MOTOROLA

Embedded SDK (Software Development Kit)

Voice Activity Detector Library

SDK122/D
Rev. 2, 07/23/2002

MOTOROLA intelligence everywhere™ digital dna™

Contents

List of Tables

List of Figures

List of Examples

About This Document

This manual describes the Voice Activity Detection, (VAD), algorithm for use with Motorola's Embedded Software Development Kit, (SDK).

Audience

This document targets software developers implementing VAD function within software applications.

Organization

This manual is arranged in the following sections:

- **Chapter 1, Introduction**—provides a brief overview of this document
- **Chapter 2, Directory Structure**—provides a description of the required core directories
- **Chapter 3, VAD Library Interfaces**—describes all of the VAD Library functions
- **Chapter 4, Building the VAD Library**—tells how to execute the system library project build
- **Chapter 5, Linking Applications with the VAD Library**—describes the organization of the VAD Library
- **Chapter 6, VAD Applications**—describes the use of VAD Library through test/demo applications
- **Chapter 7, License**—provides the license required to use this product

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800 Family Manual*, DSP56800FM/AD
- *DSP56824 User's Manual*, DSP56824UM/AD
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Commands, command parameters, code examples, expressions, datatypes, and directives	...`*Foundational include files`... ...a data structure of type `vad_tConfigure`...
Italic	Calls, functions, statements, procedures, routines, arguments, file names and applications	...the *pConfig* argument... ...defined in the C header file, *aec.h*... ...makes a call to the *Callback* procedure...
Bold	Reference sources, paths, emphasis	...refer to the **Targeting DSP56824 Platform** manual.... ... see: **C:\Program Files\Motorola\Embedded SDK\help\tutorials**
Bold/Italic	Directory name, project name	...and contains these core directories: ***applications*** contains applications software.... ...CodeWarrior project, ***3des.mcp***, is.....
Blue Text	Linkable on-line	...refer to **Chapter 7**, License...
Number	Any number is considered a positive value, unless preceded by a minus symbol to signify a negative value	3V -10 DES^{-1}
ALL CAPITAL LETTERS	Variables, directives, defined constants, files libraries	INCLUDE_DSPFUNC #define INCLUDE_STACK_CHECK
Brackets [...]	Function keys	...by pressing function key [F7]...
Quotation marks "... "	Returned messages	...the message, "Test Passed" is displayed.... ...if unsuccessful for any reason, it will return "NULL"....

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

DSP	Digital Signal Processor or Digital Signal Processing
FFT	Fast Fourier Transforms
FIR	Finite Impulse Response
I/O	Input/Output

IDE	Integrated Development Environment
IIR	Infinite Impulse Response
LSB	Least Significant Bit
MAC	Multiply/Accumulate
MIPS	Million Instructions Per Second
MSB	Most Significant Bit
OnCE™	On-Chip Emulation
OMR	Operating Mode Register
PC	Program Counter
SDK	Software Development Kit
SP	Stack Pointer
SPI	Serial Peripheral Interface
SR	Status Register
SRC	Source
VAD	Voice Activity Detector / Voice Activity Detection

References

The following sources were referenced to produce this book:

1. *DSP56800 Family Manual*, DSP56800FM/AD
2. *DSP56824 User's Manual*, DSP56824UM/AD
3. *Embedded SDK Programmer's Guide*

Chapter 1
Introduction

Welcome to Motorola's Family of Digital Signal Processors (DSPs). This document describes the Voice Activity Detection (VAD) Library, which is a part of Motorola's comprehensive Software Development Kit (SDK) for its DSPs. In this manual, you will find all the information required to use and maintain the VAD Library interface and algorithms.

Motorola provides these algorithms to you for use on the Motorola DSPs to expedite your application development and reduce the time it takes to bring your own products to market.

Motorola's VAD Library is licensed for your use on Motorola processors. Please refer to the standard Software License Agreement in **Chapter 7** for license terms and conditions; please consult with your Motorola representative for premium product licensing.

1.1 Quick Start

Motorola's Embedded SDK is targeted to a large variety of hardware platforms. To take full advantage of a particular hardware platform, use **Quick Start** from the **Targeting DSP568xx Platform** documentation.

For example, the **Targeting DSP56824 Platform** manual provides more specific information and examples about this hardware architecture. If you are developing an application for a DSP56824EVM board or any other DSP56824 development system, refer to the **Targeting DSP56824 Platform** manual for **Quick Start** or other DSP56824-specific information.

1.2 Overview of VAD

Voice activation detection is used to save bandwidth by sending packets only when speech is present. The effectiveness of the voice activity detector can be determined by measuring these factors:

1. Front-end clipping, which is the amount of time it takes a voice activity detector to detect speech and begin transmitting audio
2. Holdover time, which is the amount of time needed to determine that speech is no longer present and to stop transmitting background audio

1.2.1 Background

The VAD detects voice activity and activates or deactivates the transmission of packets to optimize bandwidth. When no activity is detected, the encoder output will not be transported across the network. Idle noise is reproduced by the remote end when there is no voice activity, so the remote user will not believe that the line has gone dead.

