TMS320C548/9

BOOT LOADER and

ON-CHIP ROM DESCRIPTION

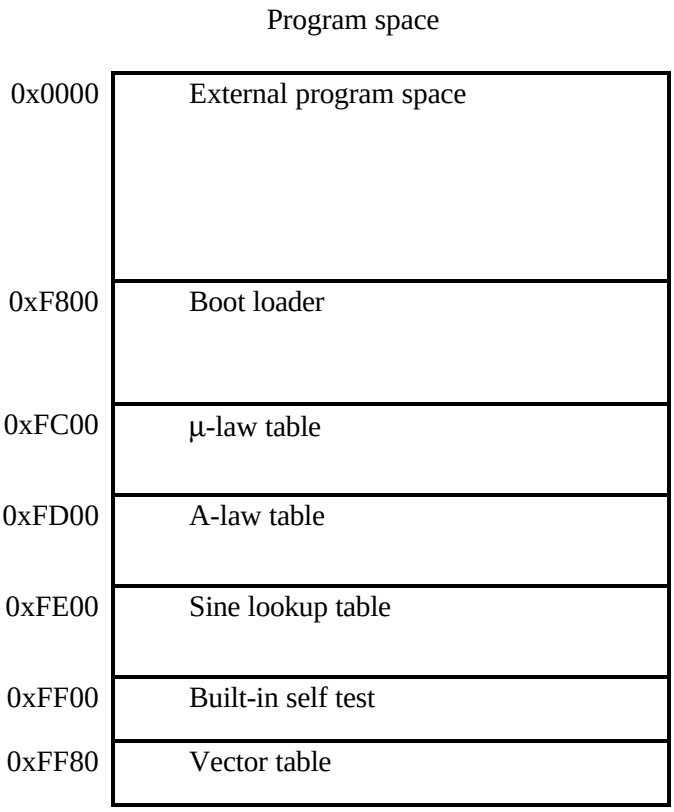
Technical Reference

1 On-chip ROM code mapping

The on-chip ROM of the C548/9 consists of :

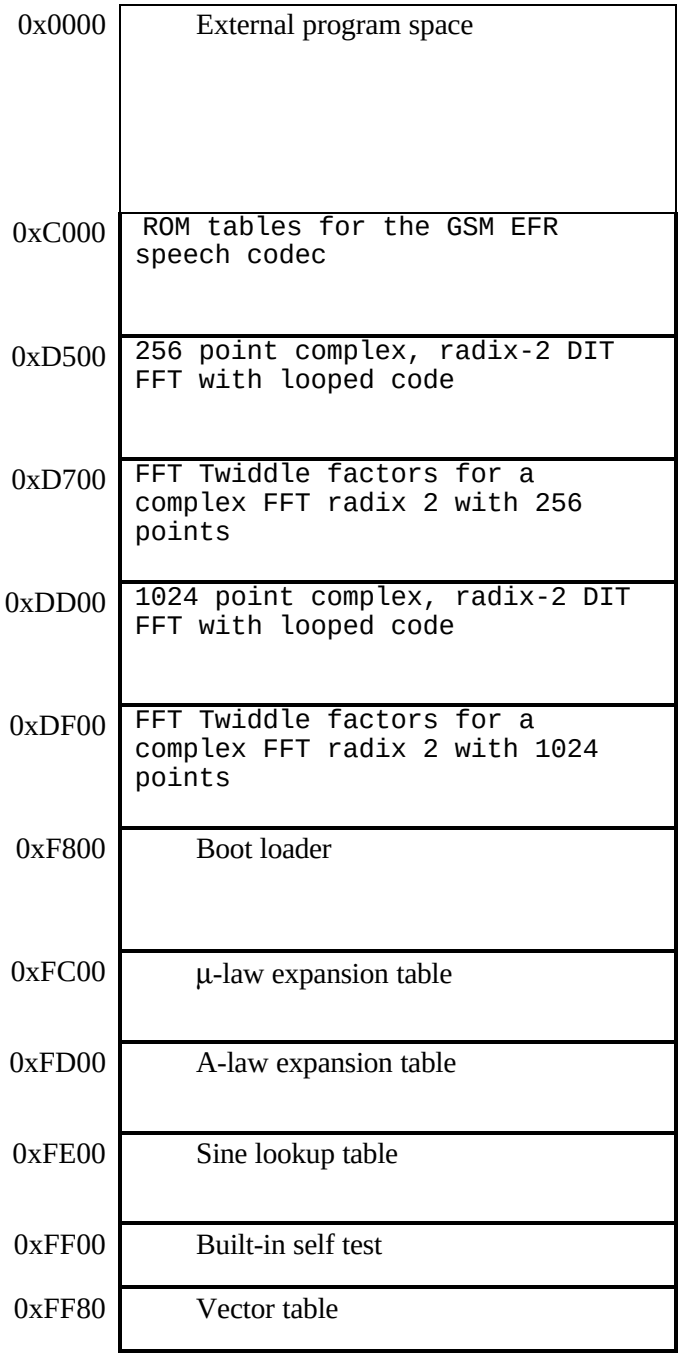
- a boot loader program which boots up from either BSP, TDM, External memory, I/O, HPI and warm boot.
- a 256 word μ-law expansion table
- a 256 word A-law expansion table
- a 256 word sine lookup table
- a built-in self test
- an interrupt vector table
- `data ROM tables for the GSM EFR speech codec (549 only)`
- `256 point complex, radix-2 DIT FFT with looped code (549 only)`
- `FFT Twiddle factors for a complex FFT radix 2 with 256 points (549 only)`
- `1024 point complex, radix-2 DIT FFT with looped code (549 only)`
- `FFT Twiddle factors for a complex FFT radix 2 with 1024 points (549 only)`

The size of this ‘C548 on-chip ROM is 2k words. It is located at 0xF800 - 0xFFFF address range in program space if MP/MC_ input pin is low.



