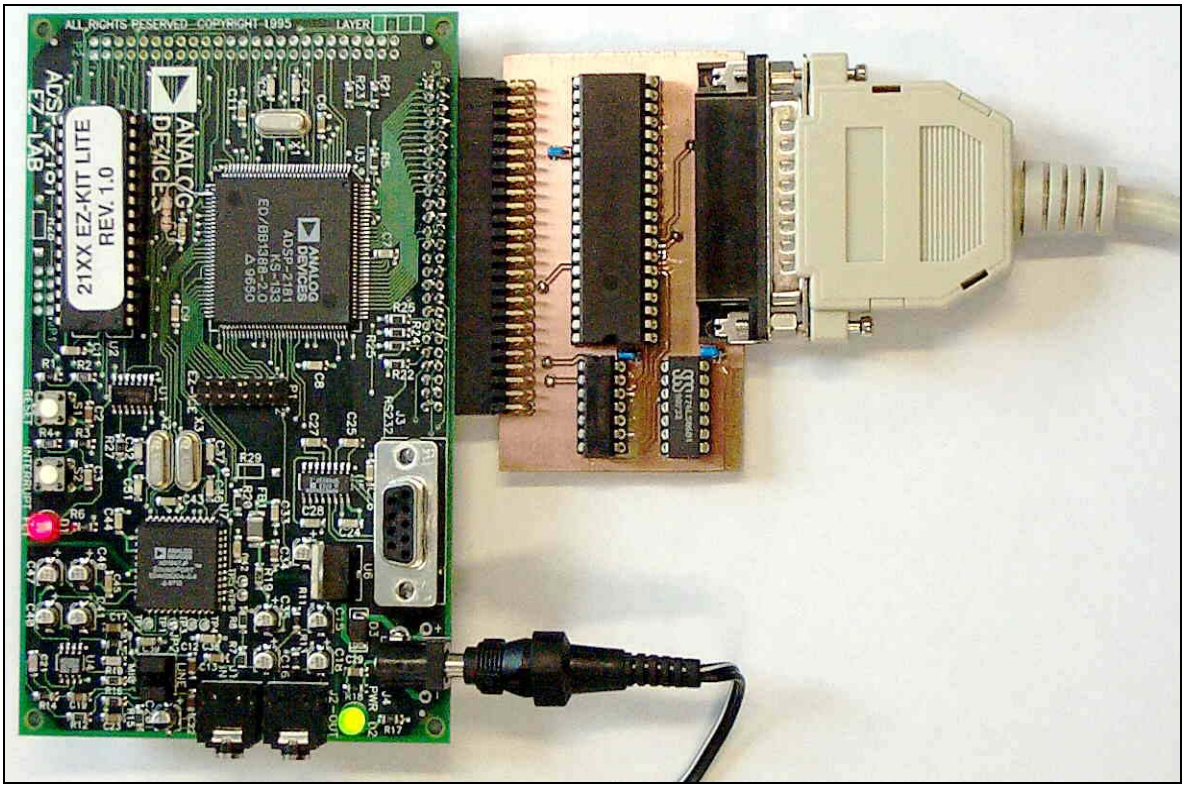

Figure 4: ADDS-2181-EZ-Lite and EZ-IDMA prototype card

```
/*******************************************************************************
 *
 *  EZ81_IDMA.CPP
 *
 *  Copyright (c) 2000 HS Scan Service, All Rights Reserved
 *                2001 adopted by ANALOG DEVICES, free redistribution allowed
 *
 *
 *  HISTORY
 *    10-SEP-2000 HS   document created
 *    02-FEB-2001 HS   updated for generic Test Shell
 *    17-MAY-2001 HS   support booting external overlays
 *
 *  COMMENT:
 *
 *
 ******************************************************************************/
#include "stdafx.h"
#include "stdio.h"
#include "stdlib.h"
#include "dos.h"
#include "conio.h"
#include "string.h"
#include "EZ81_IDMA.h"


/*******************************************************************************
 ***
 ***  Function Prototypes
 ***
 ******************************************************************************/
void Reset_Board();
int  Boot_2181(FILE* fname);
int  Boot_ext_OVLAYMGR();
int  Boot_ext_OVLAY(int type, int num, FILE* fname);
void Set_Address(int mem_type, unsigned int addr);
void Put_Data(int mem_type, unsigned long data);
long Get_Data(int mem_type);

FILE* file_open_read(char* fname);
FILE* file_open_write(char* fname);

int get_int_ff(FILE* fptr);
int get_hex_ff(FILE* fptr);

int find_port();
void delay(int count);

long ext_boot_loader[145] = {
               0x000000, 0x18030F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F, 0x0A001F,
               0x340015, 0x380017, 0x340008, 0x340009, 0x380008, 0x380009, 0x400000, 0x930010,
               0x4FFFF0, 0x930020, 0x473000, 0x93FE60, 0x40FFF0, 0x93FFE0, 0x404000, 0x93FFF0,
               0x83001A, 0x227A0F, 0x180400, 0x4FFFF0, 0x930020, 0x400104, 0x26E20F, 0x34001E,
               0x1806B0, 0x400114, 0x26E20F, 0x34001E, 0x180700, 0x400204, 0x26E20F, 0x34002E,
               0x1806B0, 0x400214, 0x26E20F, 0x34002E, 0x180700, 0x400304, 0x26E20F, 0x34001F,
               0x1807B0, 0x400314, 0x26E20F, 0x34001F, 0x180800, 0x400404, 0x26E20F, 0x34002F,
               0x1807B0, 0x400414, 0x26E20F, 0x34002F, 0x180800, 0x34000E, 0x34000F, 0x400504,
               0x26E20F, 0x1808B0, 0x18040F, 0x400000, 0x930010, 0x390000, 0x3A0001, 0x18074F,
               0x400000, 0x930010, 0x390000, 0x3B0001, 0x3D0005, 0x14077E, 0x500003, 0x580007,
               0x400000, 0x930020, 0x18040F, 0x400000, 0x930010, 0x360000, 0x340001, 0x18084F,
               0x400000, 0x930010, 0x360000, 0x350001, 0x3D0005, 0x14087E, 0x600001, 0x680005,
               0x400000, 0x930020, 0x18040F, 0x380000, 0x418000, 0x3C00F7, 0x580003, 0x18000F };
```

```
/*****************************************************************************
 ***
 ***  Functions
 ***
 *****************************************************************************/
inline void wr_to_8255(unsigned char reg, unsigned char val)
{
       _outp(CTRL, reg | D8255_WR );                        /* write to control */
       _outp(DATA, val );                                   /* write to data port */
       _outp(CTRL, reg | D8255_DL );                        /* idle 8255 */
}

inline unsigned char rd_from_8255(unsigned char reg)
{
       unsigned char read;

       _outp(CTRL, reg | D8255_DL | PPORT_BI );             /* write to control */
       _outp(CTRL, reg | D8255_RD | PPORT_BI );             /* write to control */
       read = _inp(DATA);                                   /* collect data */
       _outp(CTRL, reg | D8255_DL | PPORT_BI );             /* write to control */

       return (read);
}


/*****************************************************************************
 *
 * Reset Board
 *
 *****************************************************************************/
void Reset_Board()
{
       wr_to_8255(D8255_CC, D82_BCO);                       /* put 8255 in output mode */
       wr_to_8255(D8255_PA, D8x_DLE);                       /* init IDMA strobes on 218x */
       Set_Address(DM,0);
       Put_Data(DM,0);
}


/*****************************************************************************
 *
 * Set IDMA address
 *
 *****************************************************************************/
void Set_Address(int mem_type, unsigned int addr)
