TMS320C548/C549 Bootloader and ROM Code Contents Technical Reference

Literature Number: SPRU288A
October1998 – Revised May 2000

Texas Instruments

Preface

Read This First

About This Manual

This document describes the operation of the TMS320C548/C549 bootloader and the process used to select the operating mode. The document also provides source code for the bootloader and discusses the other on-chip ROM features of these devices.

Notational Conventions

Program listings are shown in a `special typeface`. Here is a sample program listing:

```
ser_ini
          ld        ifr, a              ; check INT3 flag
          and       #103h, a            ;
          cc        TDMSP, aneq         ;
```

Related Documentation From Texas Instruments

The following books describe the TMS320C54x™ DSP and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS320C54x Assembly Language Tools User's Guide (literature number SPRU102) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C54x generation of devices.

Trademarks

TMS320C54x is a trademark of Texas Instruments Incorporated.

Contents

Figures

Tables

Chapter 1

Introduction

This chapter describes the purpose and features of the TMS320C548/C549 digital signal processor (DSP) bootloader. It also discusses the other contents of the devices' on-chip ROM and identifies where all of this information is located within that memory.

A listing of all of the '548/9 ROM contents is provided in the appendix section of this document, and electronic copies of these files are available on the web.

Topic **Page**

1.1 Bootloader Features

The TMS320C548/C549 bootloader is used to transfer code from an external source into internal or external program memory following power up. This allows code to reside in slow non-volatile memory externally, and be transferred to high-speed memory to be executed. This eliminates the need for mask programming the ’C548/9 internal ROM, which may not be cost effective in some applications.

The bootloader provides a variety of different ways to download code to accommodate different system requirements. This includes multiple types of both parallel bus and serial port boot modes, as well as bootloading through the HPI, allowing for maximum system flexibility. Bootloading in both 8-bit byte and 16-bit word modes is also supported.

The bootloader uses various control signals including interrupts, $\overline{\text{BIO}}$, and XF to determine which boot mode to use. The boot mode selection process, as well as the specifics of bootloader operation, are described in detail in Chapter 2 of this document.

1.2 On-Chip ROM Description

On the ’C548/C549 device, the on-chip ROM is factory programmed with the boot-load routine and additional features. Appendix A contains the code for each of the following items:

- Bootloader program
- 256-word μ-law and A-law expansion tables. Lookup tables that are used for decoding
- 256-word sine look-up table. Table consisting of 256 signed Q15 integers, representing 360 degrees
- Built-in self-test. Reserved code used by TI for life testing the device. The code goes through a number of math functions to exercise the various parts of the core such as the on-chip RAM, multiplier, adder, arithmetic logic unit (ALU), BSPs, etc.
- Interrupt vector table. Code that allows indirect vectoring of interrupts
- Data ROM tables. Tables used for the global system for mobile communication enhanced full rate (GSM EFR) speech codec (’C549 only)
- 256-point complex, radix-2 DIT FFT with looped code (’C549 only)
- FFT Twiddle factors for a complex FFT radix-2 with 256 points (’C549 only)
- 1024-point complex, radix–2 DIT FFT with looped code (’C549 only)
- FFT Twiddle factors for a complex FFT radix-2 with 1024 points (’C549 only)

A description of the ’C548/9 on-chip ROM functions other than the bootloader, along with a source listing of their contents is also provided in Appendix A.

Figure 1–1 shows the on-chip ROM for the ’C548. It is 2K words in size and is located at the 0xF800 – 0xFFFF address range in program space when the MP/$\overline{MC}$ input pin is low.

Figure 1–1. TMS320C548 On-Chip ROM Program Space

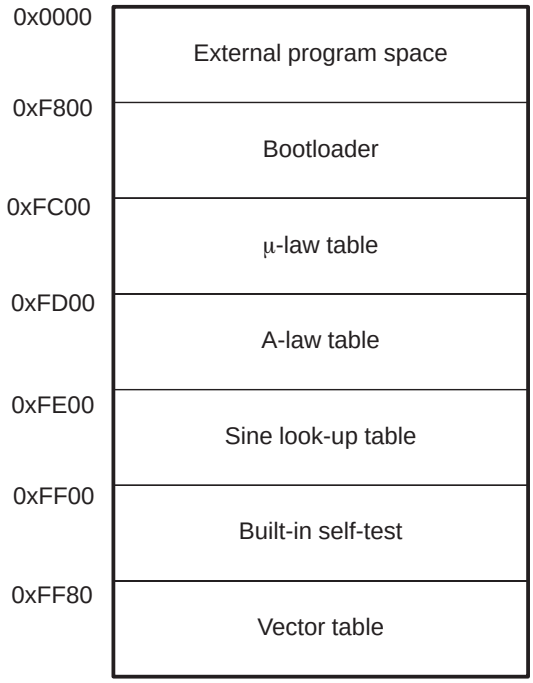
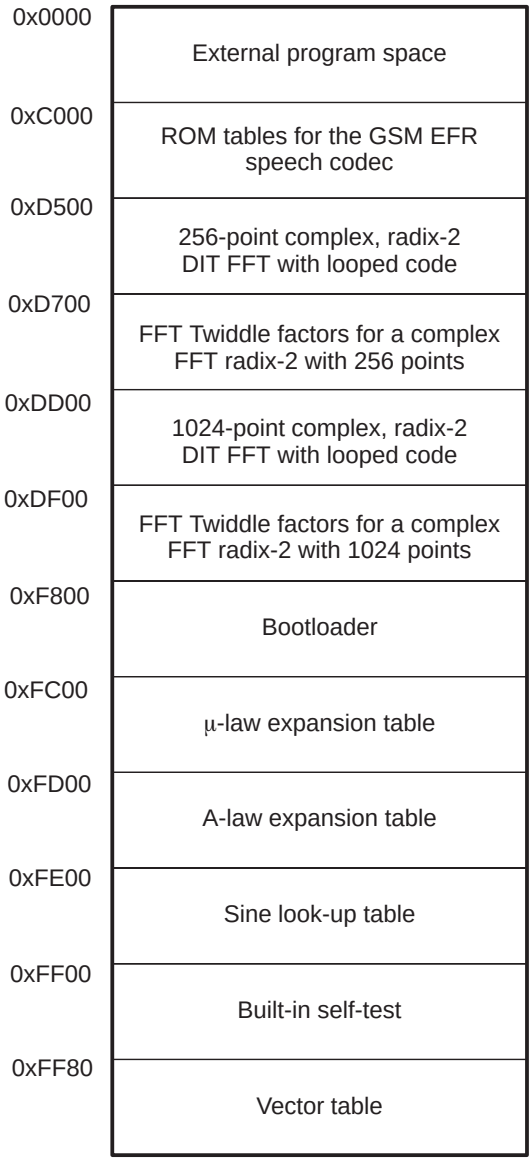


Figure 1–2 shows the ‘C549 on-chip ROM program space. It is 16K words in size and is located at the 0xC000 – 0xFFFF address range in program space when the MP/$\overline{MC}$ input pin is low.

Figure 1–2. TMS320C549 On-Chip ROM Program Space



Chapter 2

Bootloader Operation

This chapter describes in detail the boot mode selection process, as well as the specifics of bootloader operation. A source listing of the bootloader program is contained in the appendix section of this document.

Topic **Page**

2.1 Bootloader Functional Operation

The bootloader is the program located in the ’C548/9 on-chip ROM which is executed following reset when the device is in microcomputer mode (MP/$\overline{MC}$ = 0).

The function of the bootloader is to transfer user code from an external source to the program memory at power up. The bootloader sets up the CPU status registers before initiating the boot load; interrupts are globally disabled (INTM = 1) and the internal dual- and single-access RAMs are mapped into the program/data space (OVLY = 1). Seven wait states are initialized for the entire program and data spaces. The bootloader initializes bank switching on 4K boundaries, and one cycle is inserted when accesses switch between program and data space, or when crossing a page boundary.

To accommodate different system requirements, the ’C548/549 devices offer a variety of different boot modes. The following is a list of the different boot modes implemented by the bootloader, and a summary of their functional operation:

- Host Port Interface (HPI) Boot Mode:

  The code to be executed is loaded into HPI memory by an external host processor during reset, and is started by the bootloader after reset.

- Parallel Boot Modes:

  The bootloader reads the code to be loaded from the external parallel interface bus, from data space, loads the code at the indicated program memory address, and then starts execution at the indicated point within the downloaded code. Both 8-bit and 16-bit modes are supported.

- Standard Mode Serial Port Boot Modes:

  The bootloader receives the code to be loaded from one of the serial ports operating in standard mode (including the TDM port operating in standard mode), loads the code at the indicated program memory address, and then starts execution at the indicated point within the downloaded code. Both 8-bit and 16-bit modes are supported. ABU mode is not supported.

- TDM Mode Serial Port Boot Mode:

  The bootloader receives the code to be loaded from the TDM serial port operating in TDM mode, loads the code at the indicated program memory address, and then starts execution at the indicated point within the downloaded code. Only 16-bit mode is supported.

- I/O Boot Mode:

  The bootloader reads the code to be loaded from the external parallel interface bus employing an asynchronous handshake protocol using XF and $\overline{\text{BIO}}$. This allows data transfers to be performed at a rate dictated by the external device. The code is loaded at the indicated program memory address, and then starts execution at the indicated point within the downloaded code. Both 8-bit and 16-bit modes are supported.

The bootloader also offers the following additional features:

- Reprogrammable software wait states
- Reprogrammable bank switch sizes
- Multiple-section boot

The details of all of these bootload modes are described in the following sections of this chapter.

2.2 Boot Mode Selection

Once the bootloader is initiated, it performs a series of checking operations to determine which boot mode to use. The bootloader first checks for conditions that indicate it should perform an HPI boot. If the conditions are not met, it goes to the next mode and continues until it finds a mode which is selected. The flowchart shown in Figure 2–1 illustrates the process the bootloader uses to determine the desired boot mode.

The following is the sequence in which the bootloader searches, along with references to the sections where a detailed description can be found:

1) Host port interface (HPI) boot mode (see section 2.3.1 on page 2-12), selected by the presence of $\overline{\text{INT2}}$.

2) Parallel boot mode (see section 2.3.2 on page 2-14), attempted after checking for HPI boot.

3) Time division multiplexed (TDM) serial boot mode (see section 2.3.2.1 on page 2-17), selected by $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, or $\overline{\text{INT3}}$.

4) Standard serial boot mode (see section 2.3.2.2 on page 2-18), selected based on the presence of serial port interrupts.

5) I/O boot mode (see section 2.3.3 on page 2-26), selected if $\overline{\text{BIO}}$ is active.

As described above, the first boot mode tested is the HPI boot mode. If $\overline{\text{INT2}}$ is found to be active, the bootloader assumes that the code has been loaded into HPI RAM by an external host, and simply branches to the first location of this memory.

The next boot modes tested are the parallel boot modes. When using any of the parallel boot modes, although the program to be booted is expected to be located in external data memory space, the bootloader checks address 0FFFFh in both the I/O and data spaces for the source address in data space of the program to be booted. Therefore, the bootload function is not limited to requiring the source address to be only in data or I/O space. The address can be in either space, providing greater flexibility in implementing the external interface to the memory from which the user program is bootloaded. Also, the user program is not limited to being at any predetermined location in data memory; it can be located anywhere in the data memory space. Note, however, that when the source address is located in I/O space, only the six most significant bits (MSBs) of the address are read, and therefore the source address is always on a 1K-word boundary. These six most significant address bits are obtained from the six MSBs of the single-byte read from D7-D0 at I/O location 0FFFFh. This is further described in detail in section 2.3.2. When the

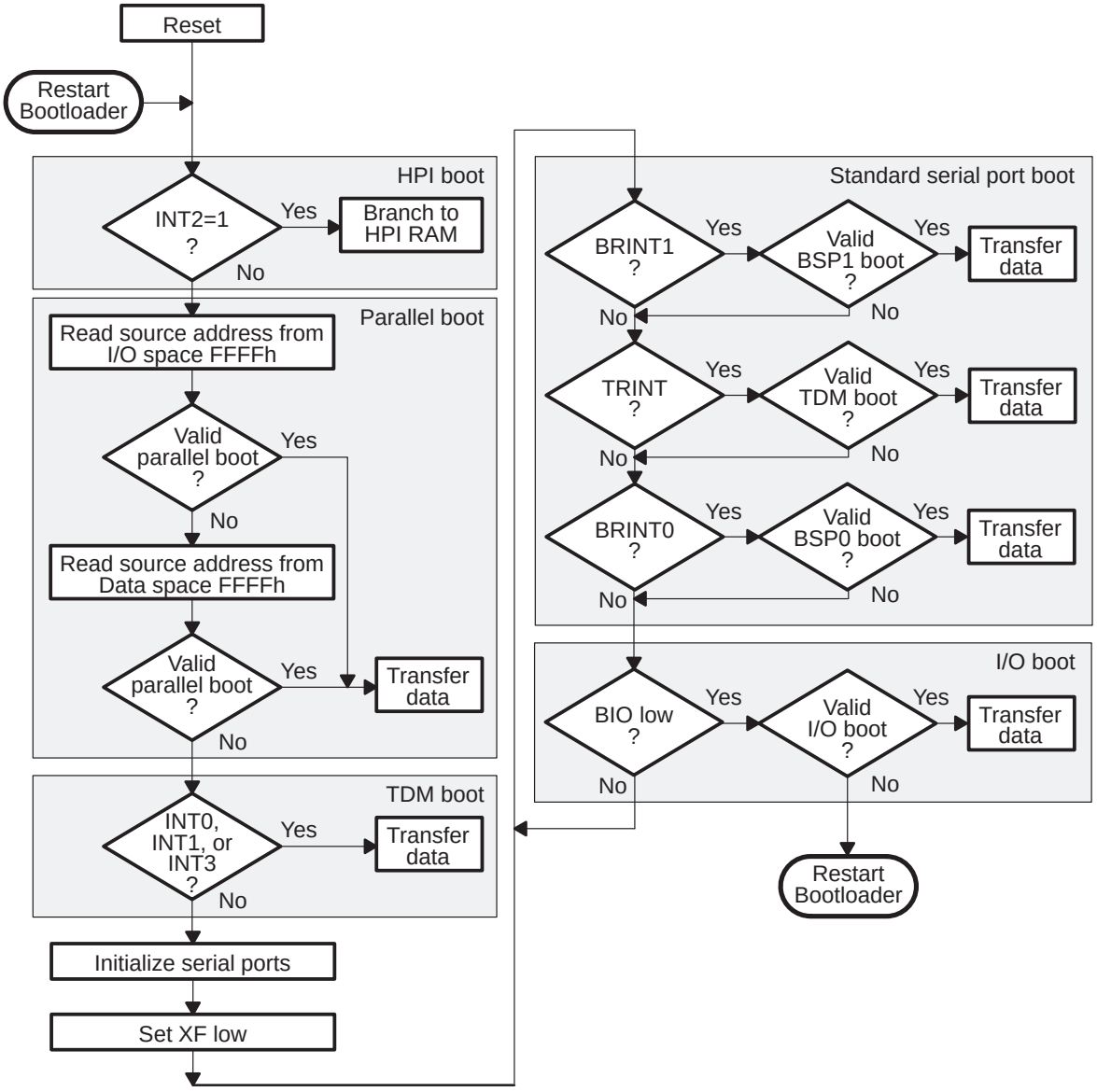
source address is located in data space, the address can be anywhere within a full 64K-word range.

Once the source address has been read, the bootloader then reads the first word from the boot table located at the source address. In the boot modes other than HPI boot mode (including serial boot modes), the first word in the boot table contains two important pieces of information: the data width option selection, and the bootloader recognition byte (BRB). The data width option selection is contained in the eight MSBs of the first word in the boot table, and tells the bootloader the width of the data: 10xxh indicates 16-bit data, and 08xx indicates 8-bit data. The eight LSBs are the BRB, 0AAh.

Since the ’C548/9 does not employ a boot routine selection (BRS) word to determine the desired boot mode (as do the ’C5x devices), the BRB, along with one of the two data width option selection values and the sequence of tests of interrupts and other control signals, is the criterion used to determine whether or not each of the boot modes is “valid” as described in the flowchart. Once the bootloader believes that it has found a possible boot mode selection, it checks for a valid data width selection and BRB (either 08AAh or 10AAh) as verification of this. If it finds a valid data width selection and BRB, it considers that it has found the selected boot mode; if not, it performs further tests in attempt to identify the selected mode. Accordingly, when using the bootloader, it is important to ensure that a valid BRB is not encountered inadvertently, and the wrong boot mode selected. This is discussed in further detail later in this chapter.

After the parallel boot modes are tested, the bootloader then checks for the serial boot modes and I/O boot mode, as shown in the flowchart. If a valid boot mode is not found after checking all of the possible boot modes, the bootloader restarts and continues to recheck each of the boot modes in the same sequence.

Figure 2–1. Bootloader Mode Selection Process



The flowchart shown in Figure 2–2 illustrates the basic process the bootloader uses to determine whether 8- or 16-bit data has been selected, transfer the data, and begin program execution. This process occurs after the bootloader finds what it believes may be a valid boot mode selected. When using the 8-bit boot modes, bytes must be ordered most significant byte first, and for parallel 8-bit modes, the bytes must be presented on the lower order eight bits of the data bus. Note that the exact sequence used for 8- and 16-bit mode processing differs somewhat depending on which specific boot mode is selected. The particular sequence used by each boot mode is shown in the sections describing the different boot modes in detail.

Figure 2–2. Basic Data-Width Selection and Data Transfer Process

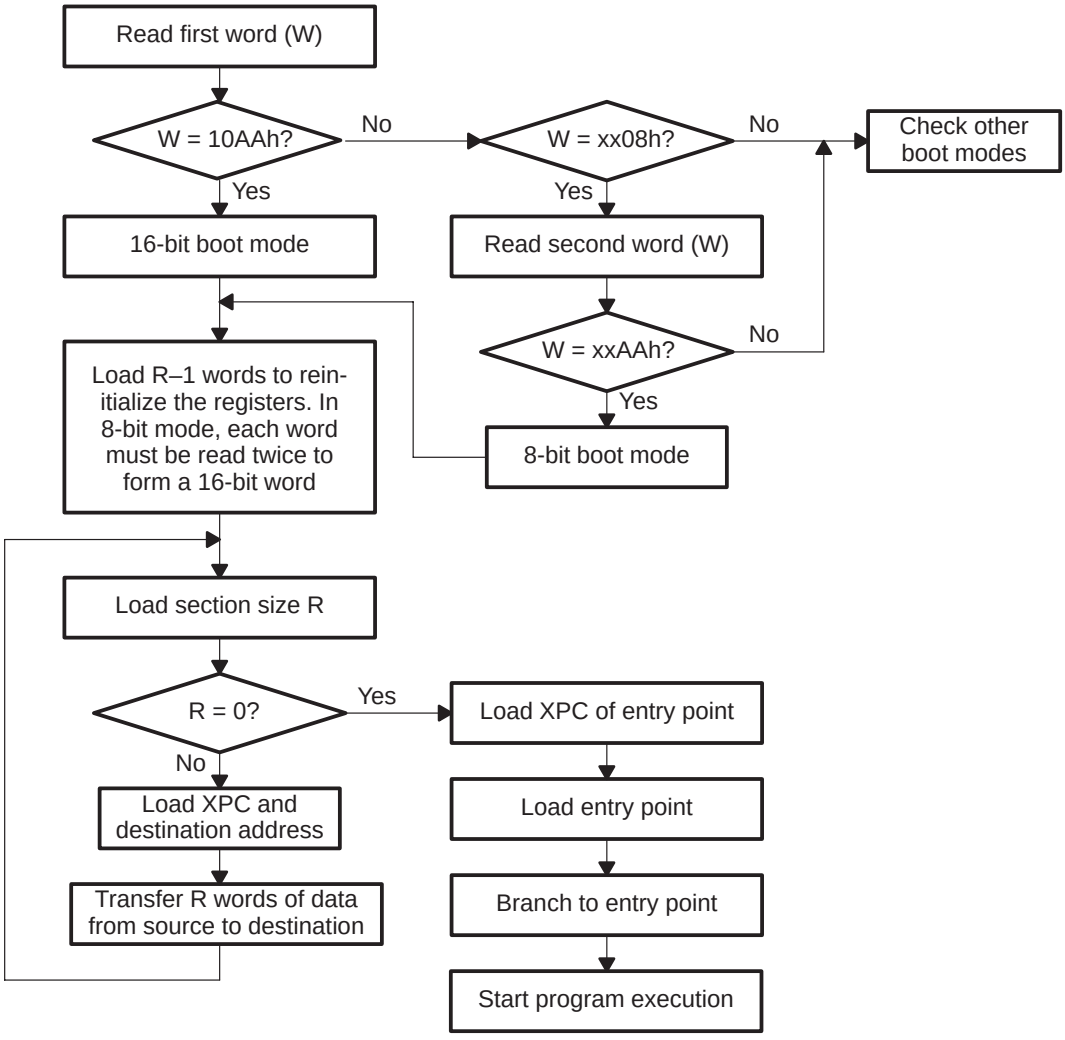


Table 2–1 and Table 2–2 show the structure of the code that is loaded for the boot modes. This structure is the same for all the modes, except HPI, which uses no specific code sequence. The first R–1 words indicate the number of words used to initialize the registers, and depends on the boot mode selected. The Rth word in the table indicates the block size of the first section to load. The contents of words R+1 through n vary for each section in the source program. The last two words contain the entry point of the program. The eight MSBs of the first word specify the memory width, and the eight LSBs of the first word are 0AAh. If 8-bit mode is selected, the MSBs are loaded first, followed by the LSBs.

Note that these tables are only included to show the basic structure of the code to be bootloaded. The exact structure of the code to be loaded varies depending on the boot mode selected. For specific information about the structure used by each of the boot modes, refer to the sections describing the different boot modes in detail.

Table 2–1. General Structure of Source Program Data Stream in 16-Bit Mode

Word	Contents
1	10AAh memory width of the source program is 16 bits
2	Value to set in the register (applied to the specified boot mode)
.	.
.	Value to set in the register
.	XPC value of the entry point (7 bits)
.	Entry point PC (16 bits)
R	Block size of the first section to load. The number of words is loaded next. If the next word is 0, this indicates the end of the source program. Otherwise, another section follows.
R+1	XPC value of the destination address of the first section (7 bits)
.	Destination address (PC) of the first section (16 bits)
.	First word of the first section of the source program
.	.
.	Last word of the first section of the source program
.	Block size of the second section to load
.	XPC value of the destination address of the second section (7 bits)
.	Destination address (PC) of the second section (16 bits)
	First word of the second section of the source program
.	.
.	Last word of the second section of the source program
.	.
.	.
.	Block size of the last section to load
.	XPC value of the destination address of the last section (7 bits)
.	Destination address (PC) of the last section (16 bits)
.	First word of the last section of the source program
.	.
.	Last word of the last section of the source program
n	0000h—indicates the end of source program

Table 2–2. General Structure of Source Program Data Stream in 8-Bit Mode

Byte	Contents
1	MSB = 08h, memory width of the source program (8 bits)
2	LSB = 0AAh
3	MSB of the value to set in the register
4	LSB of the value to set in the register
.	.
.	MSB of the value to set in the register
.	LSB of the value to set in the register
.	MSB of the XPC value of the entry point
.	LSB of the XPC value of the entry point (7 bits)
2R–1	MSB of the entry point (PC)
2R	LSB of the entry point (PC)
2R+1	MSB of the block size of the first section to load
2R+2	LSB of the block size of the first section to load
2R+3	MSB of the XPC value of the destination address of the first section
2R+4	LSB of the XPC value of the destination address of the first section (7 bits)
2R+5	MSB of the destination address (PC) of the first section
2R+6	LSB of the destination address (PC) of the first section
.	MSB of the first word of the first section of the source program
.	.
.	LSB of the last word of the first section of the source program
.	MSB of the block size of the second section to load
.	LSB of the block size of the second section to load
.	MSB of the XPC value of the destination address of the second section
.	LSB of the XPC value of the destination address of the second section (7 bits)
.	MSB of the destination address (PC) of the second section

Table 2–2. General Structure of Source Program Data Stream in 8-Bit Mode (Continued)

Byte	Contents
.	LSB of the destination address (PC) of the second section
.	MSB of the first word of the second section of the source program
.	.
.	LSB of the last word of the second section of the source program
.	.
.	MSB of the block size of the last section to load
.	LSB of the block size of the last section to load
.	MSB of the XPC value of the destination address of the last section
.	LSB of the XPC value of the destination address of the last section (7 bits)
.	MSB of the destination address (PC) of the last section
.	LSB of the destination address (PC) of the last section
.	MSB of the first word of the last section of the source program
.	.
.	LSB of the last word of the last section of the source program
2n	00h
2n+1	00h—indicates the end of the source program

2.3 Boot Mode Options

The following sections discuss the bootloader modes and the details of how the bootloader determines which mode to use.

2.3.1 HPI Boot Mode

HPI boot mode is the first boot mode tested by the bootloader. To determine whether or not to use HPI boot mode, the bootloader tests for the presence of an INT2 interrupt. In order to facilitate generation of this interrupt, the bootloader drives the $\overline{\text{HINT}}$ output low at an appropriate time such that HPI boot mode will be selected if $\overline{\text{HINT}}$ is connected to $\overline{\text{INT2}}$. Once the $\overline{\text{HINT}}$ output is asserted low, the bootloader then checks to see if the INT2 flag (IFR bit 2) is set to 1, and if so, initiates HPI boot mode.

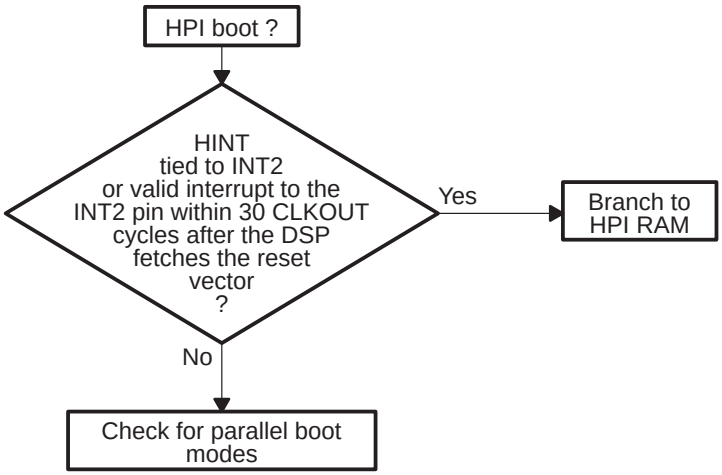
If $\overline{\text{HINT}}$ cannot be connected to $\overline{\text{INT2}}$ to select HPI boot mode, INT2 can be generated by another external signal, however, the interrupt must be generated by a falling edge on the $\overline{\text{INT2}}$ input within 30 CLKOUT cycles after the DSP fetches the reset vector.

Note, however, that since interrupts remain cleared as long as reset is asserted, and since a falling edge is required on $\overline{\text{INT2}}$ to generate an interrupt, $\overline{\text{INT2}}$ cannot simply be held low from power up to initiate an HPI boot.

If the INT2 flag is found to be 0 when tested, indicating to the bootloader that HPI boot mode has not been selected, the bootloader skips HPI mode and performs further tests to determine the selected boot mode (see Figure 2–3).

In HPI boot mode, the host must download the code to the HPI RAM before it brings the DSP out of reset. The bootloader transfers control to the start address of the on-chip HPI RAM (0x1000 in program space) and begins executing code from there. The bootloader keeps the HPI in shared-access mode during the entire boot-loading operation (SMOD = 0). When $\overline{\text{HINT}}$ has been asserted low, it stays low. The host controller can clear $\overline{\text{HINT}}$ by writing to the host port interface control register (HPIC). The source program data stream for HPI mode can only contain the program itself. It cannot include any extra information, such as section size or register values, since the bootloader does not perform any interaction with the source program data stream in HPI boot mode. The bootloader simply branches to the beginning of the HPI memory in this boot mode.

Figure 2–3. HPI Boot Mode Flow



2.3.2 Parallel Boot Mode

After verifying that an HPI boot was not selected, the bootloader checks for parallel boot mode. This mode is used if the code to be downloaded is stored in either 8- or 16-bit EPROMS or other memory located in data space. In parallel boot mode, the bootloader transfers the code from data memory to program memory to be executed.

In parallel boot mode, the bootloader obtains the source address (the address within data space of the code to be loaded into program memory) from either I/O port 0FFFFh or location 0FFFFh (and possibly 0FFFEh in 8-bit mode) in data memory. When the source address is located in I/O space, it must be on a 1K-word boundary; when it is located in data space, a 16-bit address is read, so the source address can be anywhere within a full 64K range.

To obtain the source address, the bootloader first checks at I/O port 0FFFFh. When read from I/O space, the source address is located on a 1K-word boundary, so only the most significant six bits of the address are required. Therefore, when attempting to obtain the source address from I/O space, the bootloader reads a single byte from D7–D0, and the six most significant address bits are taken from the six most significant bits of this byte. The bootloader then reads the first locations starting at the source address and checks for a valid 8- or 16-bit boot, as indicated by finding a valid width selection and BRB (010AAh or 08AAh, respectively). If a valid boot selection is not found when reading the source address from I/O space, the bootloader then checks data space for the source address.

When initially attempting to obtain the source address from data memory location 0FFFFh, the bootloader requires a 16-bit address but has not yet identified whether the memory is 8- or 16-bits. Therefore, the bootloader first assumes that the memory is 16-bit, and reads a full 16-bit address from location 0FFFFh, and then reads the first word at this location to check for a valid 16-bit boot.

If a valid 16-bit boot is not indicated (by finding a valid width selection and BRB) the bootloader then assumes the memory is 8-bit, and attempts to obtain the source address by reading a byte from location 0FFFFh for the LSB of the source address and a byte from location 0FFFEh for the MSB of the source address. Then the bootloader reads the first locations at this source address, and tests for a valid 8-bit boot.

If a valid parallel mode boot selection is found, the bootloader transfers the code from data memory space to program memory space and begins execution of the downloaded code at the specified location. If a valid parallel boot mode selection is not found, the bootloader then checks for a valid TDM serial port boot, as described in the next section.

In all of the parallel boot modes, the bank-switch control register (BSCR)and the software wait-state register (SWWSR) are reconfigurable. Since the default configuration of the software wait-state register is for seven software wait-states, reconfiguring the SWWSR can allow faster bootloads if the external memory is faster than seven wait-states. Figure 2–4 shows the sequence of actions in the parallel boot mode.