The size of this ‘C549 on-chip ROM is 16K words. It is located at 0xC000 - 0xFFFF address range in program space if MP/MC_ input pin is low.

Program space



2 Overview

The main function of the boot loader is to transfer the user code from an external source to the program memory at power-up. The TMS320C548/9 provides several different ways to download the code to accommodate varying system requirements. For some applications, a serial interface is appropriate. If the code exists in external ROM, a parallel interface is more suitable.

If the MP/MC- pin of TMS320C548/9 device is sampled low during a hardware reset, execution begins at location FF80h of the on-chip ROM. This location contains a branch instruction to the start of the boot loader program. The on-chip ROM is factory programmed with a boot-load program.

The boot load program sets up the CPU status registers before initiating the boot load. Interrupts are globally disabled (INTM=1), internal dual-access RAM and single-access RAM is mapped in program/data space (OVLY=1). Seven wait states are initialized for the entire program and data spaces. The size of the external memory bank is set to 4K words, and one cycle can be inserted when accesses switch between program space and data space, or crossing the page boundary.

The TMS320C548/9 offers the following boot loader features :

- Parallel I/O port boot
- Serial Port boot :
    - Standard or auto-buffered mode in one of the 2 Buffer Serial Port (BSPs).
    - Standard or TDM mode in the TDM serial port.
- HPI boot
- External parallel boot
- Warm boot
- SPC and SPCE register are re-configurable for serial port boot
- Re-programmable Software Wait State
- Re-programmable Bank Switch Size
- Multiple sections boot

3 Boot Mode Selection

Boot loader mode selection is determined by the interrupt flag in the IFR register and the first word read by the boot loader. In order to be compatible with the H/W design based on current C54x device, the new boot loader still reads the source address from the I/O port 0FFFFh for parallel boot and warm boot. The boot loader will also check the data memory location 0FFFFh for source address. Then the boot loader will read first word from the source address to determine whether it is a valid code to boot and the size of the data length. The first word at the source address is 08AAh or 10AAh. The eight most significant bits indicate the memory width where source program resides (8/16 bits wide). The eight least significant bits are Bootloader Recognition Byte (BRB) which is 0AAh. If the first read is not 08AAh or 010AAh, then the boot loader will try a different boot mode. Since there is not a boot routine selection(BRS) word in the new boot loader, the BRB provides extra security to insure the boot loader for selecting the right boot mode. Figure 1 and 2 show the flowcharts of how the boot loader mode is selected. The details of the boot sequence for each boot mode will be discussed in the following sections.

The first flowchart shows that the boot loader will check the HPI boot first. Next, it will find the source address from the I/O or data memory location 0FFFFh and read from the source address. If the first word read from the source address match the BRB properly, then the boot loader will do an external parallel boot. Otherwise, the boot loader will keep checking the rest of the boot modes.

For the Serial Port Boot, the boot oader check interrupt flag INT0, INT1 and INT3 for TDM mode serial boot first. If none of these flags are present, the boot loader will check all the serial port receive interrupt flags of the 2 BSPs and the TDM serial port for standard and auto-buffering mode serial boot. If more than one flag is set, the boot loader will check the BRB read from these serial ports to distinguish which serial port to be used to boot load the code. The next sections will discuss the details about the different boot load modes.