1.2.2 Features and Performance

The VAD library is multichannel and re-entrant.

For details on Memory and MIPS for a particular DSP, refer to the **Libraries** chapter of the appropriate Targeting manual.

Chapter 2
Directory Structure

2.1 Required Core Directories

Figure 2-1 details required platform directories:

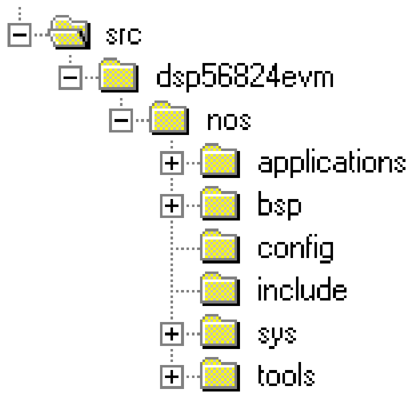


Figure 2-1. Core Directories

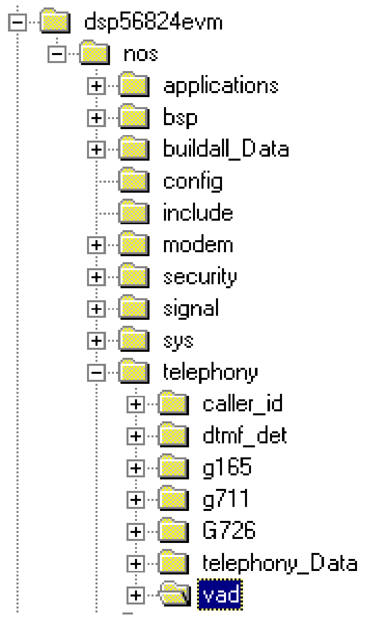
As shown in Figure 2-1, DSP56824EVM has no operating system (nos) support and contains these core directories:

- ***applications*** contains applications software that can be exercised on this platform
- ***bsp*** contains board support package specific for this platform
- ***config*** contains default hardware/software configurations for this platform
- ***include*** contains SDK header files which define the Application Programming Interface
- ***sys*** contains required system components
- ***tools*** contains useful utilities used by system components

There are also optional directories that include domain-specific libraries.

2.2 Optional (Domain-Specific) Directories

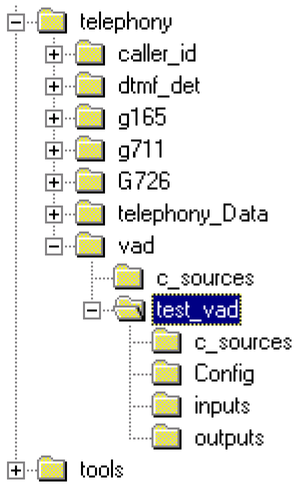
Figure 2-2 demonstrates how the VAD algorithm is encapsulated in the domain-specific directories under the directory ***telephony***.



```
dsp56824evm
  nos
    applications
    bsp
    buildall_Data
    config
    include
    modem
    security
    signal
    sys
    telephony
      caller_id
      dtmf_det
      g165
      g711
      G726
      telephony_Data
      vad
```

Figure 2-2. Domain-Specific Directories

The ***telephony*** directory includes telephony-specific algorithms. Figure 2-3 shows the ***vad*** directory structure.



```
telephony
  caller_id
  dtmf_det
  g165
  g711
  G726
  telephony_Data
  vad
    c_sources
    test_vad
      c_sources
      Config
      inputs
      outputs
tools
```

Figure 2-3. *vad* Directory Structure

The ***vad*** directory includes the following sub-directories:

- ***c_sources*** includes the APIs for VAD
- ***test_vad*** includes these sub-directories:
  - — ***c_sources*** contains an example test code
  - — ***Config*** contains configuration files *appconfig.c*, *appconfig.h* and *linker.cmd* specific to VAD
  - — ***inputs*** contains a speech file for testing VAD
  - — ***outputs*** contains reference VAD output

The ***applications*** directory includes high-level software that exercises the ***vad*** library. For example, Figure 2-4 shows the location of the ***demo_vad*** application under ***telephony*** in the ***applications*** directory.

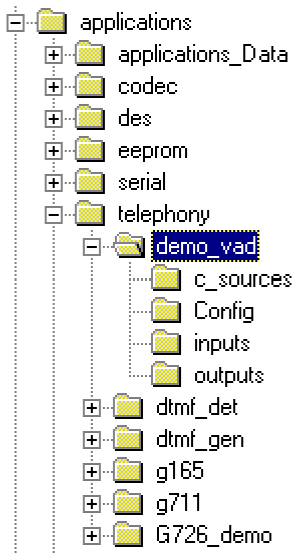


Figure 2-4. VAD Applications

Chapter 3
VAD Library Interfaces

3.1 VAD Services

The VAD library checks for speech segments in the speech signal and sets a flag if the algorithm detects a speech segment. The data to be supplied must be in 16-bit word, fixed point (1.15) format, shown in the following table.

s MSB	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i LSB

i = information bit

s = sign

3.2 Interface

The C interface for the VAD library services is defined in the C header file *vad.h*, shown in **Code Example 3-1** as a reference.

