

StarCore PortASM/56600

Application Note


by
Kim-Chyan Gan

AN1845/D
Rev. 0.1, 04/2000



StarCore and MFX are trademarks of Motorola, Inc.

This document contains information on a new product. Specifications and information herein are subject to change without notice.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer. All other tradenames, trademarks, and registered trademarks are the property of their respective owners.

How to reach us:

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

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

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

Customer Focus Center: 1-800-521-6274

Mfax™: RMFAX0@email.sps.mot.com

–TOUCHTONE 1-602-244-6609
–US & Canada ONLY 1-800-774-184
–<http://sps.motorola.com/mfax/>

HOME PAGE: <http://motorola.com/sp>

Motorola DSP Products Home Page: <http://www.motorola-dsp.com>

© Copyright Motorola, Inc., 2000

Abstract and Contents

This application note describes how the StarCore PortASM/56600 application translates DSP56600 programs into SC140 programs that the StarCore assembler and linker can then assemble. This document consists mainly of an example that details the process of porting a 56600 application to the SC140 platform.

While the PortASM/56600 saves developers time with porting applications, it increases the code size of the ported program. However, due to the superiority of the SC140 architecture, the speed of the ported program is not affected.

PortASM's code translation is nearly automatic. As the example test case demonstrates, working SC140 code is successfully produced with little effort from the programmer. Of course, the translated SC140 code should not be expected to equal hand-crafted assembly code. Nonetheless, the translated code provides a useful starting point for a fully optimized SC140 application.

- 1 Introduction** 1
- 2 Porting a DSP56600 Application to StarCore** 1
 - 2.1 Time Requirements 1
 - 2.2 Running GSM FR on StarCore Platform 2
 - 2.3 Debugging 4
- 3 Performance of Ported Program** 5
- 4 Optimizing** 6
- 5 Performance of Optimized Program** 8
- 6 Conclusions** 8

1 Introduction

The PortASM/56600 for StarCore application, developed by MicroAPL, is a tool that translates assembly programs written for the 56600 platform to the SC140 platform. The translated programs are then assembled using the StarCore assembler and linker. PortASM is capable of translating DSP56600 programs written partly in assembly and partly in high-level languages such as C or Pascal. In this case, the StarCore compiler is used to compile the high-level portions of the program.

PortASM can be integrated into the normal build process using standard MAKEFILES. Thus, one source of code can be ported for both the 56600 and the StarCore platforms.

In the translated code, the original DSP56600 assembly source code is retained as a comment. Thus, it is easy to check what the translator produces for each 56600 assembly instruction.

This document consists mainly of an example of porting a 56600 application to SC140. The procedures of porting with the PortASM program are presented in Section 2. Section 3 shows the performance of the ported application on the SC140 platform before optimization. Section 4 outlines methods for optimizing the code, and Section 5 shows the ported application's performance after those methods are applied. Section 6 contains conclusions.

2 Porting a DSP56600 Application to StarCore

The application used in this example is the GSM Full Rate (FR) Vocoder.

2.1 Time Requirements

Table 1 shows the amount of time that was required by an engineer who was not familiar with SC140 to port the GSM FR application from the 56600 platform to the SC140 platform. The actual time required to port an application may vary, depending on the complexity of the program and the experience of the engineer.

The GSM FR application contains 15 assembly files and 5709 lines of assembly code, including comments.

Table 1. Time Required to Port 56600 Application to StarCore Platform

Task	Effort (Days)
Running GSM FR on StarCore Platform	2
Debugging and Testing	8
Optimizing	3
Total Effort	13

2.2 Running GSM FR on StarCore Platform

To enable PortASM to translate the 56600 program so that it can run on the StarCore platform, the programmer modified the GSM FR program. An outline of the particular modifications used in this test case follows:

1. In GSM FR on the 56600 platform, absolute address mode was changed to relative address mode. This modification simplifies mapping between 56600 and StarCore memory. The absolute addresses are usually found where the ORG directive locates. They are also found in the program where a particular memory location is specified.

The use of absolute addressing in the 56600 code was a programming choice made by the original developer. It is always a good programming habit to limit the occurrence of absolute addresses to definitions of constants. See Example 1.

Example 1. Changing Addressing from Absolute Mode to Relative Mode

```
;          org p: $40 ;absolute address
          org p:

          xdef store ;define 'store' constant
store     ds 1
;          move x1,x:$cf55 ;the occurrence of absolute address in the program
          move x1,x:store ;delete the absolute address by definition of const.
```

After changing addressing in the 56600 program to relative mode, the programmer re-ran the GSM FR program and verified its output to be correct.