Figure 2–5 shows the source program data stream for parallel boot mode using 8- or 16-bit word mode.

Note:

If a parallel boot is not desired, the data pin D0 of the ’C548/9 should be tied high, with a weak pullup, to avoid inadvertently booting from data or I/O space. Otherwise, some other method should be used to ensure that a valid BRB is not read unexpectedly.

Figure 2–4. Parallel Boot Mode Process

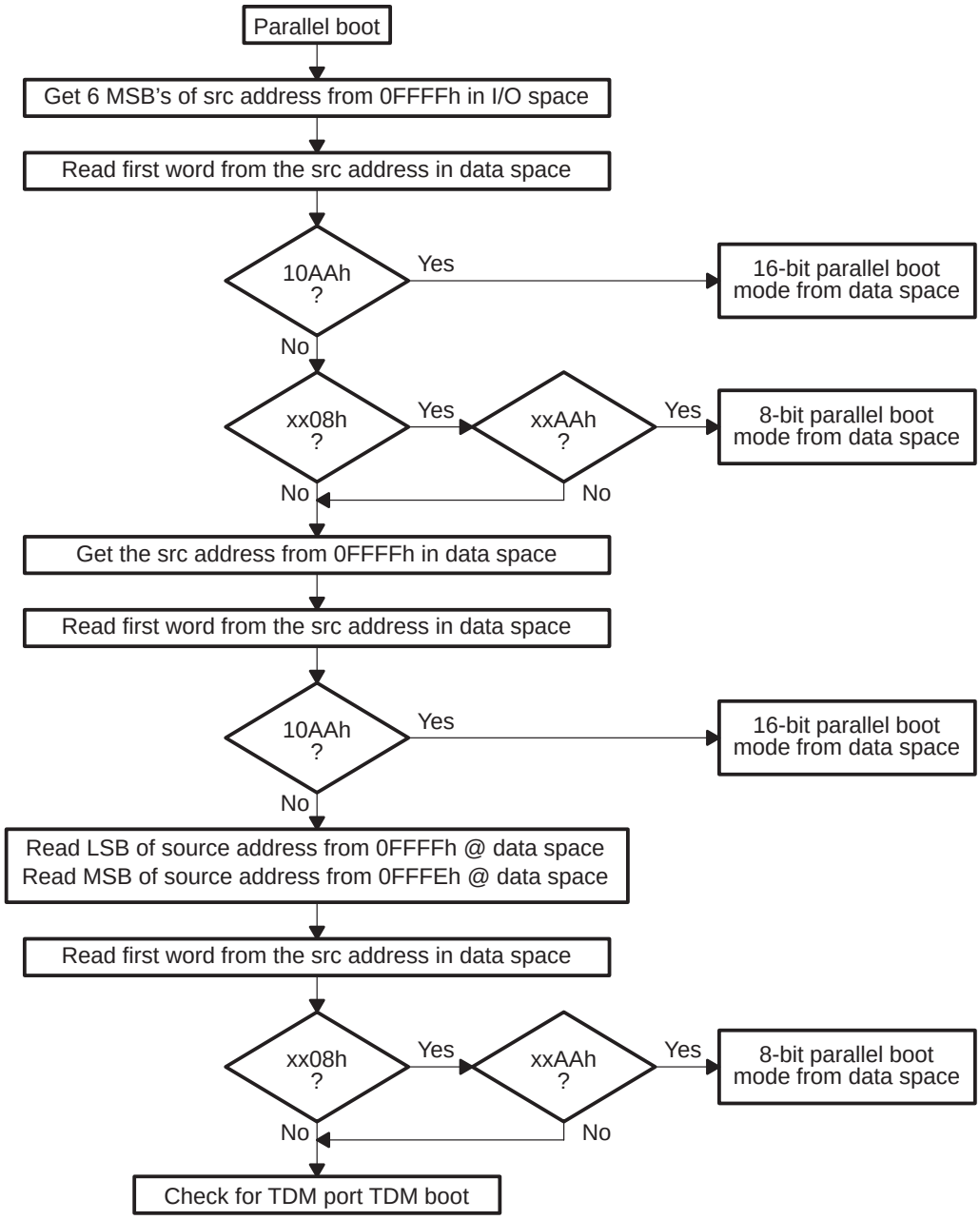


Figure 2–5. Source Program Data Stream for Parallel Boot in 8-,16-Bit-Word Mode

08AAh or 10AAh
Initialize value of $SWWSR_{16}$
Initialize value of $BSCR_{16}$
Entry point $(XPC)_7$
Entry point$(PC)_{16}$
Size of first section$_{16}$
Destination of first section $(XPC)_7$
Destination of first section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
Size of mth section$_{16}$
Destination of mth section $(XPC)_7$
Destination of mth section$(PC)_{16}$
Code word$(1)_{16}$
.
Code word$(N)_{16}$
0000h

Note: 0000h indicates the end of the program.

2.3.2.1 TDM Serial Boot Mode

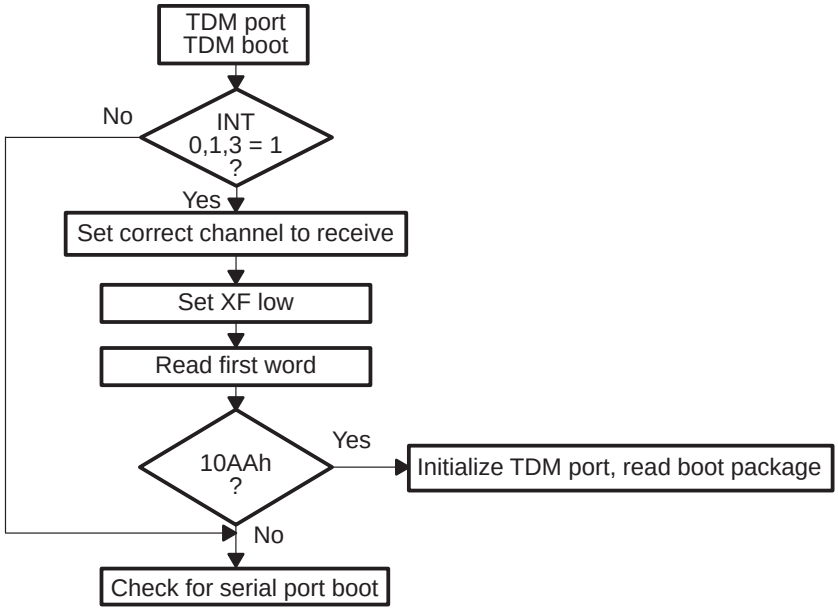
To check for TDM serial port boot, the bootloader first checks if one or more interrupt flags are set. The interrupt flags INT0, INT1, and INT3 determine the channel on which data will be received, as shown in Figure 2–6. Channel 1 is set as host channel to transmit data.

Figure 2–6. Channels Used for TDM Serial Boot Mode

[INT3, INT1, INT0]	[0,0,1]	[0,1,0]	[0,1,1]	[1,0,0]	[1,0,1]	[1,1,0]	[1,1,1]
Channel	2	3	4	5	6	7	8

The bootloader initializes the TDM receive/transmit address (TRTA) register with the correct value based on the interrupt flags and sets TFSX and TCLKX as inputs (MCM, TXM = 0). The bootloader then sets the XF pin low to indicate that the TDM port is ready to receive data. The processor then polls the RRDY bit in the serial port control (TSPC) register to read data from the TRCV register. Since the TDM mode only transmits and/or receives data in 16-bit mode, there is no 8-bit-mode boot. This means that if the first word that the TDM serial port receives is not 0x10AA, the processor skips the TDM serial boot mode and returns to check for standard serial boot modes. Figure 2–7 illustrates how the bootloader checks for a TDM serial boot.

Figure 2–7. TDM Serial Boot Mode Flow



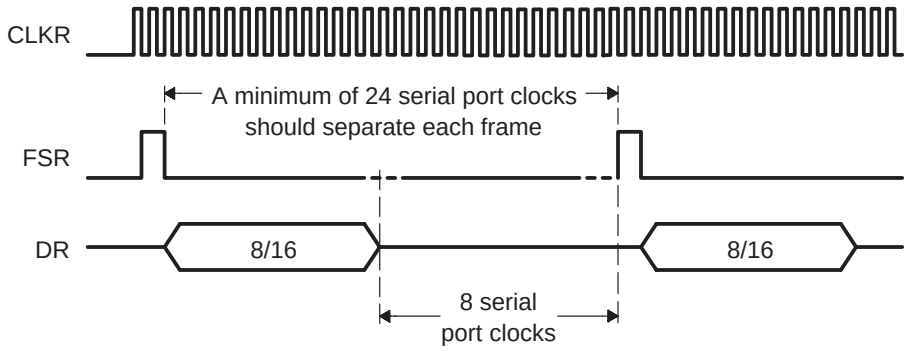
2.3.2.2 Standard Serial Boot Mode

The bootloader is capable of loading data in standard serial port mode from BSP1, the TDM port, or BSP0 (ABU mode is not supported). In standard serial boot mode, the bootloader initializes the serial port as a standard serial port (BXE = 0, TDM = 0). CLKX and FSX are configured as inputs, and burst mode is selected (FSM = 1, MCM/TXM = 0). The bootloader also sets the XF pin low

to indicate that the serial port is ready to receive data. The processor then polls the IFR to determine which serial port has data input (TRINT, BRINT0, or BRINT1).

In the standard serial boot mode, once the configuration words have been received, the serial ports are reconfigured. During this time the serial ports are reset. The external device should remain idle. No data should be sent. Figure 2–8 describes the timing conditions for the serial port boot mode.

Figure 2–8. Timing Conditions for Serial Port Boot Operation



The following conditions must be met in order to insure proper operation.

1) Serial port clocks should not exceed 1/8 clockout.

2) Delay time: At least eight serial port clocks (CLKR) are placed between the end of the current word and the next frame sync pulse.

These conditions provide the required delay for receiving consecutive words in the bootloader.

Note that the specified delay is really only required at times when the serial port is actually reset, which does not occur between every byte or word transferred. However, in a system, it may be more efficient to implement this delay between each transfer rather than implementing it at only the times at which the serial port is reset. The points at which the serial port is reset are shown in the flowcharts describing each of the serial boot modes.

Figures 2–9, 2–10, and 2–11 show the serial boot from the various serial ports supported. The bootloader then reads from the data receive register (BDRR or TRCV) to determine the data size (8 or 16 bits). The bootloader reads the initialized value of the serial port control registers to reinitialize the serial port.

Figure 2–9. Standard Serial Boot From BSP1 After BRINT = 1

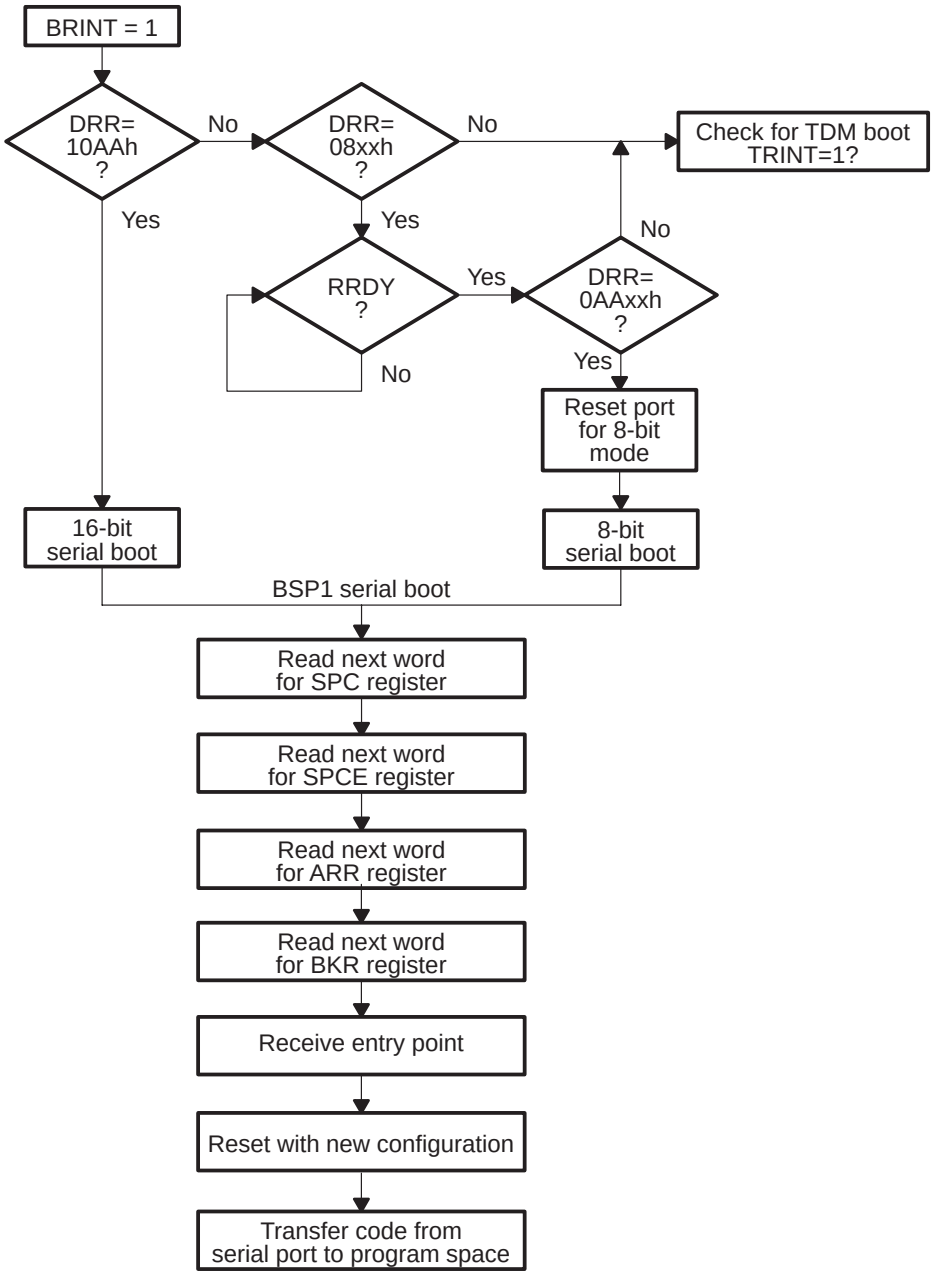


Figure 2–10. Standard Serial Boot in TDM Mode After TRINT = 1

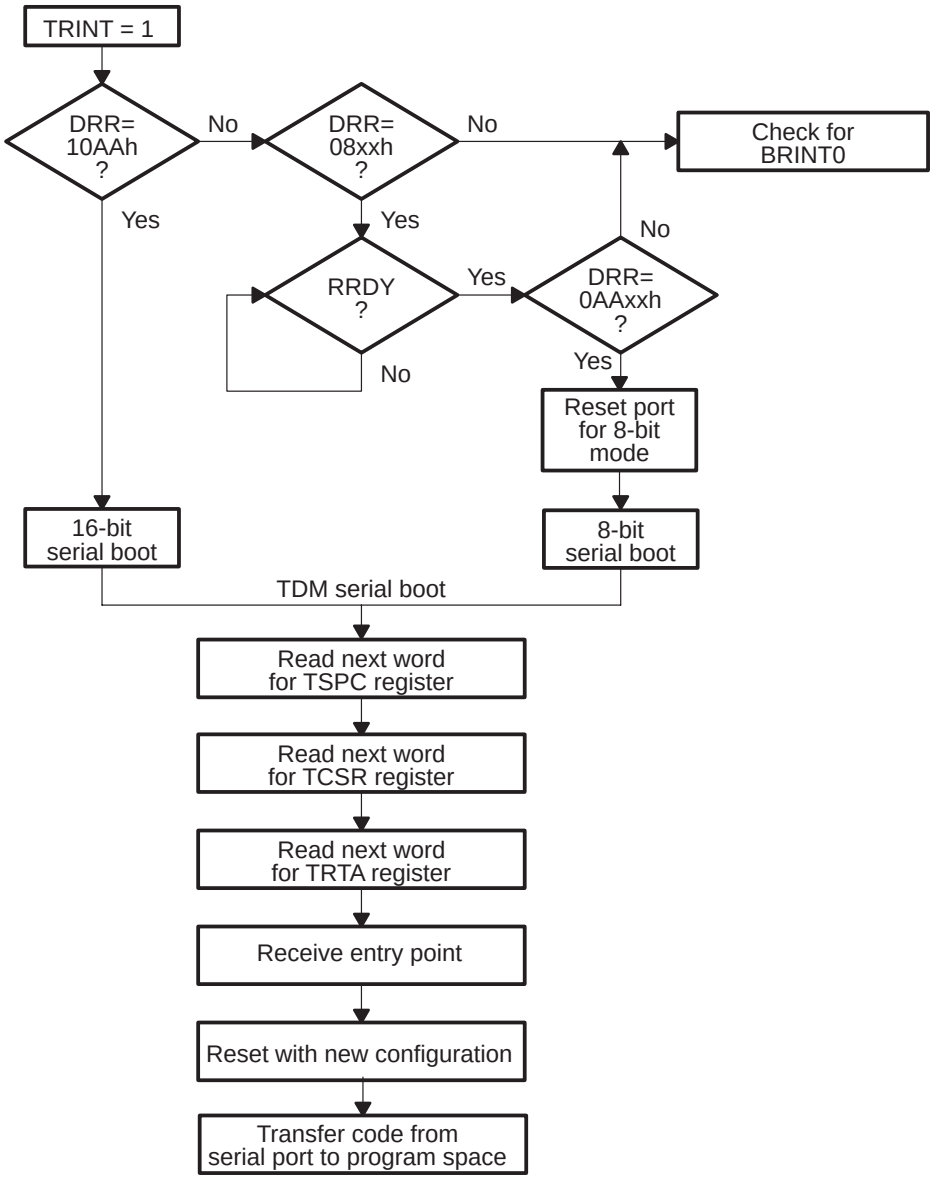
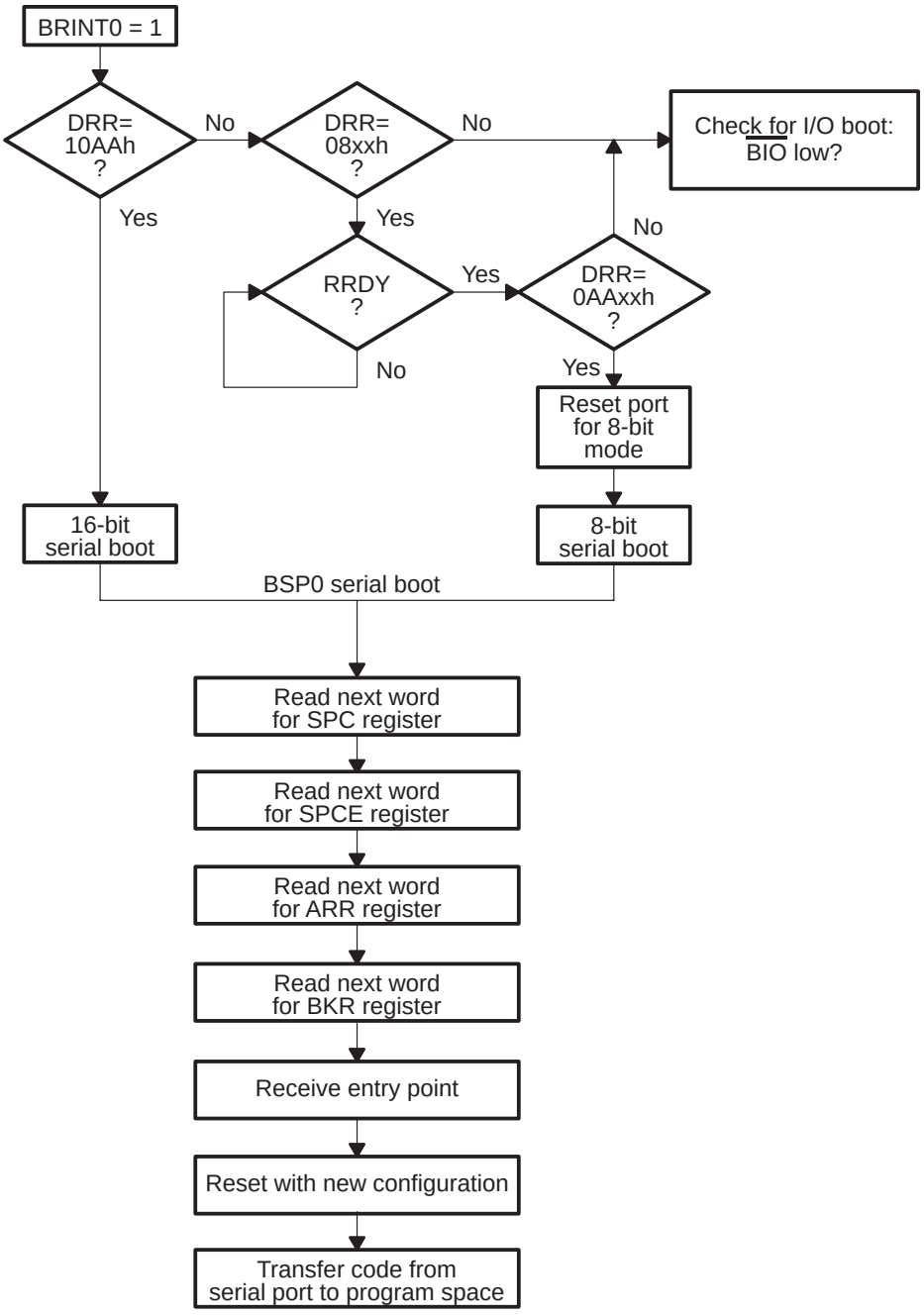


Figure 2–11. Standard Serial Boot From the BSP0 After BRINT0 = 1



Since the BSP and TDM serial ports have different control registers, the source program data stream for the TDM and standard serial boot modes are different. Figure 2–12 and Figure 2–13 show the data stream for these boots in 16-bit-word mode. For 8-bit mode, each word is stored in two memory locations with the MSB followed by the LSB.

Figure 2–12. Source Program Data Stream for 8/16-Bit BSP Boot Load in Standard Mode

08AAh or 10AAh
Initialize value of SPC_{16}
Initialize value of $SPCE_{16}$
Initialize value of ARR_{16}
Initialize value of BKR_{16}
Entry point $(XPC)_7$
Entry point$(PC)_{16}$
Size of first section$_{16}$
Destination of first section $(XPC)_7$
Destination of first section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
Size of second section$_{16}$
Destination of second section $(XPC)_7$
Destination of second section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
.
.
Size of mth section$_{16}$
Destination of mth section $(XPC)_7$
Destination of mth section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
0000h

Note: 0000h indicates the end of the program.

Figure 2–13. Source Program Data Stream for 16-Bit TDM Boot Mode

08AAh or 10AAh
Initialize value of $TSPC_{16}$
Initialize value of $TCSR_{16}$
Initialize value of $TRTA_{16}$
Entry point $(XPC)_7$
Entry $point_{16}$
Size of first $section_{16}$
Destination of first section $(XPC)_7$
Destination of first section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
Size of second $section_{16}$
Destination of second section $(XPC)_7$
Destination of second section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
.
.
Size of mth $section_{16}$
Destination of mth section $(XPC)_7$
Destination of mth section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
0000h

Note: 0000h indicates the end of the program.

2.3.3 I/O Boot Mode

The I/O boot mode is used to bootload using data transfers that occur at a slower rate than memory cycles, and the timing of which is controlled by a handshaking sequence with an external host device, rather than by the ’C548/9.

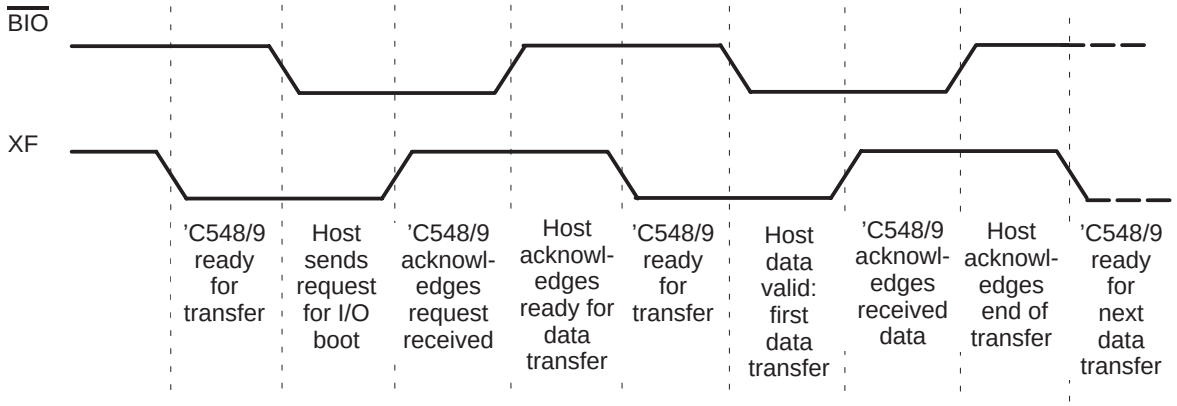
In the I/O boot mode, data is transferred between the DSP and the host via the ’C548/9 external parallel data bus, and the handshaking sequence is accomplished using the BIO and XF signals. XF is driven by the DSP to the host, and BIO is driven by the host to the DSP, allowing bidirectional handshaking to be implemented.

The handshake sequence begins with the DSP driving XF low, indicating that it is available for an I/O boot. The host responds by driving BIO low, following which the DSP drives XF high, and then the host responds back by driving BIO high again. This handshake sequence is used to accomplish two functions in the I/O boot mode: arbitration of the I/O boot mode itself, and the actual data transfers. Arbitration of the I/O boot mode is the first handshaking sequence that occurs.

Arbitration of the I/O boot mode is accomplished using the same handshake sequence that is used for data transfers, however, no specific data needs to be sent by the host at this time. In order to accomplish this arbitration, the DSP first drives XF low indicating that it is ready for communication with the host. The host should then drive BIO low, indicating a request for I/O boot mode. The DSP then drives XF high, indicating an acknowledgement of the request for I/O boot mode, following which, the host drives BIO back high, acknowledging its readiness for data transfers to begin.

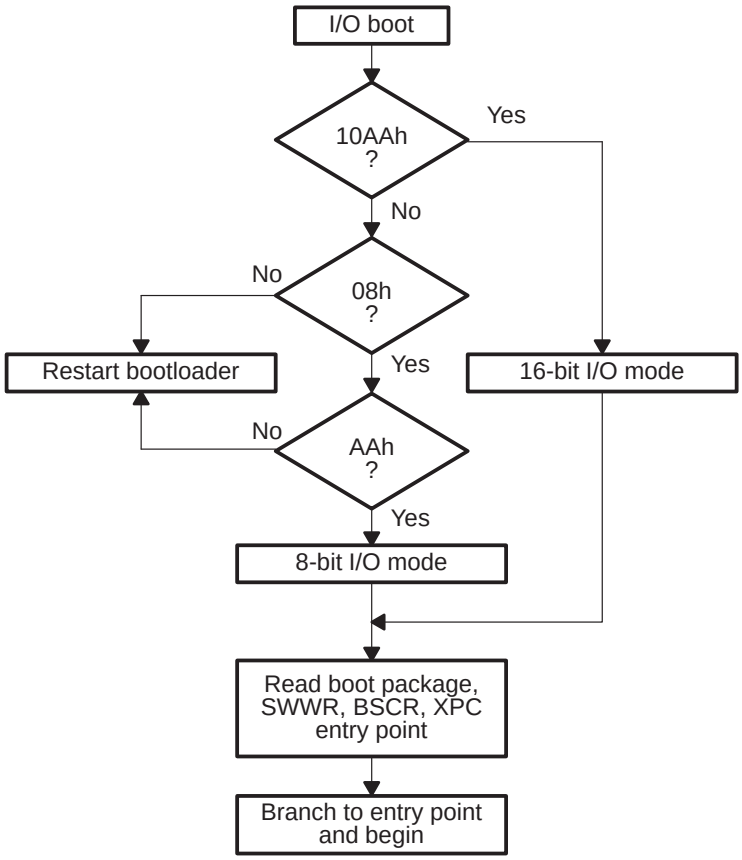
The data transfer portion of the sequence occurs with the same handshaking sequence as the I/O boot mode arbitration, starting with XF being driven low by the DSP, indicating that the DSP is ready for a data transfer to occur. The host then responds by driving BIO low. When BIO goes low, the DSP inputs the data from I/O address 0h, drives the XF pin high to indicate to the host that the data has been received, and writes the input data to the destination address in program memory. The ’C548/9 then waits for the BIO pin to be driven high and then low again by the external host for the next transfer. This sequence continues until the entire program is loaded. Figure 2–14 shows a timing diagram of the handshake protocol required to perform an I/O boot.

Figure 2–14. XF-$\overline{BIO}$ Handshake Protocol



The I/O boot mode supports both 8- and 16-bit memory widths. If the 8-bit transfer mode is selected, the lower eight data lines are read from I/O address 0h. The upper bytes on the data bus are ignored. The 'C548/C549 reads two 8-bit words to form a 16-bit word. The low byte of a 16-bit word follows the high byte. Figure 2–15 shows the events in an I/O boot.

Figure 2–15. I/O Boot Mode



For both 8-bit and 16-bit I/O modes, the first six 16-bit words received by the device must be as follows:

- Source memory width
- Initialized value of the software wait-state register
- Initialized value of the bank switch control register
- Size of the section
- Extended program counter (XPC) of the destination
- Destination address (PC)

A minimum delay of ten clock cycles is provided between the XF rising edge and the write operation to the destination address. This allows the host processor enough time to turn off its data buffers before the ’C548/C549 initiates a write operation (in case the destination is external memory). Note that the ’C548/9 only drives the external bus when XF is high.

Chapter 3

Building the Boot Table

To use the enhanced features of the ’C548/C549 bootloader, you must generate a boot table, which contains the complete data stream the bootloader needs. The boot table is generated by the hex conversion utility tool. The contents of the boot table vary, depending on the boot mode and the options selected when running the hex conversion utility.

Note:

You must use version 1.20 or higher of the ’C54x code generation tools to generate the proper boot table for the ’C548/C549. Previous versions of the code generation tools do not support the enhanced bootloader options for these devices and may produce a version of the boot table intended for earlier ’C54x devices without generating warnings or errors. Contact Texas Instruments at for information about upgrades to earlier versions of the code generation tools.