Code Example 3-1. C Header File *vad.h*

```
/* File: vad.h */

#ifndef __VAD_H
#define __VAD_H

/*
   Voice Activity Detection interface
*/

/****************************
 Foundational Include Files
*****************************/

#include "port.h"
```

```
/* VadResult which is defined in vad_sCallback
   struct below, takes one of the following
   enum values */

typedef enum
{
      VAD_VOICE_NOT_DETECTED,
      VAD_VOICE_DETECTED,
} vad_eResult;

/********************************************
     Structure for VAD Configuration
********************************************/
typedef struct
{
      void  (*pCallback) ( void   *pCallbackArg,
                           vad_eResult VadResult);
      void *pCallbackArg;
} vad_sCallback;

typedef struct
{
      vad_sCallback callback;
} vad_sConfigure;

typedef struct
{
      Int16   *pInContextBuf;
      Int16   ContextBufLength;
      Int16   beta;
      UWord32 EnergyNInit;
      UWord32 EnergySInit;
      UWord32 EnergyN;
      UWord32 EnergyS;
      UInt16  FlagF;
      UInt16  Counter;
      UInt16  HangCnt;
      UInt16  HangConst;
      UInt16  BurstCnt;
      UInt16  BurstConst;
      vad_sCallback *pCallback;
}vad_sHandle;

/***************************
 Function Prototypes
***************************/


typedef struct vad_sHandle vad_tHandle;
EXPORT vad_sHandle   *vadCreate (vad_sConfigure *pConfig);
EXPORT Result vadInit ( vad_sHandle    *pVad, vad_sConfigure *pConfig);
EXPORT Result vadProcess ( vad_sHandle *pVad, Int16 *pSamples, UInt16 NumSamples);
EXPORT void vadDestroy (vad_sHandle *pVad);

#endif
```

3.3 Specifications

The following pages characterize the VAD library functions.

Function arguments for each routine are described as *in*, *out*, or *inout*. An *in* argument means that the parameter value is an input only to the function. An *out* argument means that the parameter value is an output only from the function. An *inout* argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

Typically, *inout* parameters are input pointer variables in which the caller passes the address of a preallocated data structure to a function. The function stores its results within that data structure. The actual value of the *inout* pointer parameter is not changed.

3.3.1 *vadCreate*

Call(s):

```
vad_sHandle *vadCreate(vad_sConfigure*pConfig);
```

Required Header: “vad.h”

Arguments:

Table 3-1. *vadCreate* Arguments

pConfig	in	Points to the configuration structure for VAD

Description: The *vadCreate* function creates an instance of VAD. The *pConfig* argument points to the *vad_sConfigure* structure used to configure VAD operation. For initialization of the *vad_sConfigure* structure, see *vadInit*, **Section 3.3.2**. During the *vadCreate* call, any dynamic resources required by the VAD algorithm are allocated. In each instance, 84 words of external data memory are allocated.The library allocates memory dynamically using the *mem* library as shown in **Code Example 3-2**. The VAD library is **multichannel** and **re-entrant**.

Code Example 3-2. *mem* Library

```
#include "mem.h"
#include "vad.h"

#define FRAME_SZ 64


vad_sHandle *vadCreate (vad_sConfigure *pConfig)
{
          vad_sHandle *pVad;
          Result   res;

          pVad = (vad_sHandle *) memMallocEM (sizeof (vad_sHandle));

          if (pVad == NULL) return (NULL);

          pVad->pInContextBuf = (Int16 *) memMallocEM (FRAME_SZ * sizeof(Int16));
          if (pVad->pInContextBuf == NULL)
          {
              vadDestroy (pVad);
              return (NULL);
          }

          pVad->pCallback = (vad_sCallback *) memMallocEM (sizeof(vad_sCallback));
          if (pVad->pCallback == NULL)
          {
              vadDestroy (pVad);
              return (NULL);
          }
```

```
            res = vadInit (pVad,pConfig);
            return (pVad);

}
```

For details on the *vad_sHandle* structure, refer to **Code Example 3-1**.

If a *vadCreate* function is called to create an instance, then the *vadDestroy* function (see **Section 3.3.4**) is used to destroy the instance.

Alternatively, the user can allocate memory statically which requires duplicating all statements in the *vadCreate* function. In this case, the user can call the *vadInit* function directly, bypassing the *vadCreate* function. If the user dynamically allocates memory without calling *vadCreate*, then the user himself must destroy the memory allocated.

Returns: Upon successful completion, the *vadCreate* function will return a pointer to the specific instance of VAD created. If *vadCreate* is unsuccessful for any reason, it will return “NULL”.

Special Considerations:

- The VAD application is multichannel and re-entrant.
- If *vadCreate* is called, then the user need not call the *vadInit* function, as it is called internally in the *vadCreate* function.
- The *vadDestroy* function must be called to deallocate the memory allocated by *vadCreate*.