Figure 1: Mode Selection Flow

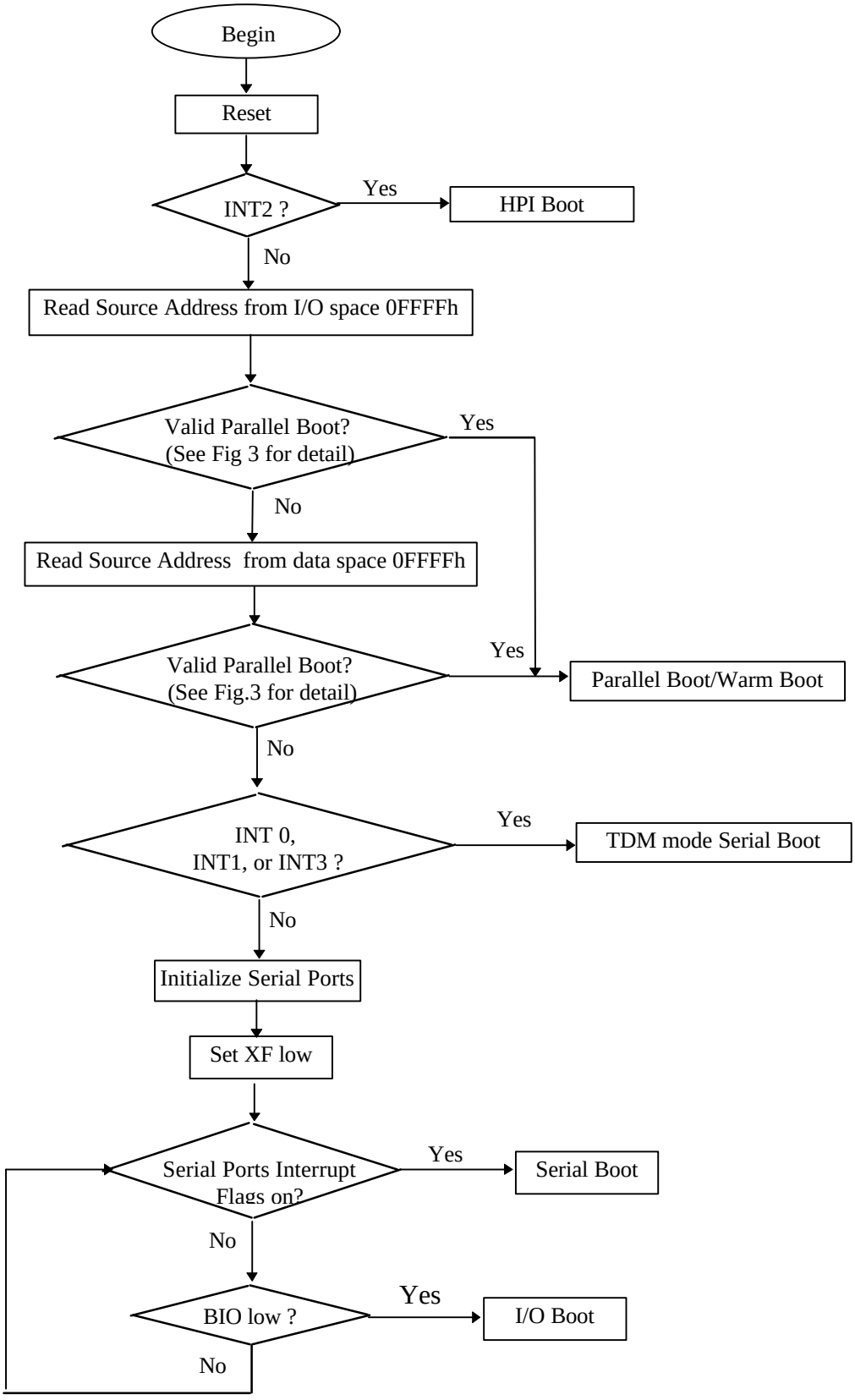


Table 1: Structure of Source Program Data Stream in 16-bit mode:

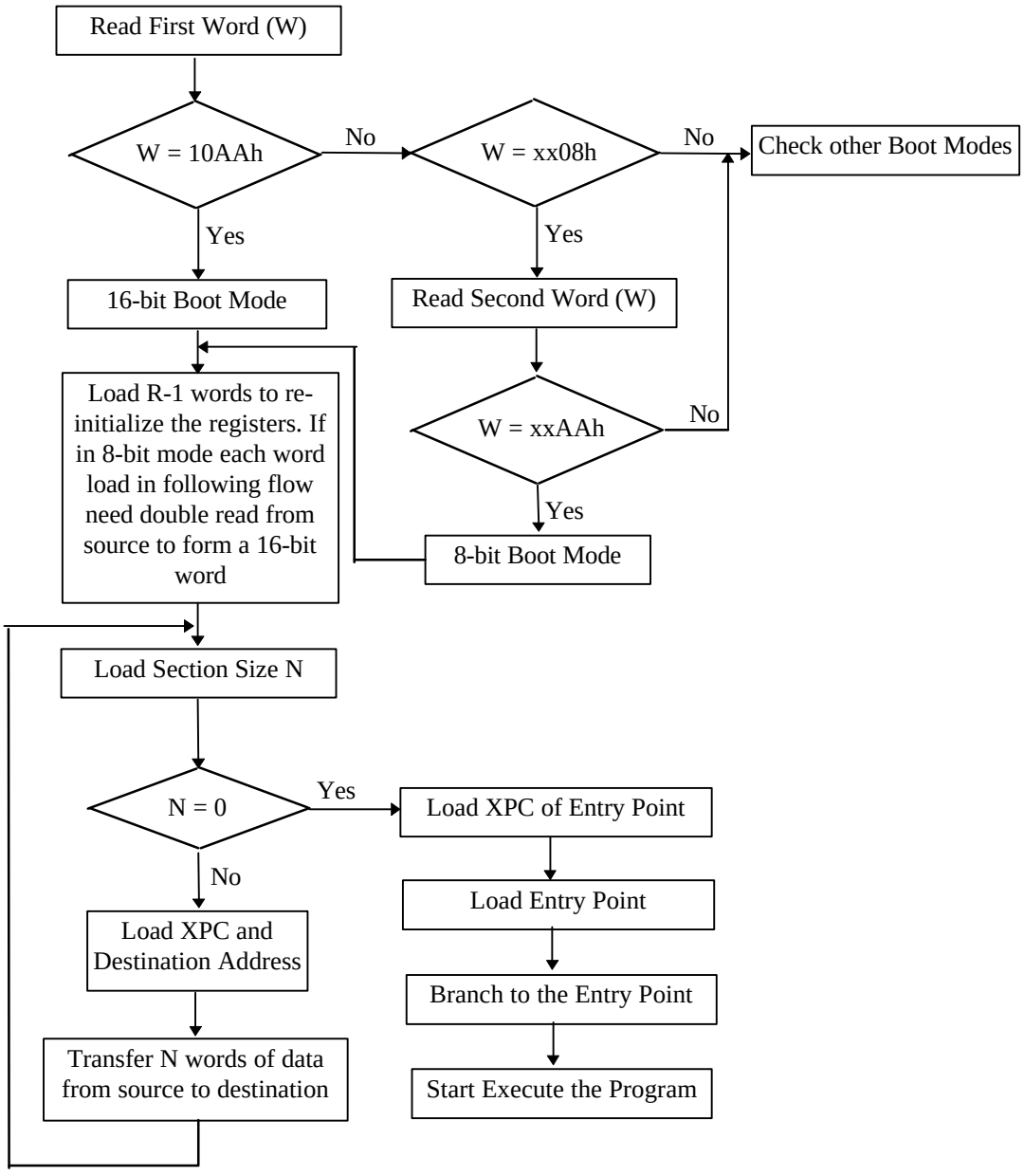
word	Contents
1	LSBs(0~7) = 0aah, MSBs(8~15) = memory width where source program resides (8, or 16 bits wide)
2	Value to set in the register
.	.
.	Value to set in the register
.	XPC value of entry point (7 bits value)
.	Entry point (16 bits)
R	Block size of the first section to load. After the number of words loaded, the next word should be zero to indicate end of source program. Otherwise, another section is assumed to follow.
R+1	XPC value of destination address of 1st section (7 bits value)
.	Destination address of 1st section (16 bits)
	First word of 1st section of source program
.	.
.	Last word of 1st section of source program
.	Block size of the second section to load.
.	XPC value of destination address of 2nd section (7 bits value)
.	Destination address of 2nd section (16 bits)
	First word of 2nd section of source program
.	.
.	Last word of 2nd section of source program
.	.
.	.
.	Block size of the last section to load.
.	XPC value of destination address of last section (7 bits value)
.	Destination address of last section (16 bits)
	First word of last section of source program
.	.
.	Last word of last section of source program
n	0000h to indicate the end of source program

