SIMULATION OF APPLICATIONS FOR THE TMS320C2X FAMILY

1. Software development in assembly language for TMS320C2X processors

The present work will cover all the necessary steps for the development and simulation of applications on DSPs of the family TMS320C2X. DSP manufacturers offer at users' disposal all the software instruments necessary for application development. In order to standardize application development Texas Instruments suggests the use of the "Common Object File Format" (COFF) for the object files. The code can be split between several modules that can be separately assembled. The object modules resulting from assembly are linkedited together thereby resulting the executable. The linkeditor allocates in an efficient manner the resources for every module.

The software instruments necessary for the development of an application are: Editor, Assembler, Linkeditor, Simulator and Debugger. To develop an application in assembly language for the TMS320C2X family of processors, the steps in figure 1 are to be followed.

The .asm source file is edited with a classic text editor and assembled with the assembler ***dspa.exe***. The assembler generates an object file (.obj) and a listing file (****.lst***). The object file is linkedited with ***dsplink.exe*** having the linkediting options in the command file link25.cdm, a file that specifies the memory map of the system the application will run on. The linkeditor provides an executable (****.out***) and a file showing the memory map occupied by the application (****.map***). The code contained in the (****.out***) can be loaded in a hardware system or in the sim2x.exe simulator.

The sim2x.exe simulator is configured to run the application through a command file siminit.cmd. The simulator also permits us to attach files in the ASCII (4 characters) at the input and output ports such that the logical signal acquisition can be simulated. Software debugging through step-by-step running and the visualization of memory zones and processor registers is also possible.

2. Software utilities overview

2.1 The dspa.exe Assembler

The dspa.exe assembler is a universal assembler for signal processors in the TMS320Cxx family in fixed point. It receives ASCII format (*.asm) source files as input and it generates an object type file (*.obj).

The use of the assembler dspa.exe is realized as follows:

dspa <input file> [<object file>[<listing file>]] [-options]

- <input file> is the source file (*.asm)

- <object file> is the file that will contain the object code obtained from assembling; its specification is optional, if not specified the file will be created with the same name as the source, but with the .obj extension.
- <listing file> is the file containing the generated application listing; its specification is optional
- options that must be granted to the assembler:
  - v10- version 320C10
  - v25- version 320C25
  - v50- version 320C5X
  - l- creates a listing file (.lst)
  - s- all relates symbols will be found in the .obj file
  - w- warns about pipeline errors