Code Example: In **Code Example 3-3**, the application creates an instance of VAD.

Code Example 3-3. Use of *vadCreate* Interface

```
#include “vad.h”
#include "mem.h"
#define VAD_BUF_LENGTH 50 /* User output buffer length */
typedef struct
{
    UInt16 *VAD_FLAG;
    UInt16 offset;
} vad_sCallbackArg;

void Callback (void *pCallbackArg, vad_eResult VAD_FLAG);
void test_vad (void)
{
   vad_sHandle *pVad;
   vad_sConfigure *pConfig;
   vad_sCallbackArg *vad_tCallbackArg;

   pConfig = (vad_sConfigure *) memMallocEM(sizeof (vad_sConfigure));

    /* User configuration of VAD */
   vad_tCallbackArg = (vad_sCallbackArg *) memMallocEM (sizeof(vad_sCallbackArg));
```

```
    vad_tCallbackArg->VAD_FLAG = (UInt16 *) memMallocEM
                             (sizeof(UInt16)*VAD_BUF_LENGTH);

    vad_tCallbackArg->offset = 0;
    pConfig->callback.pCallback = Callback;
    pConfig->callback.pCallbackArg = (vad_sCallbackArg *) vad_tCallbackArg;

    /* Create and init the instance of VAD */
    pVad = vadCreate(pConfig);}
}
```

3.3.2 *vadInit*

Call(s):

```
Result vadInit (vad_sHandle *pVad, vad_sConfigure *pConfig);
```

Required Header: "vad.h"

Arguments:

Table 3-2. *vadInit* Arguments

pVad	in	Handle to an instance of VAD
pConfig	in	A pointer to a data structure containing data for initializing the VAD algorithm

Description: The *vadInit* function will initialize the VAD algorithm. During initialization, all resources will be set to their initial values in preparation for VAD operation.

The parameter *pVad* must have been generated from a call to *vadCreate*. The parameter *pConfig* points to a data structure of type `vad_sConfigure`; its fields initialize VAD operation in the following manner:

Callback A structure of type *vad_sCallback*; it describes the procedure which VAD will call once **64** samples are processed by the algorithm. The callback procedure has the following declaration:

```
void (*pCallback) (void *pCallbackArg, vad_eResult VadResult);
```

The callback procedure parameter, *pCallbackArg*, is supplied by the user in the *vad_sCallback* structure; this value is passed back to the user during the call to the callback procedure. Typically, *pCallbackArg* points to context information used by the callback procedure, which the user must write.

The *vadResult* will take one of the following enum values:

```
VAD_VOICE_NOT_DETECTED
VAD_VOICE_DETECTED
```

An example callback procedure is shown as a reference in **Code Example 3-4**4. You must write your own callback procedure. This callback procedure stores the *vadResult* in a buffer specified by the user through the *pCallbackArg* pointer. For details about *vad_sCallbackArg*, see **Code Example 3-3**.

Code Example 3-4. Sample Callback Procedure

```
void Callback ( void *pCallbackArg, vad_eResult VAD_FLAG)
{
       vad_sCallbackArg *vad_tCallbackArg;

       vad_tCallbackArg = (vad_sCallbackArg *) pCallbackArg;
       vad_tCallbackArg->VAD_FLAG[vad_tCallbackArg->offset] = VAD_FLAG;
       vad_tCallbackArg->offset++;
       return;
}
```

Returns: Upon completion, a value of “PASS” will be returned.

Special Considerations:

- If *vadCreate* is called, then the user need not call *vadInit* function as it is called internally in the *vadCreate* function.

Code Example: In **Code Example 3-5**, the application creates an instance of VAD, which is passed to *vadInit* along with the VAD configuration structure *pConfig*.

Code Example 3-5. Use of *vadInit* Interface

```
#include “vad.h”
#include "mem.h"

#define VAD_BUF_LENGTH 50 /* User output buffer length */

typedef struct
{
    UInt16 *VAD_FLAG;
    UInt16 offset;
} vad_sCallbackArg;

void Callback (void *pCallbackArg, vad_eResult VAD_FLAG);

void test_vad (void)
{
   vad_sHandle *pVad;
   vad_sConfigure *pConfig;
   vad_sCallbackArg *vad_tCallbackArg;
   Result res;

   pConfig = (vad_sConfigure *) memMallocEM(sizeof (vad_sConfigure));

    /* User configuration of VAD */
   vad_tCallbackArg = (vad_sCallbackArg *) memMallocEM (sizeof(vad_sCallbackArg));

    vad_tCallbackArg->VAD_FLAG = (UInt16 *) memMallocEM
                                 (sizeof(UInt16)*VAD_BUF_LENGTH);
    vad_tCallbackArg->offset = 0;
    pConfig->callback.pCallback = Callback;
    pConfig->callback.pCallbackArg = (vad_sCallbackArg *) vad_tCallbackArg;

    /* Statically Create the instance of VAD */
   ....
   res = vadInit (pVad, pConfig);
}
```

3.3.3 *vadProcess*

Call(s):

```
Result vadProcess (vad_sHandle * pVad, Int16 *pSamples,
                   UInt16 NumSamples);
```

Required Header: “vad.h”

Arguments:

Table 3-3. *vadProcess* Arguments

pVad	in	Handle to an instance of VAD
pSamples	in	Pointer to speech samples to be used by the VAD algorithm
NumSamples	in	The number of samples to be processed

Description: The *vadProcess* function will process the samples supplied by *pSamples*. Once the processing is complete, the result is returned to the user by calling the *Callback* procedure. The user can call the *vadProcess* function any number of times, as long as there is data.