To build the ’C548/C549 boot table, follow these steps:

Step 1: **Assemble (or compile) the code using the –v548 assembler option.** This option marks the object files produced by the assembler specifically for the ’C548 or ’C549. The hex conversion utility uses this information to generate the correct format for the boot table. If this option is not included during assembly, the hex conversion utility may produce a version of the boot table intended for earlier ‘C54x devices without generating warnings or errors.

Step 2: **Link the file.** Each block of the boot table data corresponds to an initialized section in the COFF file. Uninitialized sections are not converted by the hex conversion utility.

Step 3: **Run the hex conversion utility.** Choose the appropriate options for the desired boot mode and run the hex conversion utility to convert the COFF file produced by the linker into a boot table. See the *TMS320C54x Assembly Language Tools User’s Guide* for a detailed description of the procedure for generating a boot table and using the options.

Appendix A

Code Listings

This appendix provides the code listings for the bootloader and for the additional features of the ’C548/C549 on-chip ROM. For more information about these features, see section 1.2, *On-Chip ROM Description*.

Topic **Page**

A.1 Bootloader Code

```
*********************************************************
***   Bootloader software version N0.   : 1.0           ***
***   Last revision date                : 10/23/1996    ***
***   Author                            : J. Chyan      ***
*********************************************************
*********************************************************
**                                                      **
**   Boot Loader Program                                **
**                                                      **
**   This code segment sets up and executes boot loader **
**   code based upon data saved in data memory          **
**                                                      **
**   WRITTEN BY:  Jason Chyan                           **
**   DATE:        06/06/96                              **
**                                                      **
** Revision History                                     **
** 1.0 Change HPI boot from c542 boot loader            **
**     Implement Paralle Boot (EPROM)       YGC 06/07/96 **
**                                                      **
** 1.1 Implement Serial Port Boot           YGC 06/17/96 **
** 1.2 Implement I/O Boot                   YGC 06/20/96 **
** 1.3 Add A-law, u-law, sinwave and                    **
**     interrupt vectors table              YGC 06/25/96 **
** 1.4 Registers reprogrammable in I/O mode YGC 06/25/96 **
** 1.5 Implement TDM mode & ABSP mode       YGC 10/23/96 **
** 1.6 Fix the SP (steak point) bug         YGC 10/24/96 **
** 1.7 Fix the BSP bug                      YGC 01/16/96 **
** 1.8 Fix the dest. address in par8/16     TNG 03/24/96 **
** 1.9 Fix the bugs in BSP/ABU mode         TNG 08/28/97 **
** 1.91 Fix the hi byte bug in par8 mode    TNG 12/10/97 **
* 1.92 Fix the par8 mode bug               PMJ2 11/09/98 **
*********************************************************
         .title   "bootc54LP"
*********************************************************
```

```
*     symbol definitions
*******************************************************
         .mnolist
         .def    boot
         .def    endboot
         .def    bootend
         .def    dest
         .def    src
         .def    lngth
         .def    hbyte
*        .ref    boota           ; reserved for ROM Code customer
boota    .set   0184h           ; Arbitrary ref (for USR bootcode)
*    Conditional Assembly Flags
HPIB     .set   1
LC548    .set   1
*
pa0      .set   0h              ; port address 0h for i/o boot load
brs      .set   60h             ; boot routine select (configuration word)
xentry   .set   61h             ; XPC of entry point
entry    .set   62h             ; entry point
hbyte    .set   63h             ; high byte of 8–bit serial word
p8word   .set   64h             ; concatenator for 8–bit memory load
src      .set   65h             ; source address
dest     .set   66h             ; destination address (dmov from above)
lngth    .set   67h             ; code length
temp0    .set   68h             ; temporary register0
temp1    .set   69h             ; temporary register1
temp2    .set   6ah             ; temporary register2
temp3    .set   6bh             ; temporary register3
nmintv   .set   6ch             ; non–maskable interrupt vector
sp_ifr   .set   6dh             ; SP IFR temp reg
*    MMR definition for c54xlp CPU register
**
ifr      .set   01h
st1      .set   07h
BL       .set   0bh
```

```
brc      .set   1ah
pmst     .set   1dh
*    MMR definition for c54xlp peripherals
**
*-------------    BSP0  ----------------------
drr0     .set   20h             ; Data Receive Register
dxr0     .set   21h             ; Data Transmit Register
spc0     .set   22h             ; Serial Port Control Register
spce0    .set   23h             ; BSP Control Extension Register
axr0     .set   38h             ; ABU(Auto Buffering Unit) Transmit Address Regis
ter
bkx0     .set   39h             ; ABU Transmit Buffer Size Register
arr0     .set   3ah             ; ABU Receive Address Register
bkr0     .set   3bh             ; ABU Receive Buffer Size Register
*-------------    BSP1  ----------------------
drr1     .set   40h             ; Data Receive Register
dxr1     .set   41h             ; Data Transmit Register
spc1     .set   42h             ; Serial Port Control Register
spce1    .set   43h             ; BSP Control Extension Register
axr1     .set   3ch             ; ABU(Auto Buffering Unit) Transmit Address Regis
ter
bkx1     .set   3dh             ; ABU Transmit Buffer Size Register
arr1     .set   3eh             ; ABU Receive Address Register
bkr1     .set   3fh             ; ABU Receive Buffer Size Register
*-------------    TDM   ----------------------
trcv     .set   30h             ; Data Receive Register
tdxr     .set   31h             ; Data Transmit Register
tspc     .set   32h             ; Serial Port Control Register
tcsr     .set   33h             ; TDM Channel Select Register
trta     .set   34h             ; TDM Receive/Transmit Register
trad     .set   35h             ; TDM Receive Address Register
swwsr    .set   28h             ; SoftWare Wait State Register
bscr     .set   29h             ; Bank Switch Control Register
hpic     .set   2ch             ; HPI Control Register
*       Bit equates
b0       .set   00h
```

```
b4       .set   04h
b8       .set   08h
bc       .set   0ch
b10      .set   010h
b14      .set   014h
b20      .set   020h
b24      .set   024h
b30      .set   030h
b34      .set   034h
hpiram   .set   01000h
int2msk  .set   004h            ; INT2_ bit position on IFR
*********************************************************
*     main program starts here
*********************************************************
         .sect ”bootload”
boot
         ssbx    intm            ; disable all interrupts
         ld      #0, dp
         stm     #0FFFFh,@ifr    ; clear IFR flag
         orm     #02b00h, @st1   ; xf=1, hm=0, intm=1, ovm=1, sxm=1
         orm     #020h, @pmst    ; ovly=1
         stm     #07fffh, swwsr  ; 7 wait states for P_,D_, and I_ spaces
         stm     #0f800h, bscr   ; full bank switching
         stm     #0007fh, sp
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*    HPI boot, simply branch to HPI RAM
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         .if     HPIB
         stm     #01010b, hpic   ; Send HINT_ low
         rpt     #16             ; wait 20 clockout cycles
         nop                     ; before check INT2
         st      #0,@xentry      ; XPC = 0
         bitf    @ifr, #int2msk  ; Check if INT2_ flag is set
                                 ; This TEST MUST BE >= 30 cycles from boot?
         stm     #int2msk, ifr   ; Clear INT2_ bit in ifr if INT2_ latched
         bcd     endboot, tc     ; If yes, branch to HPI RAM
```

```
         st      #hpiram, @entry
         .endif
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*    Check Parallel Boot
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         stm     #0h, @xentry    ; initialize the entry point
         stm     #boot, @entry   ;
         portr   #0ffffh, @brs   ; brs <-- boot load configuration word
         ld      @brs, 8, a      ; get boot value in acc AL
         and     #0fc00h, a      ; throw away 2 LSBs
         stl     a, @src         ; save as source address
         mvdk    @src, ar1       ; ar1 points at source memory (Data)
         ld      *ar1+, a        ; load accumulator A with BRW
         sub     #10AAh, a, b    ; check 16-bit Boot?
         bc      par16, beq      ; a=010AAh
         and     #0ffh, a        ; check acc AL = 08
         sub     #8h, a, b       ; check 8-bit Boot?
         bc      chk_data, bneq  ; a=08xxh
         ld      *ar1+, a        ; 8-bit mode, LSB
         and     #0ffh, a        ; check acc AL = AAh
         sub     #0AAh, a        ; LSB = 0AAh?
         bc      par08, aeq      ; 8-bit Parallel Boot
chk_data stm     #0FFFFh, ar1    ; check data memory 0xFFFF
         nop                     ; prevent pipeline conflict
         nop                     ;
         ld      *ar1+, a        ; load accumulator A with BRW
         stlm    a, ar1          ; ar1 point at source memory (Data)
         nop                     ; prevent posibble pipeline conflic
         nop                     ;
         ld      *ar1+, a        ; load acculator A with BRW
         sub     #10AAh, a, b    ; check 16-bit Boot?
         bc      par16, beq      ; a=010AAh
         stm     #0FFFFh, ar1    ; check data memory 0xFFFF & 0xFFFE
         nop                     ; prevent possible pipeline conflic
         nop                     ;
         ldu     *ar1-, a       ; acc A <-- source address
```

```
         and     #0FFh, a       ; 0 the high byte
         add     *ar1, 8, a      ;
         stlm    a, ar1          ;
         nop                     ; prvent possible pipeline conflic
         nop                     ;
         ld      *ar1+, a        ; load accumulator A with BRW
         and     #0ffh, a        ; check acc AL = 08h
         sub     #8h, a, b       ; check 8–bit Boot?
         bc      ser_ini, bneq   ; acc A = 08xxh
         ld      *ar1+, a        ; 8–bit mode, LSB
         and     #0ffh, a        ; check acc AL = AAh
         sub     #0AAh, a        ; LSB = 0AAh?
         bc      par08, aeq      ; 8–bit Parallel Boot
* * * * * * * * * * * * * * * * * * * * * *
*       Check TDM mode boot first       *
* * * * * * * * * * * * * * * * * * * * * *
ser_ini
         ld      ifr, a          ; check INT3 flag
         and     #103h, a        ;
         cc      TDMSP, aneq     ;
* * * * * * * * * * * * * * * * * * * * * *
*       Initialize serial port          *
* * * * * * * * * * * * * * * * * * * * * *
          stm     #0008h, spc1    ; configure sport (BSP0) and put in reset
         stm     #0003h, spce1  ; CLKKV=3,FSP=CLKP=FE=FIG=PCM=BXE=HLTX=BRE=HLTR=0
          orm      #00c0h, spc1   ; take sport out of reset
          stm      #0008h, spc0   ; configure sport (BSP1) and put in reset
         stm     #0003h, spce0   ; CLKKV=3,FSP=CLKP=FE=FIG=PCM=BXE=HLTX=BRE=HLTR=0
          orm      #00c0h, spc0    ; take sport out of reset
          stm      #0008h, tspc    ; configure sport (TDM) and put in reset
                                  ; CLKKV=3,FSP=CLKP=FE=FIG=PCM=BXE=HLTX=BRE=HLTR=0
          orm      #00c0h, tspc    ; take sport out of reset
         rsbx    xf              ; signal ready–to–receive
chk_ser  bitf    ifr, #400h      ; check BRINT1 flag
         cc      BSP1, tc        ;
         bitf    ifr, #40h       ; check TRNT flag
```

```
         cc      TDM, tc         ;
         bitf    ifr, #10h       ; check BRINT0 flag
         cc      BSP0, tc        ;
         bc      pasyini, bio    ; no sport int. go to I/O boot
         b       chk_ser
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       Warm–boot, simply branch to source address      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
endboot
         ldu     @entry,a        ; branch to the entry point
         add     @xentry,16,a    ;
         .if     LC548           ; if all lc54xlp has XIO
         fbacc   a               ; take the .if command out
         .else
         bacc    a
         .endif
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from 8–bit memory, MS byte first       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
par08
         ld      *ar1+, 8, a     ; load accumulator A MSB of SWWSR value
         mvdk    *ar1+, ar3      ; ar3   <–– junkbyte.low byte
         andm    #0ffh, @ar3     ; ar3   <–– low byte
         or      @ar3, a         ; acc A <–– high byte.low byte
         stlm    a,swwsr         ; stor A to SWWSR
         ld      *ar1+, 8, a     ; load accumulator A MSB of BSCR value
         mvdk    *ar1+, ar3      ; ar3   <–– junkbyte.low byte
         andm    #0ffh, @ar3     ; ar3   <–– low byte
         or      @ar3, a         ; acc A <–– high byte.low byte
         and     #0FFFEh,a       ; ensure EXIO bit is off
         stlm    a,bscr          ; stor A to BSCR
         ld      *ar1+, 8, a     ; load accumulator A MSB of XPC of entry
         mvdk    *ar1+, ar3      ; ar3   <–– junkbyte.low byte
         andm    #0ffh, @ar3     ; ar3   <–– low byte
         or      @ar3, a         ; acc A <–– high byte.low byte
         stl     a,@xentry       ; stor A to xentry
```

```
        ld      *ar1+, 8, a     ; load accumulator A MSB of entry
        mvdk    *ar1+, ar3      ; ar3   <-- junkbyte.low byte
        andm    #0ffh, @ar3     ; ar3   <-- low byte
        or      @ar3, a         ; acc A <-- high byte.low byte
        stl     a,@entry        ; stor A to entry
par08_1 ld      *ar1+, 8, a     ; get number of 16-bit words
        and     #0FF00h,a       ; Clear the guard bits and keep low accum (1.92)
        mvdk    *ar1+, ar3      ; ar3   <-- junkbyte.low byte
        andm    #0ffh, @ar3     ; ar3   <-- low byte
        or      @ar3, a         ; acc A <-- high byte.low byte
        bcd     endboot,aeq     ; section size =0 indicate boot end
        sub     #1,a,b          ; brc = section size - 1
        stlm    b, brc          ; update block repeat counter register
        ld      *ar1+, 8, a     ; get XPC of destiantion
        mvdk    *ar1+, ar3      ; ar3   <-- junkbyte.low byte
        andm    #0ffh, @ar3     ; ar3   <-- low byte
        or      @ar3, a         ; acc A <-- high byte.low byte
        stl     a,@dest         ; @dest <-- XPC
        ld      *ar1+, 8, a     ; get address of destiantion
************ Bug fix ***************************************************
        and     #0ff00h,a       ;force AG, AH to zero for correct calculation
                                ;of the 23-bit destination address.
                                 ;(10/14/99 BCT)
*******************************************************************************
        mvdk    *ar1+, ar3      ; ar3   <-- junkbyte.low byte
        andm    #0ffh, @ar3     ; ar3   <-- low byte
        or      @ar3, a         ; acc A <-- high byte.low byte
        stlm    a,ar2           ; ar2 <-- destination address
        rptb    xfr08-1
        ld      *ar1+, 8, a     ; acc A <-- high byte
        mvdk    *ar1+, ar3      ; ar3   <-- junkbyte.low byte
        andm    #0ffh, @ar3     ; ar3   <-- low byte
        or      @ar3, a         ; acc A <-- high byte.low byte
        stl     a, @p8word
        ldu     @ar2,a         ; load XPC to acc AH
        add     @dest,16,a      ; acc A <-- destination address
```

```
         nop                   ; 10 cycles b/w read & write
         writa   @p8word        ; write object data to destination
         add     #1, a
         stlm    a, ar2         ; update destination address
xfr08
         b       par08_1
* * * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from 16-bit memory             *
* * * * * * * * * * * * * * * * * * * * * * * * *
par16
         ld      *ar1+, a       ; load accumulator A with initialize
         stlm    a, @swwsr      ; value of SWWR and stored in SWWSR
         ld      *ar1+, a       ; load accumulator A with initialize
         and     #0FFFEh,a      ; ensure EXIO bit is off
         stlm    a, @bscr       ; value of BSCR and stored in BSCR
         ld      *ar1+, a       ; load accumulator A with initialize
         stl     a, @xentry     ; value of XPC of entry point
         ld      *ar1+, a       ; load accumulator A with initialize
         stl     a, @entry      ; value of entry point
par16_1  ld      *ar1+,a        ; load the size of section to A
         bcd     endboot,aeq    ; section size =0 indicate boot end
         sub     #1,a,b         ; brc = section size – 1
         stlm    b, brc         ; current destination in block repeat
         ld      *ar1+,a        ; get the XPC of destination
         stl     a,@dest        ; store XPC at data memory @dest
         ldu     *ar1+,a        ; get address of destiantion in A(0..22)
         stlm    a,ar2          ; store dest address at ar2
         add     @dest,16,a     ; acc A <––– destination address
         rptb    xfr16–1
         mvdk    *ar1+, ar3     ; read object data
         ldu     @ar2,a         ; get previous destination address
         add     @dest,16,a     ; get previous destination address
         add     #1, a          ; these instructions also
         stlm    a, ar2         ; serve the purpose of inserting
         sub     #1, a          ; 10 cycles b/w read & write
         writa   @ar3           ; write object data to destination
```

```
xfr16
         b       par16_1
ser_in
         rsbx    tc              ; clear flag
         bcd     $, ntc          ; begin receive data routine
         bitf    *ar2, #0400h    ; if rrdy = 1
         ret
DBsreadA
         call    ser_in          ; call SP input sub
         ld      *ar1, 8, a      ; acc A <-- junkbyte.high byte
         and     #0ff00h, a      ; acc A <-- high.byte
         stl     a, @hbyte       ; save high byte
         call    ser_in          ; call SP input sub
         ldu     *ar1, a         ; acc A <-- junkbyte.low byte
         and     #0ffh, a        ; acc A <-- low byte
         or      @hbyte, a       ; acc A <-- high byte.low byte
         ret
auto_in  rsbx    tc              ; clear flag
         bcd     $, ntc          ; begin receive data routine
         bitt    ifr             ; if BRINTx = 1
         ld      *ar3, b
         and     #4000h, b
         rc     beq
         mvdk    *ar4, @ar7
         ret
* * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from BSP serial port       *
* * * * * * * * * * * * * * * * * * * * * * *
BSP0
         stm     #800h,ar0       ; ar0 <-- start address of BSP0 RAM
         stm     #drr0,ar1       ; ar1 <-- drr0
         stm     #spc0,ar2       ; ar2 <-- spc0
         stm     #spce0,ar3      ; ar3 <-- spce0
         stm     #arr0,ar4       ; ar4 <-- arr0
         stm     #bkr0,ar5       ; ar5 <-- bkr0
         stm     #0bh,@sp_ifr    ; temp reg for IFR
```

```
        stm     #010h, @ifr     ; clear BRINT0 flag
        b       BSP_ini         ; check BSP
BSP1
        stm     #1800h,ar0      ; ar0 <-- start address of BSP1 RAM
        stm     #drr1,ar1       ; ar1 <-- drr0
        stm     #spc1,ar2       ; ar2 <-- spc0
        stm     #spce1,ar3      ; ar3 <-- spce0
        stm     #arr1,ar4       ; ar4 <-- arr0
        stm     #bkr1,ar5       ; ar5 <-- bkr0
        stm     #005h,@sp_ifr   ; temp reg for IFR
        stm     #400h, @ifr     ; clear BRINT1 flag
BSP_ini
        ld      @sp_ifr, T      ; TREG <-- bit code for BITT test
        ldm     *ar1, a         ; acc A <-- DRR
        sub     #10AAh, a, b    ; acc A = 0x10AA ?
        bc      bser_16, beq    ; 16-bit serial mode
        and     #0FF00h, a
        sub     #800h, a        ; acc A = 0x8xx
        rc      aneq
        call    ser_in          ; call SP input sub
        ldm     *ar1,  a        ; acc A <-- DRR
        and     #0FF00h, a
        sub     #0AA00h, a      ; acc A = 0xaaxx
        and     #0FFFFh, a
        rc      aneq
bser_08
        andm    #0ff3fh, *ar2   ; put SP (BSP0) in reset
        orm     #000ch, *ar2    ; configure sport (BSP0)
        orm     #0080h, *ar2    ; take sport receiver out of reset
        call    DBsreadA        ; call SP double read byte from DR
        stl     a, @temp0       ; save SPC value in temp0
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @temp1       ; save SPCE value in temp1
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @temp2       ; save ARR value in temp2
        call    DBsreadA        ; call SP double read byte from DRR
```

```
        stl     a, @temp3       ; save BKR value in temp3
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @xentry      ; save XPC of entry point
        call    DBsreadA        ; call SP double read byte from DRR
        rsbx    tc              ; clear flag
        bitf    @temp1, #2000h  ; if bre = 1, set TC bit
        bcd     BSP_08, tc      ; if TC = 1, begin receive data routine
        stl     a, @entry       ; save entry point
SP08
        ld      @temp0, a       ; load SPC to acc A
        and     #0FF3Fh, a      ; clear rrst & xrst bits
        stlm    a, *ar2         ; reset serial ports
        ldu     @temp1, a       ; reinitialize SPCE/TCSR of SP
        stlm    a, *ar3
        orm     #00c0h,*ar2     ; take trax and recv out of reset
SP08_1  call    DBsreadA        ; call SP double read byte from DRR
                                ; read section size
        bcd     endboot,aeq     ; section size =0 indicate boot end
        sub     #1,a,b          ; brc = section size – 1
        stlm    b, brc          ; update block repeat counter register
        call    DBsreadA        ; call SP double read byte from DRR
                                ; read XPC of destination address
        stl     a,@dest         ; XPC of destination
        call    DBsreadA        ; call SP double read byte from DRR
                                ; read destination address
        add     @dest, 16, a    ;
        rptb    sfxr08–1
         call    ser_in         ; call SP input sub
         ld      *ar1, 8, b     ; acc B <–– junkbyte.high byte
         and     #0ff00h, b     ; acc B <–– high.byte
         stl     b, @hbyte      ; save high byte
         call    ser_in         ; call SP input sub
         ldu     *ar1, b        ; acc B <–– junkbyte.low byte
         and     #0ffh, b       ; acc B <–– low byte
         or      @hbyte, b      ; acc B <–– high byte.low byte
         writa   @BL            ; [acc A] <–– acc BL
```

```
          add     #1, a          ; increment dest add
sfxr08
         b        SP08_1         ; check next section
****************************************
*        BSP0 16-bit mode
****************************************
bser_16
         call    ser_in          ; call SP input sub
         mvdk    *ar1, temp0     ; temp0 <-- drr0 (SPC0)
         call    ser_in          ; call SP input sub
         mvdk    *ar1, temp1     ; temp1 <-- drr0 (SPCE0)
         call    ser_in          ; call SP input sub
         mvdk    *ar1, temp2     ; temp2 <-- drr0 (ARR)
         call    ser_in          ; call SP input sub
         mvdk    *ar1, temp3     ; temp3 <-- drr0 (BKR)
         call    ser_in          ; call SP input sub
         mvdk    *ar1, xentry    ; xentry <-- drr0 (XPC of entry point)
         call    ser_in          ; call SP input sub
         rsbx    tc              ; clear flag
         bitf    @temp1,#2000h   ; if bre = 1, then set TC bit
         bcd     BSP_16, tc      ; if TC = 1, begin recive data routine
         mvdk    *ar1, entry     ; temp0 <-- drr0 (entry point)
SP16
         ld      @temp0, a       ; load SPC to acc A
         and     #0FF3Fh, a      ; clear rrst & xrst bits
         stlm    a, *ar2         ; reset serial ports
         ldu     @temp1, a       ; reinitialize SPCE/TCSR of SP
         stlm    a, *ar3
         ldu     @temp2, a       ; reinitialize ARR/TRTA of SP
         stlm    a, *ar4
         orm     #00c0h,*ar2     ; take trax and recv out of reset
SP16_1   call    ser_in          ; call SP input sub
         ldu     *ar1, a         ; acc A <-- drr0 (section size)
         bcd     endboot,aeq     ; section size =0 indicate boot end
         sub     #1,a,b          ; brc = section size - 1
         stlm    b, brc          ; update block repeat counter register
```

```
        call    ser_in          ; call SP input sub
        mvdk    *ar1, dest      ; xentry <-- drr0 (XPC of dest)
        call    ser_in          ; call SP input sub
        ldu     *ar1, a         ; acc A <-- destination addr
        add     @dest, 16, a    ;
        rptb    sfxr16-1
         call   ser_in          ; call SP input sub
         ldu    *ar1, b         ; acc B <-- drr0 (input dada)
         writa  @BL             ; [acc A] <-- acc BL
         add    #1, a           ; increment dest add
sfxr16
        b       SP16_1          ; check next section
BSP_08
        ld      @temp0, a       ; load SPC to acc A
        and     #0FF3Fh, a      ; clear rrst & xrst bits
        stlm    a, *ar2         ; reset serial ports
        ldu     @temp1, a       ; reinitialize SPCE/TCSR of SP
        stlm    a, *ar3
        ldu     @temp3, a       ; reinitialize BKR of SP
        and     #0fffch, a      ; make buffer size be multiper of 4
        stlm    a, *ar5         ;
        ldu     @temp2, a       ; reinitialize ARR/TRTA of SP
        stlm    a, *ar4
        orm     #00c0h,*ar2     ; take trax and recv out of reset
        stlm    a, @ar7         ; AR7 <-- starting address
        add     *ar5, -1, a     ; acc A <--- address at half of buff
        add     #1, a
        stl     a, @ar0         ; AR0 <-- acc A
        call    auto_in
N_sec_08
        rsbx    tc              ; clear flag
        ld      *ar7+, 8, a     ; acc A <-- junkbyte.high byte
        and     #0ff00h, a      ; acc A <-- high.byte
        stl     a, @hbyte       ; save high byte
        ldu     *ar7+, a        ; acc A <-- junkbyte.low byte
        and     #0ffh, a        ; acc A <-- low byte
```

```
or      @hbyte, a       ; acc A <-- high byte.low byte
bcd     endboot,aeq     ; section size =0 indicate boot end
orm     #8000h,*ar3     ; halt receive
sub     #1,a,b          ; repeat = section size – 1
cmpr    0, ar7
stlm    b, @ar6         ; store block size to AR6
cc      auto_in, tc
rsbx    tc
ld      *ar7+, 8, a     ; acc A <-- junkbyte.high byte
and     #0ff00h, a      ; acc A <-- high.byte
stl     a, @hbyte       ; save high byte
ldu     *ar7+, a        ; acc A <-- junkbyte.low byte
and     #0ffh, a        ; acc A <-- low byte
or      @hbyte, a       ; acc A <-- high byte.low byte
cmpr    0, ar7
stl     a, @dest        ; store XPC of detination add
cc      auto_in, tc
rsbx    tc
ld      *ar7+, 8, a     ; acc A <-- junkbyte.high byte
and     #0ff00h, a      ; acc A <-- high.byte
stl     a, @hbyte       ; save high byte
ldu     *ar7+, a        ; acc A <-- junkbyte.low byte
and     #0ffh, a        ; acc A <-- low byte
or      @hbyte, a       ; acc A <-- high byte.low byte
cmpr    0, ar7
add     @dest, 16, a    ; acc A <-- XPC + dest add
cc      auto_in, tc
rsbx    tc
ld      *ar7+, 8, b     ; acc A <-- junkbyte.high byte
and     #0ff00h, b      ; acc A <-- high.byte
stl     a, @hbyte       ; save high byte
ldu     *ar7+, b        ; acc A <-- junkbyte.low byte
and     #0ffh, b        ; acc A <-- low byte
or      @hbyte, b       ; acc A <-- high byte.low byte
writa   @BL
cmpr    0, ar7
```

```
        add     #1, a
        cc      auto_in, tc
        banz    N_sec_08, *ar6– ;
BSP_16
        ld      @temp0, a       ; load SPC to acc A
        and     #0FF3Fh, a      ; clear rrst & xrst bits
        stlm    a, *ar2         ; reset serial ports
        ldu     @temp1, a       ; reinitialize SPCE/TCSR of SP
        stlm    a, *ar3
        ldu     @temp3, a       ; reinitialize BKR of SP
        and      #0fffeh, a     ; make buffer size be multiple of 2
        stlm    a, *ar5         ;
        ldu     @temp2, a       ; reinitialize ARR/TRTA of SPd
        stlm    a, *ar4
        orm     #00c0h,*ar2     ; take trax and recv out of reset
        stlm    a, @ar7         ; AR7 <–– starting address
        add     *ar5, –1, a     ; acc A <––– address at half of buff
        add     #1, a           ;
        stl     a, @ar0         ; AR0 <–– acc A
        call    auto_in
N_sec_16
        rsbx    tc              ; clear flag
        ldu     *ar7+, a        ; acc A <–– section size
        bcd     endboot,aeq     ; section size =0 indicate boot end
        orm     #8000h,*ar3     ; halt receive
        sub     #1,a,b          ; repeat = section size – 1
        cmpr    0, ar7
        stlm    b, @ar6         ; store block size to AR6
        cc      auto_in, tc
        rsbx    tc
        cmpr    0, ar7
        mvdk    *ar7+, @dest    ; store XPC of detination add
        cc      auto_in, tc
        rsbx    tc
        ldu     *ar7+, a        ; acc A <–– dest addr
        cmpr    0, ar7
```

```
         add     @dest, 16, a    ; acc A <-- XPC + dest add
         cc      auto_in, tc
         rsbx    tc
         ldu     *ar7+, b        ; acc A <-- code
         writa   @BL
         cmpr    0, ar7
         ccd     auto_in, tc
         add     #1, a
         banz    N_sec_16, *ar6- ;
* * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from TDM serial port       *
* * * * * * * * * * * * * * * * * * * * * * * *
TDM
         stm     #0800h,ar0      ; ar0 <-- start address of BSP0 RAM
         stm     #trcv,ar1       ; ar1 <-- trcv
         stm     #tspc,ar2       ; ar2 <-- tspc
         stm     #tcsr,ar3       ; ar3 <-- tcsr
        stm     #trta,ar4       ; ar4 <-- trta
         stm     #trad,ar5       ; ar5 <-- trad
         ldm     *ar1, a         ; acc A <-- DRR
         sub     #10AAh, a, b    ; acc A = 0x10AA ?
         bc      tser_16, beq    ; 16-bit serial mode
         and     #0FF00h, a
         sub     #800h, a        ; acc A = 0x8xx
         rc      aneq
         call    ser_in          ; call SP input sub
         ldm     *ar1, a
         and     #0FF00h, a
         sub     #0AA00h, a      ; acc A = 0xaaxx
         and     #0FFFFh, a
         rc      aneq
tser_08
         andm    #0ff3fh, *ar2   ; put SP (TDM) in reset
         orm     #000ch, *ar2    ; configure sport (TDM)
         orm     #0080h, *ar2    ; take sport receiver out of reset
         call    DBsreadA        ; call SP double read byte from DRR
```

```
        stl     a, @temp0       ; save TSPC value in temp0
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @temp1       ; save TCSR value in temp1
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @temp2       ; save TRTA value in temp2
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @xentry      ; save XPC of entry point
        call    DBsreadA        ; call SP double read byte from DRR
        stl     a, @entry       ; save entry point
        b       SP08
*****************************************
*       TDM 16-bit mode
*****************************************
tser_16
        call    ser_in          ; call SP input sub
        mvdk    *ar1, temp0     ; temp0 <-- drr0 (TSPC)
        call    ser_in          ; call SP input sub
        mvdk    *ar1, temp1     ; temp1 <-- drr0 (TCSR)
        call    ser_in          ; call SP input sub
        mvdk    *ar1, temp2     ; temp2 <-- drr0 (TRTA)
        call    ser_in          ; call SP input sub
        mvdk    *ar1, xentry    ; xentry <-- drr0 (XPC of entry point)
        call    ser_in          ; call SP input sub
        mvdk    *ar1, entry     ; temp0 <-- drr0 (entry point)
        b       SP16
* * * * * * * * * * * * * * * * * * * *
*  TDM serial boot in TDM mode
* * * * * * * * * * * * * * * * * * * *
TDMSP
        stl     a, -6, temp0
        or      temp0, a
        stl     a, temp0
        ld      temp0, ASM
        ld      #1, b
        ld      b, ASM, a
        stlm    a, trta
```

```
stm     #0009h, tspc    ; configure sport (TDM) and put in reset
orm     #0080h, tspc    ; take sport receiver out of reset
rsbx    xf              ; signal ready-to-receive
stm     #trcv,ar1       ; ar1 <-- trcv
stm     #tspc,ar2       ; ar2 <-- tspc
stm     #tcsr,ar3       ; ar3 <-- tcsr
stm     #trta,ar4       ; ar4 <-- trta
stm     #trad,ar5       ; ar5 <-- trad
andm    #40h, ifr       ; clear TRNT flag
bcd     $, ntc          ;
bitf    ifr, #40h       ; check TRNT flag
ldm     *ar1, a         ; acc A <-- DRR
sub     #10AAh, a, b    ; acc A = 0x10AA ?
rc      bneq            ; 16-bit serial mode
rsbx    tc              ; ckear TC bit
andm    #40h, ifr       ; clear TRNT flag
bcd     $, ntc          ;
bitf    ifr, #40h       ; check TRNT flag
mvdk    *ar1, temp0     ; temp0 <-- drr (TSPC)
rsbx    tc              ; ckear TC bit
andm    #40h, ifr       ; clear TRNT flag
bcd     $, ntc          ;
bitf    ifr, #40h       ; check TRNT flag
mvdk    *ar1, temp1     ; temp1 <-- drr (TCSR)
rsbx    tc              ; ckear TC bit
andm    #40h, ifr       ; clear TRNT flag
bcd     $, ntc          ;
bitf    ifr, #40h       ; check TRNT flag
mvdk    *ar1, temp2     ; temp2 <-- drr (TRTA)
rsbx    tc              ; ckear TC bit
andm    #40h, ifr       ; clear TRNT flag
bcd     $, ntc          ;
bitf    ifr, #40h       ; check TRNT flag
mvdk    *ar1, xentry    ; XPC of entry point
rsbx    tc              ; ckear TC bit
andm    #40h, ifr       ; clear TRNT flag
```

```
         bcd     $, ntc          ;
         bitf    ifr, #40h       ; check TRNT flag
         mvdk    *ar1, entry     ; entry point
         b       SP16_1
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*       Bootload from parallel I/O port (pa0)           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pasyini
         call    handshake
         portr   pa0, @temp0     ; read BSW 10AAh or 8AAh
         ld      @temp0, a      ; check BSW
         sub     #10aah, a, b   ; acc A = 10aah ?
         bcd     pasync16, beq  ;
         and     #0ffh, a       ; check acc AL = 08
         sub     #8,a           ;
         bc      endboot, aneq  ; not a boot mode
         call    handshake
         portr   pa0, @temp0    ; read BSW 10AAh or 8AAh
         ld      @temp0, a      ; check BSW
         and     #0ffh, a       ; check acc AL = 08
         sub     #0aah, a       ; acc A = 0aah ?
         bc      pasync08, aeq  ;
         b       endboot
*       Bootload from I/O port (8–bit parallel), MS byte first
pasync08
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @temp0
        ldu     @temp0, a       ; read low byte from port
        and     #0ffh, a        ; clear upper byte
         or      @hbyte, a       ; combine high and low byte
        stl     a, @swwsr       ; save swwsr ini–value to SWWSR
        call    handshake
```

```
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @temp0
        ldu     @temp0, a       ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        stl     a, @bscr        ; save bscr ini–value to BSCR
*      get destination address from 1st two byte
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @xentry
        ldu     @xentry, a      ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        stl     a, @xentry      ; save XPC of entry point
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @entry
        ldu     @entry, a       ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        stl     a, @entry       ; save entry point
pasy08_1
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
```

```
        portr   pa0, @lngth
        ldu     @lngth, a       ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        bcd     endboot, aeq    ; if size = 0, branch to endboot
        sub     #1, a, b        ; otherwise acc B = acc A – 1
        stlm    b, brc          ; brc <–– acc B
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @dest
        ldu     @dest, a        ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        stl     a, @dest        ; save XPC of destination addr
*       get code length from 2nd two byte
        call    handshake
        portr   pa0, @hbyte
        ld      @hbyte, 8, a    ; read high byte from port
        stl     a, @hbyte       ; save high byte
        call    handshake
        portr   pa0, @temp0
        ldu     @temp0, a       ; read low byte from port
        and     #0ffh, a        ; clear upper byte
        or      @hbyte, a       ; combine high and low byte
        add     @dest, 16, a    ; save code length
        ld      a, b            ; acc B <–– destination address
        rptb    pfxr08–1
         call    handshake
         portr   pa0, @hbyte
         ld      @hbyte, 8, a   ; read high byte from port
         stl     a, @hbyte      ; save high byte
         call    handshake
        portr   pa0, @temp0
```

```
         ssbx    xf              ; acknowledge byte as soon as it's read
         ldu     @temp0, a       ; read low byte from port
         and     #0ffh, a        ; clear upper byte
         or      @hbyte, a       ; combine high and low byte
         stl     a, @temp0       ; save code word
         ld      b, a            ; acc A <-- destination address
         nop
         nop                     ; 10 cycles delay between xf and write
         writa   @temp0          ; write code word to program memory
         add     #1, a           ; increment destination address
         ld      a, b            ; save new destination address
pfxr08
        b        pasy08_1        ; branch to next section
*       Bootload from I/O port (16-bit parallel)
pasync16
        call    handshake
        portr   pa0, @swwsr     ; read word from port to SWWSR
        call    handshake
        portr   pa0, @bscr      ; read word from port to BSCR
        call    handshake
        portr   pa0, @xentry    ; read word from port to XPC of
                                ; entry point
        call    handshake
        portr   pa0, @entry     ; read word from port to entry
pasy16_1
        call    handshake
        portr   pa0, @lngth     ; read word from port to length
        ldu     @lngth, a       ; check size
        bcd     endboot, aeq    ; size = 0, end of boot
        sub     #1, a, b        ;
        stlm    b, brc
        call    handshake
        portr   pa0, @dest      ; read word from port to XPC of
                                ; destination addr
        call    handshake
        portr   pa0, @temp0     ; read from port to temp for
```

```
                              ; destiantion addr
        ldu     @temp0, a     ; acc A <-- destination address
        add     @dest, 16, a  ;
        rptb    pfxr16-1
         call   handshake     ; check BIO low ?
         portr  pa0, @temp0   ; read word from port to temp
         ssbx   xf            ; acknowledge word as soon as it's read
         rpt    #8
         nop                  ; 10 cycles delay between xf and write
         writa  @temp0        ; write word to destination
         add    #1, a         ; increment destination address
pfxr16
        b       pasy16_1
*       Handshake with BIO signal using XF
handshake
        ssbx    xf            ; acknowledge previous data word
biohigh
        bc      biohigh,bio   ; wait till host sends request
        rsbx    xf            ; indicate ready to receive new data
biolow
        rc      bio           ; wait till new data ready
        b       biolow
bootend
         .end
```