Example: DSPA sin.asm-v25 -1

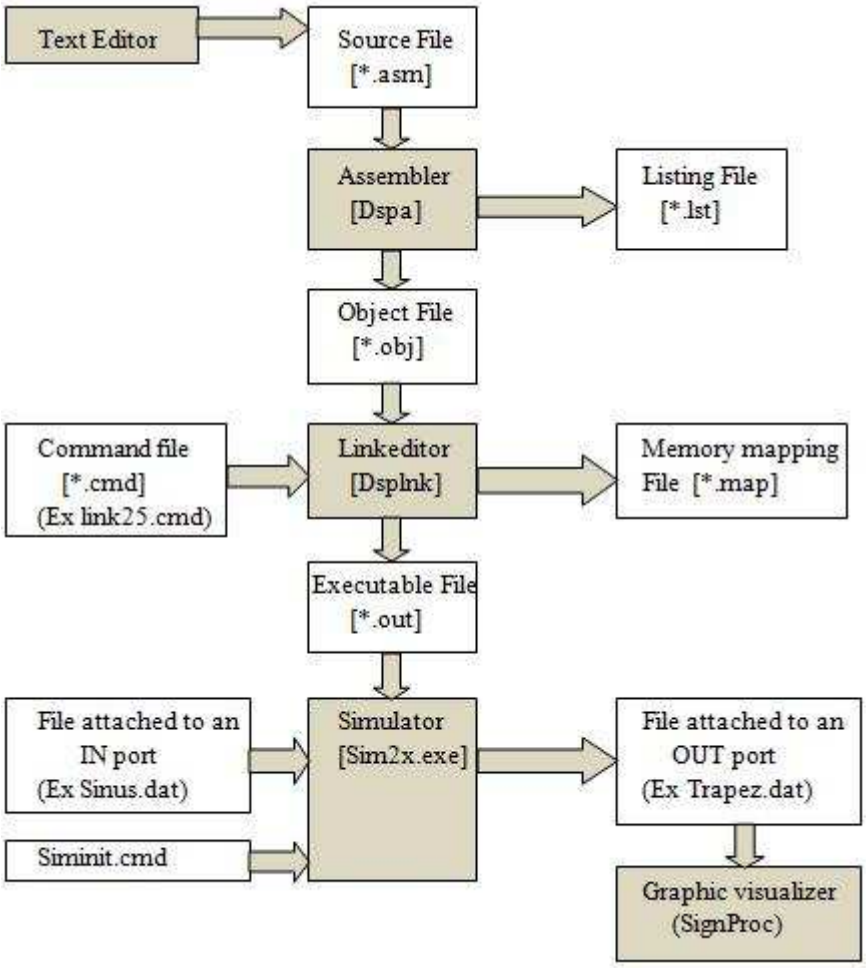

Fig. 1 Developing steps of an application for simulation

The **definition of a section** is realized based on the following syntax:

.sect <section name>

Predefined sections

.text- is the section where software instructions are defined

Instructions need to respect the following syntax:

<label>: <mnemonic> <operands>; <comment>

<label> - is optional and must begin on the first column

<mnemonic> - instruction's name

<operands> - can be missing or may be more than one

<comment> - everything in the line and preceded by ";"

If on the first column the character "*" is present then the entire line is considered a comment.

.data - This section contains the definition of the software variables.

A **variable definition** is done as follows:

<label>: **.word** <value_list>; <comment>

Memory space allocation for a non initialized variable is done as:

.bss <symbol>, <length>, <flag>

<symbol> - is the variable name

<length> - the size of the memory zone allocated to the variable (word)

<flag> - a parameter that specifies if the reserved area must or must not be continuous.

Reserving of a memory area

<symbol>**.space** <dimension>

<symbol> - is the symbol that defines the beginning address for the reserved memory

<dimension> - the dimension of the reserved area

Defining a constant

<symbol>**.set** <value>

<symbol> - constant's name

<value> - value attributed to the constant

Defining global constants recognized in other in multiple files

.global <symbol_1>, … <symbol_n>

.mmregs – permits the in-software use of symbolic names given by memory mapped registers

Software title

.title <titlu>

Software termination

.end is compulsory for the last line in the source file

2.2 The dsplnk.exe Linkeditor

The dsplnk.exe Linkeditor is configurable and can linkedit multiple object files based on a memory map described in the configuring file link*.cmd. The configuring file is specific to every type of processor (for TMS320C25 we have link25.cmd).

The linkeditor call is done as follows:

dsplnk <**options**> <**file name_1… file name_N**>

- m <file name>.map – creates a file containing memory mapping based the linkediting
- o<file name>.out – specifies the creation of an executable (.out)

Example: dsplnk sin.obj link25.cmd –m sin.map –o sin.out

sin.obj – this is the object software resulted from assembly

link25.cmd – is the command file of the linkeditor that contains the memory map and the addresses at which the linkeditor will put the different sections of the executable. A file example for LINK25.CMD is presented in Annex 1.

2.3 The sim2x simulator

The **sim2x** simulator has a user-friendly interface with windows and menus. It can be operated with a mouse and keyboard. The simulator offers the following facilities:

1. Configuration of the software memory and the data memory
2. Visualization and modification of the disassembled software memory
3. Saving/loading of memory zones in/from files
4. File visualization in the simulator window
5. Visualization and modification of data memory
6. File connection to I/O ports (I/O simulation of sample signals)
7. Visualization and modification of processor registers
8. Software execution
9. Step-by-step software running

The execution of the simulator is done using the command:

sim2x <file.out><options>

where:

-<file.out> is the file generated after linkediting the object file (.obj) by dsplink.exe

-<options> is a list of optional parameters that can transmit additional information to the simulator

These options are:

-i <path name> specifies the search path of the simulator

-mv <version> specifies the processor version

(e.g. –mv25 for sim2x processor C25)

-s- loads the symbol table

-v- loads without the symbols table

-t- <file name> identifies a new configuration file that loads instead of the siminit.cmd file

The sim2x simulator runs under DOS OS in text mode. The software's main window shows the command bar with accepted commands and four work purposed sub-windows.

