

DSPLib



DSP56xxx Interface Libraries

DSP56xxx Interface Libraries

January 21, 1998

DSPs supported by this software:

DSP560xx

DSP561xx

DSP563xx

DSP566xx

DSP568xx

MC68356 (DSP side)

Trademarks:

IBM PC, XT, AT are trademarks of International Business Machines Corporation.

OnCE is a trademark of Motorola, Inc.

Windows, Microsoft Windows, and Windows 95 are trademarks of Microsoft Inc.

Disclaimer of Warranty

This software package is provided on an 'AS IS' basis and without warranty. In no event shall Domain Technologies be liable for incidental or consequential damages arising from the use of this software. This disclaimer of warranty extends to LICENSEE, to LICENSEE's customers or users of products and is in lieu of all warranties whether expressed, implied, or statutory, including implied warranties of merchantability or fitness for a particular purpose. Domain Technologies does not warrant that software furnished hereunder is free of infringement of any third party patents, copyrights, or trade secrets.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION	5
Resource usage for different interfaces.	5
CHAPTER 2 - LIBRARY FUNCTIONS	7
EmuInt	7
EmuClose	8
EmuError	9
EmuCall	9
CHAPTER 3 - COMMANDS	11
TRG_BLOCK_READ	11
TRG_BLOCK_WRITE	11
TRG_BLOCK_COPY	12
TRG_VALUE_FILL	12
TRG_REG_READ	13
TRG_REG_WRITE	13
TRG_BP_SET	14
TRG_BP_CLR	14
TRG_BP_RESET	14
TRG_RUN	15
TGR_HALT	15
TRG_STEP	15
TRG_STAT	16
TRG_HWRST	16
TRG_READ_IDCODE	16
TRG_CHANGE_BAUD	17
TGR_HALT	17
TRG_MRUN	18
TRG_MSTEP	18
TRG_MAP_SET	19
TRG_MAP_GET	20
TRG_HOST_WRITE	21
TRG_HOST_READ	21
TRG_HOST_BLK_WR	22
TRG_HOST_BLK_RD	22
TRG_HOST_RESET	23
CHAPTER 4 - LIBRARY STRUCTURES AND EQUATES	25

CHAPTER 1 - Introduction

DSP56000 Interface Libraries allow building of custom applications accessing various DSP processors. The user programs can communicate through different interfaces between host computer and target processor.

Main access to the target DSP is provided by group of libraries utilizing OnCE logic of the target processor. The access through the OnCE port is most powerful, but also has few disadvantages: speed, and requirement to stop the processor. To overcome this problem, also available are interfaces allowing real-time communication with the target hardware. Currently supported are SCI (RS-232) and HOST port (ISA interface or Printer Port).

Availability matrix is as follows:

	OnCE/JTAG	SCI	Host Port
DSP56002EVM	x	x	x
DSP56009EVM	x		
DSP5630xEVM	x	x	x
DSP5660xEVM	x	x	x
DSP56811EVM	x		
SB-56K	x		
LINK-56K	x		

Libraries are available for 3 different platforms and 2 different compilers:

- Win-32 (Windows 95 and Windows NT) DLL for Microsoft C
- Win-16 (Windows 3.1x) static library for Microsoft C and BorlandC
- DOS (Windows 3.1x) static library for Microsoft C and BorlandC

Libraries are delivered with sample application generated in the corresponding environment, demonstrating interface initialization, and functions usage.

Resource usage for different interfaces.

All OnCE/JTAG interfaces do not require any DSP resources. For the SCI and Host port interfaces there are some restrictions, which should be observed by the user applications.

For SCI interfaces baud rate clock is generated from the DSP clock, so PLL and SCI settings should be not changed.

Host port and SCI interfaces are utilizing respective interrupt vectors. During program upload vector location can be overwritten, it will be restored by the monitor code. During user code execution, modifying the vectors (including SWI), will disable debugger interface.

Monitor code requirements are as follows:

- For DSP56002 monitor code occupies \$400 words in PRAM starting at address \$7C00. None of the X or Y memory locations are used.
- For DSP5630x monitor code occupies \$100 words in PRAM starting at address \$000F00. None of the X or Y memory locations are used.

1

CHAPTER 2 - Library Functions

Interface to the emulation software is provided by following procedures:
Emulnit , EmuClose, EmuError, EmuCall.