{
       if (addr!=0xffff)
       {
              addr |= mem_type;

              /* set up address */
              wr_to_8255(D8255_PB, (unsigned char) (addr & 0x00ff) );
              wr_to_8255(D8255_PC, (unsigned char) ((addr & 0xff00) >> 8) );

              /* latch it */
              wr_to_8255(D8255_PA, D8x_IAL);
              wr_to_8255(D8255_PA, D8x_DLE);
       }
}


/*****************************************************************************
 *
 * Put Data to IDMA port
 *
 *****************************************************************************/
void Put_Data(int mem_type, unsigned long data)
{
       if (mem_type == 0)            /* PM memory */
       {
              /* set up data */
              wr_to_8255(D8255_PB, (unsigned char) ((data & 0x00ff00) >>  8) );
              wr_to_8255(D8255_PC, (unsigned char) ((data & 0xff0000) >> 16) );
```

```
                /* latch it */
                wr_to_8255(D8255_PA, D8x_DLE);
                wr_to_8255(D8255_PA, D8x_IWR);
                wr_to_8255(D8255_PA, D8x_DLE);

                /* set up data */
                wr_to_8255(D8255_PB, (unsigned char) (data & 0x0000ff) );

                /* latch it */
                wr_to_8255(D8255_PA, D8x_DLE);
                wr_to_8255(D8255_PA, D8x_IWR);
                wr_to_8255(D8255_PA, D8x_DLE);
        }
        else                                    /* DM Memory */
        {
                /* set up data */
                wr_to_8255(D8255_PB, (unsigned char) (data & 0x00ff) );
                wr_to_8255(D8255_PC, (unsigned char) ((data & 0xff00) >> 8) );

                /* latch it */
                wr_to_8255(D8255_PA, D8x_DLE);
                wr_to_8255(D8255_PA, D8x_IWR);
                wr_to_8255(D8255_PA, D8x_DLE);
        }
}


/*************************************************************************
 *
 * Get Data from IDMA port
 *
 *************************************************************************/
long Get_Data(int mem_type)
{
        long data = 0;
        unsigned char rl, rm, rh;

        if (mem_type == 0)                                      /* PM memory */
        {
                /* switch 8255 to input */
                wr_to_8255(D8255_CC, D82_BCI);

                /* first access */
                wr_to_8255(D8255_PA, D8x_DLE);
                wr_to_8255(D8255_PA, D8x_IRD);                  /* open latch */
                        /* latch BC port */
                        rm = rd_from_8255(D8255_PB);            /* get mid */
                        rh = rd_from_8255(D8255_PC);            /* get high */
                wr_to_8255(D8255_PA, D8x_DLE);                  /* close latch */

                /* second access */
                wr_to_8255(D8255_PA, D8x_IRD);                  /* open latch */
                        /* latch B port */
                        rl = rd_from_8255(D8255_PB);            /* get low */
                wr_to_8255(D8255_PA, D8x_DLE);

                /* switch PPORT back to output */
                wr_to_8255(D8255_CC, D82_BCO);
                wr_to_8255(D8255_PA, D8x_DLE);

                data = (rh<<16) | (rm<<8) | rl;
        }

        else                                                    /* DM Memory */
        {
                /* switch 8255 to input */
                wr_to_8255(D8255_CC, D82_BCI);

                wr_to_8255(D8255_PA, D8x_DLE);
                wr_to_8255(D8255_PA, D8x_IRD);                  /* open latch */
                        /* latch BC port */
                        rl = rd_from_8255(D8255_PB);            /* get low */
                        rh = rd_from_8255(D8255_PC);            /* get high */
                wr_to_8255(D8255_PA, D8x_DLE);
```

```
                /* switch PPORT back to output */
                wr_to_8255(D8255_CC, D82_BCO);
                wr_to_8255(D8255_PA, D8x_DLE);

                data = (rh<<8) | rl;
        }

        return data;
}


/*************************************************************************
 *
 * Boot ADSP-218x
 *
 *************************************************************************/
int Boot_2181(FILE* fname)
{
        static char    *pam = "@PA";