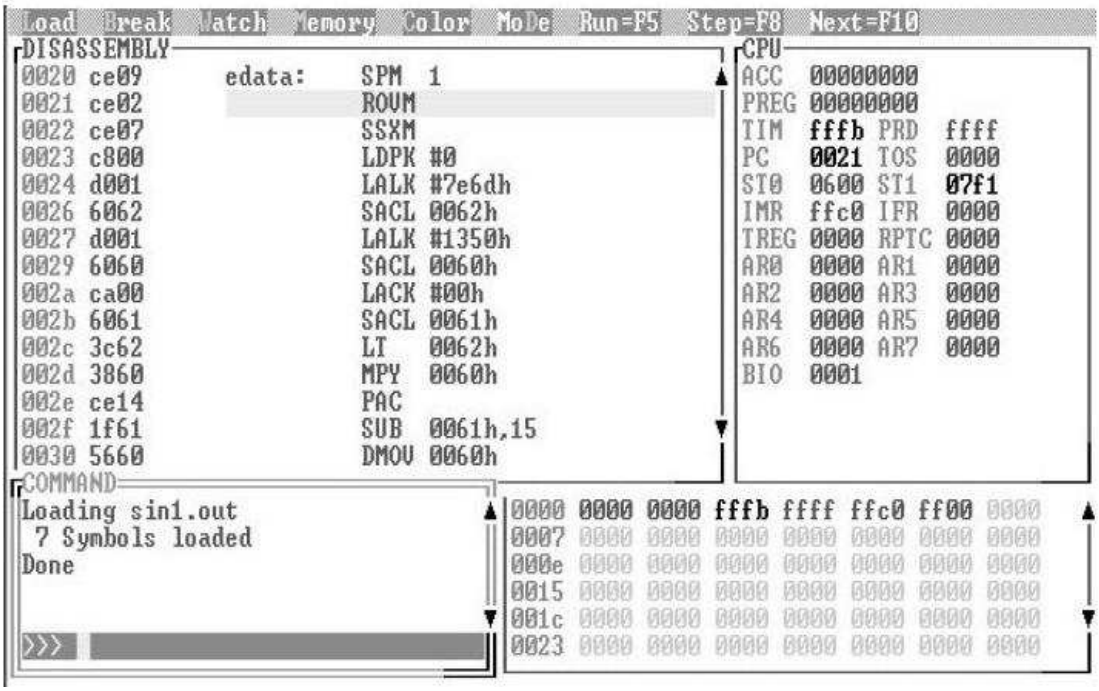

Fig. 2 sim2x Simulator's window

The Disassembly window (DISASSEMBLY)

It contains the disassembled code from the object code loaded in the simulator. The disassembled code is shown in the following format:

|hexa 4 digit address|hexa 4 digit object code|label|assembly language code|
e.g. 0020 | bf00 |edata:| SPM 0 |

The memory date window (MEMORY)

It shows the content of the data memory as follows:

|hexa 4 digit address|hexa 4 digit memory content|
e.g. 0023 |**0000 FFFF FFFF 0000 0000 0000 0000**|

Values that are modified due to running the software are shown intensely. Using the **mem** command we can go quickly in the memory window and specify the start address of the displayed area. The words in the memory zone that are not contained in the simulator configuration are displayed in red.

The processor registries window (CPU)

It displays the content of CPU registers in hexa 4 digit format. Values that are modified due to software running are shown in intense white.

The command window (COMAND)

Allows command entry using the prompter ">>>" and the display of the output messages of the simulator. The window retains the last 50 commands. Scrolling the retained commands can be done using "TAB + back arrow" and "SHIFT + TAB + forward arrow".

Commands that can be launched using the prompter in the command bar:

- **take <command file name>** It allows launching of a command file.
- **system <DOS command>** It allows launching of DOS commands without leaving the simulator. Return to the simulator window is done by pressing Enter.
- **Quit** allows simulator exit.
- **MA Address, Type, Length, Allocation Type**
  MA = memory add
  Address = the start address of the memory zone
  Type = memory zone type
  0 = program memory
  1 = data memory
  2 = I/O space
  Length = length of memory block
  Allocation Type = RAM, OPORT/INPORT

  The command allows the adding of a memory zone in the simulator's configuration.