Emulnt

Initialize interface with the target DSP

params:

TRG_INFO *init - structure defining target connection

returns:

handle (= 0) or error (0).

```
_EXTERN_C_ short DLENTY EmuInit (TRG_INFO *init);
```

The Emulnit function will return valid handle even if the initialization fails. Allows it user program to retrieve all errors which were queued during target initialization. If none error is returned, than initialization was successful. Structure defining interface to the target is defined below

```
typedef struct
{
    short  structSize;      /* to verify compatibility      */
    short  procType;       /* target processor to access   */
    short  ifType;        /* interface type to the target */
    char  *procName;      /* name of the processor found  */
    short  scanDefLen;    /* size of the scanDef table (bytes)*/
    void  *scanDef;      /* structure of multiprocess (TBD) */
    short  comPort;      /* com port number to use (-1: all) */
    short  comBaud;      /* com baud to connect at      */
    short  ioPort;       /* port address to access      */
    short  scanNum;      /* scan-chain # to access      */
    char  *dllName;      /* name of this DLL (VERSION cmd) */
    long   dllVer;       /* rev # of this DLL (VERSION cmd) */
    long   fwVer;        /* rev # of firmware          */
    char  ipStr[IP_STR_LEN]; /* string representing location */
    short  dspCount;     /* number of DSPs in the scan chain */
    short  dspNum;      /* DSP # we are talking to      */
    short  resetReq;    /* reload firmware request      */
}TRG_INFO;
```

Example:

```
BOOL initInterface (void)
{
    TRG_CMD  cmd;
    char     *error;

    if (trgHandle = 0) /* target already initialized */
        return FALSE;
    evmInit.structSize = sizeof(TRG_INFO); /* initialize str. size */
    evmInit.procType = DSP56300;
    evmInit.ifType = ISA5630XEVM;
    evmInit.dspCount = 1; /* number of DSPs in the scan chain */
    evmInit.dspNum = 0; /* DSP # we are talking to */
    evmInit.ioPort = 0x240;
    trgHandle = EmuInit (&evmInit);
    if (trgHandle = 0)
```

```
{
  if ((error = EmuError (trgHandle, EMU_LAST_ERROR)) != NULL)
  {
    systemError (ERROR_COMMAND, error);
    EmuClose (trgHandle);
    trgHandle = -1;
  }
}
if (trgHandle = 0)
  return TRUE;
else
  return FALSE;
}
```

EmuClose

Close interface to the target DSP.

params:

short handle - target reference

returns:

error code

```
_EXTERN_C_ short DLENTY EmuClose (short handle);
```

Functions closes interface with the target DSP, releasing reserved resources: COM port, I/O ports (depending on the target interface)

Example:

```
EmuClose (trgHandle);
```


EmuError

Return text description of the specified error code.

params:

short handle - target reference
int error - error code inquired

returns:

char *errorStr - error description

```
_EXTERN_C_ char * DLLENTRY EmuError (short handle, int error);
```

Error codes are listed in the `trg_def.h`. Error codes are negative numbers. The positive numbers are used to query the information about the target configuration. This methods are used in communication with the emulation server.

Example:

```
if ((error = EmuCall (trgHandle, TRG_HALT, &cmd)) != EMU_NO_ERROR)
{
printf ("Command failed. Error: %d - %s\n",
error, EmuError (trgHandle, error));
}
```

2

EmuCall

Access target DSP.

params:

short handle - target reference
short cmd - command id
TRG_CMD *cmdarg - structure defining command

returns:

error code

```
_EXTERN_C_ short DLLENTRY EmuCall (short handle, short cmd, TRG_CMD *cmdarg);
```

This is main function accessing target DSPs. As a reference is used the handle received after Emulnit call. Cmd denotes numeric ID of the operation requested and cmdarg structure passes all the parameters required for the command execution.

Target Command Structure is used to pass arguments to all standard commands (via EmuCall). Various commands use different combinations of these parameters, but in general each parameter has a specified use. Size of the DATABUF depends on the processor type, but it should be 32 bits for universal HLL debugger

```
typedef struct
{
    SPACE    Type;    /* Memory space to access      */
    ADDR     Addr;    /* Address to access           */
    SPACE    Type2;   /* Memory space for copy destination */
    ADDR     Addr2;   /* Address for copy destination */
    long     Count;   /* Count of words to access     */
    DATABUF *Buf;    /* input/output data buffer     */
    long     WrLen;   /* Count of words in Buf to write */
    long     RdLen;   /* Count of words in Buf to read  */
}TRG_CMD;
```

List of the commands available through the EmuCall. Command parameters are passed within the Target Command Structure described above. Not all elements of the structure need to be initialized. In case of accessing the target through the TCP/IP server, values for WrLen and RdLen need to be always initialized. API to the server is identical to the library interface, so initializing those values, will make the application compatible with the TCP/IP transport interface. The target interface library is not using the values passed within WrLen and RdLen.