        static char    *dam = "@DA";
        static char    *bend = "#";
        char           *p;

        char           in_str[12];
        char           addr_str[6];
        char           inst_str[8];
        char           data_str[6];

        unsigned int   addr, data;
        unsigned long  inst,inst_0;

        Reset_Board();                                          /* set 8255 state */

        do
        {
                fgets(in_str, 12, fname);                       /* fetch textline */

                if (strstr(in_str, pam) != NULL)                /* is Instruction */
                {
                        strcpy(addr_str, fgets(in_str, 12, fname));
                        addr = strtoul(addr_str,&p,16);

                        do {
                                strcpy(inst_str, fgets(in_str, 12, fname));
                                if (strstr(in_str, bend) == NULL)
                                {
                                        inst = strtoul(inst_str, &p, 16);
                                        if(addr==0)
                                        {
                                                inst_0=inst;    /* save instruction 0 */
                                                addr++;
                                        }
                                        else
                                        {
                                                Set_Address(PM, addr);
                                                Put_Data(PM, inst);
                                                addr=0xffff;
                                        }
                                }
                        }
                        while(strstr(in_str, bend) == NULL);  /* until end of block */
                }
        }
        while (!feof(fname));



        rewind(fname);                                          /* next parse for data */

        do
        {
                fgets(in_str, 12, fname);                       /* get textline */
```

```
		if (strstr(in_str, dam) != NULL)			/* data follows */
		{
			strcpy(addr_str, fgets(in_str, 12, fname));
			addr = strtoul(addr_str, &p, 16);

			do {
				strcpy(data_str, fgets(in_str, 12, fname));
				if (strstr(in_str, bend) == NULL)
				{
					data = strtoul(data_str, &p, 16);
					Set_Address(DM, addr);
					Put_Data(DM, data);
					addr=0xffff;
				}
			}
			while(strstr(in_str, bend) == NULL);  /* until end of block */
		}
	}
	while (!feof(fname));

	Set_Address(PM, 0);					/* kick off DSP */
	Put_Data(PM, inst_0);					/* write instructon 0 */

	return(1);

}


/***************************************************************************
 *
 * Download Boot Manager
 *
 ***************************************************************************/
int Boot_ext_OVLAYMGR()
{
	int i;
	long inst0;

	Set_Address(PM, 0x0001);
	for (i=0; i< 145; i++)
	{
		if (i == 0)
			inst0 = ext_boot_loader[i];
		else
			Put_Data(PM, ext_boot_loader[i]);	/* write instructon */
	}
	Set_Address(PM, 0x0000);				/* kick off DSP */
	Put_Data(PM, inst0);					/* write instructon 0 */

	return 1;
}


/***************************************************************************
 *
 * Boot external ADSP-218x
 *
 ***************************************************************************/
int Boot_ext_OVLAY(int type, int num, FILE* fname)
{
	static char	*pam = "@PA";
	static char	*dam = "@DA";
	static char	*bend = "#";
	char		*p;

	char		in_str[12];
	char		addr_str[6];
	char		inst_str[8];
	char		data_str[6];

	unsigned int	addr, command, i, result;
	unsigned int	data[8192];
	unsigned long	inst[8192];
```

```
        /*
         * Implemented commands in 2181_Boot_Manager
         *
         * 0x00 = wait
         *
         * 0x10 = PMOVLAY 1, lower 4k
         * 0x11 = PMOVLAY 1, upper 4k
         * 0x20 = PMOVLAY 2, lower 4k
         * 0x21 = PMOVLAY 2, upper 4k
         *
         * 0x30 = DMOVLAY 1, lower 4k
         * 0x31 = DMOVLAY 1, upper 4k
         * 0x40 = DMOVLAY 2, lower 4k
         * 0x41 = DMOVLAY 2, upper 4k
         *
         * 0x50 = final init, boot internal
         */

        // test for final init
        if (num == 0)
        {