2. The original MAKEFILE was changed, and a project file was created. The new MAKEFILE was given the name Makefile.sc. In Makefile.sc, first the PortASM application translates assembly files, and then assembling and linking occur on the SC140 platform. See Example 2. The project file (named Encoder.prj) is used by the PortASM application to keep track of the PortASM command-line options and to keep track of which source files are linked together to compose the application.

Example 2. Makefile.sc

```
PORTASM = pasc
PAOPTIONS = -project encoder.prj -o obj/
ASM = asmsc100
ASMOPTIONS = -g -dARCH "'56600'" -llst/$*.lst -bobj/$*.cln

#####
# Assembling translated files          #
#####
obj/%.cln:          obj/%.s
          $(ASM) $(ASMOPTIONS) obj/$*.s

#####
# Translating files                    #
#####
obj/%.s:            %.asm
          $(PORTASM) $(PAOPTIONS) $*.asm
```

3. The errors generated by PortASM and the assembler were removed.
 - a) Error 12105: Defining a label through EQU can cause an incorrect translation. A real label was used instead. See Example 3 on page 3.

Example 3. Resolving Error 12105: Using a Real Label

```

;dec_drp      ds 160
;dec_drp120   equ dec_drp
;dec_drp80    equ dec_drp+40
;dec_drp0     equ dec_drp+120
dec_drp120    ds 40
dec_drp80     ds 80
dec_drp0      ds 40

```

- b) The SCP (assembler built-in functions) is ignored by PortASM. The modifications that were made are shown in Example 4.

Example 4. PortASM Ignores SCP

```

;          if @scp("ARCH", '56600')
;          if (ARCH==56600)

```

4. The startup code was placed in the main program (Encoder.asm). See Example 5. The initialization code does the following:
- Creates a stack
 - Allocates additional memory to store N0–N7 and M0–M7
 - Uses the linear addressing mode
 - Initializes N0 to hold the value Y_MEM_OFFSET/2

Example 5. Initialization Code

```

;          if @def(StarCore_version)
;          asm
;          section spill_section
;          global spill_block
;          align 4
spill_block    ds      100
;          endsec
;
;          section starcore_stack
;          global stack_base
;          align 4
stack_base     ds      256
;          endsec
;          endasm
;          endif
;
;          if @def(StarCore_version)
;          asm
;          global ProgramStart
ProgramStart
;          move.l #Y_MEM_OFFSET/2,n0 ; Y memory offset reg
;          move.l #stack_base,r0
;          tfra r0,sp ; Create stack
;          move.l #0,mctl ; All memory linear
;          bsr enco_start
;          endasm
;          endif

```

2.3 Debugging

Although the translated program ran on the StarCore platform, the output of the GSM FR program was not correct. The following bugs were identified:

1. Before a function is called, PortASM analyzes how many hardware loops have been used. If the total number of required hardware loops is more than four in the current routine and sub-routine, PortASM saves the hardware loop register before the function is called. Saving the SR register will not clear the SLF, LF3, LF2, LF1, and LF0 bits in the SR register, regardless of whether these bits are set or clear. If these bits are set and a new loop is executed, the program will execute the wrong path. The new loop's starting address is saved in register SA0 by the translated program. The program, however, loops to the starting address SA3 if LF3 is not clear. The SLF, LF3, LF2, LF1, and LF0 bits must be cleared explicitly to make the program run correctly. See Example 6.

Example 6. Identifying Saved Hardware Loop Data and Clearing SLF and LFx Bits

```
push lc0
push sa0
push lc1
push sa1
push lc2
push sa2
push lc3
push sa3
push sr
bmclr #$ff00,sr.h ; ===== Add this line
jsr rp_update
```

If a PortASM translation of any 56600 program includes this bug, you can use the following general procedure to find and fix the bug: Search for the instruction `PUSH SR` in the translated assembly files. If any hardware loop data is saved, clear the corresponding LFx bit.

2. There is an ASLL operation bug. See Example 7.

Example 7. ASLL Translation Bug

```
;      asl      #2,b,b
;      asll     #2,d1
;      move    b,x:(r1)+
;      moves.f  d1,(r1)+
```

The translation in Example 7 seems to be correct because the values in registers “a” and “d0” are the same. However, when the data in register “a” is moved into memory, a problem might occur: If the data is overflow, the data shifter/limiter circuits in 56600 architecture may perform the limiting operation. Nevertheless, the limiting operation will not be performed in SC140 because the L1 bit will always be clear in ASLL operation. In order to validate the L1 bit before `MOVES.F` is performed, insert a `TFR D1,D1` instruction. Example 8 shows the correct translation.