CHAPTER 3 - Commands

TRG_BLOCK_READ

Read a memory block from DSP memory spaces P, X or Y. Function fills the buffer with data read from the target DSP.

Type	Memory space to read from
Addr	Address to read from
Count	Count of words to read
*Buf	pointer to data buffer for target data
WrLen	0
RdLen	Count of words in Buf to read

Example:

```
cmd.Type    = XMEM;
cmd.Addr    = 0;
cmd.Count   = 0x10;   cmd.Buf    = lBuffer;
cmd.RdLen   = 0x10;
cmd.WrLen   = 0;
if (EmuCall (trgHandle, TRG_BLOCK_READ, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_BLOCK_WRITE

Write a memory block to the DSP memory spaces P, X or Y. The data from the buffer will be send to the target DSP.

Type	Memory space to write
Addr	Address to write
Count	Count of words to write
*Buf	input/output data buffer
WrLen	Count of words in Buf to write
RdLen	0

Example:

```
cmd.Type    = XMEM;
cmd.Addr    = 0;
cmd.Count   = 0x10;
cmd.Buf     = lBuffer;
cmd.WrLen   = 0x10;
cmd.RdLen   = 0;
if (EmuCall (trgHandle, TRG_BLOCK_WRITE, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_BLOCK_COPY

Copies memory block between DSPs memory spaces P, X or Y.

Type	Memory space to read (source)
Addr	Address to read (source)
Type2	Memory space for copy destination
Addr2	Address for copy destination
Count	Count of words to copy
WrLen	0
RdLen	0

Example:

```
cmd.Type = XMEM;
cmd.Addr = 0;
cmd.Type2 = XMEM;
cmd.Addr2 = 0x100;
cmd.Count = 0x10;
cmd.WrLen = 0;
cmd.RdLen = 0;
if EmuCall (trgHandle, TRG_BLOCK_COPY, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

3

TRG_VALUE_FILL

Fills specified range of memory of the target DSP with the data.

Type	Memory space to fill
Addr	Starting address to fill
Count	Count of words to fill
*Buf	Buffer with data to fill
WrLen	1
RdLen	0

Example:

```
cmd.Type = XMEM;
cmd.Addr = 0x20;
cmd.Count = 0x4;
cmd.Buf = &lValue;
cmd.WrLen = 1;
cmd.RdLen = 0;
if (EmuCall (trgHandle, TRG_VALUE_FILL, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_REG_READ

Allows to read any of the registers listed below as DSP Registers.

Addr	ID of the first register to read
Count	Number of register to read
*Buf	Buffer to store the read register values
WrLen	0
RdLen	Count of words in the Buffer

Example:

```
cmd.Addr   = DSP_REG_PC;
cmd.Count  = 1;
cmd.Buf    = &pc;
cmd.WrLen  = 0;
cmd.RdLen  = 1;
if (EmuCall (trgHandle, TRG_REG_READ, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_REG_WRITE

Allows to set a value of any of the registers listed below as DSP Registers.

Addr	ID of the first register to write
Count	Number of register to write
*Buf	Buffer with data to write to registers
WrLen	Count of words in Buf to write
RdLen	0

Example:

```
cmd.Addr   = DSP_REG_M0;
cmd.Count  = 8;
cmd.Buf    = &lBuffer;
cmd.WrLen  = 8;
cmd.RdLen  = 0;
if (EmuCall (trgHandle, TRG_REG_WRITE, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_BP_SET

Allows to set a software breakpoint (only within DSP's Program RAM).

Addr	Address to set a software breakpoint
WrLen	0
RdLen	0

Example:

```
cmd.Addr = brkAddr;
cmd.RdLen = 0;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_BP_SET, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_BP_CLR

Removes software breakpoint (only within DSP's Program RAM).

Addr	Address to remove the breakpoint
WrLen	0
RdLen	0

Example:

```
cmd.Addr = brkAddr;
cmd.RdLen = 0;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_BP_CLR, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_BP_RESET

Removes software breakpoint (only within DSP's Program RAM).