                command = 0x50;
                Set_Address(DM, 0x3001);
                Put_Data(DM, command);
                return (1);
        }

        /* rewind file */
        rewind(fname);

        if (type == PM)                                         /* parse for PM memory */
        {
                do {
                        fgets(in_str, 12, fname);               /* fetch textline */
                        if (strstr(in_str, pam) != NULL)        /* is Instruction */
                        {
                                strcpy(addr_str, fgets(in_str, 12, fname));
                                addr = strtoul(addr_str,&p,16);
                                if (addr < 0x2000)
                                {
                                        printf("\n\n FILE ERROR - Executable contains code for non overlay
section @ addr 0x%04x", addr);
                                        printf("\n Section skipped, cf. MAP file \n");
                                }
                                else
                                {
                                        addr = addr - 0x2000;   /* subtract ovlay */
                                        do {
                                                strcpy(inst_str, fgets(in_str, 12, fname));
                                                if (strstr(in_str, bend) == NULL)
                                                        inst[addr++] = strtoul(inst_str, &p, 16);
                                        }
                                        while(strstr(in_str, bend) == NULL);  /* until end of block */
                                }
                        }
                }
                while (!feof(fname));
        }

        else                                                    /* parse for DM memory */
        {
                do {
                        fgets(in_str, 12, fname);               /* get textline */
                        if (strstr(in_str, dam) != NULL)        /* data follows */
                        {
                                strcpy(addr_str, fgets(in_str, 12, fname));
                                addr = strtoul(addr_str, &p, 16);
                                if (addr > 0x2000)
                                {
                                        printf("\n\n FILE ERROR - Executable contains data for non overlay
section @ addr 0x%04x", addr);
                                        printf("\n Section skipped, cf. MAP file \n");
                                }
                                else
```

```
                        {
                                do {
                                        strcpy(data_str, fgets(in_str, 12, fname));
                                        if (strstr(in_str, bend) == NULL)
                                                data[addr++] = strtoul(data_str, &p, 16);
                                }
                                while(strstr(in_str, bend) == NULL);  /* until end of block */
                        }
                }
        }
        while (!feof(fname));
}

/* download sections */
if (type == PM)
{
        command = (num << 4);                   /* low block */
        Set_Address(PM, 0x1000);                /* set download address */
        for (i=0; i< 0x1000; i++)
        {
                Put_Data(PM, inst[i]);          /* dump code */
        }
        Set_Address(DM, 0x3001);                /* set command address */
        Put_Data(DM, command);                  /* write command */
        do
        {
                Set_Address(DM, 0x3002);
                result = Get_Data(DM);
        }
        while (result != 0);                    /* wait for SUCCESS */

        command = command + 1;                  /* high block */
        Set_Address(PM, 0x1000);                /* set download address */
        for (i=0x1000; i< 0x2000; i++)
        {
                Put_Data(PM, inst[i]);          /* dump code */
        }
        Set_Address(DM, 0x3001);                /* set command address */
        Put_Data(DM, command);                  /* write commadn */
        do
        {
                Set_Address(DM, 0x3002);
                result = Get_Data(DM);