Example 8. Corrected ASLL Translation

```
;      asl      #2,b,b
;      asll     #2,d1
;      tfr     d1,d1
;      move    b,x:(r1)+
;      moves.f  d1,(r1)+
```

To find and fix this bug in any translated 56600 program, use the following procedure: Once PortASM translates the 56600 program, search for the instruction `ASL` followed by `MOVE`. Add the instruction `TFR` before the `MOVE` instruction.

- The SR register is saved if the hardware needs to continue to loop after all four hardware loops are used. After the SR register is saved, the SR register is modified. In Example 9, the saturation arithmetic is turned on before the SR register is restored.

Example 9. Saturation Arithmetic Preceding SR Register Restoration

```

push    sr
...
bmset   #$4, sr.l
...
pop     sr                ; Restore loop state

```

Thus, the SR register has to be set again after the restoring instruction. See Example 10.

Example 10. Setting SR Register After Executing Restoring Instruction

```

push    sr
...
bmset   #$4, sr.l
...
pop     sr                ; Restore loop state
bmset   #$4, sr.l

```

To find and fix this bug in any translated 56600 program, use the following procedure: Once PortASM translates the 56600 program, search for the instruction BMSET. Make sure no SR register is saved before the SR register is changed.

In the SC140 simulation program, the 56600 source code is displayed if the PortASM application is run with the option `-symSrc`. With this option, PortASM explicitly puts debugging information in the translated source code. However, in a new version of the SC140 simulation program, 56600 source code cannot be displayed in the SC140 simulation program. The debugging information put explicitly in the translated code cannot be interpreted by the current version of the simulator (version 6.3.92).

3 Performance of Ported Program

Two sets of input (seq01.inp and seq02.inp) were tested in the GSM FR program on the SC140 platform. The output of the GSM FR program on the SC140 platform was verified to match the reference file of the two input vectors.

Table 2 shows the translated program's performance. The results are based on the testing of vector input seq02.inp, which contains 947 frames.

Table 2. Performance of Translated GSM FR in StarCore Platform

Platform (Version)	Average (Cycles/Frame)	Worst Case (Cycles/Frame)	Code Size (Bytes)
StarCore (PortASM)	117 403.87	117837	6976
56600	68 654.92	69011	4370

Because the architectures of the 56600 platform and the SC140 platform are not the same, the translation of one instruction on the 56600 platform yields one or more instructions on the SC140 platform. The translated code is bound to be inefficient in terms of code size. Moreover, the translation also increases the number of clock cycles or decreases speed.

Nonetheless, using PortASM is still a good starting point for making the transition from the 56600 platform to the SC140 platform. For programmers who prefer to hand craft SC140 assembly code, PortASM output can be a reference. For programmers who are concerned about speeding up development, PortASM will be very helpful.

4 Optimizing

Profiling the GSM FR application in the 56600 simulator identified the most time-consuming functions. The four most time-consuming functions are shown in Table 3.

Table 3. Four Most Time-Consuming Functions in GSM FR

Function	Cycle	% Cycle
lt_calc	33296	48.8
st_anal	9920	14.5
block_filter	5920	8.7
autocorr	3450	5.1

Thus, the effort to optimize was concentrated on these four functions. An outline of the optimization methods follows:

1. PortASM produces unparallelized code, even when the original 56600 code contains parallel instructions. Example 11 shows an example.

Example 11. Unparallelized Code

```

;          mac x0,y0,a x:(r0)+,x0
;          mac d2,d4,d0
;          move.f (r0)+,d2

```

After the instructions are parallelized, the translated code appears as shown in Example 12.

Example 12. Parallelized Code

```

mac d2,d4,d0 move.f (r0)+,d2

```

Most of the parallel instructions in 56600 can be found in the MAC, MACR, and MPY instructions. Searching for these instructions in the program saves time when you review line by line.