WrLen	0
RdLen	0

Example:

```
cmd.RdLen = 0;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_BP_RESET, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_RUN

Changes state of the target DSP from the DEBUG mode to free execution of the target object code (program).

WrLen 0
RdLen 0

Example:

```
cmd.RdLen = 0;  
cmd.WrLen = 0;  
if (EmuCall (trgHandle, TRG_RUN, &cmd) != EMU_NO_ERROR)  
    return FALSE;
```

TGR_HALT

Forces target DSP into the DEBUG mode, allowing to access all of the DSP's resources.

WrLen 0
RdLen 0

Example:

```
cmd.RdLen = 0;  
cmd.WrLen = 0;  
if (EmuCall (trgHandle, TGR_HALT, &cmd) != EMU_NO_ERROR)  
    return FALSE;
```

TRG_STEP

Executes single step of the target object code.

WrLen 0
RdLen 0

Example:

```
cmd.RdLen = 0;  
cmd.WrLen = 0;  
if (EmuCall (trgHandle, TRG_STEP, &cmd) != EMU_NO_ERROR)  
    return FALSE;
```

TRG_STAT

Reports current status of the DSP (executing program or DEBUG mode). TRUE value means device is in the debug mode, FALSE - target DSP is executing user code.

*Buf	Buffer to return the status of the DSP
WrLen	0
RdLen	1

Example:

```
cmd.Buf = &status;
cmd.RdLen = 1;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_STAT, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_HWRST

Performs Hardware reset of the target DSP.

WrLen	0
RdLen	0

Example:

```
cmd.RdLen = 0;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_HWRST, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_READ_IDCODE

Reads the silicon JTAG ID register. Applicable only for the JTAG interfaces.

*Buf	Single word buffer to store thr ID value.
WrLen	0
RdLen	1

Example:

```
cmd.Buf = &idCode;
cmd.RdLen = 1;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_IDCODE, &cmd) != EMU_NO_ERROR)
    return FALSE;
```


TRG_CHANGE_BAUD

Function queries or changes value of the baudrate. Applicable only for the RS-232 interface. If the value passed at *Buf equals -1, function will return current baud rate, otherwise function will set requested baudrate. Allowed values are as follows:

B9600
 B19200
 B28800
 B38400
 B57600
 B115200

*Buf	input/output data buffer
WrLen	1
RdLen	1

Example:

```
cmd.Buf    = B115200;
cmd.RdLen  = 0;
cmd.WrLen  = 1;
if (EmuCall (trgHandle, TRG_CHANGE_BAUD, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TGR_HALT

Forces target DSP into the DEBUG mode, allowing to access all of the DSP's resources.

WrLen	0
RdLen	0

Example:

```
cmd.RdLen  = 0;
cmd.WrLen  = 0;
if (EmuCall (trgHandle, TGR_HALT, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_MRUN

Synchronously starts code execution on selected processors. Processors are identified by the scan sequence number. Command passes the variable length buffer of the processors identification numbers.

The command operation requires that all selected processors were initialized with the Emulnit call.

Command applies only to the multi-DSP emulators (SB-56K)

Count	Number of processors to be started
*Buf	pointer to buffer with processors sequence numbers
WrLen	Number of processors to be started
RdLen	0

Example:

```

IBuffer[0] = 2;
IBuffer[1] = 5;
cmd.Count = 2;
cmd.Buf = IBuffer;
cmd.RdLen = 0;
cmd.WrLen = 2;
if (EmuCall (trgHandle, TRG_MRUN, &cmd) != EMU_NO_ERROR)
    return FALSE;
    
```

3

TRG_MSTEP

Single step list of processors. Synchronously single step selected processors. Processors are identified by the scan sequence number. Command passes the variable length buffer of the processors identification numbers.

The command operation requires that all selected processors were initialized with the Emulnit call.

Command applies only to the multi-DSP emulators (SB-56K)

Count	Number of processors to be single stepped
*Buf	pointer to buffer with processors sequence numbers
WrLen	Number of processors to be single stepped
RdLen	0

Example:

```

IBuffer[0] = 2;
IBuffer[1] = 5;
cmd.Count = 2;
cmd.Buf = IBuffer;
cmd.RdLen = 0;
cmd.WrLen = 2;
if (EmuCall (trgHandle, TRG_MSTEP, &cmd) != EMU_NO_ERROR)
    return FALSE;
    