Table 2. Structure of Source Program Data Stream in 8-bit mode:

Byte	Contents
1	MSB = memory width where source program resides (8, or 16 bits wide)
2	LSB = 0aah
3	MSB of value to set in the register
4	LSB of value to set in the register
.	.
.	MSB of value to set in the register
.	LSB of value to set in the register
.	MSB of XPC value of entry point (00)
.	LSB of XPC value of entry point (7 bits value)
2R-1	MSB of entry point
2R	LSB of entry point
2R+1	MSB of block size of the first section to load.
2R+2	LSB of block size of the first section to load.
2R+3	MSB of XPC value of destination address (00)
2R+4	LSB of XPC value of destination address of 1st section (7 bits value)
2R+5	MSB of destination address of 1st section
2R+6	LSB of destination address of 1st section
.	MSB of the first word of 1st section of source program
.	.
	LSB of the last word of 1st source program
.	MSB of block size of the second section to load.
.	LSB of block size of the second section to load.
.	MSB of XPC value of destination address (00)
.	LSB of XPC value of destination address of 2nd section (7 bits value)
.	MSB of destination address of 2nd section
.	LSB of destination address of 2nd section
.	MSB of the first word of 2nd section of source program
.	.
	LSB of the last word of 2nd section of source program
.	.
.	MSB of block size of the last section to load.
.	LSB of block size of the last section to load.
.	MSB of XPC value of destination address (00)
.	LSB of XPC value of destination address of last section (7 bits value)
.	MSB of destination address of last section
.	LSB of destination address of last section
.	MSB of the first word of last section of source program
.	.
	LSB of the last word of last section of source program
2n	00
2n+1	00h to indicate the end of source program

In order to support the multiple sections boot and reinitialize the related registers' value, the data stream with the source program should be in the format shown in Table 1. The contents of words R+1 through n vary for each sections in the source program loaded throughout the entire data stream. The first R words are non-variables that affect each of the source program sections. The last 2 words contain the entry point of the program. The eight most significant bits (MSBs) of the first word specify the memory width and the eight least significant bits (LSBs) of the first word will be 0AAh. If byte wide is selected, the loading sequence is MSBs first then LSBs. The size of R depends on the boot mode that is selected. The detailed structure of source program data stream will be discussed in each boot mode.

Figure 2: Boot Load Flow



4 HPI Boot

The first step of the boot loader is to check if HPI (Host Port Interface) boot option is selected. In order to do that, it asserts HINT\ (Host Interrupt Signal) low. This signal should normally be tied externally to INT2\ (External Interrupt 2) input pin if HPI Boot mode is to be selected. This will result in setting corresponding IFR (Interrupt Flag Register) bit to one. The boot loader waits for 20 CLOCKOUT cycles after asserting HINT\ and then reads IFR bit #2. If it is set to one (indicating that INT2\ interrupt is recognized), the boot loader transfers control to the start address of the on-chip HPI RAM (0x1000 in program space) and starts executing code from there. If IFR bit #2 is not set (indicating that HINT\ is not tied to INT2\), the boot routine skips HPI boot mode and goes on to set the XF pin low and reads IFR to determine the boot mode.

If HPI boot mode is selected, the host must download code to the C548/9 on-chip HPI RAM before it brings the DSP out of reset. Note that the boot loader keeps HPI in Shared-Access mode (SMOD = 1) during the entire boot loading operation. Once HINT is asserted low by the boot loader, it will stay low until a host controller (if any) clears it by writing to the host port interface control(HPIC) register. In the HPI boot mode, it boot directly to the HPI RAM, then executes the program The source program data stream should not include any extra information such as section size, registers' value. The data stream should contain only the program itself.

Instead of tying the HINT\ signal to INT2\ , you can send a valid interrupt to INT2\ input pin within 30 CLOCKOUT cycles after DSP fetches the reset vector.

Another alternative to HPI boot mode is the Warm boot option described in a later section. Warm boot option may be preferred over the standard HPI boot option if it is not convenient to tie HINT\ to INT2\. After verify there is no HPI boot, the boot loader code will check for the parallel/warm boot.