- **MC Address, Type, File, Opening Parameter**
  MC = memory connect
  Address = the start address of the memory zone
  Type = memory type zone
  0 **=** program memory
  1 = data memory
  2 = I/O space
  File = file name
  Length = length of memory block
  Allocation Type = RAM, OPORT/INPORT
  Opening parameter = Read/Write

  This command allows the connection of INPORT, OPORT, IOPORT type of memories to a file.

- **MR** (memory reset) allows erasing all the settings referring to the memory in the simulator configuration.
- **MD Address, Type, Length, Allocation Type**
  MD = memory delete
  Address = start address of the memory zone
  Type = memory zone type
    0 = software memory
    1 = data memory
    2 = I/O space
  Length = length of memory block
  Allocation Type = RAM, OPORT/INPORT

  The command enables memory erase of the INPORT/OPORT/IOPORT type from the simulator's configuration.

- **MI Address, Type, File, Opening Parameter**
  MI =memory disconnect
  Address = start address of the memory zone
  Type = memory zone type
    0 = software memory
    1 = data memory
    2 = I/O space
  Length = length of memory block
  Allocation Type = RAM, OPORT/INPORT
  Opening Parameter = READ/WRITE

  The command enables memory disconnection of the INPORT/OPORT/IOPORT from a file.

Command bar

Is arranged in the upper part of the window and besides usual commands, it allows entering bulk commands in the command window. The usual commands are:

a. **Software load.** Using the **Load** command we can type inside the window the file name and the OK command is entered.
b. **Software running.** To run the software, F5 will be pressed.
c. **Cancel the software run.** To do this, ESC will be pressed.
d. **Step-by-step run.** F8 is pressed for the step-by-step run. This method functions for the instructions inside the called functions as well.
e. **Instruction-by-instruction run.** When pressing F10 the software is ran instruction-by-instruction without entering the called functions.
f. **Breakpoint addition.** (stopping the software run at a desired moment). Right click on the instruction where the breakpoint is desired to be placed.
g. **Breakpoint erasure.** Right click on the instruction that contains the breakpoint.

Simulator's configuration

Configuration of the simulator is done by the implicit loading of the siminit.cfg file at start. Loading a different configuration file instead of the implicit file can be made with the option –**t**(take) when launching the simulator.

The siminit.cfg file contains commands that are accessible at simulator's prompter too. If simulator's reconfiguration is desired, another configuration file from the prompter is loaded by using the **take** command.

Example of configuration:

```
;Program memory
MA 0x0000 , 0 , 0x0020 , RAM          ; INTERRUPTION VECTORS
MA 0x0020 , 0 , 0x0F00 , RAM          ; ROM
MA 0x0FB0 , 0 , 0x0050 , RAM          ; RAM
MA 0x1000 , 0 , 0x0400 , RAM          ; OFF CHIP

;Data memory
MA 0x0000 , 1 , 0x0006 , RAM          ; MMR
MA 0x0060 , 1 , 0x0020 , RAM          ; B2
MA 0x0200 , 1 , 0x0400 , RAM          ; B0, B1
MA 0x0800 , 1 , 0x0800 , RAM          ; OFF CHIP

;Ports
MA 0x0000 , 2 , 0x0001, OPORT
MC 0x0000 , 2 , SINUS.DAT , WRITE
```

It is very important that the configuration of the simulator to correspond with the one of the linkeditor. If this is not done, the application cannot be run on the simulator. Usually a configuration file is constructed for each application.

2.4 The graphical viewer for "SignProc" files

The viewer ("the oscilloscope") is a program that allows visualization of data files (.dat) on 16 bit integer format on a graphical form, files which are "connected" to the simulator as input or output signal.