2. Instruction grouping increases the parallelization, thereby speeding up the processor. See Example 13 on page 7.

Example 13. Grouping Instructions to Speed Up Processing

```

;          dosetup0 ltcSA
;          doen0 #20
;          loopstart0
;ltcSA    mpy d2,d3,d1 move.f (r0)+,d2
;          mpy d2,d3,d0 move.f (r0)+,d2
;          moves.f d1,(r3)+ moves.f d1,(r4)+
;          moves.f d0,(r3)+ moves.f d0,(r4)+
;          loopend0

          mpy d2,d3,d1 move.f (r0)+,d2
          doensh0 #19
          loopstart0
          mpy d2,d3,d0 move.f (r0)+,d2 moves.f d1,(r3)+ moves.f d1,(r4)+
          mpy d2,d3,d1 move.f (r0)+,d2 moves.f d0,(r3)+ moves.f d0,(r4)+
          loopend0
          mpy d2,d3,d0 move.f (r0)+,d2 moves.f d1,(r3)+ moves.f d1,(r4)+
          moves.f d0,(r3)+ moves.f d0,(r4)+

```

This optimization method can be applied to almost any function in the translated code. It is advisable to apply this method to some of the most time-consuming functions.

3. All instruction loops in 56600 are translated into long loops in SC140. In the translated SC140 code, if a long loop contains three instructions, and if one or more of the instructions is an NOP instruction, then the long loop should be changed to a short loop.

For example, in 56600, the REP instruction executes a single-instruction loop. However, the translated program uses a long loop when the REP instruction is encountered. In long loops, the minimum number of instructions is three. To make the long loop legal, two NOP instructions are added to the translated SC140 code.

Thus, in the translated program, the NOPs should be removed to change the long loop back into a short loop. In the translated code, search for NOPs in long loops and delete them.

4. There are some new instructions introduced in StarCore that can reduce the number of instructions in 56600 code but that perform the same task. The new instructions can be used to optimize the translated code.

On the 56600 platform, storing 40-bit data in the ALU accumulator register (A or B register) requires three MOVE instructions. For these three MOVE instructions on the 56600 platform, more than three instructions are performed in the translated SC140 code. In SC140, the ALU accumulator register can be saved by two PUSH instructions and restored by two POP instructions.

Most of the program uses the MOVE instruction heavily. On the SC140 platform, more than 16-bit data width can be moved in one instruction cycle. For example, MOVE.L and MOVE.2W move 32-bit data in one cycle.

To use this optimization method, search for MOVE in the translated code. If multiple MOVES are performed for data located in contiguous memory, use the higher-bandwidth MOVE. However, pay attention to memory alignment. The data must be located at a memory location that is a multiple of four if MOVE.L or MOVE.2W is used. See Example 14 on page 8.

Example 14. Using MOVE.L to Move 32-Bit Data

```
;      doensh0 #40
;      loopstart0
;      move.f (r0)+,d2
;      move.f d2,(r3)+
;      loopend0
;      move.l (r0)+,d2
;      doensh0 #19
;      loopstart0
;      move.l d2,(r3)+ move.l (r0)+,d2
;      loopend0
;      move.l d2,(r3)+
```

Plans for Motorola's StarCore LLT (low-level transformation) program include providing support for raw assembly input. When this support is available, the LLT program can be used to do most of the optimizing of code that is ported from the 56600 platform to the SC140 platform.

5 Performance of Optimized Program

Table 4 shows the optimized, translated program's performance and compares it to the performance of hand-crafted code. The latter code was developed from scratch and was optimized to run on StarCore architecture.

Table 4. Performance of Optimized GSM FR on StarCore Platform

Platform (Version)	Average (Cycles/Frame)	Worst Case (Cycles/Frame)	Code Size (Bytes)
StarCore (optimized translation)	64 715.01	65019	6842
StarCore (optimized, hand-crafted code)	14704	14914	4456

The code size of the optimized translation is still larger than that of the original 56600 program. However, the architecture of the SC140 platform is superior to the architecture of the 56600 platform because the SC140 has larger numbers of ALUs and AGUs. Thus, with little optimization effort, the speed of the translated GSM FR program on the SC140 platform is almost the same as on the 56600 platform (see Table 2 on page 5).

The optimized, hand-crafted GSM FR is more than four times faster than the translated version. The memory that is required by the hand-crafted version is almost identical to the memory required by the 56600 program, which, again, is less than the memory required by the translated version.

The optimizing methods outlined in Section 4 do not exhaust all of the possibilities. The particular set of presented methods demonstrates the relatively large benefit that can result from relatively little effort. Numerous other optimizing methods could be used to improve the results further.

6 Conclusions

The PortASM program greatly reduces the time of porting applications in DSP56600 to SC140. The speed of the ported program is not an issue due to the superiority of SC140 architecture. However, the ported program is bound to be inefficient in terms of code size.