A.2 alaw.asm

```
;***************************************************************************
;
;  CCITT expansion table
;  The is the table A–law expansion table for ADI–coded samples
;
;***************************************************************************
   .DEF   AEXPTAB

   .TEXT
AEXPTAB   .WORD    –688
   .WORD    –656
   .WORD    –752
   .WORD    –720
   .WORD    –560
   .WORD    –528
   .WORD    –624
   .WORD    –592
   .WORD    –944
   .WORD    –912
   .WORD    –1008
   .WORD    –976
   .WORD    –816
   .WORD    –784
   .WORD    –880
   .WORD    –848
   .WORD    –344
   .WORD    –328
   .WORD    –376
   .WORD    –360
   .WORD    –280
   .WORD    –264
   .WORD    –312
   .WORD    –296
   .WORD    –472
```

```
.WORD    -456
.WORD    -504
.WORD    -488
.WORD    -408
.WORD    -392
.WORD    -440
.WORD    -424
.WORD    -2752
.WORD    -2624
.WORD    -3008
.WORD    -2880
.WORD    -2240
.WORD    -2112
.WORD    -2496
.WORD    -2368
.WORD    -3776
.WORD    -3648
.WORD    -4032
.WORD    -3904
.WORD    -3264
.WORD    -3136
.WORD    -3520
.WORD    -3392
.WORD    -1376
.WORD    -1312
.WORD    -1504
.WORD    -1440
.WORD    -1120
.WORD    -1056
.WORD    -1248
.WORD    -1184
.WORD    -1888
.WORD    -1824
.WORD    -2016
.WORD    -1952
.WORD    -1632
```

```
    .WORD    -1568
    .WORD    -1760
    .WORD    -1696
    .WORD    -43
    .WORD    -41
    .WORD    -47
    .WORD    -45
    .WORD    -35
    .WORD    -33
    .WORD    -39
    .WORD    -37
    .WORD    -59
    .WORD    -57
    .WORD    -63
    .WORD    -61
    .WORD    -51
    .WORD    -49
    .WORD    -55
    .WORD    -53
    .WORD    -11
    .WORD    -9
    .WORD    -15
    .WORD    -13
    .WORD    -3
    .WORD    -1
    .WORD    -7
    .WORD    -5
    .WORD    -27
    .WORD    -25
    .WORD    -31
    .WORD    -29
    .WORD    -19
    .WORD    -17
    .WORD    -23
    .WORD    -21
    .WORD    -172
```

```
.WORD    -164
.WORD    -188
.WORD    -180
.WORD    -140
.WORD    -132
.WORD    -156
.WORD    -148
.WORD    -236
.WORD    -228
.WORD    -252
.WORD    -244
.WORD    -204
.WORD    -196
.WORD    -220
.WORD    -212
.WORD    -86
.WORD    -82
.WORD    -94
.WORD    -90
.WORD    -70
.WORD    -66
.WORD    -78
.WORD    -74
.WORD    -118
.WORD    -114
.WORD    -126
.WORD    -122
.WORD    -102
.WORD    -98
.WORD    -110
.WORD    -106
.WORD    688
.WORD    656
.WORD    752
.WORD    720
.WORD    560
```

```
    .WORD    528
    .WORD    624
    .WORD    592
    .WORD    944
    .WORD    912
    .WORD    1008
    .WORD    976
    .WORD    816
    .WORD    784
    .WORD    880
    .WORD    848
    .WORD    344
    .WORD    328
    .WORD    376
    .WORD    360
    .WORD    280
    .WORD    264
    .WORD    312
    .WORD    296
    .WORD    472
    .WORD    456
    .WORD    504
    .WORD    488
    .WORD    408
    .WORD    392
    .WORD    440
    .WORD    424
    .WORD    2752
    .WORD    2624
    .WORD    3008
    .WORD    2880
    .WORD    2240
    .WORD    2112
    .WORD    2496
    .WORD    2368
    .WORD    3776
```

```
.WORD    3648
.WORD    4032
.WORD    3904
.WORD    3264
.WORD    3136
.WORD    3520
.WORD    3392
.WORD    1376
.WORD    1312
.WORD    1504
.WORD    1440
.WORD    1120
.WORD    1056
.WORD    1248
.WORD    1184
.WORD    1888
.WORD    1824
.WORD    2016
.WORD    1952
.WORD    1632
.WORD    1568
.WORD    1760
.WORD    1696
.WORD    43
.WORD    41
.WORD    47
.WORD    45
.WORD    35
.WORD    33
.WORD    39
.WORD    37
.WORD    59
.WORD    57
.WORD    63
.WORD    61
.WORD    51
```

```
    .WORD    49
    .WORD    55
    .WORD    53
    .WORD    11
    .WORD    9
    .WORD    15
    .WORD    13
    .WORD    3
    .WORD    1
    .WORD    7
    .WORD    5
    .WORD    27
    .WORD    25
    .WORD    31
    .WORD    29
    .WORD    19
    .WORD    17
    .WORD    23
    .WORD    21
    .WORD    172
    .WORD    164
    .WORD    188
    .WORD    180
    .WORD    140
    .WORD    132
    .WORD    156
    .WORD    148
    .WORD    236
    .WORD    228
    .WORD    252
    .WORD    244
    .WORD    204
    .WORD    196
    .WORD    220
    .WORD    212
    .WORD    86
```

```
.WORD    82
.WORD    94
.WORD    90
.WORD    70
.WORD    66
.WORD    78
.WORD    74
.WORD    118
.WORD    114
.WORD    126
.WORD    122
.WORD    102
.WORD    98
.WORD    110
.WORD    106
```

A.3 bist548.asm

```
;*****************************************************************************
;***  File Name     : bist.asm
;***  Creation DATE : 06/04/1996
;***  Purpose       : Built–In Self Test
;***  Mod History   : 7/17/96 to support lc548 device
;***
;*****************************************************************************
;***    Entry Conditions  :
;***                    RESET initialization =>>>>>> TC=1 <<<<<<<
;***                    Burning mode:       BIO_=1, entry point 0xff00
;***                    Self test mode:     BIO_=1, entry point 0xff01
;***                    Serial Port Loader: BIO_=0, entry point 0xff01
;***
;***    Sections :
;*****************************************************************************
;

*********************************************************
*     Program
*********************************************************

        .mmregs
SPC2    .set    00042h
DXR2    .set  00041h
psa     .set    00004h

device  .set    548             ; Choose your device

        .if (device–540)
        .else
romstrt .set    0c000h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    01000h–ramstrt
```

```
        .endif
        .if (device–541)
        .else
romstrt .set    09000h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    01400h–ramstrt
        .endif
        .if (device–542)
        .else
romstrt .set    0f800h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    02800h–ramstrt
        .endif
        .if (device–543)
        .else
romstrt .set    0f800h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    02800h–ramstrt
        .endif
        .if (device–544)
        .else
romstrt .set    0a000h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    01000h–ramstrt
        .endif
        .if (device–545)
        .else
romstrt .set    03000h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    01800h–ramstrt
        .endif
```

```
        .if (device–546)
        .else
romstrt .set    03000h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    01800h–ramstrt
        .endif
        .if (device–548)
        .else
romstrt .set    0f800h
romsize .set    0FFFFh–romstrt
ramstrt .set    00060h
ramsize .set    08000h–ramstrt
        .endif

        .title  ”BIST”
        .text
**************TYPICAL MAIN PROGRAM CALLING BIST************************
;       RESET           ; Hardware or Software RESET
;MAIN   STM     #00061,SP
;       SSBX    TC      ; NO LOOPING
;       CALL    BIST
************************************************************************
BURNING
        RSBX    TC                      ; Clear TC to repeat BIST forever

BIST_ENTRY
        STM     #010a0h,PMST            ; remap interrupt, DROM = 0
                                        ; MP\MC_=0 , OVLY=1

******** save call return address

          POPM    BK                    ; return address in BRC
          STM     #00061h,SP            ; Set stack pointer

RESTART   STM     #psa,AR4              ; AR4 pointing on psa address
```

```
                                    ; So we are sure that AR4 isn't
                                    ; null for first
                                    ; SP0 receive in the case of SP loader.
          STM       #ramstrt,AR5    ; ram start address
          STM       #(ramsize–2),BRC  ; ram size
          BC        Selftest,NBIO,TC  ; take branch if Self Test

******** activate Serial Ports

          LD        #0c8h,A         ; init Serial Port for SP loader
          XC        1,NTC
          LD        #0f8h,A         ; init Serial Port for Self Test
          if (device–540)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–541)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–542)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–543)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–544)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
```

```
          .endif
          .if (device–545)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–546)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          .endif
          .if (device–548)
          .else
          STLM      A,SPC0
          STLM      A,SPC1
          STLM      A,SPC2
          .endif
          BC        Selfever,NBIO,NTC      ; take branch if burning

******** init SP boot loader

          STM       #00013h,IMR            ; Unmask INT0, INT1 and RSP0 interrupt
          RSBX      INTM                   ; INTM = 0 : all interrupts are en-
abled
          STM       #DRR0,AR2              ; SP0 data receive
          LD        #AR_CHG,B              ; Auxiliary Register Modification
                                           ; address

IDLE1   IDLE    1                          ; Wait for interrupt
          B         IDLE1

******** test Serial Port only il TC = 0

Selfever
          .if (device–540)
          .else
```

```
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–541)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–542)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–543)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–544)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–545)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–546)
       .else
STLM   A,DXR0
STLM   A,DXR1
       .endif
       .if (device–548)
       .else
STLM   A,DXR0
```

```
   STLM   A,DXR1
   STLM   A,DXR2
          .endif

******** rom test

Selftest
        MVMM    AR5,AR2                 ; Temporary buffer
        MVMM    AR5,AR3                 ; copy AR2 for psa_result
        RPT     #3
        MVPD    0ff7ch,*+AR2            ; Read rom_psa result and place it
                                        ; in temporary buffer

        MAR     *AR3+
        LDU     *AR3+,A
        AND     #0ff00h,A               ; Keep only good data
        STH     A,8,*AR3+
        MAR     *AR3+

        RPT     #6
        MVPD    DATA,*+AR2              ; Add in Temporary buffer all

        LD      #0,A
                                        ; psa's configuration word necessary
        RPT     #2
        MVDD    *AR2–,*AR4              ; psa initialization sequence: Clear
                                        ; psa accumulator

        RPT     #romsize–100h           ; Stop before TI reserved area
        MVPD    romstrt,TRN
        RPT     #07fh                   ; Interrupt Vector 0ff80h–0ffffh
        MVPD    0ff80h,TRN

        MVDD    *AR2–,*AR4              ; Stop psa
        MVDD    *AR2–,*AR4              ; Read psa0H
        LD      #0,B
```

```
        B       NEXT


******** Int interrupts routines

RRINT0  LD      #SP_CHG,A               ; Modify Acc A for next Serial Port
                                        ; interrupt for changing return address
        RETE                            ; end interrupt routine
        .word   00000h
RRINT1  LD      B,A                     ; Modify Acc A for next Serial Port
                                        ; interrupt for modifying AR2
        RETE                            ; end interrupt routine

DATA    .word   00002h,00001h,00000h,0000ch,00007h,0000ch,0000dh

        .if (device–540)
        .else
RESULT  .word   0ea0fh,0a821h,0000ah
        .endif
        .if (device–541)
        .else
RESULT  .word   0a00fh,0d164h,00014h
        .endif
        .if (device–542)
        .else
RESULT  .word   02e0fh,09b02h,000a6h
        .endif
        .if (device–543)
        .else
RESULT  .word   02e0fh,09b02h,000a6h
        .endif
        .if (device–544)
        .else
RESULT  .word   0ea0fh,0a821h,0000ah
        .endif
```

```
        .if (device–545)
        .else
RESULT  .word   0560fh,0fa57h,00023h
        .endif
        .if (device–546)
        .else
RESULT  .word   0560fh,0fa57h,00023h
        .endif

        .if (device–548)
        .else
RESULT  .word   01011h,07fb1h,00004h
        .endif
;
; Before adding new device, used sim5xx simulator
; to compute ram ”signature”
;
;       .if (device–546)
;       .else
;RESULT .word   ram_signature_low,ram_signature_high,ram_signature_guard
;       .endif
;

******** Serial Port interrupt

RSP0    BACC    A                       ; branch to RE_DATA, SP_CHG or AR_CHR.
RE_DATA MVDD    *AR2,*AR4+              ; receive data and place it
        RETE                            ; bye
SP_CHG  MVDD    *AR2,*AR5               ; (1) Change StackPointer (pointing
                                        ; by AR5) to enter prg
AR_CHG  MVMD    *AR2,00014h             ; sinon set AR4 pointer
        LD      #RE_DATA,A              ; Load ACC A to next Serial Port
                                        ; interruption
        RETE

******** Self test : to be continued...
```

```
NEXT
        ABDST   *AR4,*AR3–              ; Check result
        MVDD    *AR2–,*AR4              ; Read psa1H        ABDST   *AR4,*AR3–
                                        ; Check result
        MVDD    *AR2–,*AR4              ; Read psa1G/2H
        RSBX    XF                      ; Clear XF
        ABDST   *AR4,*AR3–              ; Check result
        ADD     A,B                     ; don't forget last ABDST

******** ram / multiplier / adder / alu test

        LD      #0,ASM                  ; Clear ASM
        MVMM    AR5,AR2                 ; AR2 is pointing on ram address
        LD      AR5,16,A                ;
        ST      A,*AR2                  ; load address into address
        ||LD    *AR3–,T                 ; Only for *AR3–
        RPTB    END–1                   ; repeat block
        LD      *AR2,T                  ;   Modify T register for MAS et MAC
                                        ;   instruction
        XOR     #0FFFFh,16,A            ;   A= – A + 1
        ST      A,*AR2+                 ;   write address_ in address
        ||MAC   *AR3,B                  ;   and compute ram signature
        NEG     A                       ;   A = – A <=> address++
        ST      A,*AR2                  ;   write next address into next
                                        ;   address
        ||MAS   *AR3+,B                 ;   and compute ram signature

END     STM     #0008h,AR4              ; pointing on accumulator A
        RPT     #2
        MVPD    RESULT,*AR4+            ; expected value move to ACC A

        SUB     A,B                     ; compute mismatch

        BCD     RESTART,NTC             ; burning mode : repeat Self test
                                        ; forever
```

```
        XC      1,BEQ                   ; but if B is equal zero
        SSBX    XF                      ; (1) Self Test Pass
        PSHM    BK                      ; push return address into stack
RETURN  RET                             ; end subroutine
    .end
    .END
```

A.4 buttfly.asm

```
*************************************************************************
*  MACRO:  'buttfly'   optimized general butterfly radix 2 for 320C5xx  *
*                                                                       *
*  version 0.99                          update:  2. May 94             *
*  version 1.00                                   22  Nov 96            *
*                                                                       *
*  Update: Modified Butterfly now takes only 8 cycles. One butterfly is *
*           done outside the RPTB loop.                                 *
*                                                                       *
*                                                                       *
*                                                                       *
*                                                                       *
*  ASM = -1  / SXM=1 / BK=0 / FRCT=1 / BK=0                             *
*                                                                       *
*      Cycles:                                                          *
*      Prolog code: number of words/number of instr/cycle=19            *
*                    (19 words includes the RPTB and first step of      *
*                     butterfly)                                        *
*      Butterfly  : 8 * (num-2) cycles                                  *
*      Epilog code: number of words/number of cycles=2                  *
*                                                                       *
*        CALCULATE (startup code):                                      *
*        QR'=(PR-QR)/2      <-AR2                                       *
*        PR'=(PR+QR)/2      <-AR3+1                                     *
*        PI'=(PI+QI)/2      <-AR2                                       *
*        PI'=(PI-QI)/2      <-AR3+1                                     *
*************************************************************************
*                                                                       *
*  Def.: AR2 -> QR   (input)  AR2 -> QR+1     (output)                  *
*  Def.: AR3 -> PR   (input)  AR3 -> PR+1     (output)                  *
*  Def.: AR4 -> Cxxx (input)  AR4 -> Cxxx+1   (output) --> WR=cosine    *
*  Def.: AR5 -> Sxxx (input)  AR5 -> Sxxx+1   (output) --> WI=sine      *
*                                                                       *
*        PR' = (PR+(QR*WR+QI*WI))/2       WR=COS(W)   WI=SIN(W)       *
```

```
*        PI' = (PI+(QI*WR-QR*WI))/2  = (PI-(QR*WI-QI*WR))/2             *
*        QR' = (PR-(QR*WR+QI*WI))/2                                     *
*        QI' = (PI-(QI*WR-QR*WR))/2  = (PI+(QR*WI-QI*WR))/2             *
************************************************************************
buttfly .macro num         ;             (contents of register after exec.)
      .asg  AR2, P
      .asg  AR3, Q
      .asg  AR4,WR
      .asg  AR5,WI

;X    STM  #-2,AR0         ; index = -2
      STM  #:num:-3,BRC    ; execute startup + num-3 times general BUTTFLY
;                                                          AR2  AR3  AR4  AR5
; takes 17 words-/cycles (including RPTB)                  ---  ---  ---  ---
      LD   *P,16,A         ;A  :=  PR                       PR   QR   WR   WI
      SUB  *Q,16,A,B       ;B  :   PR-QR                    PR   QR   WR   WI
      ST   B,*Q            ;<<ASM;QR':= (PR-QR)/2
||    ADD  *Q+,B           ;B  := (PR+QR)                   PR   QI   WR   WI
      ST   B,*P+           ;<<ASM;PR':= (PR+QR)/2
||    LD   *Q,A            ;<<16 ;A  :=  QI                 PI   QI   WR   WI
      ADD  *P,16,A,B       ;B  := (PI+QI)                   PI   QI   WR   WI
      ST   B,*P            ;<<ASM;PI':= (PI+QI)/2
||    SUB  *P+,B           ;B  := (PI-QI)                   PR+1 QR   WR   WI
      STH   B,ASM,*Q+      ;QI':= (PI-QI)/2                 PR+1 QR+1 WR   WI
      MPY   *WR,*Q+,A      ;A  := QR*WR                     PR+1 QI+1 WR   WI
      MAC   *WI+,*Q-,A     ;A  := (QR*WR+QI*WI) || T=WI     PR+1 QR+1 WR   WI+1
      ADD   *P,16,A,B      ;B  := (PR+(QR*WR+QI*WI))        PR+1 QR+1 WR   WI+1
      ST    B,*P           ;<<ASM;PR':= (PR+(QR*WR+QI*WI))/2
||    SUB   *P+,B          ;B  := (PR-(QR*WR+QI*WI))        PI+1 QR+1 WR   WI+1
      ST    B,*Q           ;<<ASM;QR':= (PR-(QR*WR+QI*WI))/2
||    MPY   *Q+,A          ;A  := QR*WI     [t=WI]          PI+1 QI+1 WR   WI+1
      MAS   *WR+,*Q,A      ;A  := (   (QR*WI-QI*WR))        PI+1 QI+1 WR+1 WI+1
      RPTBD end?-1         ;delayed block repeat
      ST    A,*Q+          ;dummy write
||    SUB   *P,B           ;B  := (PI-(QR*WI-QI*WR))        PI+1 QR+2 WR+1 WI+1
      ST    B,*P           ;<<ASM;PI':= (PI-(QR*WI-QI*WR))/2
```

```
||    ADD   *P+,B          ;B  := (PI+(QR*WI–QI*WR))        PR+2 QR+2 WR+1 WI+1
;
; Butterfly kernel with 8 instructions / 8 cycles
;
      MPY   *WR,*Q+,A      ;A  :=  QR*WR                    PR+2 QI+2 WR+1 WI+1
      MAC   *WI+,*Q+0%,A   ;A  := (QR*WR+QI*WI) || T=WI     PR+2 QI+1 WR+1 WI+2
      ST    B,*Q+          ;<<ASM;QI’:= (PI+(QR*WI–QI*WR))/2
||    ADD   *P,B           ;B  := (PR+(QR*WR+QI*WI))        PR+2 QR+2 WR+1 WI+2
      ST    B,*P           ;<<ASM;PR’:= (PR+(QR*WR+QI*WI))/2
||    SUB   *P+,B          ;B  := (PR–(QR*WR+QI*WI))        PI+2 QR+2 WR+1 WI+2
      ST    B,*Q           ;<<ASM;QR’:= (PR–(QR*WR+QI*WI))/2
||    MPY   *Q+,A          ;A  := QR*WI     [t=WI]          PI+2 QI+2 WR+1 WI+2
      MAS   *WR+,*Q,A      ;A  := (   (QR*WI–QI*WR))        PI+2 QI+2 WR+2 WI+2
      ST    A,*Q+          ;dummy write
||    SUB   *P,B           ;B  := (PI–(QR*WI–QI*WR))        PI+2 QR+3 WR+2 WI+2
      ST    B,*P           ;<<ASM;PI’:= (PI–(QR*WI–QI*WR))/2
||    ADD   *P+,B          ;B  := (PI+(QR*WI–QI*WR))        PR+3 QR+3 WR+2 WI+2
end?
      MAR   *Q–
      STH   B,ASM,*Q+      ;QI’:= (PI+(QR*WI–QI*WR))/2      PR+3 QR+3 WR+2 WI+2
      .endm
```