        }
        while (result != 0);                    /* wait for SUCCESS */
}
else    /* type == DM */
{
        command = ((num + 2) << 4);             /* low block */
        Set_Address(DM, 0x2000);                /* set download address */
        for (i=0; i< 0x1000; i++)
        {
                Put_Data(DM, data[i]);          /* dumpü data */
        }
        Set_Address(DM, 0x3001);                /* set command address */
        Put_Data(DM, command);                  /* write command */
        do
        {
                Set_Address(DM, 0x3002);
                result = Get_Data(DM);
        }
        while (result != 0);                    /* wait for SUCCESS */

        command = command + 1;                  /* high block */
        Set_Address(DM, 0x2000);                /* set download address */
        for (i=0x1000; i< 0x2000; i++)
        {
                Put_Data(DM, data[i]);          /* dump data */
        }
        Set_Address(DM, 0x3001);                /* set command address */
        Put_Data(DM, command);                  /* write command */
        do
        {
                Set_Address(DM, 0x3002);
```

```
                result = Get_Data(DM);
            }
            while (result != 0);                /* write for SUCCESS */
        }
        return (1);
}



/*************************************************************************
 *
 * File open for write access
 *
 *************************************************************************/
FILE* file_open_write(char* fname)
{
        FILE *fptr;

        if ((fptr = fopen(fname,"wt")) == NULL)
                printf("\nCan not open file %s\n", fname);
        else
                printf("\nsuccess opening %s for write", fname);

        return fptr;
}


/*************************************************************************
 *
 * File open for read access
 *
 *************************************************************************/
FILE* file_open_read(char* fname)
{
        FILE *fptr;

        if ((fptr = fopen(fname,"rt")) == NULL)
                printf("\nCan not open file %s\n", fname);
        else
                printf("\nsuccess opening %s for read", fname);

        return fptr;
}


/*************************************************************************
 *
 * Get INT value from current file position
 *
 *************************************************************************/
int get_int_ff(FILE* fptr)
{
        int result, readval;

        result = fscanf(fptr, "%d", &readval);
        if (result == EOF) readval = 0;

        return readval;
}


/*************************************************************************
 *
 * Get HEX value from current file position
 *
 *************************************************************************/
int get_hex_ff(FILE* fptr)
{
        int result, readval;

        result = fscanf(fptr, "%lx", &readval);
        if (result == EOF) readval = 0;

        return (int) readval;
```

```
}


/*************************************************************************
 *
 * Find Printer Port
 *
 *************************************************************************/
int find_port()
{
        unsigned int *ptraddr;
        unsigned int address;
        int a;

        ptraddr=(unsigned int *)0x00000408;                 /*Bios Location */

        for (a = 0; a < 3; a++)
        {
                address = *ptraddr;
                if (address == 0)
                        printf("No port found for LPT%d \n",a+1);
                else
                        printf("Address assigned to LPT%d is %Xh\n",a+1,address);
                *ptraddr++;
        }

        return (address);
}

/*************************************************************************
 *
 * Delay
 *
 *************************************************************************/
void delay(int count)
{
        int i, res;

        for (i=0; i<count; i++)
                res = _inpw(0x0);
}
```

Listing 1: API sources

```
/*****************************************************************************
 *
 *  EZ81_IDMA.H
 *
 *  Copyright (c) 2000 HS Scan Service, All Rights Reserved
 *                2001 adopted by ANALOG DEVICES, free redistribution allowed