```

TRG_MAP_SET

Set memory map attributes for the specified memory address range. This command allows to control access from the emulator to the target memory. Memory access attributes are as follow:

MAP_READ
MAP_WRITE
MAP_FLASH

To remove memory map attribute, remove bit needs to be set:

MAP_REMOVE

Every TRG_MAP_SET command controls global enable flag with the enable bit:

MAP_ENABLE

Type	Memory address space
Addr	Starting address of the memory block
Count	Length of the memory block
*Buf	pointer to the attribute bits
WrLen	1
RdLen	0

Example:

```
attribute = MAP_READ | MAP_WRITE | MAP_ENABLE
cmd.Type = XMEM;
cmd.Addr = 0;
cmd.Count = 0x10;
cmd.Buf = &attribute;
cmd.RdLen = 0;
cmd.WrLen = 1;
if (EmuCall (trgHandle, TRG_MAP_SET, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_MAP_GET

Get next memory map address range with the associated attributes. This command allows to retrieve the list of the memory maps set with the TRG_MAP_SET.

Command returns next element from the list, where the element is identified by its address and length. Length of 0, indicates beginning of the list.

Four data words are returned:

data[0] - memory address space
data[1] - starting address value
data[2] - length of the memory block
data[3] - attributes

Example:

```
cmd.Type = 0;
cmd.Addr = 0;
cmd.Count = 0;
cmd.Buf = data;
cmd.RdLen = 4;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_MAP_GET, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

```
else
{
do {
    displayMap (data[0], data[1], data[2], data[3]);
    cmd.Type = data[0];
    cmd.Addr = data[1];
    cmd.Count = data[2];
}
while (EmuCall (trgHandle, TRG_MAP_GET, &cmd) == EMU_NO_ERROR);
}
```

3

TRG_HOST_WRITE

Write a word to the DSP's host port register through the initialized interface. This allows to write value to each of the registers of the Host Port interface.

This command applies to the libraries providing interface to the DSP's Host Port from the Host ISA interface or Printer Port interface.

Addr	register address (0..7)
*Buf	pointer to the buffer with data word
WrLen	1
RdLen	0

Example:

```
cmd.Addr = HOST_ICR;
cmd.Buf = &value;
cmd.RdLen = 0;
cmd.WrLen = 1;
if (EmuCall (trgHandle, TRG_HOST_WRITE, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_HOST_READ

Read a word from the DSP's host port register through the initialized interface. This allows to read any of the registers of the Host Port interface.

This command applies to the libraries providing interface to the DSP's Host Port from the Host ISA interface or Printer Port interface.

Addr	register address (0..7)
*Buf	pointer to the buffer with data word
WrLen	1
RdLen	0

Example:

```
cmd.Buf = &value;
cmd.Addr = HOST_ISR;
cmd.RdLen = 0;
cmd.WrLen = 1;
if (EmuCall (trgHandle, TRG_HOST_READ, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_HOST_BLK_WR

Write a block of data to the DSP's host port through the initialized interface. Function will check if the HRX register is empty by reading the ISR register, and then perform writes to High, Mid and Low host port data registers, until all requested words are written. Function waits for the empty HRX register for 400 ms, and if the time expires, it will return the error.

This command applies to the libraries providing interface to the DSP's Host Port from the Host ISA interface or Printer Port interface.

*Buf	pointer to the buffer with data values
Count	number of words to write
WrLen	number of words to write
RdLen	0

Example:

```
cmd.Buf = dataBuf;
cmd.Count = len;
cmd.RdLen = 0;
cmd.WrLen = len;
if (EmuCall (trgHandle, TRG_HOST_BLK_WR, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

3

TRG_HOST_BLK_RD

Read a word from the DSP's host port through the initialized interface. Function will check if the HTX register is full by reading the ISR register, and then perform read from High, Mid and Low host port data registers until all requested words are read. Function waits for the full HTX register for 400 ms, and if the time expires, it will return the error.

This command applies to the libraries providing interface to the DSP's Host Port from the Host ISA interface or Printer Port interface.