5 Parallel Boot/Warm Boot Options

If the code to be download is stored in EPROMs (8-bit or 16-bit wide in global data space) then parallel boot options ought to be selected. The code is transferred from data memory to program memory. The source address resides in the data memory location 0FFFFh. To be compatible with current boot loader, the new boot loader use the same mechanism to get the source address from the I/O port 0FFFFh. Since the source address in the data memory 0FFFFh is 16-bit data, the source program could reside in any space in the 64K range. The Software Wait-State Register (SWWSR) and Bank-Switch Control Register (BSCR) in this boot options are reconfigurable. This will provide the boot loader the capability to boot from faster EPROM with less software wait-states. The default wait-state defined by the boot loader is 7. The section size information inside the source program data stream allows the boot loader to combine the parallel boot option and warm boot option. Since there is no BSR word in new boot loader, the new boot loader will not know the memory width before it reads the address. In other words, the new boot loader needs to check both 0FFFFh (LSB source address)and 0FFFEh (MSB source address) of the address to obtain the correct source address. Figure 3 and Table 3 and 4 show the flow of the parallel boot and the source program data stream for parallel boot and warm boot in 16-bit word mode.

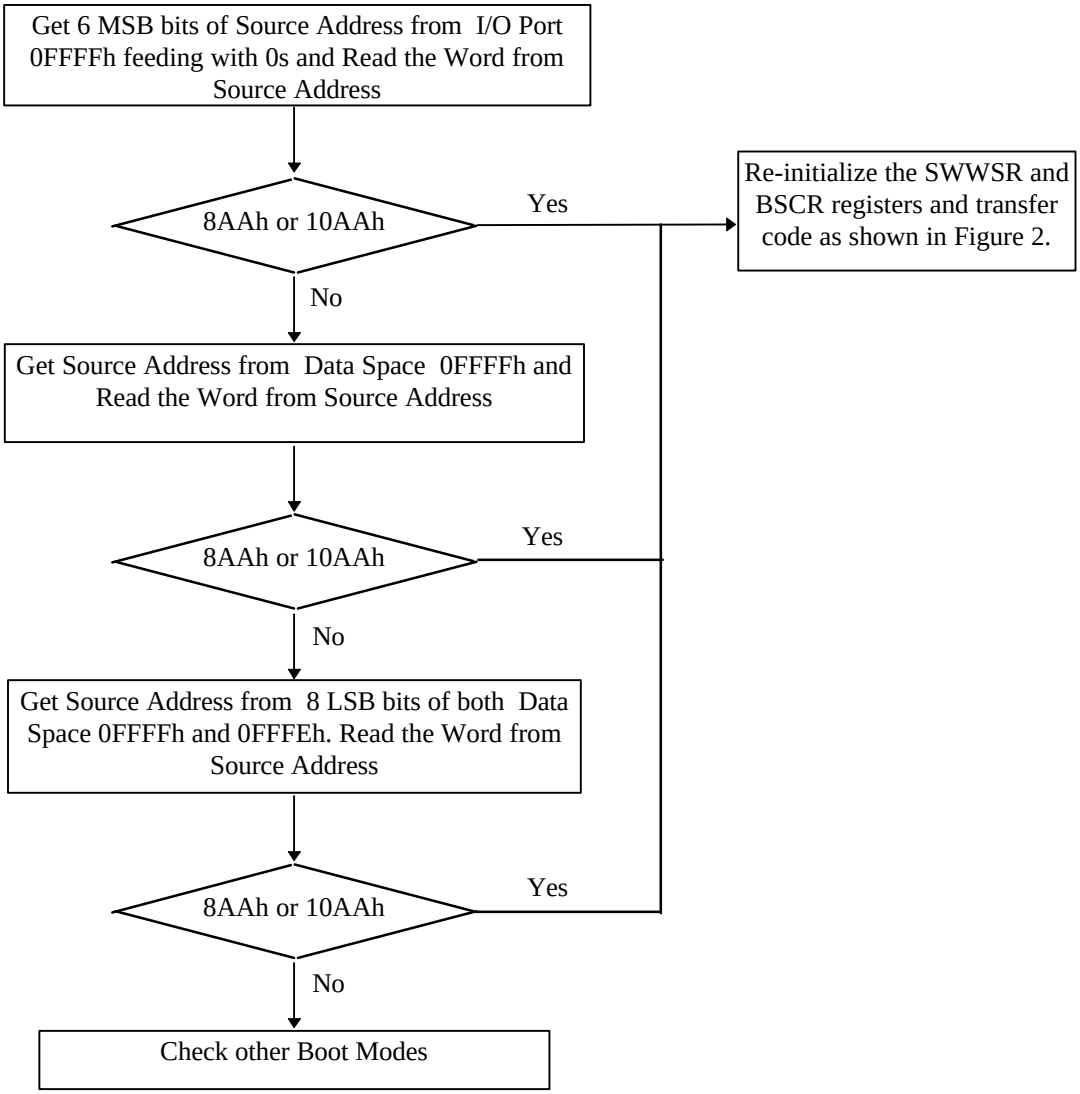
Table 3. Source Program Data Stream for Parallel Boot in Word Mode

O8AAh or 10AAh
Initialize value of $SWWSR_{16}$
Initialize value of $BSCR_{16}$
Entry $Point(XPC)_7$
$Entry\ Point_{16}$
$Size\ of\ 1st\ section_{16}$
$Destination\ of\ 1st\ section(XPC)_7$
$Destination\ of\ 1st\ section_{16}$
$Code\ Word(1)_{16}$
.
.
$Code\ Word(N)_{16}$
$Size\ of\ Mth\ section_{16}$
$Destination\ of\ Mth\ section(XPC)_7$
$Destination\ of\ Mth\ section_{16}$
$Code\ Word(1)_{16}$
.
$Code\ Word(N)_{16}$
0000h

Table 4. Source Program Data Stream for Warm Boot

O8AAh or 10AAh
Initialize value of $SWWSR_{16}$
Initialize value of $BSCR_{16}$
Entry $Point(XPC)_7$
$Entry\ Point_{16}$
0000h

Figure 3. Parallel Boot Mode Flow



6 Serial Port Boot

As mentioned in section A-2, the serial boot start with checking the interrupt flags INT0, INT1 and INT3 for TDM mode serial boot first. If none of these flag is present, the boot loader will check normal serial port boot. The following two sections will discuss TDM mode and standard/ABU mode respectively.