Returns: Upon completion, *vadProcess* will return “PASS”.

Special Considerations:

- The *vadProcess* function makes a call to the *Callback* procedure only when **64** samples of data are processed.
- After processing 64 samples, *vadProcess* function does a callback with either *VAD_VOICE_DETECTED* (speech frame), or *VAD_VOICE_NOT_DETECTED* (silence frame).

Code Example 3-6. Use of *vadProcess* Interface

```
#include “vad.h”
#include "mem.h"

#define VAD_BUF_LENGTH 50 /* User output buffer length */

typedef struct
{
    UInt16 *VAD_FLAG;
    UInt16 offset;
} vad_sCallbackArg;

void Callback (void *pCallbackArg, vad_eResult VAD_FLAG);

void test_vad (void)
{
   vad_sHandle *pVad;
   vad_sConfigure *pConfig;
   vad_sCallbackArg *vad_tCallbackArg;
   Int16 InBuffer[100];
   UInt16 length = 100;
   Result res;
   pConfig = (vad_sConfigure *) memMallocEM(sizeof (vad_sConfigure));

    /* User configuration of VAD */
   vad_tCallbackArg = (vad_sCallbackArg *) memMallocEM (sizeof(vad_sCallbackArg));

    vad_tCallbackArg->VAD_FLAG = (UInt16 *) memMallocEM
                                 (sizeof(UInt16)*VAD_BUF_LENGTH);

    vad_tCallbackArg->offset = 0;
    pConfig->callback.pCallback = Callback;
    pConfig->callback.pCallbackArg = (vad_sCallbackArg *) vad_tCallbackArg;

    /* Create and init the instance of VAD */
    pVad = vadCreate(pConfig);}

   ...

   res = vadProcess (pVad, InBuffer, length);

}
```

3.3.4 *vadDestroy*

Call(s):

```
void vadDestroy (vad_sHandle *pVad);
```

Required Header: “vad.h”

Arguments:

Table 3-4. *vadDestroy* Arguments

pVad	in	Handle to an instance of VAD generated by a call to *vadCreate*

Description: The *vadDestroy* function destroys the instance of the VAD originally created by a call to *vadCreate*.

Returns: None

Special Considerations: During a call to *vadDestroy*, any remaining samples in the context buffer which do not make a 64 sample frame are not processed.

Code Example 3-7. Use of *vadDestroy* Interface

```
#include “vad.h”
#include "mem.h"

#define VAD_BUF_LENGTH 50 /* User output buffer length */

typedef struct
{
    UInt16 *VAD_FLAG;
    UInt16 offset;
} vad_sCallbackArg;

void Callback (void *pCallbackArg, vad_eResult VAD_FLAG);

void test_vad (void)
{
   vad_sHandle *pVad;
   vad_sConfigure *pConfig;
   vad_sCallbackArg *vad_tCallbackArg;
   Int16 InBuffer[100];
   UInt16 length = 100;
   Result res;

   pConfig = (vad_sConfigure *) memMallocEM(sizeof (vad_sConfigure));

    /* User configuration of VAD */
   vad_tCallbackArg = (vad_sCallbackArg *) memMallocEM (sizeof(vad_sCallbackArg));

    vad_tCallbackArg->VAD_FLAG = (UInt16 *) memMallocEM
                             (sizeof(UInt16)*VAD_BUF_LENGTH);
```

```
     vad_tCallbackArg->offset = 0;
     pConfig->callback.pCallback = Callback;
     pConfig->callback.pCallbackArg = (vad_sCallbackArg *) vad_tCallbackArg;

     /* Create and init the instance of VAD */
     pVad = vadCreate(pConfig);}

    ...

    res = vadProcess (pVad, InBuffer, length);

    vadDestroy (pVad);
}
```