A.5 cfft256b.asm

```
;*****************************************************************************
;  Filename:     cfft#.asm
;  Version:      0.01
;  Description:  complex FFT;
-----------------------------------------------------------------------------;
Synopsys:;
;  short oflag =  cfft#(DATA *x);
;  x[#*2]     :  Pointer to complex input data vector (in bit-reversed order)
;                of size nx = #    complex numbers(Re-Im format). On output x
;                will contain the FFT(x);
;-----------------------------------------------------------------------------
;  Description:  complex FFT (radix2 DIT). Input in bit-reversed order.
;                Output in normal order. 1st 2 stages implemented as a
;                radix-4. Last stage also unrolled. Twiddle factors are built-in
;                (provided in the sintab.asm file).;
;                This is an FFT optimized for time. Space consumption is
;                high due to the usage of a separate sine table in each stage.
;                This reduce MIPS count but also increases twiddle table data
;                space.
;-----------------------------------------------------------------------------
;  Algorithm;
;  DFT formula;
;-----------------------------------------------------------------------------
;  Benchmarks;;  - 8 cycles      (butterfly core);;
;-----------------------------------------------------------------------------
; Revision History:;
; 0.00    M. Christ/M. Chishtie. Original code
; 0.01    R. Piedra, C-callable version. Corrected also functional problem
;               with M. Chisthie code   8/31/98
; 0.02    P.Jones II, Added bit reversal. 9/1/98
;*****************************************************************************
N    .set      256   ; NUMBER OF POINTS FOR FFT


    .include "macros.asm"
    .include "sintab.q15"


    .mmregs
```

```
; Far–mode adjustment
; ------------------
       .if __far_mode
offset .set 1          ; far mode uses one extra location for ret addr  ll
       .else
offset .set 0
       .endif
       .asg   (0), TWID
       .asg   (1), INPUT
       .asg   (2), DATA
       .asg   (3), SIN45
       .asg   (4), save_ar7    ; stack description
       .asg   (5), save_ar6    ; stack description
       .asg   (6), save_ar1
       .asg   (7), ret_addr
                               ; x in A
       .asg   (8 + offset), arg_y
;*****************************************************************************
; BIT REV
       .asg   ar2,ar_dst
       .asg   ar3,ar_src
                               ; register usage
                               ; ar0 : bit reversing idx
;*****************************************************************************
       .def   _cfft256b
       .global   _cbrev
       .text
_cfft256b

; Set modes
; ---------
       ssbx   frct         ; fractional mode is on
       ssbx   sxm
offset .set 0
                           ; stack description
       .asg   (0), ret_addr
```

```
                        ; x in A
                        ; register usage
                        ; ar0 : bit reversing idx

; Get arguments
; -------------
       stlm   a, ar4
       stlm   a, ar_src    ; pointer to src
       stlm   a, ar_dst    ; pointer to src
       stm #N, AR0         ; AR0 = n = 1/2 size of circ buffer
       ld  #N-3,a       ; a = n-3 (by pass 1st and last elem)
       stlm   a, brc       ; brc = n-3
; In-place bit-reversing
; ----------------------
in_place:
       mar *ar_src+0B      ; bypass first and last element
       mar *+ar_dst(2)

       rptbd  in_place_end-1
       ldm ar_src,a        ; b = src_addr
       ldm ar_dst, b       ; a = dst_addr

       sub b,a             ; a =  src_addr  - dst_addr
                           ; if >=0  bypass move just increment
       bcd bypass, ageq    ; if (src_addr >= dst_addr) then skip
       ld  *ar_dst+, a     ; a = Re dst element (preserve)
       ld  *ar_dst-, b     ; b = Im dst element (preserve)

       mvdd   *ar_src+, *ar_dst+  ; Re dst = Re src
       mvdd   *ar_src , *ar_dst-  ; Im dst = Im src ; point to Re
       stl b, *ar_src-     ; Im src = b = Im dst; point to Re
       stl a, *ar_src      ; Re src = a = Re dst
bypass
       mar *ar_src+0B
       mar *+ar_dst(2)
```

```
       ldmar_src,a     ; b = src_addr
       ldmar_dst, b    ; a = dst_addr
in_place_end

       ldm    ar4, a
; Return
; ------
       stm   #0100010101011110b,ST1 ; ASM=-2 , FRACT=1, sbx=1;CPL=1(compiler)

; Preserve registers
; ------------------
       pshm   ar1
       pshm   ar6
       pshm   ar7

; Preserve local variables
; ------------------------
       frame  -4
       nop
       nop

; Get Arguments
; -------------
       stla,*sp(INPUT) ; INPUT = *SP(INPUT)
       stla, *sp(DATA) ; DATA = *sp(DATA)
       .if   N>4    ; ??? no need
       st    #5a82h,*sp(SIN45)
       .endif

; Execute
; -------
       combo5xx            ; FFT CODE for STAGES 1 and 2
       stage3                        ; MACRO WITH CODE FOR STAGE 3
       stdmacro 4,16,8,16,sin4,cos4     ; stage,outloopcnter,loopcnter,index
       stdmacro 5,8,16,32,sin5,cos5  ; stage,outloopcnter,loopcnter,index
       stdmacro 6,4,32,64,sin6,cos6  ; stage,outloopcnter,loopcnter,index
```

```
       stdmacro 7,2,64,128,sin7,cos7    ; stage,outloopcnter,loopcnter,index
       laststag 8,sin8,cos8          ; MACRO WITH CODE FOR STAGE 8

; Return
;--------
       frame  +4
       popm   ar7
       popm   ar6
       popm   ar1

       .if __far_mode
       fret
       .else
       ret
       .endif
```

A.6 c5xx1024.asm

```
         .file      ”c5xx1024.asm”
         .title     ”1024 point DIT Radix–2, Complex FFT,TMS320C5xx”
         .width     120
N        .set       1024               ; NUMBER OF POINTS FOR FFT
*************************************************************************
*
*  1024  – POINT COMPLEX, RADIX–2 DIF FFT WITH THE TMS320C5xx / LOOPED CODE
*  ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
*
* THE PROGRAM IS OPTIMIZED FOR THE TMS320C5x INCLUDING BIT REVERSAL
* ADDRESSING MODE.
*
*      REVISION: 0.99    2. May  94
*
*      COPYRIGHT TEXAS INSTRUMENTS INC. 1994
*
*************************************************************************
*
*    USED  REGISTER: INDX,AR0..AR5,A,B,T,ST1,BRC, 1 Stacklevel
*
*    PROGRAM MEMORY:  ––– WORDS (’FFTLEN’) WITHOUT INITIALIZATION
*
*    COEFFICIENTS  :   16 BITS  (Q15 Format)  SCALING:   1/2^12
*
*    CYCLES     : 47952 CYLCES (from ’callfft’ to ’wait’)
*
*    MEMORY USAGE: FFT PROGRAM    [ 0F000H –  0FXXXH ]
*                  INPUT DATA     [ 01000H –  017FFH ]
*                  DATA WORKSPACE [ 00800H –  00FFFH ]
*                  TWIDDLES       [ 00080H –  005ECH ]
*
*    INPUT DATA AT ADDRESS 0800H–0FFFH:
*    ––––––––––––––––––––––––––––––––––
*    THE DATA ARE STORED IN ’input’ IN THE SEQUENCE: X(0),X(1),...,X(1023)
*                                                    Y(0),Y(1),...,Y(1023)
```

```
*
*    INPUT DATA AT ADDRESS 01000h–017FFh:
*    ____________________________________
*    THE DATA ARE STORED IN section 'fftdata' IN THE SEQUENCE:
*    X(0),Y(0),X(1),Y(1),... ... ,X(1023),Y(1023)
*************************************************************************
*
*    THIS PROGRAM INCLUDES FOLLOWING FILES:
*    ______________________________________
*    'TWIDDLES.Q15' contains all twiddle factors in Q15 format (sine,cosine)
*    'C5XXRAD2.ASM' consists of all macros for all FFTs
*    'INIT–FFT.ASM' for initialization
*************************************************************************
        .include init–fft.asm
        .sect    "fftprog"
;    FFT PROGRAM executes the DIT complex radix 2 algorithm
FFT:
   ld    #0,DP           ; DP = Page 0
        stm   #STACK+stacksize,sp
        .if   N>4
        ld    #SIN45,DP
        .bss  SIN45,1
        st    #5a82h,SIN45
        .endif
;
; EXECUTE THE FFT
;
        stm   #0000101010111110b,ST1     ; ASM=–2 , FRACT=1, sbx=1
                                         ; shift 1 from Product to A,B
   combo5xx                              ; FFT CODE for STAGES 1 and 2
        stage3                           ; MACRO WITH CODE FOR STAGE 3
        stdmacro 4,64,8,16,sin4,cos4     ; stage,outloopcnter,loopcnter,index
        stdmacro 5,32,16,32,sin5,cos5    ; stage,outloopcnter,loopcnter,index
        stdmacro 6,16,32,64,sin6,cos6    ; stage,outloopcnter,loopcnter,index
        stdmacro 7,8,64,128,sin7,cos7    ; stage,outloopcnter,loopcnter,index
        stdmacro 8,4,128,256,sin8,cos8   ; stage,outloopcnter,loopcnter,index
```

```
        stdmacro 9,2,256,512,sin9,cos9  ; stage,outloopcnter,loopcnter,index
        laststag 10,sina,cosa           ; MACRO WITH CODE FOR STAGE 10
FFTLEN: ret
        .end
```

A.7 cfft1kb.asm

```
;*****************************************************************************
;  Filename:      cfft#.asm
;  Version :      0.01
;  Description:   complex FFT
;-----------------------------------------------------------------------------
;  Synopsys:
;
;  short oflag =  cfft#(DATA *x)
;
;  x[#*2]      : Pointer to complex input data vector (in bit–reversed order)
;                of size nx = #    complex numbers(Re–Im format). On output x
;                will contain the FFT(x)
;
;-----------------------------------------------------------------------------
;  Description:   complex FFT (radix2 DIT). Input in bit–reversed order.
;                Output in normal order. 1st 2 stages implemented as a
;                radix–4. Last stage also unrolled. Twiddle factors are built–in
;                (provided in the sintab.asm file).
;
;                This is an FFT optimized for time. Space consumption is
;                high due to the usage of a separate sine table in each stage.
;                This reduce MIPS count but also increases twiddle table data
;                space.
;-----------------------------------------------------------------------------
;  Algorithm
;
;  DFT formula
;
;-----------------------------------------------------------------------------
;  Benchmarks
;
;  – 8 cycles    (butterfly core)
;
;
```

```
;-------------------------------------------------------------------------
; Revision History:
;
; 0.00 M.  Christ/M. Chishtie. Original code
; 0.01 R.  Piedra, C–callable version. Corrected also functional problem
;             with M. Chisthie code  8/31/98
; 0.02    P.Jones II, Added bit reversal. 9/1/98
;*************************************************************************
N       .set      1024  ; NUMBER OF POINTS FOR FFT
       .include ”macros.asm”
       .include ”sintab.q15”
        .mmregs
; Far–mode adjustment
; ––––––––––––––––––
       .if __far_mode
offset .set 1                ; far mode uses one extra location for ret addr  ll
       .else
offset .set 0
       .endif
       .asg   (0), TWID
       .asg   (1), INPUT
       .asg   (2), DATA
       .asg   (3), SIN45
       .asg   (4), save_ar7    ; stack description
       .asg   (5), save_ar6    ; stack description
       .asg   (6), save_ar1
       .asg   (7), ret_addr
                               ; x in A
       .asg   (8 + offset), arg_y
;*************************************************************************
; BIT REV
       .asg   ar2,ar_dst
       .asg   ar3,ar_src

                           ; register usage
                           ; ar0 : bit reversing idx
```

```
;***************************************************************************
        .def  _cfft1024b
        .text
_cfft1024b
; Set modes
; ---------
       ssbx   frct          ; fractional mode is on
       ssbx   sxm
offset .set 0
                             ; stack description
       .asg   (0), ret_addr
                             ; x in A
                             ; register usage
                             ; ar0 : bit reversing idx
; Get arguments
; -------------

       stlm   a, ar4
       stlm   a, ar_src      ; pointer to src
       stlm   a, ar_dst      ; pointer to src
       stm #N, AR0           ; AR0 = n = 1/2 size of circ buffer
       ld  #N-3,a            ; a = n-3 (by pass 1st and last elem)
       stlm   a, brc         ; brc = n-3
; In-place bit-reversing
; ----------------------
in_place:
       mar *ar_src+0B        ; bypass first and last element
       mar *+ar_dst(2)
       rptbd  in_place_end-1
       ldm ar_src,a          ; b = src_addr
       ldm ar_dst, b         ; a = dst_addr
       sub b,a               ; a =  src_addr  - dst_addr
                             ; if >=0  bypass move just increment
       bcd bypass, ageq      ; if (src_addr >= dst_addr) then skip
       ld  *ar_dst+, a       ; a = Re dst element (preserve)
       ld  *ar_dst-, b       ; b = Im dst element (preserve)
```

```
       mvdd   *ar_src+, *ar_dst+  ; Re dst = Re src
       mvdd   *ar_src , *ar_dst–  ; Im dst = Im src ; point to Re
       stlb, *ar_src–      ; Im src = b = Im dst; point to Re
       stla, *ar_src       ; Re src = a = Re dst
bypass
       mar *ar_src+0B
       mar *+ar_dst(2)
       ldmar_src,a         ; b = src_addr
       ldmar_dst, b        ; a = dst_addr
in_place_end
       ldm    ar4, a
; Return
; ------
; Preserve registers
; ------------------
       pshm   ar1
       pshm   ar6
       pshm   ar7
; Preserve local variables
; ------------------------
       frame  –4
       nop
       nop
; Get Arguments
; -------------
       stla,*sp(INPUT)     ; INPUT = *SP(INPUT)
       stla, *sp(DATA)     ; DATA = *sp(DATA)
       .if   N>4           ; ??? no need
       st    #5a82h,*sp(SIN45)
        .endif
; Set modes
; ---------
       stm   #0100010101011110b,ST1 ; ASM=–2 , FRACT=1, sbx=1;CPL=1(compiler)
; Execute
; -------
       combo5xx             ; FFT CODE for STAGES 1 and 2
```

```
       stage3                          ; MACRO WITH CODE FOR STAGE 3
       stdmacro 4,64,8,16,sin4,cos4    ; stage,outloopcnter,loopcnter,index
       stdmacro 5,32,16,32,sin5,cos5   ; stage,outloopcnter,loopcnter,index
       stdmacro 6,16,32,64,sin6,cos6   ; stage,outloopcnter,loopcnter,index
       stdmacro 7,8,64,128,sin7,cos7   ; stage,outloopcnter,loopcnter,index
       stdmacro 8,4,128,256,sin8,cos8  ; stage,outloopcnter,loopcnter,index
       stdmacro 9,2,256,512,sin9,cos9  ; stage,outloopcnter,loopcnter,index
       laststag 10,sina,cosa           ; MACRO WITH CODE FOR STAGE 10
; Return
;--------
       frame  +4
       popm   ar7
       popm   ar6
       popm   ar1
       .if __far_mode
       fret
       .else
       ret
       .endif
```

A.8 lsptab.asm

```
*********************************************************
*                        SCCS    INFO
* MODULE         : isptab.asm
* VERSION NO.    : 1.0
* FILE CREATED   : 1/3/92
* PREVIOUS DELTA :
* SCCS FILE      : //node_1d62d/local_db/c50cr/tests/SCCS/s.isptab.asm
* copyright      Texas Instruments, Inc.
* SccsId         : '@(#)        isptab.asm 1.2  10/4/90 11:13:04        '
*
*********************************************************
***********************************************************
**                                                       **
**   Interrupt Service Pointer Table                     **
**                                                       **
**   This code allows indirect vectoring of interrupts   **
**   based on the addresses loaded in RAM block B2.      **
**   This allows the definition of addresses to be       **
**   deferred until the RAM resident code is assembled.  **
**                                                       **
**   DATE:        1/3/92                                 **
**                                                       **
**   Revision History                                    **
**   1.1                                                 **
**                                                       **
***********************************************************
        .length 60
        .def    int0v,int1v,int2v,int3v
        .def    tintv
        .def    brint0v,bxint0v
        .def    brint1v,bxint1v
        .def    trntv,txntv
        .def    hpintv
        .def    nmintv, rstv
```

```
        .def    sint17v,sint18v,sint19v,sint20v
        .def    sint21v,sint22v,sint23v,sint24v,sint25v
        .def    sint26v,sint27v,sint28v,sint29v,sint30v
        .ref    boot
rstv    .set    60h             ; external interrupt vectors
nmintv  .set    61h             ; Nonmaskable interrupt
sint17v .set    62h             ; Software INT #17
sint18v .set    63h             ; Software INT #18
sint19v .set    64h             ; Software INT #19
sint20v .set    65h             ; Software INT #20
sint21v .set    66h             ; Software INT #21
sint22v .set    67h             ; Software INT #22
sint23v .set    68h             ; Software INT #23
sint24v .set    69h             ; Software INT #24
sint25v .set    6ah             ; Software INT #25
sint26v .set    6bh             ; Software INT #26
sint27v .set    6ch             ; Software INT #27
sint28v .set    6dh             ; Software INT #28
sint29v .set    6eh             ; Software INT #29
sint30v .set    6fh             ; Software INT #30
int0v   .set    70h             ; External INT0
int1v   .set    71h             ; External INT1
int2v   .set    72h             ; External INT2
tintv   .set    73h             ;
brint0v .set    74h             ; BSP0 Receive Int
bxint0v .set    75h             ; BSP0 Transmit Int
trntv   .set    76h             ; TDM Receive Int
txntv   .set    77h             ; TDM Transmit Int
int3v   .set    78h             ; External INT3
hpintv  .set    79h             ; HPI Interrupt
brint1v .set    7ah             ; BSP1 Receive Int
bxint1v .set    7bh             ; BSP1 Transmit Int
        .sect   ”int_vect”
reset   bd      0f800h          ; 0 – power on reset
        stm     #06fh,SP
        ld      nmintv, a       ; 1 – NMI interrupt
```

```
fbacc   a
nop
nop
ld      sint17v, a      ; 2 – software interrupt #17
fbacc   a
nop
nop
ld      sint18v, a      ; 3 – external interrupt #18
fbacc   a
nop
nop
ld      sint19v, a      ; 4 – software interrupt #19
fbacc   a
nop
nop
ld      sint20v, a      ; 5 – software interrupt #20
fbacc   a
nop
nop
ld      sint21v, a      ; 6 – external interrupt #21
fbacc   a
nop
nop
ld      sint22v, a      ; 7 – software interrupt #22
fbacc   a
nop
nop
ld      sint23v, a      ; 8 – software interrupt #23
fbacc   a
nop
nop
ld      sint24v, a      ; 9 – external interrupt #24
fbacc   a
nop
nop
ld      sint25v, a      ; 10 – software interrupt #25
```

```
fbacc   a
nop
nop
ld      sint26v, a      ; 11 – software interrupt #26
fbacc   a
nop
nop
ld      sint27v, a      ; 12 – external interrupt #27
fbacc   a
nop
nop
ld      sint28v, a      ; 13 – software interrupt #28
fbacc   a
nop
nop
ld      sint29v, a      ; 14 – software interrupt #29
fbacc   a
nop
nop
ld      sint30v, a      ; 15 – software interrupt #30
fbacc   a
nop
nop
b       0ff00h          ; 16 – external interrupt INT0
nop
nop
call    0ff01h          ; 17 – external interrupt INT1
idle    1
nop
idle    2               ; 18 – external interrupt INT2
nop
nop
nop
ld      tintv, a        ; 19 – timer interrupt
fbacc   a
nop
```

```
nop
ld      brint0v, a      ; 20 – BSP0 receive interrupt
fbacc   a
nop
nop
ld      bxint0v, a      ; 21 – BSP0 transmit interrupt
fbacc   a
nop
nop
ld      trntv, a        ; 22 – TDM port receive interrupt
fbacc   a
nop
nop
ld      txntv, a        ; 23 – TDM port transmit interrupt
fbacc   a
nop
nop
ld      int3v, a        ; 24 – external interrupt INT3
fbacc   a
nop
nop
ld      hpintv, a       ; 25 – HPI interrupt
fbacc   a
nop
nop
ld      brint1v, a      ; 26 – BSP1 receive interrupt
fbacc   a
nop
nop
ld      bxint1v, a      ; 27 – BSP1 transmit interrupt
fbacc   a
nop
nop
.end
```

A.9 sin.asm

```
;***********************************************************************
;       SINE LOOKUP TABLE
;***********************************************************************
    .def SINE
    .text
SINE:    .WORD  00324h
    .WORD  00647h
    .WORD  0096Ah
    .WORD  00C8Bh
    .WORD  00FABh
    .WORD  012C7h
    .WORD  015E1h
    .WORD  018F8h
    .WORD  01C0Bh
    .WORD  01F19h
    .WORD  02223h
    .WORD  02527h
    .WORD  02826h
    .WORD  02B1Eh
    .WORD  02E10h
    .WORD  030FBh
    .WORD  033DEh
    .WORD  036B9h
    .WORD  0398Ch
    .WORD  03C56h
    .WORD  03F16h
    .WORD  041CDh
    .WORD  0447Ah
    .WORD  0471Ch
    .WORD  049B3h
    .WORD  04C3Fh
    .WORD  04EBFh
    .WORD  05133h
    .WORD  0539Ah
```

```
.WORD  055F4h
.WORD  05842h
.WORD  05A81h
.WORD  05CB3h
.WORD  05ED6h
.WORD  060EBh
.WORD  062F1h
.WORD  064E7h
.WORD  066CEh
.WORD  068A5h
.WORD  06A6Ch
.WORD  06C23h
.WORD  06DC9h
.WORD  06F5Eh
.WORD  070E1h
.WORD  07254h
.WORD  073B5h
.WORD  07503h
.WORD  07640h
.WORD  0776Bh
.WORD  07883h
.WORD  07989h
.WORD  07A7Ch
.WORD  07B5Ch
.WORD  07C29h
.WORD  07CE2h
.WORD  07D89h
.WORD  07E1Ch
.WORD  07E9Ch
.WORD  07F08h
.WORD  07F61h
.WORD  07FA6h
.WORD  07FD7h
.WORD  07FF5h
.WORD  07FFEh
.WORD  07FF5h
```

```
.WORD  07FD7h
.WORD  07FA6h
.WORD  07F61h
.WORD  07F08h
.WORD  07E9Ch
.WORD  07E1Ch
.WORD  07D89h
.WORD  07CE2h
.WORD  07C29h
.WORD  07B5Ch
.WORD  07A7Ch
.WORD  07989h
.WORD  07883h
.WORD  0776Bh
.WORD  07640h
.WORD  07503h
.WORD  073B5h
.WORD  07254h
.WORD  070E1h
.WORD  06F5Eh
.WORD  06DC9h
.WORD  06C23h
.WORD  06A6Ch
.WORD  068A5h
.WORD  066CEh
.WORD  064E7h
.WORD  062F1h
.WORD  060EBh
.WORD  05ED6h
.WORD  05CB3h
.WORD  05A81h
.WORD  05842h
.WORD  055F5h
.WORD  0539Ah
.WORD  05133h
.WORD  04EBFh
```

```
.WORD  04C3Fh
.WORD  049B3h
.WORD  0471Ch
.WORD  0447Ah
.WORD  041CDh
.WORD  03F16h
.WORD  03C56h
.WORD  0398Ch
.WORD  036B9h
.WORD  033DEh
.WORD  030FBh
.WORD  02E10h
.WORD  02B1Eh
.WORD  02826h
.WORD  02527h
.WORD  02223h
.WORD  01F19h
.WORD  01C0Bh
.WORD  018F8h
.WORD  015E1h
.WORD  012C7h
.WORD  00FABh
.WORD  00C8Bh
.WORD  0096Ah
.WORD  00647h
.WORD  00324h
.WORD  00000h
.WORD  0FCDCh
.WORD  0F9B9h
.WORD  0F696h
.WORD  0F375h
.WORD  0F056h
.WORD  0ED39h
.WORD  0EA1Fh
.WORD  0E708h
.WORD  0E3F5h
```

```
.WORD  0E0E7h
.WORD  0DDDDh
.WORD  0DAD9h
.WORD  0D7DAh
.WORD  0D4E2h
.WORD  0D1F0h
.WORD  0CF05h
.WORD  0CC22h
.WORD  0C947h
.WORD  0C674h
.WORD  0C3AAh
.WORD  0C0EAh
.WORD  0BE33h
.WORD  0BB86h
.WORD  0B8E4h
.WORD  0B64Dh
.WORD  0B3C1h
.WORD  0B141h
.WORD  0AECDh
.WORD  0AC66h
.WORD  0AA0Ch
.WORD  0A7BEh
.WORD  0A57Fh
.WORD  0A34Dh
.WORD  0A12Ah
.WORD  09F15h
.WORD  09D0Fh
.WORD  09B19h
.WORD  09932h
.WORD  0975Bh
.WORD  09594h
.WORD  093DDh
.WORD  09237h
.WORD  090A2h
.WORD  08F1Fh
.WORD  08DACh
```

```
.WORD  08C4Ch
.WORD  08AFDh
.WORD  089C0h
.WORD  08895h
.WORD  0877Dh
.WORD  08677h
.WORD  08584h
.WORD  084A4h
.WORD  083D8h
.WORD  0831Eh
.WORD  08277h
.WORD  081E4h
.WORD  08164h
.WORD  080F8h
.WORD  0809Fh
.WORD  0805Ah
.WORD  08029h
.WORD  0800Bh
.WORD  08002h
.WORD  0800Bh
.WORD  08029h
.WORD  0805Ah
.WORD  0809Fh
.WORD  080F8h
.WORD  08164h
.WORD  081E4h
.WORD  08277h
.WORD  0831Eh
.WORD  083D7h
.WORD  084A4h
.WORD  08584h
.WORD  08677h
.WORD  0877Dh
.WORD  08895h
.WORD  089C0h
.WORD  08AFDh
```

```
.WORD  08C4Bh
.WORD  08DACh
.WORD  08F1Fh
.WORD  090A2h
.WORD  09237h
.WORD  093DDh
.WORD  09594h
.WORD  0975Bh
.WORD  09932h
.WORD  09B19h
.WORD  09D0Fh
.WORD  09F15h
.WORD  0A12Ah
.WORD  0A34Dh
.WORD  0A57Fh
.WORD  0A7BEh
.WORD  0AA0Bh
.WORD  0AC66h
.WORD  0AECDh
.WORD  0B141h
.WORD  0B3C1h
.WORD  0B64Dh
.WORD  0B8E4h
.WORD  0BB86h
.WORD  0BE33h
.WORD  0C0EAh
.WORD  0C3AAh
.WORD  0C674h
.WORD  0C947h
.WORD  0CC22h
.WORD  0CF05h
.WORD  0D1F0h
.WORD  0D4E1h
.WORD  0D7DAh
.WORD  0DAD9h
.WORD  0DDDDh
```

```
.WORD  0E0E7h
.WORD  0E3F5h
.WORD  0E708h
.WORD  0EA1Eh
.WORD  0ED38h
.WORD  0F055h
.WORD  0F375h
.WORD  0F696h
.WORD  0F9B9h
.WORD  0FCDCh
.WORD  00000h
.END
```