6-1 TDM Mode Serial Boot

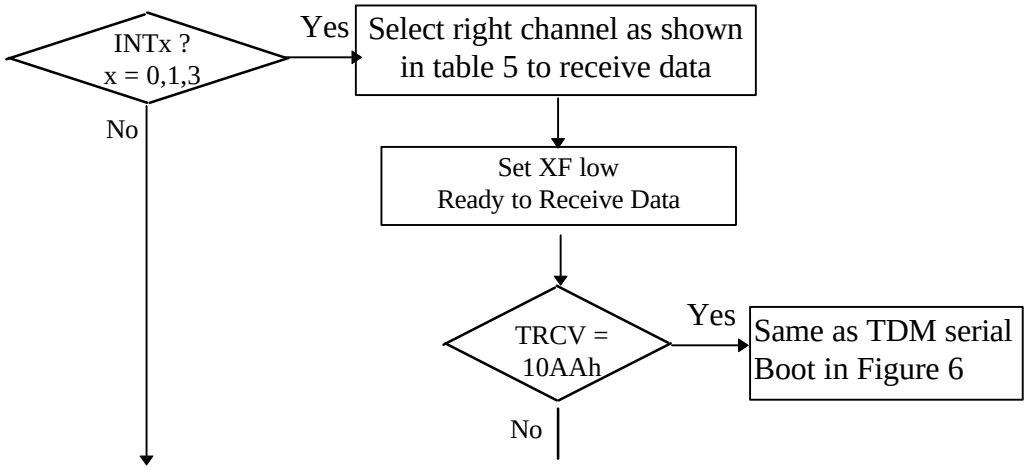
Since the signal pin connection is different between standard/ABU serial port and TDM mode serial port, the external interrupt pins are used to recognize the TDM mode serial boot and these interrupt flags are used to determine the channel to receive data. Table 5 showed the corresponding channel that boot loader uses to receive data by setup different INTx (x=0,1,3)flags. Channel 1 is set as host channel to transmit data.

Table 5. Corresponding Channel used for TDM mode boot

[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 boot loader will initialize TDM Receive/Transmit Address register (TRTA) with correct value based on the interrupt flags and set the clock as input (MCM=0) and also set the XF pin to low to indicate that the TDM port is ready to receive data. After XF pin is set low, the processor polls the RRDY bit in the Serial Port Control register (TSPC) to read data from TRCV register. Since the TDM mode only transmits and/or receives data in 16-bit mode, there is no 8-bit mode boot option in the TDM mode serial boot. Therefore, if the first word that TDM serial port receive is not 0x10AA, then the processor will skip the TDM mode boot and return to check the other boot mode. Figure 4 shows the flow of how TDM mode serial boot works.

Figure 4 TDM Mode Serial Boot Flow



6-2 Standard/ABU Serial Boot

In standard/ABU serial boot, the boot loader initializes the serial port as a standard serial port (BXE=0, TDM=0), and sets the transmit clock as output and frame pulse as input (MCM=1, TXM=0) and also sets the XF pin to low to indicate that the serial port is ready to receive data. After setting XF pin low, the processor polls the interrupt flag in the IFR register to check which serial port has data input, and reads the data from DRR to make decision and determines the data size. After the right serial port is recognized, the boot loader will read the initialized value of serial port control registers to re-initialize the serial port. Figure 5 and 6 show the flowcharts of the serial boot in detail. Since the BSP and TDM serial ports have different control registers, the source program data stream for BSP and TDM serial boot are different. Table 6 and 7 show the data stream for BSP serial boot and TDM serial boot in 16-bit word mode respectively. For 8-bit mode, every word is stored in 2 memory locations with MSB first then LSB.