Chapter 4
Building the VAD Library

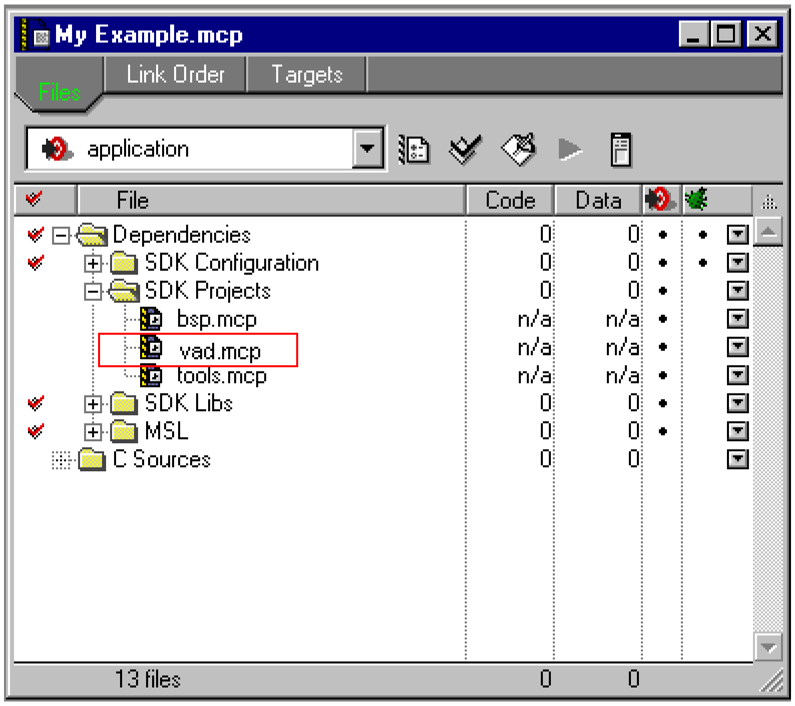
4.1 Building the VAD Library

The VAD library combines all of the components described in previous sections into one library: *vad.lib*. To build this library, a Metrowerks' CodeWarrior project, ***vad.mcp***, is provided. This project and all the necessary components to build the VAD library are located in the ***...\nos\telephony\vad*** directory of the SDK directory structure.

There are two methods to execute a system library project build: dependency build and direct build.

4.1.1 Dependency Build

Dependency build is the easiest approach and requires no additional work on the user's part. If you add the VAD library project, ***vad.mcp,*** to your application project as shown in **Figure 4-1**, the VAD library will automatically build when the application is built.


My Example.mcp

Files | Link Order | Targets

application

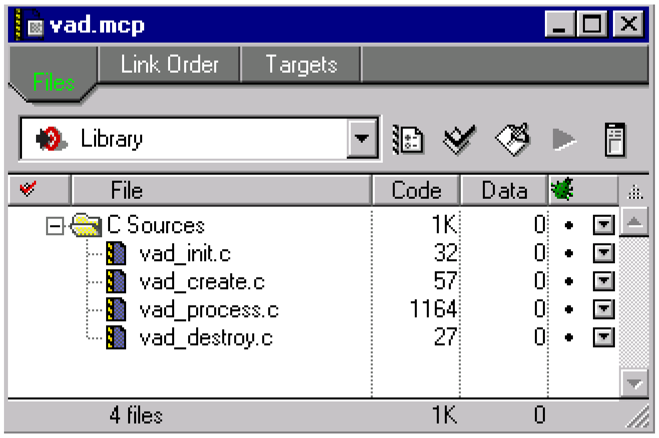
File	Code	Data
Dependencies	0	0
SDK Configuration	0	0
SDK Projects	0	0
bsp.mcp	n/a	n/a
vad.mcp	n/a	n/a
tools.mcp	n/a	n/a
SDK Libs	0	0
MSL	0	0
C Sources	0	0
13 files	0	0

Figure 4-1. Dependency Build for VAD Library

4.1.2 Direct Build

Direct build allows you to build a VAD library independently of any other build. Follow these steps:

Step 1. Open ***vad.mcp*** project, as shown in **Figure 4-2**.


vad.mcp

Files | Link Order | Targets

Library

File	Code	Data
C Sources	1K	0
vad_init.c	32	0
vad_create.c	57	0
vad_process.c	1164	0
vad_destroy.c	27	0
4 files	1K	0

Figure 4-2. *vad.mcp* Project

Step 2. Execute the build by pressing function key [*F7]* or by choosing *Make* from the Project menu; see **Figure 4-3**.

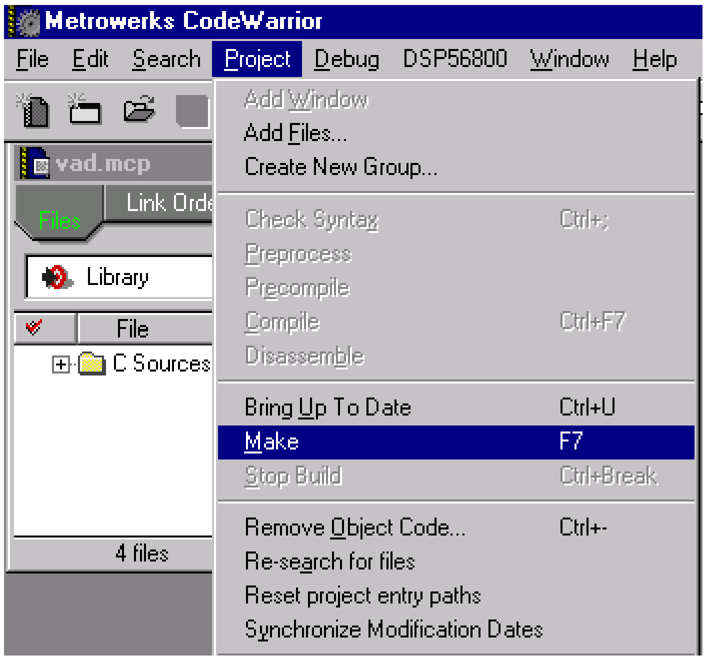


Figure 4-3. Execute Make

At this point, if the build is successful, the *vad.lib* library file is created in the ***...\nos\telephony\vad\Debug*** directory.