A.10 macros.asm

```
   .mmregs
; ----------------------combo5xx.asm----------------------------------------
combo5xx .macro                    ; REPEAT MACRO 'combo5xx': N/4 times
    .global STAGE1,COMBO,end?
*                                                                          *
* R1  := [(R1+R2)+(R3+R4)]/4    INPUT               OUTPUT                 *
* R2  := [(R1-R2)+(I3-I4)]/4    ------------------  ------------------     *
* R3  := [(R1+R2)-(R3+R4)]/4    AR0 =  7                                   *
* R4  := [(R1-R2)-(I3-I4)]/4    AR1 -> R1,I1        AR1 - > R5,I5          *
* I1  := [(I1+I2)+(I3+I4)]/4    AR2 -> R2,I2        AR2 - > R6,I6          *
* I2  := [(I1-I2)-(R3-R4)]/4    ARP-> AR3 -> R3,I3  ARP - > AR3 - > R7,I7  *
* I3  := [(I1+I2)-(I3+I4)]/4    AR4 -> R4,I4        AR4 - > R8,I8          *
* I4  := [(I1-I2)+(R3-R4)]/4                                               *
*                                                                          *
* RMP : INPUT = OUTPUT
STAGE1:
      mvdk   *sp(DATA),ar2 ; (RMP) pointer to DATA   r1,i1
      mvdk   *sp(DATA),ar3
  nop
  nop
      mvmm   ar3,ar4
      mvmm   ar3,ar5
  nop
  nop
      mar *+ar3(2)   ; pointer to DATA + 2  r2,i2
      mar *+ar4(4)   ; pointer to DATA + 4  r3,i3
      mar *+ar5(6)   ; pointer to DATA + 6  r4,i4
  nop
  nop
       .if     N>4
       stm     #7,ar0        ; index
       stm     #0,BK         ; blocksize to zero!
       stm     #N/4-1,BRC    ; execute N/4-1 times 'combo5xx'
```

```
   rptb   end?         ;
        .endif            ;                AR2 AR3 AR4 AR5
                               ;                               --- --- --- ---
COMBO   sub     *ar2,*ar3,B    ; B  := (R1–R2)              R1  R2  R3  R4
        add     *ar2,*ar3,A    ; A  := (R1+R2)              R1  R2  R3  R4
        sth     B,ASM,*ar3     ; R2 := (R1–R2)/4            R1  R2  R3  R4
        add     *ar4,*ar5,B    ; B  := (R3+R4)              R1  R2  R3  R4
        add     B,A            ; A  := (R1+R2) + (R3+R4)    R1  R2  R3  R4
        sth     A,ASM,*ar2+    ; R1':=((R1+R2) + (R3+R4))/4 I1  R2  R3  R4
        sub     B,1,A          ; B  :=((R1+R2) – (R3+R4))   I1  R2  R3  R4
        sub    *ar4,*ar5,B     ; B  := (R3–R4)              I1  R2  R3  R4
        st      A,*ar4+ ;ASM   ; R3':=((R1+R2) – (R3+R4))/4 I1  R2  I3  I4
   ||   ld      *ar3,A ; 16    ; A  := (R1–R2)/4            I1  R2  I3  I4
        sth     B,ASM,*ar5+    ; R4':= (R3–R4)/4            I1  R2  I3  I4
        sub    *ar4,*ar5–,B    ; B  := (I3–I4)              I1  R2  I3  R4
        add     B,ASM,A        ; A  := (R1–R2) + (I3 –I4)/4 I1  R2  I3  R4
        sth     A,*ar3+        ; R2':= (R1–R2) + (I3 –I4)/4 I1  I2  I3  R4
        sub     B,–1,A         ; A  :=((R1+R2) – (R3+R4))   I1  I2  I3  R4
        ld      *ar5,16,B      ; B=R3–R4
        sth     A,*ar5+        ; R3':=((R1+R2) – (R3+R4))/4 I1  I2  I3  I4
        add     *ar4,*ar5,A    ; A  := (I3+I4)              I1  I2  I3  I4
        sth     A,ASM,*ar4     ; I3':= (I3+I4)/4            I1  I2  I3  I4
        sub     *ar2,*ar3,A    ; A  := (I1–I2)              I1  I2  I3  I4
        add     B,2,A          ; A  := (I1–I2)+ (r3–r4)     I1  I2  I3  I4
        sth     A,ASM,*ar5+0   ; I4':= (I1–I2)+ (r3–r4)/4   I1  I2  I3  R4'
        sub     B,3,A          ; A  := (I1–I2)– (r3–r4)     I1  I2  I3  R4'
        add     *ar2,*ar3,B    ; B  := (I1+I2)              I1  I2  I3  R4'
        st      A,*ar3+0% ;asm; I2':= (I1–I2)–(R3–R4)/4    I1  R2' I3  R4'
   ||   ld      *ar4,A    ;16  ; A  := (I3+I4)/4            I1  R2' I3  R4'
        add     A,2,B          ; B  := (I1+I2)+(I3+I4)      I1  R2' I3  R4'
        sth     B,ASM,*ar2+0   ; I1':= (I1+I2)+(I3+I4)/4    R1' R2' I3  R4'
        sub     A,3,B          ; B  := (I1+I2)–(I3+I4)/4    R1' R2' I3  R4'
end?    sth     B,ASM,*ar4+0   ; I3':= (I1+I2)–(I3+I4)/4    R1' R2' R3' R4'
       .endm
; ----------------------stage3.asm-----------------------------------------
stage3      .macro
```

```
        .global STAGE3,MCR3,end?

        .asg    AR2,P
        .asg    AR3,Q

STAGE3:
       ld *sp(DATA),a   ; a = DATA
       stlm   a, P      ; pointer to DATA   pr,pi
       add#8,a      ; a = DATA + #8
       stlm   a, Q      ; pointer to DATA + 8  qr,qi
        LD      #–1,ASM     ; ASM=–1
        STM     #9,AR1
        STM     #2,AR4
        .if     N>8
        STM     #N/8–1,BRC  ; execute N/8–1 times '4 macros'
        RPTBD   end?        ;
        .endif              ;
       LD *sp(SIN45),T ; load to sin(45)
       nop
*************************************************************************
*                                                                       *
*     MACRO requires   number of words/number of cycles: 6.5            *
*                                                                       *
*        PR'=(PR+QR)/2      PI'=(PI+QI)/2                               *
*        QR'=(PR–QR)/2      QI'=(PI–QI)/2                               *
*                                                                       *
*  version 0.99    from Manfred Christ     update:  2. May. 94          *
*************************************************************************
;                                        (contents of register after exec.)
;                                                          AR2  AR3
;                                                          ---  ---
MCR3  LD    *P,16,A         ; A :=       PR                 PR   QR
      SUB   *Q,16,A,B       ; B :        PR–QR              PR   QR
      ST    B,*Q            ; QR:= (1/2)(PR–QR)
||    ADD   *Q+,B           ; B :=      (PR+QR)             PR   QI
      ST    B,*P+           ; PR:= (1/2)(PR+QR)
```

```
||    LD    *Q,A           ; A :=          QI            PI   QI
      ST    A,*Q           ; Dummy write
||    SUB   *P,B           ; B :=      (PI-QI)           PI   QI
      ST    B,*Q+          ; QI:= (1/2)(PI-QI)           PI   QR+1
||    ADD   *P,B           ; B :=      (PI+QI)
      ST    B,*P+          ; PI:= (1/2)(PI+QI)           PR+1 QR+1


*************************************************************************
*                                                                       *
*     MACRO requires   number of words/number of cycles: 9              *
*                                                                       *
*  T=SIN(45)=COS(45)=W45                                                *
*                                                                       *
*                                                                       *
*        PR'= PR + (W*QI + W*QR) = PR + W * QI + W * QR    (<- AR2)     *
*        QR'= PR - (W*QI + W*QR) = PR - W * QI - W * QR    (<- AR3)     *
*        PI'= PI + (W*QI - W*QR) = PI + W * QI - W * QR    (<- AR2+1)   *
*        QI'= PI - (W*QI - W*QR) = PI - W * QI + W * QR    (<- AR3+2)   *
*                                                                       *
*        PR'= PR +  W * (QI + QR)    (<- AR2)                           *
*        QR'= PR -  W * (QI + QR)    (<- AR3)                           *
*        PI'= PI +  W * (QI - QR)    (<- AR2+1)                         *
*        QI'= PI -  W * (QI - QR)    (<- AR3+1)                         *
*                                                                       *
*  version 0.99    from Manfred Christ     update: 2. May. 94           *
*                                                                       *
*************************************************************************
||    MPY   *Q+,A         ;A   =  QR*W                  PR   QI
      MVMM  AR4,AR0       ;Index = 2
      MAC   *Q-,A         ;A  :=     (QR*W +QI*W)       PR   QR
      ADD   *P,16,A,B     ;B  := (PR+(QR*W +QI*W ))     PR   QR
      ST    B,*P    ;<<ASM;PR':= (PR+(QR*W +QI*W ))/2   PI   QR
||    SUB   *P+,B         ;B  := (PR-(QR*W +QI*W ))     PI   QR
      ST    B,*Q    ;<<ASM;QR':= (PR-(QR*W +QI*W ))/2
||    MPY   *Q+,A         ;A  :=      QR*W              PI   QI
```

```
      MAS   *Q,A          ;A  := (   (QR*W –QI*W ))       PI   QI
      ADD   *P,16,A,B     ;B  := (PI+(QR*W –QI*W ))       PI   QI
      ST    B,*Q+0%       ;QI':= (PI+(QR*W –QI*W ))/2     PI   QI+1
||    SUB   *P,B          ;B  := (PI–(QR*W –QI*W ))       PI   QI+1
      ST    B,*P+         ;PI':= (PI–(QR*W –QI*W ))/2     PR+1 QI+1
*****************************************************************************
*                                                                           *
*     MACRO 'PBY2I'    number of words/number of cycles: 6                  *
*                                                                           *
*        PR'=(PR+QI)/2      PI'=(PI–QR)/2                                   *
*        QR'=(PR–QI)/2      QI'=(PI+QR)/2                                   *
*                                                                           *
*  version 0.99    from Manfred Christ     update:  2. May. 94              *
*                                                                           *
*****************************************************************************
;                                       (contents of register after exec.)
;                                                    AR2      AR3
;                                                    ---      ---
||    LD    *Q–,A           ; A   :=     QI          PR       QR
      ADD   *P,16,A,B  ; RMP  ; B   := (PR+QI)    PR  QR
      ST    B,*P            ; PR' := (PR+QI)/2
||    SUB   *P+,B           ; B   := (PR–QI)         PI       QR
      ST    B,*Q            ; QR' := (PR–QI)/2
||    LD    *Q+,A           ; A   :=     QR          PI       QI
      ADD   *P,16,A,B  ; RMP   ; B   := (PI+QR)    PI  QI
      ST    B,*Q+           ; QI' := (PI+QR)/2       PI       QR+1
||    SUB   *P,B            ; B   := (PI–QR)
      ST    B,*P+           ; PI' := (PI–QR)/2       PR+1     QR+1
*****************************************************************************
*                                                                           *
*     MACRO requires   number of words/number of cycles: 9.5                *
*                                                                           *
*  version 0.99    from: Manfred Christ   update:  2. May. 94               *
*                                                                           *
*   ENTRANCE IN THE MACRO: AR2–>PR,PI                                       *
*                          AR3–>QR,QI                                       *
```

```
*                          TREG=W=COS(45)=SIN(45)                              *
*                                                                              *
*       EXIT OF THE MACRO: AR2->PR+1,PI+1                                      *
*                          AR3->QR+1,QI+1                                      *
*                                                                              *
*        PR'= PR + (W*QI - W*QR) = PR + W * QI - W * QR    (<- AR1)          *
*        QR'= PR - (W*QI - W*QR) = PR - W * QI + W * QR    (<- AR2)          *
*        PI'= PI - (W*QI + W*QR) = PI - W * QI - W * QR    (<- AR1+1)        *
*        QI'= PI + (W*QI + W*QR) = PI + W * QI + W * QR    (<- AR1+2)        *
*                                                                              *
*        PR'= PR + W*(QI - QR)  = PR - W *(QR -QI)   (<- AR2)                *
*        QR'= PR - W*(QI - QR)  = PR - W *(QR -QI)   (<- AR3)                *
*        PI'= PI - W*(QI + QR)     (<- AR2+1)                                *
*        QI'= PI + W*(QI + QR)     (<- AR3+1)                                *
*                                                                              *
* BK==0 !!!!!                                                                  *
*                                                                              *
********************************************************************************
;                                                          AR2 AR3
;                                                          --- ---
||    MPY   *Q+,A          ;A  :=  QR*W                    PR  QI
      MVMM  AR1,AR0        ;Index = 9
      MAS   *Q-,A          ;A  := (QR*W -QI*W )            PR  QR
      ADD   *P,16,A,B      ;B  := (PR+(QR*W -QI*W ))       PR  QR
      ST    B,*Q+    ;<<ASM;QR':= (PR+(QR*W -QI*W ))/2     PR  QI
||    SUB   *P,B           ;B  := (PR-(QR*W -QI*W ))
      ST    B,*P+    ;<<ASM;PR':= (PR-(QR*W -QI*W ))/2
||    MAC   *Q,A           ;A  := QR*W                     PI  QI
      MAC   *Q,A           ;A  := (   (QR*W +QI*W ))       PI  QI
      ADD   *P,16,A,B      ;B  := (PI+(QR*W +QI*W ))       PI  QI
      ST    B,*Q+0%   ;<ASM;QI':= (PI+(QR*W +QI*W ))/2     PI  QR+1
||    SUB   *P,B           ;B  := (PI-(QR*W +QI*W ))
      STH   B,ASM,*P+0%    ;PI':= (PI-(QR*W +QI*W ))/2     PR+1QR+1
end?  .set  $-1

      STM   #-2,AR0        ;Index used in stdmacro macro
```

```
      .endm
; ------------------------------c5xxrad2.asm---------------------------------
laststag .macro  stage,sin,cos
         .global STAGE:stage:,end?
STAGE:stage: .set $
       ld *sp(DATA),a
       stlm   a, ar2    ; ar2 –> DATA
       add#N,a
       stlm   a, ar3    ; ar3 –> DATA+(offset=N)
      stm   #cos,ar4         ; start of cosine in stage 'stg'
      stm   #sin,ar5         ; start of sine in stage   'stg'
      buttfly N/2            ; execute N/2 butterflies
      .endm
; ------------------------------c5xxrad2.asm---------------------------------
stdmacro .macro  stage,l1,l2,idx,sin,cos
         .global STAGE:stage:,end?
STAGE:stage: .set $
       ld *sp(DATA),a
       stlm   a,ar2     ; ar2 –> DATA
       add#idx,a    ; ar3 –> DATA+(offset=idx)
       stlm   a,ar3
      stm   #l1–1,ar1        ; outer loop counter
      stm   #cos,ar6         ; start of cosine in stage 'stg'
      stm   #sin,ar7         ; start of sine in stage 'stg'
loop? mvmm  ar6,ar4          ; start of cosine in stage 'stg'
      mvmm  ar7,ar5          ; start of sine in stage 'stg'
      buttfly l2             ; execute l2 butterflies
      mar   *+ar2(idx)
      banzd loop?,*ar1–
      mar   *+ar3(idx)
      .endm
; ---------------------------------------------------------------------------
buttfly .macro num         ;            (contents of register after exec.)
      .asg  AR2, P
      .asg  AR3, Q
      .asg  AR4,WR
```

```
      .asg  AR5,WI

      STM  #:num:-3,BRC     ; execute startup + num-3 times general BUTTFLY
;                                                          AR2  AR3  AR4  AR5
; takes 17 words-/cycles (including RPTB)                  ---  ---  ---  ---
      LD   *P,16,A        ;A  :=  PR                     PR   QR   WR   WI
      SUB  *Q,16,A,B      ;B  :   PR-QR                  PR   QR   WR   WI
      ST   B,*Q     ;<<ASM;QR':= (PR-QR)/2
||    ADD  *Q+,B          ;B  := (PR+QR)                 PR   QI   WR   WI
      ST   B,*P+    ;<<ASM;PR':= (PR+QR)/2
||    LD   *Q,A     ;<<16 ;A  :=  QI                     PI   QI   WR   WI
      ADD  *P,16,A,B      ;B  := (PI+QI)                 PI   QI   WR   WI
      ST   B,*P     ;<<ASM;PI':= (PI+QI)/2
||    SUB  *P+,B          ;B  := (PI-QI)                 PR+1 QR   WR   WI
      STH   B,ASM,*Q+     ;QI':= (PI-QI)/2               PR+1 QR+1 WR   WI
      MPY   *WR,*Q+,A     ;A  := QR*WR                   PR+1 QI+1 WR   WI
      MAC   *WI+,*Q-,A    ;A  := (QR*WR+QI*WI) || T=WI   PR+1 QR+1 WR   WI+1
      ADD   *P,16,A,B     ;B  := (PR+(QR*WR+QI*WI))      PR+1 QR+1 WR   WI+1
      ST    B,*P    ;<<ASM;PR':= (PR+(QR*WR+QI*WI))/2
||    SUB   *P+,B         ;B  := (PR-(QR*WR+QI*WI))      PI+1 QR+1 WR   WI+1
      ST    B,*Q    ;<<ASM;QR':= (PR-(QR*WR+QI*WI))/2
||    MPY   *Q+,A         ;A  := QR*WI     [t=WI]        PI+1 QI+1 WR   WI+1
      MAS   *WR+,*Q,A     ;A  := (   (QR*WI-QI*WR))      PI+1 QI+1 WR+1 WI+1
      RPTBD end?-1        ;delayed block repeat
      ST    A,*Q+         ;dummy write
||    SUB   *P,B          ;B  := (PI-(QR*WI-QI*WR))      PI+1 QR+2 WR+1 WI+1
      ST    B,*P    ;<<ASM;PI':= (PI-(QR*WI-QI*WR))/2
||    ADD   *P+,B         ;B  := (PI+(QR*WI-QI*WR))      PR+2 QR+2 WR+1 WI+1
;
; Butterfly kernal with 8 instructions / 8 cycles
;
      MPY   *WR,*Q+,A     ;A  :=  QR*WR                  PR+2 QI+2 WR+1 WI+1
      MAC   *WI+,*Q+0%,A  ;A  := (QR*WR+QI*WI) || T=WI   PR+2 QI+1 WR+1 WI+2
      ST    B,*Q+   ;<<ASM;QI':= (PI+(QR*WI-QI*WR))/2
||    ADD   *P,B          ;B  := (PR+(QR*WR+QI*WI))      PR+2 QR+2 WR+1 WI+2
      ST    B,*P    ;<<ASM;PR':= (PR+(QR*WR+QI*WI))/2
```

```
||    SUB   *P+,B          ;B  := (PR–(QR*WR+QI*WI))       PI+2 QR+2 WR+1 WI+2
      ST    B,*Q    ;<<ASM;QR’:= (PR–(QR*WR+QI*WI))/2
||    MPY   *Q+,A          ;A  := QR*WI     [t=WI]         PI+2 QI+2 WR+1 WI+2
      MAS   *WR+,*Q,A      ;A  := (   (QR*WI–QI*WR))       PI+2 QI+2 WR+2 WI+2
      ST    A,*Q+          ;dummy write
||    SUB   *P,B           ;B  := (PI–(QR*WI–QI*WR))       PI+2 QR+3 WR+2 WI+2
      ST    B,*P    ;<<ASM;PI’:= (PI–(QR*WI–QI*WR))/2
||    ADD   *P+,B          ;B  := (PI+(QR*WI–QI*WR))       PR+3 QR+3 WR+2 WI+2
end?
      MAR   *Q–
      STH   B,ASM,*Q+      ;QI’:= (PI+(QR*WI–QI*WR))/2     PR+3 QR+3 WR+2 WI+2
      .endm
```