Figure 5. Standard/ABU Serial Boot Flow

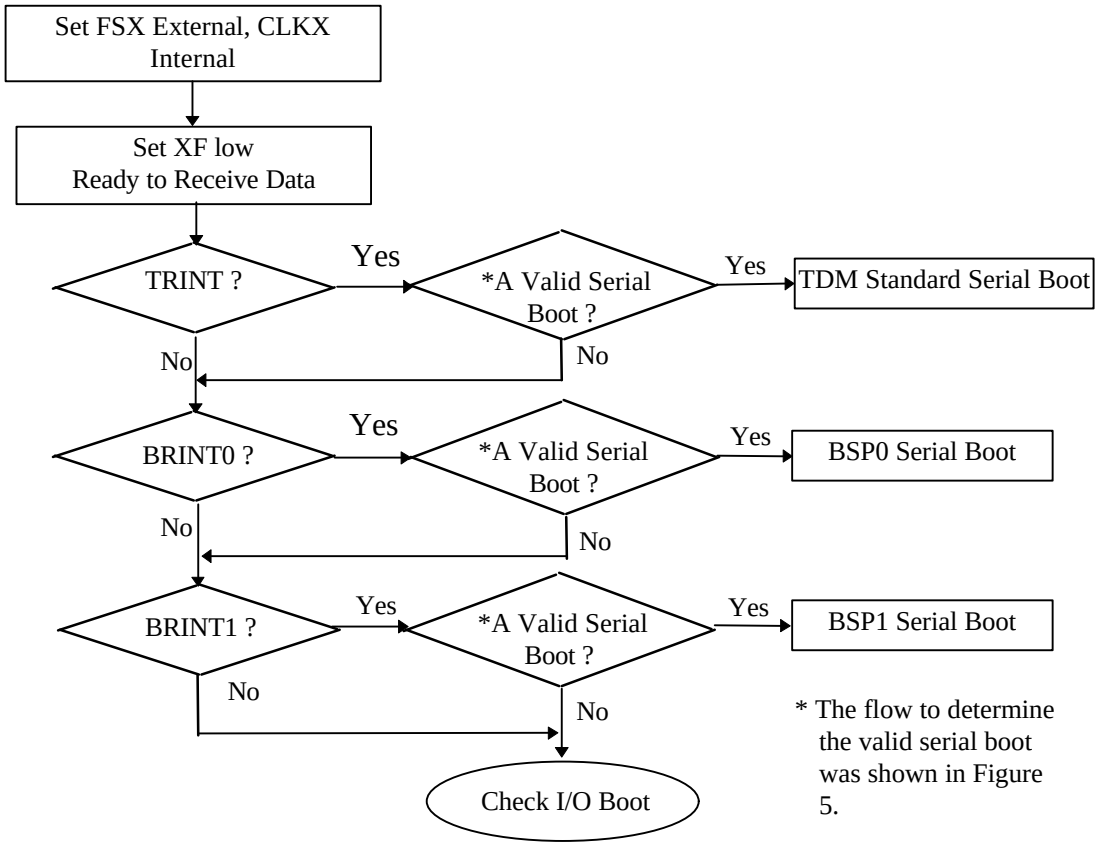


Figure 6. Serial Boot Sequence after IFR Flag is set.

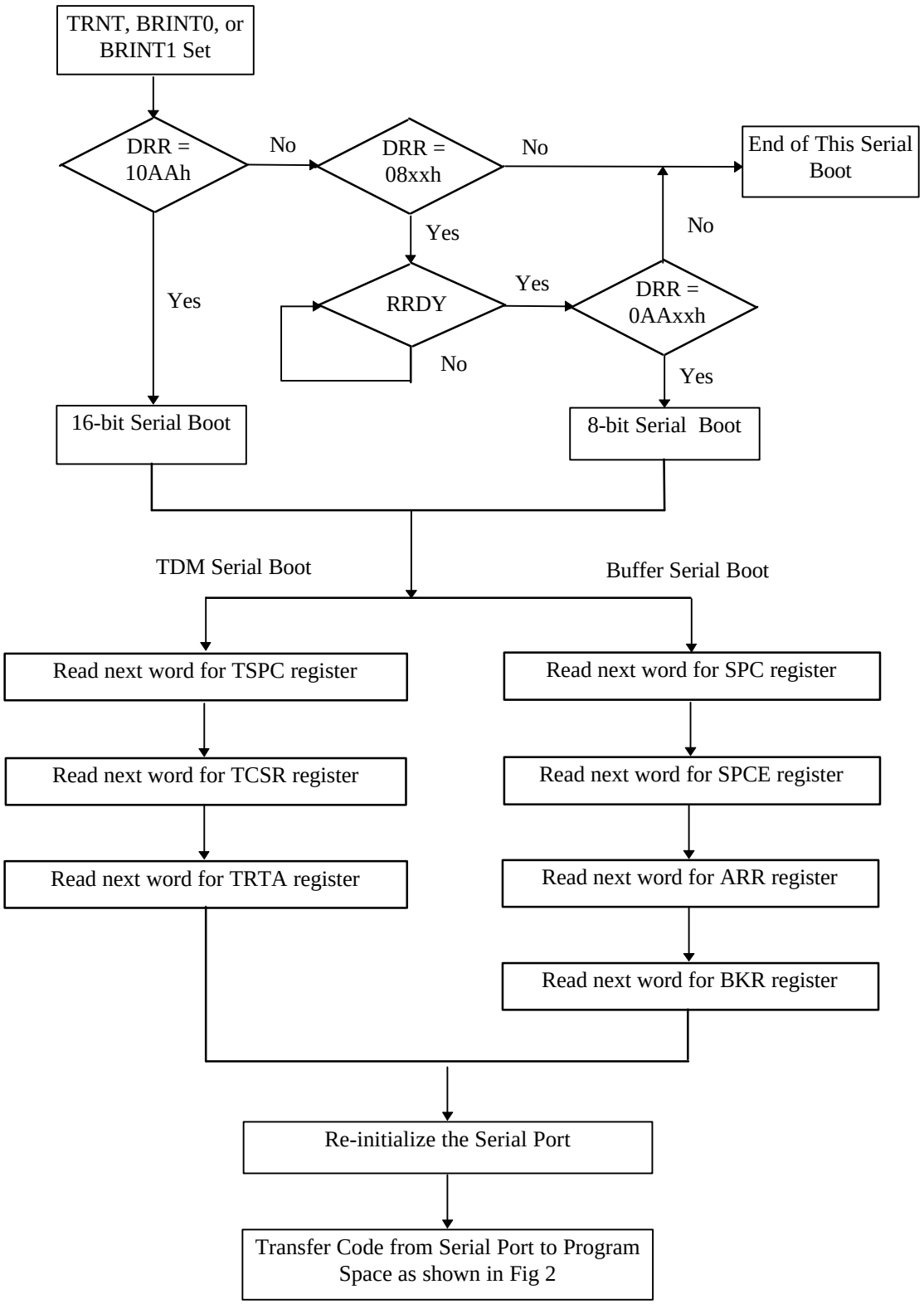


Table 6. Source Program Data Stream for 16-bit BSPs boots is:

O8AAh or 10AAh
Initialize value of SPC_{16}
Initialize value of $SPCE_{16}$
Initialize value of ARR_{16}
Initialize value of BKR_{16}
Entry $Point(XPC)_7$
$Entry\ Point_{16}$
Size of 1st $section_{16}$
Destination of 1st $section(XPC)_7$
Destination of 1st $section_{16}$
Code $Word(1)_{16}$
.
.
Code $Word(N)_{16}$
Size of 2nd $section_{16}$
Destination of 2nd $section(XPC)_7$
Destination of 2nd $section_{16}$
Code $Word(1)_{16}$
.
.
Code $Word(N)_{16}$
.
.
Size of Mth $section_{16}$
Destination of Mth $section(XPC)_7$
Destination of Mth $section_{16}$
Code $Word(1)_{16}$
.
.
Code $Word(N)_{16}$
0000h

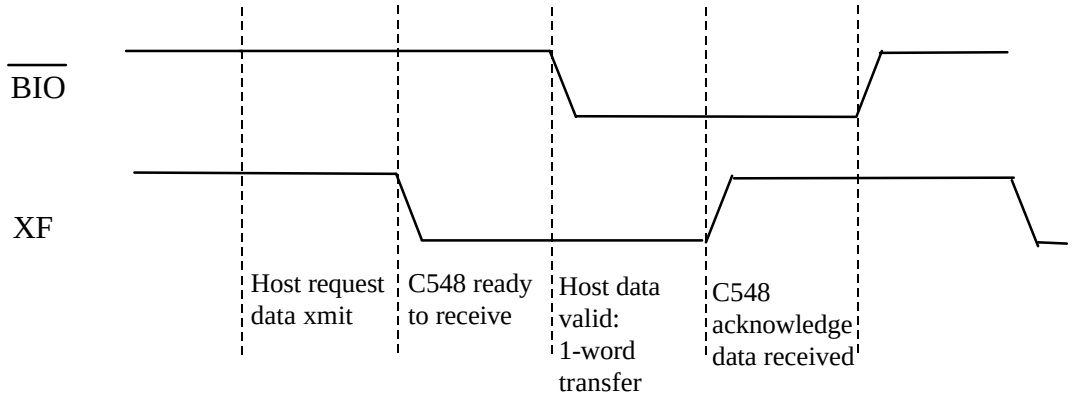
Table 7. Source Program Data Stream for 16-bit TDM boot is:

O8AAh 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 1st $section_{16}$
Destination of 1st $section(XPC)_7$
Destination of 1st $section_{16}$
$Code\ Word(1)_{16}$
.
.
$Code\ Word(N)_{16}$
Size of 2nd $section_{16}$
Destination of 2nd $section(XPC)_7$
Destination of 2nd $section_{16}$
$Code\ Word(1)_{16}$
.
.
$Code\ Word(N)_{16}$
.
.
Size of Mth $section_{16}$
Destination of Mth $section(XPC)_7$
Destination of Mth $section_{16}$
$Code\ Word(1)_{16}$
.
.
$Code\ Word(N)_{16}$
0000h

7 I/O Boot Options:

This mode is provided to asynchronously transfer code from I/O port 0h to internal/external program memory. Each word may either be 16-bit or 8-bit long. The C548/9 communicates with the external device using $\overline{\text{BIO}}$ and XF handshake lines. This allows a slow host processor to easily communicate with the TMS320C548/9 by polling/driving the XF/$\overline{\text{BIO}}$ lines. The following handshake protocol is required to successfully transfer each word via I/O address 0h:

Figure 6. XF-BIO Handshake



A data transfer is initiated by the host, driving the $\overline{\text{BIO}}$ pin low. When the $\overline{\text{BIO}}$ pin goes low, the C548/9 input the data from I/O address 0h, drives the XF pin high to indicate to the host that the data has been received, then write the input data to the destination address. The C548/9 then waits for the $\overline{\text{BIO}}$ pin to polled by the host for the next data transfer.

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 C548/9 reads two 8-bit words to form a 16-bit word. The low byte of a 16-bit word should follow the high byte.

For both 8-bit and 16-bit I/O modes, the first six 16-bit words received by the C548/9 must be the source memory width, initialized value of software wait-state register, initialized value of bank switch control register, the size of the section, the XPC of destination and the destination address, respectively. See the parallel boot section for a description of the destination and the length of the code words. 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 sufficient time to turn off its data buffers before the C548/9 initiates a write operation (in case the destination is external memory). Note that the C548/9 only accesses the external bus when XF is high.

BOOT LOADER 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, sine and                            **
**     interrupt vectors table           YGC 06/25/96        **
** 1.4 Registers re-programmable 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     **
*************************************************************

        .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 Register
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 Register
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=0
                                     ;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=FIG0
                                     ;=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=0
                                     ;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      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;warmboot
;       delay   @src                        ; dest <-- src
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                       ; store 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                  ; store 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               ; store 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                ; store A to entry
par08_1 ld      *ar1+, 8, a             ; get number of 16-bit words
        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 destination
        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 destination
        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 destination 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
;        rcd     beq
;        and     #4000h, b
        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 DRR
      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
;       ldu     @temp2, a               ; reinitialize ARR/TRTA of SP
;       stlm    a, *ar4
        orm     #00c0h,*ar2             ; take trans 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 destination 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     #0ffffh, a          ; make buffer size be multiper of 2
            and     #0fffeh, a          ; make buffer size be multiple of 2
        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_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
                                        ; CLKKV=3,FSP=CLKP=FE=FIG=0
                                        ; PCM=BXE=HLTX=BRE=HLTR=0
        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
```