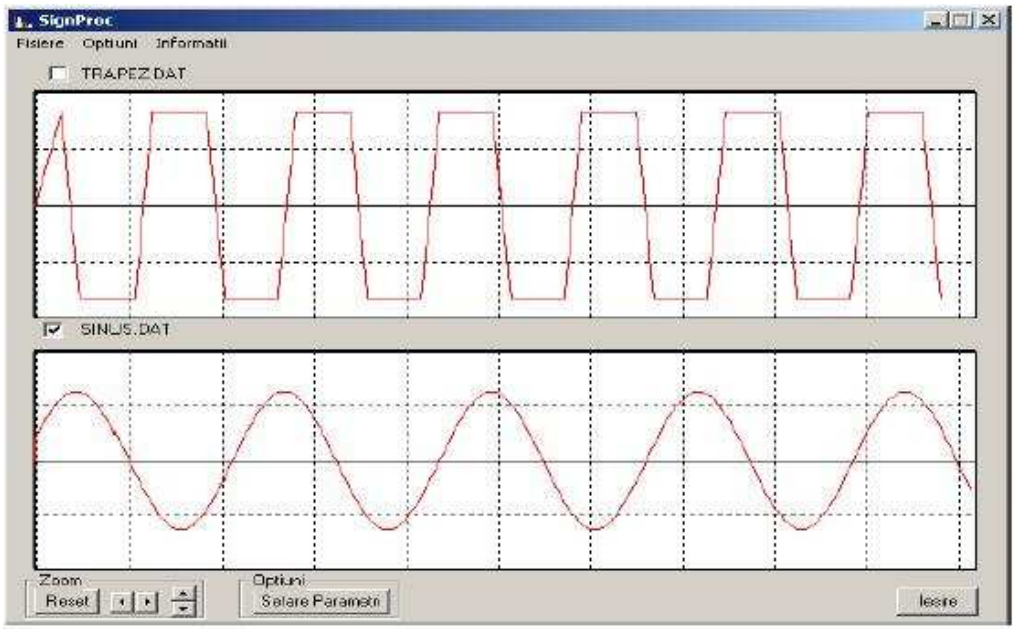

Fig. 3 Graphical viewer for files (.dat) "SignProc"

The program has 2 graphical windows that can display 2 different files. Choosing a file for display is done from the **Files** menu by choosing **Open**. Saving the file under a different name is done by selecting the option **Save** from **Files** menu. Modifying the parameters is done with the help of the Zoom button group placed on the bottom of the main window.

3. Application

The proposed application generates a trapezoidal signal out of a triunghiular signal which is amplified in order to be limited after like in figure 4.