A.11 sintab.q15

```
;***************************************************************************
;  Filename:  sintab.q15
;  Version :  0.01
;  Description:  twiddle table to include for CFFT
;---------------------------------------------------------------------------
;  Description: a separate sine table is provided for each stage to increase
;      FFT speed. This is at the expense of an increased Data Memory
;      size.
;      Format: a 1/4–cycle sine values followed by a 1/2–cycle cosine
;      values for a total of (3/4 * FFTSIZE –1) values
;---------------------------------------------------------------------------
; Revision History:
;
; 0.00 M. Christ/M. Chishtie. Original code
;
;***************************************************************************
     .sect   ".sintab"
TWIDSTRT
     .if N>8
; STAGE 4
sin4 .word 030fch     ;   22.500 ø
     .word 05a82h     ;   45.000 ø
     .word 07642h     ;   67.500 ø
     .word 07fffh     ;   90.000 ø
cos4 .word 07642h     ;  112.500 ø
     .word 05a82h     ;  135.000 ø
     .word 030fch     ;  157.500 ø
     .word 00000h     ;  180.000 ø
     .word 0cf04h     ;  202.500 ø
     .word 0a57eh     ;  225.000 ø
     .word 089beh     ;  247.500 ø
; Numbers in stage 4 = 11
     .endif
     .if N>16
```

```
; STAGE 5
sin5 .word 018f9h     ;   11.250 ø
     .word 030fch     ;   22.500 ø
     .word 0471dh     ;   33.750 ø
     .word 05a82h     ;   45.000 ø
     .word 06a6eh     ;   56.250 ø
     .word 07642h     ;   67.500 ø
     .word 07d8ah     ;   78.750 ø
     .word 07fffh     ;   90.000 ø
cos5 .word 07d8ah     ;  101.250 ø
     .word 07642h     ;  112.500 ø
     .word 06a6eh     ;  123.750 ø
     .word 05a82h     ;  135.000 ø
     .word 0471dh     ;  146.250 ø
     .word 030fch     ;  157.500 ø
     .word 018f9h     ;  168.750 ø
     .word 00000h     ;  180.000 ø
     .word 0e707h     ;  191.250 ø
     .word 0cf04h     ;  202.500 ø
     .word 0b8e3h     ;  213.750 ø
     .word 0a57eh     ;  225.000 ø
     .word 09592h     ;  236.250 ø
     .word 089beh     ;  247.500 ø
     .word 08276h     ;  258.750 ø
; Numbers in stage 5 = 23
     .endif
     .if N>32
; STAGE 6
sin6 .word 00c8ch     ;    5.625 ø
     .word 018f9h     ;   11.250 ø
     .word 02528h     ;   16.875 ø
     .word 030fch     ;   22.500 ø
     .word 03c57h     ;   28.125 ø
     .word 0471dh     ;   33.750 ø
     .word 05134h     ;   39.375 ø
     .word 05a82h     ;   45.000 ø
```

```
      .word 062f2h     ;   50.625 ø
      .word 06a6eh     ;   56.250 ø
      .word 070e3h     ;   61.875 ø
      .word 07642h     ;   67.500 ø
      .word 07a7dh     ;   73.125 ø
      .word 07d8ah     ;   78.750 ø
      .word 07f62h     ;   84.375 ø
      .word 07fffh     ;   90.000 ø
cos6  .word 07f62h     ;   95.625 ø
      .word 07d8ah     ;  101.250 ø
      .word 07a7dh     ;  106.875 ø
      .word 07642h     ;  112.500 ø
      .word 070e3h     ;  118.125 ø
      .word 06a6eh     ;  123.750 ø
      .word 062f2h     ;  129.375 ø
      .word 05a82h     ;  135.000 ø
      .word 05134h     ;  140.625 ø
      .word 0471dh     ;  146.250 ø
      .word 03c57h     ;  151.875 ø
      .word 030fch     ;  157.500 ø
      .word 02528h     ;  163.125 ø
      .word 018f9h     ;  168.750 ø
      .word 00c8ch     ;  174.375 ø
      .word 00000h     ;  180.000 ø
      .word 0f374h     ;  185.625 ø
      .word 0e707h     ;  191.250 ø
      .word 0dad8h     ;  196.875 ø
      .word 0cf04h     ;  202.500 ø
      .word 0c3a9h     ;  208.125 ø
      .word 0b8e3h     ;  213.750 ø
      .word 0aecch     ;  219.375 ø
      .word 0a57eh     ;  225.000 ø
      .word 09d0eh     ;  230.625 ø
      .word 09592h     ;  236.250 ø
      .word 08f1dh     ;  241.875 ø
      .word 089beh     ;  247.500 ø
```

```
     .word 08583h     ;  253.125 ø
     .word 08276h     ;  258.750 ø
     .word 0809eh     ;  264.375 ø
; Numbers in stage 6 = 47
     .endif
     .if N>64
; STAGE 7
sin7 .word 00648h     ;    2.812 ø
     .word 00c8ch     ;    5.625 ø
     .word 012c8h     ;    8.438 ø
     .word 018f9h     ;   11.250 ø
     .word 01f1ah     ;   14.062 ø
     .word 02528h     ;   16.875 ø
     .word 02b1fh     ;   19.688 ø
     .word 030fch     ;   22.500 ø
     .word 036bah     ;   25.312 ø
     .word 03c57h     ;   28.125 ø
     .word 041ceh     ;   30.938 ø
     .word 0471dh     ;   33.750 ø
     .word 04c40h     ;   36.562 ø
     .word 05134h     ;   39.375 ø
     .word 055f6h     ;   42.188 ø
     .word 05a82h     ;   45.000 ø
     .word 05ed7h     ;   47.812 ø
     .word 062f2h     ;   50.625 ø
     .word 066d0h     ;   53.438 ø
     .word 06a6eh     ;   56.250 ø
     .word 06dcah     ;   59.062 ø
     .word 070e3h     ;   61.875 ø
     .word 073b6h     ;   64.688 ø
     .word 07642h     ;   67.500 ø
     .word 07885h     ;   70.312 ø
     .word 07a7dh     ;   73.125 ø
     .word 07c2ah     ;   75.938 ø
     .word 07d8ah     ;   78.750 ø
     .word 07e9dh     ;   81.562 ø
```

```
     .word 07f62h     ;   84.375 ø
     .word 07fd9h     ;   87.188 ø
     .word 07fffh     ;   90.000 ø
cos7 .word 07fd9h     ;   92.812 ø
     .word 07f62h     ;   95.625 ø
     .word 07e9dh     ;   98.438 ø
     .word 07d8ah     ;  101.250 ø
     .word 07c2ah     ;  104.062 ø
     .word 07a7dh     ;  106.875 ø
     .word 07885h     ;  109.688 ø
     .word 07642h     ;  112.500 ø
     .word 073b6h     ;  115.312 ø
     .word 070e3h     ;  118.125 ø
     .word 06dcah     ;  120.938 ø
     .word 06a6eh     ;  123.750 ø
     .word 066d0h     ;  126.562 ø
     .word 062f2h     ;  129.375 ø
     .word 05ed7h     ;  132.188 ø
     .word 05a82h     ;  135.000 ø
     .word 055f6h     ;  137.812 ø
     .word 05134h     ;  140.625 ø
     .word 04c40h     ;  143.438 ø
     .word 0471dh     ;  146.250 ø
     .word 041ceh     ;  149.062 ø
     .word 03c57h     ;  151.875 ø
     .word 036bah     ;  154.688 ø
     .word 030fch     ;  157.500 ø
     .word 02b1fh     ;  160.312 ø
     .word 02528h     ;  163.125 ø
     .word 01f1ah     ;  165.938 ø
     .word 018f9h     ;  168.750 ø
     .word 012c8h     ;  171.562 ø
     .word 00c8ch     ;  174.375 ø
     .word 00648h     ;  177.188 ø
     .word 00000h     ;  180.000 ø
     .word 0f9b8h     ;  182.812 ø
```

```
     .word 0f374h     ;  185.625 ø
     .word 0ed38h     ;  188.438 ø
     .word 0e707h     ;  191.250 ø
     .word 0e0e6h     ;  194.062 ø
     .word 0dad8h     ;  196.875 ø
     .word 0d4e1h     ;  199.688 ø
     .word 0cf04h     ;  202.500 ø
     .word 0c946h     ;  205.312 ø
     .word 0c3a9h     ;  208.125 ø
     .word 0be32h     ;  210.938 ø
     .word 0b8e3h     ;  213.750 ø
     .word 0b3c0h     ;  216.562 ø
     .word 0aecch     ;  219.375 ø
     .word 0aa0ah     ;  222.188 ø
     .word 0a57eh     ;  225.000 ø
     .word 0a129h     ;  227.812 ø
     .word 09d0eh     ;  230.625 ø
     .word 09930h     ;  233.438 ø
     .word 09592h     ;  236.250 ø
     .word 09236h     ;  239.062 ø
     .word 08f1dh     ;  241.875 ø
     .word 08c4ah     ;  244.688 ø
     .word 089beh     ;  247.500 ø
     .word 0877bh     ;  250.312 ø
     .word 08583h     ;  253.125 ø
     .word 083d6h     ;  255.938 ø
     .word 08276h     ;  258.750 ø
     .word 08163h     ;  261.562 ø
     .word 0809eh     ;  264.375 ø
     .word 08027h     ;  267.188 ø
; Numbers in stage 7 = 95
     .endif
     .if N>128
; STAGE 8
sin8 .word 00324h     ;    1.406 ø
     .word 00648h     ;    2.812 ø
```

```
.word 0096bh     ;    4.219 ø
.word 00c8ch     ;    5.625 ø
.word 00fabh     ;    7.031 ø
.word 012c8h     ;    8.438 ø
.word 015e2h     ;    9.844 ø
.word 018f9h     ;   11.250 ø
.word 01c0ch     ;   12.656 ø
.word 01f1ah     ;   14.062 ø
.word 02224h     ;   15.469 ø
.word 02528h     ;   16.875 ø
.word 02827h     ;   18.281 ø
.word 02b1fh     ;   19.688 ø
.word 02e11h     ;   21.094 ø
.word 030fch     ;   22.500 ø
.word 033dfh     ;   23.906 ø
.word 036bah     ;   25.312 ø
.word 0398dh     ;   26.719 ø
.word 03c57h     ;   28.125 ø
.word 03f17h     ;   29.531 ø
.word 041ceh     ;   30.938 ø
.word 0447bh     ;   32.344 ø
.word 0471dh     ;   33.750 ø
.word 049b4h     ;   35.156 ø
.word 04c40h     ;   36.562 ø
.word 04ec0h     ;   37.969 ø
.word 05134h     ;   39.375 ø
.word 0539bh     ;   40.781 ø
.word 055f6h     ;   42.188 ø
.word 05843h     ;   43.594 ø
.word 05a82h     ;   45.000 ø
.word 05cb4h     ;   46.406 ø
.word 05ed7h     ;   47.812 ø
.word 060ech     ;   49.219 ø
.word 062f2h     ;   50.625 ø
.word 064e9h     ;   52.031 ø
.word 066d0h     ;   53.438 ø
```

```
      .word 068a7h     ;   54.844 ø
      .word 06a6eh     ;   56.250 ø
      .word 06c24h     ;   57.656 ø
      .word 06dcah     ;   59.062 ø
      .word 06f5fh     ;   60.469 ø
      .word 070e3h     ;   61.875 ø
      .word 07255h     ;   63.281 ø
      .word 073b6h     ;   64.688 ø
      .word 07505h     ;   66.094 ø
      .word 07642h     ;   67.500 ø
      .word 0776ch     ;   68.906 ø
      .word 07885h     ;   70.312 ø
      .word 0798ah     ;   71.719 ø
      .word 07a7dh     ;   73.125 ø
      .word 07b5dh     ;   74.531 ø
      .word 07c2ah     ;   75.938 ø
      .word 07ce4h     ;   77.344 ø
      .word 07d8ah     ;   78.750 ø
      .word 07e1eh     ;   80.156 ø
      .word 07e9dh     ;   81.562 ø
      .word 07f0ah     ;   82.969 ø
      .word 07f62h     ;   84.375 ø
      .word 07fa7h     ;   85.781 ø
      .word 07fd9h     ;   87.188 ø
      .word 07ff6h     ;   88.594 ø
      .word 07fffh     ;   90.000 ø
cos8  .word 07ff6h     ;   91.406 ø
      .word 07fd9h     ;   92.812 ø
      .word 07fa7h     ;   94.219 ø
      .word 07f62h     ;   95.625 ø
      .word 07f0ah     ;   97.031 ø
      .word 07e9dh     ;   98.438 ø
      .word 07e1eh     ;   99.844 ø
      .word 07d8ah     ;  101.250 ø
      .word 07ce4h     ;  102.656 ø
      .word 07c2ah     ;  104.062 ø
```

```
.word 07b5dh     ;  105.469 ø
.word 07a7dh     ;  106.875 ø
.word 0798ah     ;  108.281 ø
.word 07885h     ;  109.688 ø
.word 0776ch     ;  111.094 ø
.word 07642h     ;  112.500 ø
.word 07505h     ;  113.906 ø
.word 073b6h     ;  115.312 ø
.word 07255h     ;  116.719 ø
.word 070e3h     ;  118.125 ø
.word 06f5fh     ;  119.531 ø
.word 06dcah     ;  120.938 ø
.word 06c24h     ;  122.344 ø
.word 06a6eh     ;  123.750 ø
.word 068a7h     ;  125.156 ø
.word 066d0h     ;  126.562 ø
.word 064e9h     ;  127.969 ø
.word 062f2h     ;  129.375 ø
.word 060ech     ;  130.781 ø
.word 05ed7h     ;  132.188 ø
.word 05cb4h     ;  133.594 ø
.word 05a82h     ;  135.000 ø
.word 05843h     ;  136.406 ø
.word 055f6h     ;  137.812 ø
.word 0539bh     ;  139.219 ø
.word 05134h     ;  140.625 ø
.word 04ec0h     ;  142.031 ø
.word 04c40h     ;  143.438 ø
.word 049b4h     ;  144.844 ø
.word 0471dh     ;  146.250 ø
.word 0447bh     ;  147.656 ø
.word 041ceh     ;  149.062 ø
.word 03f17h     ;  150.469 ø
.word 03c57h     ;  151.875 ø
.word 0398dh     ;  153.281 ø
.word 036bah     ;  154.688 ø
```

```
.word 033dfh     ;  156.094 ø
.word 030fch     ;  157.500 ø
.word 02e11h     ;  158.906 ø
.word 02b1fh     ;  160.312 ø
.word 02827h     ;  161.719 ø
.word 02528h     ;  163.125 ø
.word 02224h     ;  164.531 ø
.word 01f1ah     ;  165.938 ø
.word 01c0ch     ;  167.344 ø
.word 018f9h     ;  168.750 ø
.word 015e2h     ;  170.156 ø
.word 012c8h     ;  171.562 ø
.word 00fabh     ;  172.969 ø
.word 00c8ch     ;  174.375 ø
.word 0096bh     ;  175.781 ø
.word 00648h     ;  177.188 ø
.word 00324h     ;  178.594 ø
.word 00000h     ;  180.000 ø
.word 0fcdch     ;  181.406 ø
.word 0f9b8h     ;  182.812 ø
.word 0f695h     ;  184.219 ø
.word 0f374h     ;  185.625 ø
.word 0f055h     ;  187.031 ø
.word 0ed38h     ;  188.438 ø
.word 0ea1eh     ;  189.844 ø
.word 0e707h     ;  191.250 ø
.word 0e3f4h     ;  192.656 ø
.word 0e0e6h     ;  194.062 ø
.word 0dddch     ;  195.469 ø
.word 0dad8h     ;  196.875 ø
.word 0d7d9h     ;  198.281 ø
.word 0d4e1h     ;  199.688 ø
.word 0d1efh     ;  201.094 ø
.word 0cf04h     ;  202.500 ø
.word 0cc21h     ;  203.906 ø
.word 0c946h     ;  205.312 ø
```

```
.word 0c673h     ;  206.719 ø
.word 0c3a9h     ;  208.125 ø
.word 0c0e9h     ;  209.531 ø
.word 0be32h     ;  210.938 ø
.word 0bb85h     ;  212.344 ø
.word 0b8e3h     ;  213.750 ø
.word 0b64ch     ;  215.156 ø
.word 0b3c0h     ;  216.562 ø
.word 0b140h     ;  217.969 ø
.word 0aecch     ;  219.375 ø
.word 0ac65h     ;  220.781 ø
.word 0aa0ah     ;  222.188 ø
.word 0a7bdh     ;  223.594 ø
.word 0a57eh     ;  225.000 ø
.word 0a34ch     ;  226.406 ø
.word 0a129h     ;  227.812 ø
.word 09f14h     ;  229.219 ø
.word 09d0eh     ;  230.625 ø
.word 09b17h     ;  232.031 ø
.word 09930h     ;  233.438 ø
.word 09759h     ;  234.844 ø
.word 09592h     ;  236.250 ø
.word 093dch     ;  237.656 ø
.word 09236h     ;  239.062 ø
.word 090a1h     ;  240.469 ø
.word 08f1dh     ;  241.875 ø
.word 08dabh     ;  243.281 ø
.word 08c4ah     ;  244.688 ø
.word 08afbh     ;  246.094 ø
.word 089beh     ;  247.500 ø
.word 08894h     ;  248.906 ø
.word 0877bh     ;  250.312 ø
.word 08676h     ;  251.719 ø
.word 08583h     ;  253.125 ø
.word 084a3h     ;  254.531 ø
.word 083d6h     ;  255.938 ø
```

```
     .word 0831ch     ;  257.344 ø
     .word 08276h     ;  258.750 ø
     .word 081e2h     ;  260.156 ø
     .word 08163h     ;  261.562 ø
     .word 080f6h     ;  262.969 ø
     .word 0809eh     ;  264.375 ø
     .word 08059h     ;  265.781 ø
     .word 08027h     ;  267.188 ø
     .word 0800ah     ;  268.594 ø
; Numbers in stage 8 = 191
     .endif
     .if N>256
; STAGE 9
sin9 .word 00192h     ;    0.703 ø
     .word 00324h     ;    1.406 ø
     .word 004b6h     ;    2.109 ø
     .word 00648h     ;    2.812 ø
     .word 007d9h     ;    3.516 ø
     .word 0096bh     ;    4.219 ø
     .word 00afbh     ;    4.922 ø
     .word 00c8ch     ;    5.625 ø
     .word 00e1ch     ;    6.328 ø
     .word 00fabh     ;    7.031 ø
     .word 0113ah     ;    7.734 ø
     .word 012c8h     ;    8.438 ø
     .word 01455h     ;    9.141 ø
     .word 015e2h     ;    9.844 ø
     .word 0176eh     ;   10.547 ø
     .word 018f9h     ;   11.250 ø
     .word 01a83h     ;   11.953 ø
     .word 01c0ch     ;   12.656 ø
     .word 01d93h     ;   13.359 ø
     .word 01f1ah     ;   14.062 ø
     .word 0209fh     ;   14.766 ø
     .word 02224h     ;   15.469 ø
     .word 023a7h     ;   16.172 ø
```

```
.word 02528h     ;   16.875 ø
.word 026a8h     ;   17.578 ø
.word 02827h     ;   18.281 ø
.word 029a4h     ;   18.984 ø
.word 02b1fh     ;   19.688 ø
.word 02c99h     ;   20.391 ø
.word 02e11h     ;   21.094 ø
.word 02f87h     ;   21.797 ø
.word 030fch     ;   22.500 ø
.word 0326eh     ;   23.203 ø
.word 033dfh     ;   23.906 ø
.word 0354eh     ;   24.609 ø
.word 036bah     ;   25.312 ø
.word 03825h     ;   26.016 ø
.word 0398dh     ;   26.719 ø
.word 03af3h     ;   27.422 ø
.word 03c57h     ;   28.125 ø
.word 03db8h     ;   28.828 ø
.word 03f17h     ;   29.531 ø
.word 04074h     ;   30.234 ø
.word 041ceh     ;   30.938 ø
.word 04326h     ;   31.641 ø
.word 0447bh     ;   32.344 ø
.word 045cdh     ;   33.047 ø
.word 0471dh     ;   33.750 ø
.word 0486ah     ;   34.453 ø
.word 049b4h     ;   35.156 ø
.word 04afbh     ;   35.859 ø
.word 04c40h     ;   36.562 ø
.word 04d81h     ;   37.266 ø
.word 04ec0h     ;   37.969 ø
.word 04ffbh     ;   38.672 ø
.word 05134h     ;   39.375 ø
.word 05269h     ;   40.078 ø
.word 0539bh     ;   40.781 ø
.word 054cah     ;   41.484 ø
```

```
.word 055f6h     ;   42.188 ø
.word 0571eh     ;   42.891 ø
.word 05843h     ;   43.594 ø
.word 05964h     ;   44.297 ø
.word 05a82h     ;   45.000 ø
.word 05b9dh     ;   45.703 ø
.word 05cb4h     ;   46.406 ø
.word 05dc8h     ;   47.109 ø
.word 05ed7h     ;   47.812 ø
.word 05fe4h     ;   48.516 ø
.word 060ech     ;   49.219 ø
.word 061f1h     ;   49.922 ø
.word 062f2h     ;   50.625 ø
.word 063efh     ;   51.328 ø
.word 064e9h     ;   52.031 ø
.word 065deh     ;   52.734 ø
.word 066d0h     ;   53.438 ø
.word 067bdh     ;   54.141 ø
.word 068a7h     ;   54.844 ø
.word 0698ch     ;   55.547 ø
.word 06a6eh     ;   56.250 ø
.word 06b4bh     ;   56.953 ø
.word 06c24h     ;   57.656 ø
.word 06cf9h     ;   58.359 ø
.word 06dcah     ;   59.062 ø
.word 06e97h     ;   59.766 ø
.word 06f5fh     ;   60.469 ø
.word 07023h     ;   61.172 ø
.word 070e3h     ;   61.875 ø
.word 0719eh     ;   62.578 ø
.word 07255h     ;   63.281 ø
.word 07308h     ;   63.984 ø
.word 073b6h     ;   64.688 ø
.word 07460h     ;   65.391 ø
.word 07505h     ;   66.094 ø
.word 075a6h     ;   66.797 ø
```

```
     .word 07642h     ;   67.500 ø
     .word 076d9h     ;   68.203 ø
     .word 0776ch     ;   68.906 ø
     .word 077fbh     ;   69.609 ø
     .word 07885h     ;   70.312 ø
     .word 0790ah     ;   71.016 ø
     .word 0798ah     ;   71.719 ø
     .word 07a06h     ;   72.422 ø
     .word 07a7dh     ;   73.125 ø
     .word 07aefh     ;   73.828 ø
     .word 07b5dh     ;   74.531 ø
     .word 07bc6h     ;   75.234 ø
     .word 07c2ah     ;   75.938 ø
     .word 07c89h     ;   76.641 ø
     .word 07ce4h     ;   77.344 ø
     .word 07d3ah     ;   78.047 ø
     .word 07d8ah     ;   78.750 ø
     .word 07dd6h     ;   79.453 ø
     .word 07e1eh     ;   80.156 ø
     .word 07e60h     ;   80.859 ø
     .word 07e9dh     ;   81.562 ø
     .word 07ed6h     ;   82.266 ø
     .word 07f0ah     ;   82.969 ø
     .word 07f38h     ;   83.672 ø
     .word 07f62h     ;   84.375 ø
     .word 07f87h     ;   85.078 ø
     .word 07fa7h     ;   85.781 ø
     .word 07fc2h     ;   86.484 ø
     .word 07fd9h     ;   87.188 ø
     .word 07feah     ;   87.891 ø
     .word 07ff6h     ;   88.594 ø
     .word 07ffeh     ;   89.297 ø
     .word 07fffh     ;   90.000 ø
cos9 .word 07ffeh     ;   90.703 ø
     .word 07ff6h     ;   91.406 ø
     .word 07feah     ;   92.109 ø
```

```
.word 07fd9h     ;   92.812 ø
.word 07fc2h     ;   93.516 ø
.word 07fa7h     ;   94.219 ø
.word 07f87h     ;   94.922 ø
.word 07f62h     ;   95.625 ø
.word 07f38h     ;   96.328 ø
.word 07f0ah     ;   97.031 ø
.word 07ed6h     ;   97.734 ø
.word 07e9dh     ;   98.438 ø
.word 07e60h     ;   99.141 ø
.word 07e1eh     ;   99.844 ø
.word 07dd6h     ;  100.547 ø
.word 07d8ah     ;  101.250 ø
.word 07d3ah     ;  101.953 ø
.word 07ce4h     ;  102.656 ø
.word 07c89h     ;  103.359 ø
.word 07c2ah     ;  104.062 ø
.word 07bc6h     ;  104.766 ø
.word 07b5dh     ;  105.469 ø
.word 07aefh     ;  106.172 ø
.word 07a7dh     ;  106.875 ø
.word 07a06h     ;  107.578 ø
.word 0798ah     ;  108.281 ø
.word 0790ah     ;  108.984 ø
.word 07885h     ;  109.688 ø
.word 077fbh     ;  110.391 ø
.word 0776ch     ;  111.094 ø
.word 076d9h     ;  111.797 ø
.word 07642h     ;  112.500 ø
.word 075a6h     ;  113.203 ø
.word 07505h     ;  113.906 ø
.word 07460h     ;  114.609 ø
.word 073b6h     ;  115.312 ø
.word 07308h     ;  116.016 ø
.word 07255h     ;  116.719 ø
.word 0719eh     ;  117.422 ø
```

```
.word 070e3h     ;  118.125 ø
.word 07023h     ;  118.828 ø
.word 06f5fh     ;  119.531 ø
.word 06e97h     ;  120.234 ø
.word 06dcah     ;  120.938 ø
.word 06cf9h     ;  121.641 ø
.word 06c24h     ;  122.344 ø
.word 06b4bh     ;  123.047 ø
.word 06a6eh     ;  123.750 ø
.word 0698ch     ;  124.453 ø
.word 068a7h     ;  125.156 ø
.word 067bdh     ;  125.859 ø
.word 066d0h     ;  126.562 ø
.word 065deh     ;  127.266 ø
.word 064e9h     ;  127.969 ø
.word 063efh     ;  128.672 ø
.word 062f2h     ;  129.375 ø
.word 061f1h     ;  130.078 ø
.word 060ech     ;  130.781 ø
.word 05fe4h     ;  131.484 ø
.word 05ed7h     ;  132.188 ø
.word 05dc8h     ;  132.891 ø
.word 05cb4h     ;  133.594 ø
.word 05b9dh     ;  134.297 ø
.word 05a82h     ;  135.000 ø
.word 05964h     ;  135.703 ø
.word 05843h     ;  136.406 ø
.word 0571eh     ;  137.109 ø
.word 055f6h     ;  137.812 ø
.word 054cah     ;  138.516 ø
.word 0539bh     ;  139.219 ø
.word 05269h     ;  139.922 ø
.word 05134h     ;  140.625 ø
.word 04ffbh     ;  141.328 ø
.word 04ec0h     ;  142.031 ø
.word 04d81h     ;  142.734 ø
```

```
.word 04c40h     ;  143.438 ø
.word 04afbh     ;  144.141 ø
.word 049b4h     ;  144.844 ø
.word 0486ah     ;  145.547 ø
.word 0471dh     ;  146.250 ø
.word 045cdh     ;  146.953 ø
.word 0447bh     ;  147.656 ø
.word 04326h     ;  148.359 ø
.word 041ceh     ;  149.062 ø
.word 04074h     ;  149.766 ø
.word 03f17h     ;  150.469 ø
.word 03db8h     ;  151.172 ø
.word 03c57h     ;  151.875 ø
.word 03af3h     ;  152.578 ø
.word 0398dh     ;  153.281 ø
.word 03825h     ;  153.984 ø
.word 036bah     ;  154.688 ø
.word 0354eh     ;  155.391 ø
.word 033dfh     ;  156.094 ø
.word 0326eh     ;  156.797 ø
.word 030fch     ;  157.500 ø
.word 02f87h     ;  158.203 ø
.word 02e11h     ;  158.906 ø
.word 02c99h     ;  159.609 ø
.word 02b1fh     ;  160.312 ø
.word 029a4h     ;  161.016 ø
.word 02827h     ;  161.719 ø
.word 026a8h     ;  162.422 ø
.word 02528h     ;  163.125 ø
.word 023a7h     ;  163.828 ø
.word 02224h     ;  164.531 ø
.word 0209fh     ;  165.234 ø
.word 01f1ah     ;  165.938 ø
.word 01d93h     ;  166.641 ø
.word 01c0ch     ;  167.344 ø
.word 01a83h     ;  168.047 ø
```

```
.word 018f9h     ;  168.750 ø
.word 0176eh     ;  169.453 ø
.word 015e2h     ;  170.156 ø
.word 01455h     ;  170.859 ø
.word 012c8h     ;  171.562 ø
.word 0113ah     ;  172.266 ø
.word 00fabh     ;  172.969 ø
.word 00e1ch     ;  173.672 ø
.word 00c8ch     ;  174.375 ø
.word 00afbh     ;  175.078 ø
.word 0096bh     ;  175.781 ø
.word 007d9h     ;  176.484 ø
.word 00648h     ;  177.188 ø
.word 004b6h     ;  177.891 ø
.word 00324h     ;  178.594 ø
.word 00192h     ;  179.297 ø
.word 00000h     ;  180.000 ø
.word 0fe6eh     ;  180.703 ø
.word 0fcdch     ;  181.406 ø
.word 0fb4ah     ;  182.109 ø
.word 0f9b8h     ;  182.812 ø
.word 0f827h     ;  183.516 ø
.word 0f695h     ;  184.219 ø
.word 0f505h     ;  184.922 ø
.word 0f374h     ;  185.625 ø
.word 0f1e4h     ;  186.328 ø
.word 0f055h     ;  187.031 ø
.word 0eec6h     ;  187.734 ø
.word 0ed38h     ;  188.438 ø
.word 0ebabh     ;  189.141 ø
.word 0ea1eh     ;  189.844 ø
.word 0e892h     ;  190.547 ø
.word 0e707h     ;  191.250 ø
.word 0e57dh     ;  191.953 ø
.word 0e3f4h     ;  192.656 ø
.word 0e26dh     ;  193.359 ø
```

```
.word 0e0e6h     ;  194.062 ø
.word 0df61h     ;  194.766 ø
.word 0dddch     ;  195.469 ø
.word 0dc59h     ;  196.172 ø
.word 0dad8h     ;  196.875 ø
.word 0d958h     ;  197.578 ø
.word 0d7d9h     ;  198.281 ø
.word 0d65ch     ;  198.984 ø
.word 0d4e1h     ;  199.688 ø
.word 0d367h     ;  200.391 ø
.word 0d1efh     ;  201.094 ø
.word 0d079h     ;  201.797 ø
.word 0cf04h     ;  202.500 ø
.word 0cd92h     ;  203.203 ø
.word 0cc21h     ;  203.906 ø
.word 0cab2h     ;  204.609 ø
.word 0c946h     ;  205.312 ø
.word 0c7dbh     ;  206.016 ø
.word 0c673h     ;  206.719 ø
.word 0c50dh     ;  207.422 ø
.word 0c3a9h     ;  208.125 ø
.word 0c248h     ;  208.828 ø
.word 0c0e9h     ;  209.531 ø
.word 0bf8ch     ;  210.234 ø
.word 0be32h     ;  210.938 ø
.word 0bcdah     ;  211.641 ø
.word 0bb85h     ;  212.344 ø
.word 0ba33h     ;  213.047 ø
.word 0b8e3h     ;  213.750 ø
.word 0b796h     ;  214.453 ø
.word 0b64ch     ;  215.156 ø
.word 0b505h     ;  215.859 ø
.word 0b3c0h     ;  216.562 ø
.word 0b27fh     ;  217.266 ø
.word 0b140h     ;  217.969 ø
.word 0b005h     ;  218.672 ø
```

```
.word 0aecch     ;  219.375 ø
.word 0ad97h     ;  220.078 ø
.word 0ac65h     ;  220.781 ø
.word 0ab36h     ;  221.484 ø
.word 0aa0ah     ;  222.188 ø
.word 0a8e2h     ;  222.891 ø
.word 0a7bdh     ;  223.594 ø
.word 0a69ch     ;  224.297 ø
.word 0a57eh     ;  225.000 ø
.word 0a463h     ;  225.703 ø
.word 0a34ch     ;  226.406 ø
.word 0a238h     ;  227.109 ø
.word 0a129h     ;  227.812 ø
.word 0a01ch     ;  228.516 ø
.word 09f14h     ;  229.219 ø
.word 09e0fh     ;  229.922 ø
.word 09d0eh     ;  230.625 ø
.word 09c11h     ;  231.328 ø
.word 09b17h     ;  232.031 ø
.word 09a22h     ;  232.734 ø
.word 09930h     ;  233.438 ø
.word 09843h     ;  234.141 ø
.word 09759h     ;  234.844 ø
.word 09674h     ;  235.547 ø
.word 09592h     ;  236.250 ø
.word 094b5h     ;  236.953 ø
.word 093dch     ;  237.656 ø
.word 09307h     ;  238.359 ø
.word 09236h     ;  239.062 ø
.word 09169h     ;  239.766 ø
.word 090a1h     ;  240.469 ø
.word 08fddh     ;  241.172 ø
.word 08f1dh     ;  241.875 ø
.word 08e62h     ;  242.578 ø
.word 08dabh     ;  243.281 ø
.word 08cf8h     ;  243.984 ø
```

```
.word 08c4ah     ;  244.688 ø
.word 08ba0h     ;  245.391 ø
.word 08afbh     ;  246.094 ø
.word 08a5ah     ;  246.797 ø
.word 089beh     ;  247.500 ø
.word 08927h     ;  248.203 ø
.word 08894h     ;  248.906 ø
.word 08805h     ;  249.609 ø
.word 0877bh     ;  250.312 ø
.word 086f6h     ;  251.016 ø
.word 08676h     ;  251.719 ø
.word 085fah     ;  252.422 ø
.word 08583h     ;  253.125 ø
.word 08511h     ;  253.828 ø
.word 084a3h     ;  254.531 ø
.word 0843ah     ;  255.234 ø
.word 083d6h     ;  255.938 ø
.word 08377h     ;  256.641 ø
.word 0831ch     ;  257.344 ø
.word 082c6h     ;  258.047 ø
.word 08276h     ;  258.750 ø
.word 0822ah     ;  259.453 ø
.word 081e2h     ;  260.156 ø
.word 081a0h     ;  260.859 ø
.word 08163h     ;  261.562 ø
.word 0812ah     ;  262.266 ø
.word 080f6h     ;  262.969 ø
.word 080c8h     ;  263.672 ø
.word 0809eh     ;  264.375 ø
.word 08079h     ;  265.078 ø
.word 08059h     ;  265.781 ø
.word 0803eh     ;  266.484 ø
.word 08027h     ;  267.188 ø
.word 08016h     ;  267.891 ø
.word 0800ah     ;  268.594 ø
.word 08002h     ;  269.297 ø
```

```
; Numbers in stage 9 = 383
     .endif
     .if N>512
; STAGE 10
     .def SINE10
SINE10 .set $
sina .word 000c9h     ;    0.352 ø
     .word 00192h     ;    0.703 ø
     .word 0025bh     ;    1.055 ø
     .word 00324h     ;    1.406 ø
     .word 003edh     ;    1.758 ø
     .word 004b6h     ;    2.109 ø
     .word 0057fh     ;    2.461 ø
     .word 00648h     ;    2.812 ø
     .word 00711h     ;    3.164 ø
     .word 007d9h     ;    3.516 ø
     .word 008a2h     ;    3.867 ø
     .word 0096bh     ;    4.219 ø
     .word 00a33h     ;    4.570 ø
     .word 00afbh     ;    4.922 ø
     .word 00bc4h     ;    5.273 ø
     .word 00c8ch     ;    5.625 ø
     .word 00d54h     ;    5.977 ø
     .word 00e1ch     ;    6.328 ø
     .word 00ee4h     ;    6.680 ø
     .word 00fabh     ;    7.031 ø
     .word 01073h     ;    7.383 ø
     .word 0113ah     ;    7.734 ø
     .word 01201h     ;    8.086 ø
     .word 012c8h     ;    8.438 ø
     .word 0138fh     ;    8.789 ø
     .word 01455h     ;    9.141 ø
     .word 0151ch     ;    9.492 ø
     .word 015e2h     ;    9.844 ø
     .word 016a8h     ;   10.195 ø
     .word 0176eh     ;   10.547 ø
```

```
.word 01833h     ;   10.898 ø
.word 018f9h     ;   11.250 ø
.word 019beh     ;   11.602 ø
.word 01a83h     ;   11.953 ø
.word 01b47h     ;   12.305 ø
.word 01c0ch     ;   12.656 ø
.word 01cd0h     ;   13.008 ø
.word 01d93h     ;   13.359 ø
.word 01e57h     ;   13.711 ø
.word 01f1ah     ;   14.062 ø
.word 01fddh     ;   14.414 ø
.word 0209fh     ;   14.766 ø
.word 02162h     ;   15.117 ø
.word 02224h     ;   15.469 ø
.word 022e5h     ;   15.820 ø
.word 023a7h     ;   16.172 ø
.word 02467h     ;   16.523 ø
.word 02528h     ;   16.875 ø
.word 025e8h     ;   17.227 ø
.word 026a8h     ;   17.578 ø
.word 02768h     ;   17.930 ø
.word 02827h     ;   18.281 ø
.word 028e5h     ;   18.633 ø
.word 029a4h     ;   18.984 ø
.word 02a62h     ;   19.336 ø
.word 02b1fh     ;   19.688 ø
.word 02bdch     ;   20.039 ø
.word 02c99h     ;   20.391 ø
.word 02d55h     ;   20.742 ø
.word 02e11h     ;   21.094 ø
.word 02ecch     ;   21.445 ø
.word 02f87h     ;   21.797 ø
.word 03042h     ;   22.148 ø
.word 030fch     ;   22.500 ø
.word 031b5h     ;   22.852 ø
.word 0326eh     ;   23.203 ø
```

```
.word 03327h     ;   23.555 ø
.word 033dfh     ;   23.906 ø
.word 03497h     ;   24.258 ø
.word 0354eh     ;   24.609 ø
.word 03604h     ;   24.961 ø
.word 036bah     ;   25.312 ø
.word 03770h     ;   25.664 ø
.word 03825h     ;   26.016 ø
.word 038d9h     ;   26.367 ø
.word 0398dh     ;   26.719 ø
.word 03a40h     ;   27.070 ø
.word 03af3h     ;   27.422 ø
.word 03ba5h     ;   27.773 ø
.word 03c57h     ;   28.125 ø
.word 03d08h     ;   28.477 ø
.word 03db8h     ;   28.828 ø
.word 03e68h     ;   29.180 ø
.word 03f17h     ;   29.531 ø
.word 03fc6h     ;   29.883 ø
.word 04074h     ;   30.234 ø
.word 04121h     ;   30.586 ø
.word 041ceh     ;   30.938 ø
.word 0427ah     ;   31.289 ø
.word 04326h     ;   31.641 ø
.word 043d1h     ;   31.992 ø
.word 0447bh     ;   32.344 ø
.word 04524h     ;   32.695 ø
.word 045cdh     ;   33.047 ø
.word 04675h     ;   33.398 ø
.word 0471dh     ;   33.750 ø
.word 047c4h     ;   34.102 ø
.word 0486ah     ;   34.453 ø
.word 0490fh     ;   34.805 ø
.word 049b4h     ;   35.156 ø
.word 04a58h     ;   35.508 ø
.word 04afbh     ;   35.859 ø
```

```
.word 04b9eh     ;   36.211 ø
.word 04c40h     ;   36.562 ø
.word 04ce1h     ;   36.914 ø
.word 04d81h     ;   37.266 ø
.word 04e21h     ;   37.617 ø
.word 04ec0h     ;   37.969 ø
.word 04f5eh     ;   38.320 ø
.word 04ffbh     ;   38.672 ø
.word 05098h     ;   39.023 ø
.word 05134h     ;   39.375 ø
.word 051cfh     ;   39.727 ø
.word 05269h     ;   40.078 ø
.word 05303h     ;   40.430 ø
.word 0539bh     ;   40.781 ø
.word 05433h     ;   41.133 ø
.word 054cah     ;   41.484 ø
.word 05560h     ;   41.836 ø
.word 055f6h     ;   42.188 ø
.word 0568ah     ;   42.539 ø
.word 0571eh     ;   42.891 ø
.word 057b1h     ;   43.242 ø
.word 05843h     ;   43.594 ø
.word 058d4h     ;   43.945 ø
.word 05964h     ;   44.297 ø
.word 059f4h     ;   44.648 ø
.word 05a82h     ;   45.000 ø
.word 05b10h     ;   45.352 ø
.word 05b9dh     ;   45.703 ø
.word 05c29h     ;   46.055 ø
.word 05cb4h     ;   46.406 ø
.word 05d3eh     ;   46.758 ø
.word 05dc8h     ;   47.109 ø
.word 05e50h     ;   47.461 ø
.word 05ed7h     ;   47.812 ø
.word 05f5eh     ;   48.164 ø
.word 05fe4h     ;   48.516 ø
```

```
.word 06068h     ;   48.867 ø
.word 060ech     ;   49.219 ø
.word 0616fh     ;   49.570 ø
.word 061f1h     ;   49.922 ø
.word 06272h     ;   50.273 ø
.word 062f2h     ;   50.625 ø
.word 06371h     ;   50.977 ø
.word 063efh     ;   51.328 ø
.word 0646ch     ;   51.680 ø
.word 064e9h     ;   52.031 ø
.word 06564h     ;   52.383 ø
.word 065deh     ;   52.734 ø
.word 06657h     ;   53.086 ø
.word 066d0h     ;   53.438 ø
.word 06747h     ;   53.789 ø
.word 067bdh     ;   54.141 ø
.word 06832h     ;   54.492 ø
.word 068a7h     ;   54.844 ø
.word 0691ah     ;   55.195 ø
.word 0698ch     ;   55.547 ø
.word 069fdh     ;   55.898 ø
.word 06a6eh     ;   56.250 ø
.word 06addh     ;   56.602 ø
.word 06b4bh     ;   56.953 ø
.word 06bb8h     ;   57.305 ø
.word 06c24h     ;   57.656 ø
.word 06c8fh     ;   58.008 ø
.word 06cf9h     ;   58.359 ø
.word 06d62h     ;   58.711 ø
.word 06dcah     ;   59.062 ø
.word 06e31h     ;   59.414 ø
.word 06e97h     ;   59.766 ø
.word 06efbh     ;   60.117 ø
.word 06f5fh     ;   60.469 ø
.word 06fc2h     ;   60.820 ø
.word 07023h     ;   61.172 ø
```

```
.word 07083h     ;   61.523 ø
.word 070e3h     ;   61.875 ø
.word 07141h     ;   62.227 ø
.word 0719eh     ;   62.578 ø
.word 071fah     ;   62.930 ø
.word 07255h     ;   63.281 ø
.word 072afh     ;   63.633 ø
.word 07308h     ;   63.984 ø
.word 0735fh     ;   64.336 ø
.word 073b6h     ;   64.688 ø
.word 0740bh     ;   65.039 ø
.word 07460h     ;   65.391 ø
.word 074b3h     ;   65.742 ø
.word 07505h     ;   66.094 ø
.word 07556h     ;   66.445 ø
.word 075a6h     ;   66.797 ø
.word 075f4h     ;   67.148 ø
.word 07642h     ;   67.500 ø
.word 0768eh     ;   67.852 ø
.word 076d9h     ;   68.203 ø
.word 07723h     ;   68.555 ø
.word 0776ch     ;   68.906 ø
.word 077b4h     ;   69.258 ø
.word 077fbh     ;   69.609 ø
.word 07840h     ;   69.961 ø
.word 07885h     ;   70.312 ø
.word 078c8h     ;   70.664 ø
.word 0790ah     ;   71.016 ø
.word 0794ah     ;   71.367 ø
.word 0798ah     ;   71.719 ø
.word 079c9h     ;   72.070 ø
.word 07a06h     ;   72.422 ø
.word 07a42h     ;   72.773 ø
.word 07a7dh     ;   73.125 ø
.word 07ab7h     ;   73.477 ø
.word 07aefh     ;   73.828 ø
```

```
.word 07b27h     ;   74.180 ø
.word 07b5dh     ;   74.531 ø
.word 07b92h     ;   74.883 ø
.word 07bc6h     ;   75.234 ø
.word 07bf9h     ;   75.586 ø
.word 07c2ah     ;   75.938 ø
.word 07c5ah     ;   76.289 ø
.word 07c89h     ;   76.641 ø
.word 07cb7h     ;   76.992 ø
.word 07ce4h     ;   77.344 ø
.word 07d0fh     ;   77.695 ø
.word 07d3ah     ;   78.047 ø
.word 07d63h     ;   78.398 ø
.word 07d8ah     ;   78.750 ø
.word 07db1h     ;   79.102 ø
.word 07dd6h     ;   79.453 ø
.word 07dfbh     ;   79.805 ø
.word 07e1eh     ;   80.156 ø
.word 07e3fh     ;   80.508 ø
.word 07e60h     ;   80.859 ø
.word 07e7fh     ;   81.211 ø
.word 07e9dh     ;   81.562 ø
.word 07ebah     ;   81.914 ø
.word 07ed6h     ;   82.266 ø
.word 07ef0h     ;   82.617 ø
.word 07f0ah     ;   82.969 ø
.word 07f22h     ;   83.320 ø
.word 07f38h     ;   83.672 ø
.word 07f4eh     ;   84.023 ø
.word 07f62h     ;   84.375 ø
.word 07f75h     ;   84.727 ø
.word 07f87h     ;   85.078 ø
.word 07f98h     ;   85.430 ø
.word 07fa7h     ;   85.781 ø
.word 07fb5h     ;   86.133 ø
.word 07fc2h     ;   86.484 ø
```

```
      .word 07fceh     ;   86.836 ø
      .word 07fd9h     ;   87.188 ø
      .word 07fe2h     ;   87.539 ø
      .word 07feah     ;   87.891 ø
      .word 07ff1h     ;   88.242 ø
      .word 07ff6h     ;   88.594 ø
      .word 07ffah     ;   88.945 ø
      .word 07ffeh     ;   89.297 ø
      .word 07fffh     ;   89.648 ø
      .word 07fffh     ;   90.000 ø
cosa  .word 07fffh     ;   90.352 ø
      .word 07ffeh     ;   90.703 ø
      .word 07ffah     ;   91.055 ø
      .word 07ff6h     ;   91.406 ø
      .word 07ff1h     ;   91.758 ø
      .word 07feah     ;   92.109 ø
      .word 07fe2h     ;   92.461 ø
      .word 07fd9h     ;   92.812 ø
      .word 07fceh     ;   93.164 ø
      .word 07fc2h     ;   93.516 ø
      .word 07fb5h     ;   93.867 ø
      .word 07fa7h     ;   94.219 ø
      .word 07f98h     ;   94.570 ø
      .word 07f87h     ;   94.922 ø
      .word 07f75h     ;   95.273 ø
      .word 07f62h     ;   95.625 ø
      .word 07f4eh     ;   95.977 ø
      .word 07f38h     ;   96.328 ø
      .word 07f22h     ;   96.680 ø
      .word 07f0ah     ;   97.031 ø
      .word 07ef0h     ;   97.383 ø
      .word 07ed6h     ;   97.734 ø
      .word 07ebah     ;   98.086 ø
      .word 07e9dh     ;   98.438 ø
      .word 07e7fh     ;   98.789 ø
      .word 07e60h     ;   99.141 ø
```

```
.word 07e3fh     ;   99.492 ø
.word 07e1eh     ;   99.844 ø
.word 07dfbh     ;  100.195 ø
.word 07dd6h     ;  100.547 ø
.word 07db1h     ;  100.898 ø
.word 07d8ah     ;  101.250 ø
.word 07d63h     ;  101.602 ø
.word 07d3ah     ;  101.953 ø
.word 07d0fh     ;  102.305 ø
.word 07ce4h     ;  102.656 ø
.word 07cb7h     ;  103.008 ø
.word 07c89h     ;  103.359 ø
.word 07c5ah     ;  103.711 ø
.word 07c2ah     ;  104.062 ø
.word 07bf9h     ;  104.414 ø
.word 07bc6h     ;  104.766 ø
.word 07b92h     ;  105.117 ø
.word 07b5dh     ;  105.469 ø
.word 07b27h     ;  105.820 ø
.word 07aefh     ;  106.172 ø
.word 07ab7h     ;  106.523 ø
.word 07a7dh     ;  106.875 ø
.word 07a42h     ;  107.227 ø
.word 07a06h     ;  107.578 ø
.word 079c9h     ;  107.930 ø
.word 0798ah     ;  108.281 ø
.word 0794ah     ;  108.633 ø
.word 0790ah     ;  108.984 ø
.word 078c8h     ;  109.336 ø
.word 07885h     ;  109.688 ø
.word 07840h     ;  110.039 ø
.word 077fbh     ;  110.391 ø
.word 077b4h     ;  110.742 ø
.word 0776ch     ;  111.094 ø
.word 07723h     ;  111.445 ø
.word 076d9h     ;  111.797 ø
```

```
.word 0768eh     ;  112.148 ø
.word 07642h     ;  112.500 ø
.word 075f4h     ;  112.852 ø
.word 075a6h     ;  113.203 ø
.word 07556h     ;  113.555 ø
.word 07505h     ;  113.906 ø
.word 074b3h     ;  114.258 ø
.word 07460h     ;  114.609 ø
.word 0740bh     ;  114.961 ø
.word 073b6h     ;  115.312 ø
.word 0735fh     ;  115.664 ø
.word 07308h     ;  116.016 ø
.word 072afh     ;  116.367 ø
.word 07255h     ;  116.719 ø
.word 071fah     ;  117.070 ø
.word 0719eh     ;  117.422 ø
.word 07141h     ;  117.773 ø
.word 070e3h     ;  118.125 ø
.word 07083h     ;  118.477 ø
.word 07023h     ;  118.828 ø
.word 06fc2h     ;  119.180 ø
.word 06f5fh     ;  119.531 ø
.word 06efbh     ;  119.883 ø
.word 06e97h     ;  120.234 ø
.word 06e31h     ;  120.586 ø
.word 06dcah     ;  120.938 ø
.word 06d62h     ;  121.289 ø
.word 06cf9h     ;  121.641 ø
.word 06c8fh     ;  121.992 ø
.word 06c24h     ;  122.344 ø
.word 06bb8h     ;  122.695 ø
.word 06b4bh     ;  123.047 ø
.word 06addh     ;  123.398 ø
.word 06a6eh     ;  123.750 ø
.word 069fdh     ;  124.102 ø
.word 0698ch     ;  124.453 ø
```

```
.word 0691ah     ;  124.805 ø
.word 068a7h     ;  125.156 ø
.word 06832h     ;  125.508 ø
.word 067bdh     ;  125.859 ø
.word 06747h     ;  126.211 ø
.word 066d0h     ;  126.562 ø
.word 06657h     ;  126.914 ø
.word 065deh     ;  127.266 ø
.word 06564h     ;  127.617 ø
.word 064e9h     ;  127.969 ø
.word 0646ch     ;  128.320 ø
.word 063efh     ;  128.672 ø
.word 06371h     ;  129.023 ø
.word 062f2h     ;  129.375 ø
.word 06272h     ;  129.727 ø
.word 061f1h     ;  130.078 ø
.word 0616fh     ;  130.430 ø
.word 060ech     ;  130.781 ø
.word 06068h     ;  131.133 ø
.word 05fe4h     ;  131.484 ø
.word 05f5eh     ;  131.836 ø
.word 05ed7h     ;  132.188 ø
.word 05e50h     ;  132.539 ø
.word 05dc8h     ;  132.891 ø
.word 05d3eh     ;  133.242 ø
.word 05cb4h     ;  133.594 ø
.word 05c29h     ;  133.945 ø
.word 05b9dh     ;  134.297 ø
.word 05b10h     ;  134.648 ø
.word 05a82h     ;  135.000 ø
.word 059f4h     ;  135.352 ø
.word 05964h     ;  135.703 ø
.word 058d4h     ;  136.055 ø
.word 05843h     ;  136.406 ø
.word 057b1h     ;  136.758 ø
.word 0571eh     ;  137.109 ø
```

```
.word 0568ah     ;  137.461 ø
.word 055f6h     ;  137.812 ø
.word 05560h     ;  138.164 ø
.word 054cah     ;  138.516 ø
.word 05433h     ;  138.867 ø
.word 0539bh     ;  139.219 ø
.word 05303h     ;  139.570 ø
.word 05269h     ;  139.922 ø
.word 051cfh     ;  140.273 ø
.word 05134h     ;  140.625 ø
.word 05098h     ;  140.977 ø
.word 04ffbh     ;  141.328 ø
.word 04f5eh     ;  141.680 ø
.word 04ec0h     ;  142.031 ø
.word 04e21h     ;  142.383 ø
.word 04d81h     ;  142.734 ø
.word 04ce1h     ;  143.086 ø
.word 04c40h     ;  143.438 ø
.word 04b9eh     ;  143.789 ø
.word 04afbh     ;  144.141 ø
.word 04a58h     ;  144.492 ø
.word 049b4h     ;  144.844 ø
.word 0490fh     ;  145.195 ø
.word 0486ah     ;  145.547 ø
.word 047c4h     ;  145.898 ø
.word 0471dh     ;  146.250 ø
.word 04675h     ;  146.602 ø
.word 045cdh     ;  146.953 ø
.word 04524h     ;  147.305 ø
.word 0447bh     ;  147.656 ø
.word 043d1h     ;  148.008 ø
.word 04326h     ;  148.359 ø
.word 0427ah     ;  148.711 ø
.word 041ceh     ;  149.062 ø
.word 04121h     ;  149.414 ø
.word 04074h     ;  149.766 ø
```

```
.word 03fc6h     ;  150.117 ø
.word 03f17h     ;  150.469 ø
.word 03e68h     ;  150.820 ø
.word 03db8h     ;  151.172 ø
.word 03d08h     ;  151.523 ø
.word 03c57h     ;  151.875 ø
.word 03ba5h     ;  152.227 ø
.word 03af3h     ;  152.578 ø
.word 03a40h     ;  152.930 ø
.word 0398dh     ;  153.281 ø
.word 038d9h     ;  153.633 ø
.word 03825h     ;  153.984 ø
.word 03770h     ;  154.336 ø
.word 036bah     ;  154.688 ø
.word 03604h     ;  155.039 ø
.word 0354eh     ;  155.391 ø
.word 03497h     ;  155.742 ø
.word 033dfh     ;  156.094 ø
.word 03327h     ;  156.445 ø
.word 0326eh     ;  156.797 ø
.word 031b5h     ;  157.148 ø
.word 030fch     ;  157.500 ø
.word 03042h     ;  157.852 ø
.word 02f87h     ;  158.203 ø
.word 02ecch     ;  158.555 ø
.word 02e11h     ;  158.906 ø
.word 02d55h     ;  159.258 ø
.word 02c99h     ;  159.609 ø
.word 02bdch     ;  159.961 ø
.word 02b1fh     ;  160.312 ø
.word 02a62h     ;  160.664 ø
.word 029a4h     ;  161.016 ø
.word 028e5h     ;  161.367 ø
.word 02827h     ;  161.719 ø
.word 02768h     ;  162.070 ø
.word 026a8h     ;  162.422 ø
```

```
.word 025e8h     ;  162.773 ø
.word 02528h     ;  163.125 ø
.word 02467h     ;  163.477 ø
.word 023a7h     ;  163.828 ø
.word 022e5h     ;  164.180 ø
.word 02224h     ;  164.531 ø
.word 02162h     ;  164.883 ø
.word 0209fh     ;  165.234 ø
.word 01fddh     ;  165.586 ø
.word 01f1ah     ;  165.938 ø
.word 01e57h     ;  166.289 ø
.word 01d93h     ;  166.641 ø
.word 01cd0h     ;  166.992 ø
.word 01c0ch     ;  167.344 ø
.word 01b47h     ;  167.695 ø
.word 01a83h     ;  168.047 ø
.word 019beh     ;  168.398 ø
.word 018f9h     ;  168.750 ø
.word 01833h     ;  169.102 ø
.word 0176eh     ;  169.453 ø
.word 016a8h     ;  169.805 ø
.word 015e2h     ;  170.156 ø
.word 0151ch     ;  170.508 ø
.word 01455h     ;  170.859 ø
.word 0138fh     ;  171.211 ø
.word 012c8h     ;  171.562 ø
.word 01201h     ;  171.914 ø
.word 0113ah     ;  172.266 ø
.word 01073h     ;  172.617 ø
.word 00fabh     ;  172.969 ø
.word 00ee4h     ;  173.320 ø
.word 00e1ch     ;  173.672 ø
.word 00d54h     ;  174.023 ø
.word 00c8ch     ;  174.375 ø
.word 00bc4h     ;  174.727 ø
.word 00afbh     ;  175.078 ø
```

```
.word 00a33h     ;  175.430 ø
.word 0096bh     ;  175.781 ø
.word 008a2h     ;  176.133 ø
.word 007d9h     ;  176.484 ø
.word 00711h     ;  176.836 ø
.word 00648h     ;  177.188 ø
.word 0057fh     ;  177.539 ø
.word 004b6h     ;  177.891 ø
.word 003edh     ;  178.242 ø
.word 00324h     ;  178.594 ø
.word 0025bh     ;  178.945 ø
.word 00192h     ;  179.297 ø
.word 000c9h     ;  179.648 ø
.word 00000h     ;  180.000 ø
.word 0ff37h     ;  180.352 ø
.word 0fe6eh     ;  180.703 ø
.word 0fda5h     ;  181.055 ø
.word 0fcdch     ;  181.406 ø
.word 0fc13h     ;  181.758 ø
.word 0fb4ah     ;  182.109 ø
.word 0fa81h     ;  182.461 ø
.word 0f9b8h     ;  182.812 ø
.word 0f8efh     ;  183.164 ø
.word 0f827h     ;  183.516 ø
.word 0f75eh     ;  183.867 ø
.word 0f695h     ;  184.219 ø
.word 0f5cdh     ;  184.570 ø
.word 0f505h     ;  184.922 ø
.word 0f43ch     ;  185.273 ø
.word 0f374h     ;  185.625 ø
.word 0f2ach     ;  185.977 ø
.word 0f1e4h     ;  186.328 ø
.word 0f11ch     ;  186.680 ø
.word 0f055h     ;  187.031 ø
.word 0ef8dh     ;  187.383 ø
.word 0eec6h     ;  187.734 ø
```

```
.word 0edffh     ;  188.086 ø
.word 0ed38h     ;  188.438 ø
.word 0ec71h     ;  188.789 ø
.word 0ebabh     ;  189.141 ø
.word 0eae4h     ;  189.492 ø
.word 0ea1eh     ;  189.844 ø
.word 0e958h     ;  190.195 ø
.word 0e892h     ;  190.547 ø
.word 0e7cdh     ;  190.898 ø
.word 0e707h     ;  191.250 ø
.word 0e642h     ;  191.602 ø
.word 0e57dh     ;  191.953 ø
.word 0e4b9h     ;  192.305 ø
.word 0e3f4h     ;  192.656 ø
.word 0e330h     ;  193.008 ø
.word 0e26dh     ;  193.359 ø
.word 0e1a9h     ;  193.711 ø
.word 0e0e6h     ;  194.062 ø
.word 0e023h     ;  194.414 ø
.word 0df61h     ;  194.766 ø
.word 0de9eh     ;  195.117 ø
.word 0dddch     ;  195.469 ø
.word 0dd1bh     ;  195.820 ø
.word 0dc59h     ;  196.172 ø
.word 0db99h     ;  196.523 ø
.word 0dad8h     ;  196.875 ø
.word 0da18h     ;  197.227 ø
.word 0d958h     ;  197.578 ø
.word 0d898h     ;  197.930 ø
.word 0d7d9h     ;  198.281 ø
.word 0d71bh     ;  198.633 ø
.word 0d65ch     ;  198.984 ø
.word 0d59eh     ;  199.336 ø
.word 0d4e1h     ;  199.688 ø
.word 0d424h     ;  200.039 ø
.word 0d367h     ;  200.391 ø
```

```
.word 0d2abh     ;  200.742 ø
.word 0d1efh     ;  201.094 ø
.word 0d134h     ;  201.445 ø
.word 0d079h     ;  201.797 ø
.word 0cfbeh     ;  202.148 ø
.word 0cf04h     ;  202.500 ø
.word 0ce4bh     ;  202.852 ø
.word 0cd92h     ;  203.203 ø
.word 0ccd9h     ;  203.555 ø
.word 0cc21h     ;  203.906 ø
.word 0cb69h     ;  204.258 ø
.word 0cab2h     ;  204.609 ø
.word 0c9fch     ;  204.961 ø
.word 0c946h     ;  205.312 ø
.word 0c890h     ;  205.664 ø
.word 0c7dbh     ;  206.016 ø
.word 0c727h     ;  206.367 ø
.word 0c673h     ;  206.719 ø
.word 0c5c0h     ;  207.070 ø
.word 0c50dh     ;  207.422 ø
.word 0c45bh     ;  207.773 ø
.word 0c3a9h     ;  208.125 ø
.word 0c2f8h     ;  208.477 ø
.word 0c248h     ;  208.828 ø
.word 0c198h     ;  209.180 ø
.word 0c0e9h     ;  209.531 ø
.word 0c03ah     ;  209.883 ø
.word 0bf8ch     ;  210.234 ø
.word 0bedfh     ;  210.586 ø
.word 0be32h     ;  210.938 ø
.word 0bd86h     ;  211.289 ø
.word 0bcdah     ;  211.641 ø
.word 0bc2fh     ;  211.992 ø
.word 0bb85h     ;  212.344 ø
.word 0badch     ;  212.695 ø
.word 0ba33h     ;  213.047 ø
```

```
.word 0b98bh     ;  213.398 ø
.word 0b8e3h     ;  213.750 ø
.word 0b83ch     ;  214.102 ø
.word 0b796h     ;  214.453 ø
.word 0b6f1h     ;  214.805 ø
.word 0b64ch     ;  215.156 ø
.word 0b5a8h     ;  215.508 ø
.word 0b505h     ;  215.859 ø
.word 0b462h     ;  216.211 ø
.word 0b3c0h     ;  216.562 ø
.word 0b31fh     ;  216.914 ø
.word 0b27fh     ;  217.266 ø
.word 0b1dfh     ;  217.617 ø
.word 0b140h     ;  217.969 ø
.word 0b0a2h     ;  218.320 ø
.word 0b005h     ;  218.672 ø
.word 0af68h     ;  219.023 ø
.word 0aecch     ;  219.375 ø
.word 0ae31h     ;  219.727 ø
.word 0ad97h     ;  220.078 ø
.word 0acfdh     ;  220.430 ø
.word 0ac65h     ;  220.781 ø
.word 0abcdh     ;  221.133 ø
.word 0ab36h     ;  221.484 ø
.word 0aaa0h     ;  221.836 ø
.word 0aa0ah     ;  222.188 ø
.word 0a976h     ;  222.539 ø
.word 0a8e2h     ;  222.891 ø
.word 0a84fh     ;  223.242 ø
.word 0a7bdh     ;  223.594 ø
.word 0a72ch     ;  223.945 ø
.word 0a69ch     ;  224.297 ø
.word 0a60ch     ;  224.648 ø
.word 0a57eh     ;  225.000 ø
.word 0a4f0h     ;  225.352 ø
.word 0a463h     ;  225.703 ø
```

```
.word 0a3d7h     ;  226.055 ø
.word 0a34ch     ;  226.406 ø
.word 0a2c2h     ;  226.758 ø
.word 0a238h     ;  227.109 ø
.word 0a1b0h     ;  227.461 ø
.word 0a129h     ;  227.812 ø
.word 0a0a2h     ;  228.164 ø
.word 0a01ch     ;  228.516 ø
.word 09f98h     ;  228.867 ø
.word 09f14h     ;  229.219 ø
.word 09e91h     ;  229.570 ø
.word 09e0fh     ;  229.922 ø
.word 09d8eh     ;  230.273 ø
.word 09d0eh     ;  230.625 ø
.word 09c8fh     ;  230.977 ø
.word 09c11h     ;  231.328 ø
.word 09b94h     ;  231.680 ø
.word 09b17h     ;  232.031 ø
.word 09a9ch     ;  232.383 ø
.word 09a22h     ;  232.734 ø
.word 099a9h     ;  233.086 ø
.word 09930h     ;  233.438 ø
.word 098b9h     ;  233.789 ø
.word 09843h     ;  234.141 ø
.word 097ceh     ;  234.492 ø
.word 09759h     ;  234.844 ø
.word 096e6h     ;  235.195 ø
.word 09674h     ;  235.547 ø
.word 09603h     ;  235.898 ø
.word 09592h     ;  236.250 ø
.word 09523h     ;  236.602 ø
.word 094b5h     ;  236.953 ø
.word 09448h     ;  237.305 ø
.word 093dch     ;  237.656 ø
.word 09371h     ;  238.008 ø
.word 09307h     ;  238.359 ø
```

```
.word 0929eh     ;  238.711 ø
.word 09236h     ;  239.062 ø
.word 091cfh     ;  239.414 ø
.word 09169h     ;  239.766 ø
.word 09105h     ;  240.117 ø
.word 090a1h     ;  240.469 ø
.word 0903eh     ;  240.820 ø
.word 08fddh     ;  241.172 ø
.word 08f7dh     ;  241.523 ø
.word 08f1dh     ;  241.875 ø
.word 08ebfh     ;  242.227 ø
.word 08e62h     ;  242.578 ø
.word 08e06h     ;  242.930 ø
.word 08dabh     ;  243.281 ø
.word 08d51h     ;  243.633 ø
.word 08cf8h     ;  243.984 ø
.word 08ca1h     ;  244.336 ø
.word 08c4ah     ;  244.688 ø
.word 08bf5h     ;  245.039 ø
.word 08ba0h     ;  245.391 ø
.word 08b4dh     ;  245.742 ø
.word 08afbh     ;  246.094 ø
.word 08aaah     ;  246.445 ø
.word 08a5ah     ;  246.797 ø
.word 08a0ch     ;  247.148 ø
.word 089beh     ;  247.500 ø
.word 08972h     ;  247.852 ø
.word 08927h     ;  248.203 ø
.word 088ddh     ;  248.555 ø
.word 08894h     ;  248.906 ø
.word 0884ch     ;  249.258 ø
.word 08805h     ;  249.609 ø
.word 087c0h     ;  249.961 ø
.word 0877bh     ;  250.312 ø
.word 08738h     ;  250.664 ø
.word 086f6h     ;  251.016 ø
```

```
.word 086b6h     ;  251.367 ø
.word 08676h     ;  251.719 ø
.word 08637h     ;  252.070 ø
.word 085fah     ;  252.422 ø
.word 085beh     ;  252.773 ø
.word 08583h     ;  253.125 ø
.word 08549h     ;  253.477 ø
.word 08511h     ;  253.828 ø
.word 084d9h     ;  254.180 ø
.word 084a3h     ;  254.531 ø
.word 0846eh     ;  254.883 ø
.word 0843ah     ;  255.234 ø
.word 08407h     ;  255.586 ø
.word 083d6h     ;  255.938 ø
.word 083a6h     ;  256.289 ø
.word 08377h     ;  256.641 ø
.word 08349h     ;  256.992 ø
.word 0831ch     ;  257.344 ø
.word 082f1h     ;  257.695 ø
.word 082c6h     ;  258.047 ø
.word 0829dh     ;  258.398 ø
.word 08276h     ;  258.750 ø
.word 0824fh     ;  259.102 ø
.word 0822ah     ;  259.453 ø
.word 08205h     ;  259.805 ø
.word 081e2h     ;  260.156 ø
.word 081c1h     ;  260.508 ø
.word 081a0h     ;  260.859 ø
.word 08181h     ;  261.211 ø
.word 08163h     ;  261.562 ø
.word 08146h     ;  261.914 ø
.word 0812ah     ;  262.266 ø
.word 08110h     ;  262.617 ø
.word 080f6h     ;  262.969 ø
.word 080deh     ;  263.320 ø
.word 080c8h     ;  263.672 ø
```

```
     .word 080b2h     ;  264.023 ø
     .word 0809eh     ;  264.375 ø
     .word 0808bh     ;  264.727 ø
     .word 08079h     ;  265.078 ø
     .word 08068h     ;  265.430 ø
     .word 08059h     ;  265.781 ø
     .word 0804bh     ;  266.133 ø
     .word 0803eh     ;  266.484 ø
     .word 08032h     ;  266.836 ø
     .word 08027h     ;  267.188 ø
     .word 0801eh     ;  267.539 ø
     .word 08016h     ;  267.891 ø
     .word 0800fh     ;  268.242 ø
     .word 0800ah     ;  268.594 ø
     .word 08006h     ;  268.945 ø
     .word 08002h     ;  269.297 ø
     .word 08001h     ;  269.648 ø
; Numbers in stage 10 = 767
     .endif
TWIDEND  .set $
TWIDLEN  .set TWIDEND-TWIDSTRT
```