 *
 *  HISTORY
 *    10-SEP-2000 HS   document created
 *    02-FEB-2001 HS   updated for generic Test Shell
 *    17-MAY-2001 HS   support booting external overlays
 *
 *  COMMENT:
 *
 *
 *****************************************************************************/

#ifndef ___EZ_IDMA81___
#define ___EZ_IDMA81___

#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "string.h"


/*****************************************************************************
 ***
 ***  Global Defines
 ***
 *****************************************************************************/

#define BASE    0x378
#define DATA    BASE + 0
#define STAT    BASE + 1
#define CTRL    BASE + 2

/*****************************************************************************
 ***
 ***  Printer Port Interface
 ***
 *****************************************************************************/
#define D8255_PA       0x08    //  00001000, Port A
#define D8255_PB       0x0C    //  00001100, Port B
#define D8255_PC       0x00    //  00000000, Port C
#define D8255_CC       0x04    //  00000100, Control

#define D8255_WR       0x01    //  00000001, /WR line at 8255
#define D8255_RD       0x02    //  00000010, /RD line at 8255
#define D8255_DL       0x00    //  00000000, idle at 8255

#define PPORT_BI       0x20    //  00100000, set before read from par_port
#define PPORT_BO       0xDF
#define PPORT_UL       0x40    //  01000000, set before read from par_port

/*****************************************************************************
 ***
 ***  8255 Defines
 ***
 *****************************************************************************/
#define D82_BCO               0x80    //  10000000, Mode0: A-out, B-out, C-out
#define D82_BCI               0x8B    //  10001011, Mode0: A-out, B-in , C-in

#define D8x_IAL               0x04    // ~xxxx1011,  IAL line at 218x
#define D8x_IWR               0x0E    // ~xxxx0001, /IWR line at 218x
#define D8x_IRD               0x0D    // ~xxxx0010, /IRD line at 218x
#define D8x_DLE               0x08    // ~xxxx0111, idle at 218x




/*****************************************************************************
```

```
 ***
 ***  IDMA Defines
 ***
 ****************************************************************************/
#define PM      0x0000
#define DM      0x4000


/****************************************************************************
 ***
 ***  Function Prototypes
 ***
 ****************************************************************************/
void Reset_Board();
int  Boot_2181(FILE* fname);
int  Boot_ext_OVLAYMGR(void);
int  Boot_ext_OVLAY(int type, int num, FILE* fname);
void Set_Address(int mem_type, unsigned int addr);
void Put_Data(int mem_type, unsigned long data);
long Get_Data(int mem_type);

FILE* file_open_read(char* fname);
FILE* file_open_write(char* fname);
int get_int_ff(FILE* fptr);
int get_hex_ff(FILE* fptr);

int find_port();

#endif  // ___EZ_IDMA81___
```

Listing 2: API include file

```
#include "EZ81_IDMA.h"
#include "EZ_Support.h"
#include "stdio.h"
#include "conio.h"

int main(int argc, char* argv[])
{
        FILE* exefile;

        printf("\n\nPlease press RESET on the ez kit lite");
        getch();

        // Reset EZ_IDMA adaptor card
        Reset_Board();

        // Boot DSP Executable Manager for External Overlays
        Boot_ext_OVLAYMGR();

        // PM External Overlay 1
        exefile = file_open_read(PROGPATH"page1.exe");
        if (exefile == NULL)
        {
                printf("program aborted\n");
                fcloseall();
                getch();
                return false;
        }
        Boot_ext_OVLAY(PM, 1, exefile);
        fclose(exefile);

        // PM External Overlay 2
        exefile = file_open_read(PROGPATH"page2.exe");
        if (exefile == NULL)
        {
                printf("program aborted\n");
                fcloseall();
                getch();
                return false;
        }
        Boot_ext_OVLAY(PM, 2, exefile);
        fclose(exefile);

        // final init
        Boot_ext_OVLAY(PM, 0, exefile);

        // do your regression test here
}
```

Listing 3: Example Boot API use on PC, function calls printed in boldface