Chapter 5
Linking Applications with the VAD Library

5.1 VAD Library

The library includes APIs, which provide interface between the user application and the VAD modules. To invoke VAD, APIs must be called in the following order:

— vadCreate (.......);
— vadInit (.......);
— vadProcess (.......);
— vadDestroy (.......);

5.1.1 Library Sections

The VAD Library contains no assembly code. To develop Debug applications with the VAD Library, use the default *linker.cmd* file. Because the VAD library is written in C language, there are no specific program and data sections to be included in the *linker.cmd* file.

Please see the *linker.cmd* example file in the **...telephony\vad\test_vad\Config\** directory, found in the Software Development Kit, (SDK). A sample linker.cmd file is included in Code Example for the reference.

Code Example 5-1. ***linker.cmd***

```
# Linker.cmd file for DSP56824EVM External RAM
#       using both internal and external data memory (EX = 0)
#    and using external program memory (Mode = 3)

MEMORY {

       .pram   (RWX) : ORIGIN = 0x0000, LENGTH = 0xFF80  # ? external program memory

       .avail  (RW)  : ORIGIN = 0x0000, LENGTH = 0x0030  # available
       .cwregs (RW)  : ORIGIN = 0x0030, LENGTH = 0x0010  # C temp registrs in
                                 CodeWarrior
```

```
      .im1    (RW)  : ORIGIN = 0x0040, LENGTH = 0x07C0  # data 1
      .rom    (R)   : ORIGIN = 0x0800, LENGTH = 0x0800  # internal data ROM
      .im2    (RW)  : ORIGIN = 0x1000, LENGTH = 0x0600  # data 2
      .hole   (R)   : ORIGIN = 0x1600, LENGTH = 0x0A00  # hole
      .data   (RW)  : ORIGIN = 0x2000, LENGTH = 0xC000  # data segment
      .em     (RW)  : ORIGIN = 0xE000, LENGTH = 0x1000  # data 3
      .stack  (RW)  : ORIGIN = 0xF000, LENGTH = 0x0F80  # stack
      .onchip1(RW)  : ORIGIN = 0xFF80, LENGTH = 0x0040  # on-chip peripheral
                          registers
      .onchip2(RW)  : ORIGIN = 0xFFC0, LENGTH = 0x0040  # on-chip peripheral
                          registers

}


FORCE_ACTIVE {FconfigInterruptVector}

SECTIONS {

      #
      # Data (X) Memory Layout
      #
            _EX_BIT     = 0;

            # Internal Memory Partitions (for mem.h partitions)
            _NUM_IM_PARTITIONS = 2;  # .im1 and .im2

            # External Memory Partition (for mem.h partitions)
            _NUM_EM_PARTITIONS = 1;   # .em


      .main_application_code :
      {
            # .text sections

            #  config.c MUST be placed first, otherwise the Interrupt Vector
            #  configInterruptVector will not be located at the correct address,
                          P:0x0000

            config.c (.text)
            * (.text)
            * (rtlib.text)
            * (fp_engine.text)
            * (user.text)

      } > .pram


      .main_application_data :
      {
            #
            # Define variables for C initialization code
            #
            F_Xdata_start_addr_in_ROM = ADDR(.rom) + SIZEOF(.rom) / 2;
            F_StackAddr               = ADDR(.stack);
            F_StackEndAddr            = ADDR(.stack) + SIZEOF(.stack) / 2  - 1;
            F_Xdata_start_addr_in_RAM = .;

            #
            # Memory layout data for SDK INCLUDE_MEMORY (mem.h) support
            #

            FmemEXbit = .;
                  WRITEH(_EX_BIT);
            FmemNumIMpartitions = .;
                  WRITEH(_NUM_IM_PARTITIONS);
            FmemNumEMpartitions = .;
                  WRITEH(_NUM_EM_PARTITIONS);
            FmemIMpartitionList = .;
            #     WRITEH(ADDR(.im1));
            #     WRITEH(SIZEOF(.im1) / 2);
                  WRITEH(ADDR(.im2));
                  WRITEH(SIZEOF(.im2) / 2);
            FmemEMpartitionList = .;
                  WRITEH(ADDR(.em));
                  WRITEH(SIZEOF(.em) /2);

            # .data sections

            * (.data)
            * (fp_state.data)
            * (rtlib.data)

            F_Xdata_ROMtoRAM_length = 0;

            F_bss_start_addr = .;
            _BSS_ADDR = .;

            * (rtlib.bss.lo)
            * (.bss)

            F_bss_length = . - _BSS_ADDR;  # Copy DATA


      } > .data

      FArchIO   = ADDR(.onchip2);

}
```

Chapter 6
VAD Applications

6.1 Test and Demo Applications

To verify the VAD algorithm, test and demo applications have been developed. Refer to the **Targeting Motorola DSP568xx Platform** Manual for the DSP you are using to see if the test and demo applications are available for your target.