A.12 ulaw.asm

```
;***********************************************************************
;
;  CCITT μ–law Expansion Table
;
;***********************************************************************
   .DEF   UEXPTAB
   .TEXT
UEXPTAB   .WORD  0e0a1h
   .WORD  0e1a1h
   .WORD  0e2a1h
   .WORD  0e3a1h
   .WORD  0e4a1h
   .WORD  0e5a1h
   .WORD  0e6a1h
   .WORD  0e7a1h
   .WORD  0e8a1h
   .WORD  0e9a1h
   .WORD  0eaa1h
   .WORD  0eba1h
   .WORD  0eca1h
   .WORD  0eda1h
   .WORD  0eea1h
   .WORD  0efa1h
   .WORD  0f061h
   .WORD  0f0e1h
   .WORD  0f161h
   .WORD  0f1e1h
   .WORD  0f261h
   .WORD  0f2e1h
   .WORD  0f361h
   .WORD  0f3e1h
   .WORD  0f461h
   .WORD  0f4e1h
   .WORD  0f561h
```

```
    .WORD  0f5e1h
    .WORD  0f661h
    .WORD  0f6e1h
    .WORD  0f761h
    .WORD  0f7e1h
    .WORD  0f841h
    .WORD  0f881h
    .WORD  0f8c1h
    .WORD  0f901h
    .WORD  0f941h
    .WORD  0f981h
    .WORD  0f9c1h
    .WORD  0fa01h
    .WORD  0fa41h
    .WORD  0fa81h
    .WORD  0fac1h
    .WORD  0fb01h
    .WORD  0fb41h
    .WORD  0fb81h
    .WORD  0fbc1h
    .WORD  0fc01h
    .WORD  0fc31h
    .WORD  0fc51h
    .WORD  0fc71h
    .WORD  0fc91h
    .WORD  0fcb1h
    .WORD  0fcd1h
    .WORD  0fcf1h
    .WORD  0fd11h
    .WORD  0fd31h
    .WORD  0fd51h
    .WORD  0fd71h
    .WORD  0fd91h
    .WORD  0fdb1h
    .WORD  0fdd1h
    .WORD  0fdf1h
```

```
.WORD  0fe11h
.WORD  0fe29h
.WORD  0fe39h
.WORD  0fe49h
.WORD  0fe59h
.WORD  0fe69h
.WORD  0fe79h
.WORD  0fe89h
.WORD  0fe99h
.WORD  0fea9h
.WORD  0feb9h
.WORD  0fec9h
.WORD  0fed9h
.WORD  0fee9h
.WORD  0fef9h
.WORD  0ff09h
.WORD  0ff19h
.WORD  0ff25h
.WORD  0ff2dh
.WORD  0ff35h
.WORD  0ff3dh
.WORD  0ff45h
.WORD  0ff4dh
.WORD  0ff55h
.WORD  0ff5dh
.WORD  0ff65h
.WORD  0ff6dh
.WORD  0ff75h
.WORD  0ff7dh
.WORD  0ff85h
.WORD  0ff8dh
.WORD  0ff95h
.WORD  0ff9dh
.WORD  0ffa3h
.WORD  0ffa7h
.WORD  0ffabh
```

```
    .WORD  0ffafh
    .WORD  0ffb3h
    .WORD  0ffb7h
    .WORD  0ffbbh
    .WORD  0ffbfh
    .WORD  0ffc3h
    .WORD  0ffc7h
    .WORD  0ffcbh
    .WORD  0ffcfh
    .WORD  0ffd3h
    .WORD  0ffd7h
    .WORD  0ffdbh
    .WORD  0ffdfh
    .WORD  0ffe2h
    .WORD  0ffe4h
    .WORD  0ffe6h
    .WORD  0ffe8h
    .WORD  0ffeah
    .WORD  0ffech
    .WORD  0ffeeh
    .WORD  0fff0h
    .WORD  0fff2h
    .WORD  0fff4h
    .WORD  0fff6h
    .WORD  0fff8h
    .WORD  0fffah
    .WORD  0fffch
    .WORD  0fffeh
    .WORD  00000h
    .WORD  01f5fh
    .WORD  01e5fh
    .WORD  01d5fh
    .WORD  01c5fh
    .WORD  01b5fh
    .WORD  01a5fh
    .WORD  0195fh
```

```
.WORD  0185fh
.WORD  0175fh
.WORD  0165fh
.WORD  0155fh
.WORD  0145fh
.WORD  0135fh
.WORD  0125fh
.WORD  0115fh
.WORD  0105fh
.WORD  00f9fh
.WORD  00f1fh
.WORD  00e9fh
.WORD  00e1fh
.WORD  00d9fh
.WORD  00d1fh
.WORD  00c9fh
.WORD  00c1fh
.WORD  00b9fh
.WORD  00b1fh
.WORD  00a9fh
.WORD  00a1fh
.WORD  0099fh
.WORD  0091fh
.WORD  0089fh
.WORD  0081fh
.WORD  007bfh
.WORD  0077fh
.WORD  0073fh
.WORD  006ffh
.WORD  006bfh
.WORD  0067fh
.WORD  0063fh
.WORD  005ffh
.WORD  005bfh
.WORD  0057fh
.WORD  0053fh
```

```
    .WORD  004ffh
    .WORD  004bfh
    .WORD  0047fh
    .WORD  0043fh
    .WORD  003ffh
    .WORD  003cfh
    .WORD  003afh
    .WORD  0038fh
    .WORD  0036fh
    .WORD  0034fh
    .WORD  0032fh
    .WORD  0030fh
    .WORD  002efh
    .WORD  002cfh
    .WORD  002afh
    .WORD  0028fh
    .WORD  0026fh
    .WORD  0024fh
    .WORD  0022fh
    .WORD  0020fh
    .WORD  001efh
    .WORD  001d7h
    .WORD  001c7h
    .WORD  001b7h
    .WORD  001a7h
    .WORD  00197h
    .WORD  00187h
    .WORD  00177h
    .WORD  00167h
    .WORD  00157h
    .WORD  00147h
    .WORD  00137h
    .WORD  00127h
    .WORD  00117h
    .WORD  00107h
    .WORD  000f7h
```

```
.WORD  000e7h
.WORD  000dbh
.WORD  000d3h
.WORD  000cbh
.WORD  000c3h
.WORD  000bbh
.WORD  000b3h
.WORD  000abh
.WORD  000a3h
.WORD  0009bh
.WORD  00093h
.WORD  0008bh
.WORD  00083h
.WORD  0007bh
.WORD  00073h
.WORD  0006bh
.WORD  00063h
.WORD  0005dh
.WORD  00059h
.WORD  00055h
.WORD  00051h
.WORD  0004dh
.WORD  00049h
.WORD  00045h
.WORD  00041h
.WORD  0003dh
.WORD  00039h
.WORD  00035h
.WORD  00031h
.WORD  0002dh
.WORD  00029h
.WORD  00025h
.WORD  00021h
.WORD  0001eh
.WORD  0001ch
.WORD  0001ah
```

```
    .WORD  00018h
    .WORD  00016h
    .WORD  00014h
    .WORD  00012h
    .WORD  00010h
    .WORD  0000eh
    .WORD  0000ch
    .WORD  0000ah
    .WORD  00008h
    .WORD  00006h
    .WORD  00004h
    .WORD  00002h
    .WORD  00000h
    .END
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265