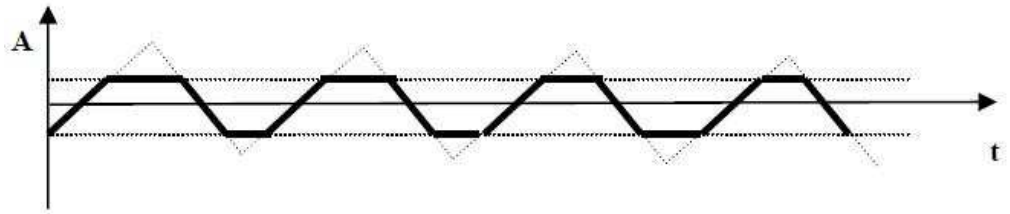

Fig. 4 The trapezoidal signal as a triunghiular limited signal

The following listing represents the trapez.asm application:

```
; Declaration of variables
maximn .set  8000h      ; the most negative number (=-1, in Q15)
pas    .set  400h       ;step between 2 consecutive samples
const  .set  4000h      ;constant = 0.5 in Q15
; Memory space is reserved for variables
       .bss  x , 1
       .bss  y , 1
       .bss  z , 1
       .bss  p , 1
       .bss  m , 1
       .bss  ct , 1
; OVM (overflow) bit is set from ST0
       begin  sovm
; DP register is initialized with the first 9 bits (most significant ones) of the x number. The x variable is the address of the first reserved word. Therefore DP will be positioned in such way that will indicate the page where the reserved memory locations are to be found through .bss
       ldp    #x
; x and m variables are initialized with the most negative value
       lalk   #maximn
       sacl   x
       sacl   m
; p variable is initialized with the step value between 2 samples
       lalk   #pas
       sacl   p
; ct variable is initialized with 0.5 in Q15
```

```
        lalk    #const
        sacl    ct
; The auxiliary register AR7 is loaded with the value 0. This register will be used to count the
; number of generated output samples. This will only help in the simulation process (we have
; the information about the number of generated samples at any moment of time); it has no    ;
; other role in the program
        lark    ar7,#0
; The auxiliary register AR7 becomes the current auxiliary register for indirect addressing
        mar     *,ar7
; Generating signal loop
; We compute: y = |x| - 0.5:
; The content of memory location with the x address is shifted towards left with 16 positions ;
; in order to place the value in ACCH and the absolute value is computed
loop    lacc    x,16
; Absolute value is applied to the content of ACC; the result will also be in ACC
        abs
; The content of the memory location with the address ct shifted towards left with 16 positions
; is subtracted from ACC
        sub     ct,16
; ACC is saved at the memory location with the address y
        sach    y
; We compute: z = 2y = 2(|x| - 0.5) = 2|x| -1
; The content of the memory location with the address y shifted towards left with 16 positions
; is added twice to ACC
        add     y,16
        add     y,16
; ACCH is saved at the memory location with the address z
        sach    z
; The previous address content memory is sent to port 0
        out     z,0
; OV bit from ST0 is reset (if an overflow occurs this bit will be 1 – it comes back to 0 only
; when an interruption is received or a conditional jump instruction is executed by that bit)
Eti     bv      eti
; ACC is loaded with  lac     x,16
; We compute: x = x + p where p is the x's increment step as long as an overflow doesn't
; occur: the content of the memory location with the address p shifted towards left with 16
; positions is added to ACC
        add     p,16
; If an overflow occurs a jump to the specified location is executed or else it continues with
; the execution of the next instruction
        bv      et
; ACC is saved in the memory location with the address x
        sach    x
```

```
; Unconditional jump at the specified address
        b       et1
; ACC is loaded with the content of the memory location with the address m shifted towards
; left with 16 positions
et      lac     ,16
;  ACC is saved in the memory location with the address x
        sach    x
; The current auxiliary register is incremented, the output sample counter is updated
et1     mar     *+
; The generating signal loop is closed through unconditional jump at the start address. The
; signal will be sent towards the port 0as long as ESC key is not pressed
        b       loop
.end
```

4. The course of the paper

a) Study the usage of the assembler (**dspa**), the linkeditor (**dsplnk**) and the simulator (**sim2x**).
b) Carefully study the application.
c) Lunch in execution the trapez.bat commands file presented below:

```
@echo off
dspa -v25 -1 -x trapex.asm
dsplnk -ar trapez.obj link25.cmd -m trapez.map -o trapez.out
sim2x trapez.out
signproc.exe
```

d) Run the program step-by-step by pressing the F8 key.
e) After running ~60 times the loop that generates the trapezoidal signal, press **quit** in the command line of the simulator in order to abandon it.
f) Lunch the graphic viewer and read trapez.dat file.

5. Homework

a) Modify trapez.asm so that it generates a triunghiular signal. In order to do that we don't amplify the signal anymore, so it fits between the limits.
b) What do we have to modify in the program that generates a triunghiular signal in order to generate a ramp signal?

LINK25.CMD file structure

Linker Command File

Microcomputer Mode MC/MP = 1

- sets the Microcomputer/Microprocessor mode

1K RAM block mapped into program space

RAM, OVLY bits = 1,0

Block B0 configured as program memory

ST1 – CNF BIT = 0 MEMORY

- the system's memory map setting

{

PAGE 0 : /* Program Memory

- Program Memory (Page())

VECS : origin = 0h, length = 20h interrupts

- sets the memory location where interruption vectors are

PROG_ROM : origin = 20h, length = 0F90h ROM

- sets the program location from the ROM memory

PROG_RAM : origin = 0FB0h, length = 0050h RAM/*

- sets the program location from the ROM memory

/* sets the progra location from external memory

EXT_PROG : origin = 1000h, length = 0C400h off-chip

PAGE 1 : /* Data Memory (Page 1)

/* sets the memory location where the registers mapped in mem are

REGS : origin = 0h, length = 6h

/* memory block 2 localization

BLK_B2 : origin = 60h, length = 20h Block B2

/* size and localization of internal RAM setting

INTL_RAM : origin = 200h, length = 400h Block B0 & B1

/* size and localization of external RAM setting

EXT_DATA : origin = 800h, length = 0F800h off-chip

}

SECTIONS – setting section specific to COFF format

{

```
.vectors: {} > VECS PAGE 0          /* section.vectors in Page 0
.text: {} > PROG_ROM PAGE 0         /* section.text in Page 0 in internal ROM
                                    /* memory
.data: {} > PROG_ROM PAGE 0         /* section.data in Page 0 in internal ROM
                                    /* memory
.bss: {} > EXT_DATA PAGE 1          /* section.bss in Page 1 in internal ROM
                                    /* memory
```

}