Chapter 7 License

7.1 Limited Use License Agreement

LIMITED USE LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THIS SOFTWARE. BY USING OR COPYING THE SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

The software in either source code form ("Source") or object code form ("Object") (cumulatively hereinafter "Software") is provided under a license agreement ("Agreement") as described herein. Any use of the Software including copying, modifying, or installing the Software so that it is usable by or accessible by a central processing unit constitutes acceptance of the terms of the Agreement by the person or persons making such use or, if employed, the employer thereof ("Licensee") and if employed, the person(s) making such use hereby warrants that they have the authority of their employer to enter this license agreement,. If Licensee does not agree with and accept the terms of this Agreement, Licensee must return or destroy any media containing the Software or materials related thereto, and destroy all copies of the Software.

The Software is licensed to Licensee by Motorola Incorporated ("Motorola") for use under the terms of this Agreement. Motorola retains ownership of the Software. Motorola grants only the rights specifically granted in this Agreement and grants no other rights. Title to the Software, all copies thereof and all rights therein, including all rights in any intellectual property including patents, copyrights, and trade secrets applicable thereto, shall remain vested in Motorola.

For the Source, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to use, copy, and make derivatives of the Source solely in a development system environment in order to produce object code solely for operating on a Motorola semiconductor device having a central processing unit ("Derivative Object").

For the Object and Derivative Object, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to copy, use, and distribute the Object and the Derivative Object solely for operating on a Motorola semiconductor device having a central processing unit.

Licensee agrees to: (a) not use, modify, or copy the Software except as expressly provided herein, (b) not distribute, disclose, transfer, sell, assign, rent, lease, or otherwise make available the Software, any derivatives thereof, or this license to a third party except as expressly provided herein, (c) not remove obliterate, or otherwise defeat any copyright, trademark, patent or proprietary notices, related to the Software (d) not in any form export, re-export, resell, ship or divert or cause to be exported, re-exported, resold, shipped, or diverted, directly or indirectly, the Software or a direct product thereof to any country which the United States government or any agency thereof at the time of export or re-export requires an export license or other government approval without first obtaining such license or approval.

THE SOFTWARE IS PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MOTOROLA BE LIABLE FOR ANY LIABILITY OR DAMAGES OF ANY KIND INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT OR INCIDENTAL OR CONSEQUENTIAL OR PUNITIVE DAMAGES OR LOST PROFITS OR LOSS OF USE ARISING FROM USE OF THE SOFTWARE OR THE PRODUCT REGARDLESS OF THE FORM OF ACTION OR THEORY OF LIABILITY (INCLUDING WITHOUT LIMITATION, ACTION IN CONTRACT, NEGLIGENCE, OR PRODUCT LIABILITY) EVEN IF MOTOROLA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS DISCLAIMER OF WARRANTY EXTENDS TO LICENSEE OR USERS OF PRODUCTS AND IS IN LIEU OF ALL WARRANTIES WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE.

Motorola does not represent or warrant that the Software is free of infringement of any third party patents, copyrights, trade secrets, or other intellectual property rights or that Motorola has the right to grant the licenses contained herein. Motorola does not represent or warrant that the Software is free of defect, or that it meets any particular requirements or need of the Licensee, or that it conforms to any documentation, or that it meets any standards.

Motorola shall not be responsible to maintain the Software, provide upgrades to the Software, or provide any field service of the Software. Motorola reserves the right to make changes to the Software without further notice to Licensee.

The Software is not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Software could create a situation where personal injury or death may occur. Should Licensee purchase or use the Software for any such unintended or unauthorized application, Licensee shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the Software.

The term of this Agreement is for as long as Licensee uses the Software for its intended purpose and is not in default of any provisions of this Agreement. Motorola may terminate this Agreement if Licensee is in default of any of the terms and conditions of this Agreement.

This Agreement shall be governed by and construed in accordance with the laws of the State of Arizona and can only be modified in a writing signed by both parties. Licensee agrees to jurisdiction and venue in the State of Arizona.

By using, modifying, installing, compiling, or copying the Software, Licensee acknowledges that this Agreement has been read and understood and agrees to be bound by its terms and conditions. Licensee agrees that this Agreement is the complete and exclusive statement of the agreement between Licensee and Motorola and supersedes any earlier proposal or prior arrangement, whether oral or written, and any other communications relative to the subject matter of this Agreement.

Index

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors/

MOTOROLA