*Buf	pointer to the buffer with data word
Count	number of words to read
WrLen	0
RdLen	number of words to read

Example:

```
cmd.Buf = &value;
cmd.Count = len;
cmd.RdLen = len;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_HOST_BLK_RD, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

TRG_HOST_RESET

Perform Hardware Reset of the target DSP. Function expects the target to be configured in the Host Bootstrap Mode, so after successful reset Host Port's IVR (Interrupt Vector Register) will contain 0x0f.

This command applies to the libraries providing interface to the DSP's Host Port from the Host ISA interface or Printer Port interface.

WrLen	0
RdLen	0

Example:

```
cmd.RdLen = 0;
cmd.WrLen = 0;
if (EmuCall (trgHandle, TRG_HOST_RESET, &cmd) != EMU_NO_ERROR)
    return FALSE;
```

3

CHAPTER 4 - Library Structures and Equates

DSP registers.

The following enumeration lists all registers for following families: DSP56000, DSP56100, DSP56300, DSP56600, DSP56800. Some registers are not valid for all devices.

```
enum regs56k
{
    DSP_REG_PC,
    DSP_REG_X0, DSP_REG_X1, DSP_REG_Y0, DSP_REG_Y1,
    DSP_REG_A0, DSP_REG_A1, DSP_REG_A2,
    DSP_REG_B0, DSP_REG_B1, DSP_REG_B2, DSP_REG_R0, DSP_REG_R1,
    DSP_REG_R2, DSP_REG_R3, DSP_REG_R4, DSP_REG_R5, DSP_REG_R6,
    DSP_REG_R7, DSP_REG_N0, DSP_REG_N1, DSP_REG_N2,
    DSP_REG_N3, DSP_REG_N4, DSP_REG_N5, DSP_REG_N6, DSP_REG_N7,
    DSP_REG_M0, DSP_REG_M1, DSP_REG_M2, DSP_REG_M3,
    DSP_REG_M4, DSP_REG_M5, DSP_REG_M6, DSP_REG_M7,
    DSP_REG_SR, DSP_REG_OMR, DSP_REG_SP, DSP_REG_SSH,
    DSP_REG_SSL, DSP_REG_LA, DSP_REG_LC,
    DSP_REG_SC, DSP_REG_SZ, DSP_REG_EP, DSP_REG_VBA,
    DSP_REG_SH0, DSP_REG_SL0, DSP_REG_SH1, DSP_REG_SL1,
    DSP_REG_SH2, DSP_REG_SL2, DSP_REG_SH3, DSP_REG_SL3,
    DSP_REG_SH4, DSP_REG_SL4, DSP_REG_SH5, DSP_REG_SL5,
    DSP_REG_SH6, DSP_REG_SL6, DSP_REG_SH7, DSP_REG_SL7,
    DSP_REG_SH8, DSP_REG_SL8, DSP_REG_SH9, DSP_REG_SL9,
    DSP_REG_SHA, DSP_REG_SLA, DSP_REG_SHB, DSP_REG_SLB,
    DSP_REG_SHC, DSP_REG_SLC, DSP_REG_SHD, DSP_REG_SLD,
    DSP_REG_SHE, DSP_REG_SLE, DSP_REG_SHF, DSP_REG_SLF,
    DSP_ONCE_SC, DSP_ONCE_TCOUNT, DSP_ONCE_BCOUNT, DSP_ONCE_BCTRL,
    DSP_ONCE_BADDR0, DSP_ONCE_BADDR1, DSP_ONCE_GDB, DSP_ONCE_PDB,
    DSP_ONCE_PDBG, DSP_ONCE_PABF, DSP_ONCE_PIL, DSP_ONCE_PABD,
    DSP_ONCE_PABE, DSP_ONCE_FIFO0, DSP_ONCE_FIFO1, DSP_ONCE_FIFO2,
    DSP_ONCE_FIFO3, DSP_ONCE_FIFO4, DSP_ONCE_FIFO5, DSP_ONCE_FIFO6,
    DSP_ONCE_FIFO7, DSP_ONCE_FIFO8, DSP_ONCE_FIFO9, DSP_ONCE_FIFO10,
    DSP_ONCE_FIFO11, DSP_ONCE_TAG0, DSP_ONCE_TAG1, DSP_ONCE_TAG2,
    DSP_ONCE_TAG3, DSP_ONCE_TAG4, DSP_ONCE_TAG5, DSP_ONCE_TAG6,
    DSP_ONCE_TAG7, DSP_ONCE_TAGST, DSP_REG_N, DSP_REG_M01,
    DSP_REG_HWS0, DSP_REG_HWS1, };
```

4