Locking Code in the NS32GX32 Instruction Cache

National Semiconductor
Application Note 608
Alon Shacham
April 1989

INTRODUCTION

The NS32GX32 provides two on-chip caches: A 512-byte Instruction Cache (IC), and a 1024-byte Data Cache. These caches are blocks of very fast, local memory that hold copies of instructions and data most frequently used by the CPU. For most general purpose applications, spatial and temporal locality ensure that more than 85% of the accesses are made from on-chip memory instead of main memory.

Having information on-chip speeds the execution of programs while reducing external bus traffic. Since this makes the processor less sensitive to memory speed, slower and less expensive main memories can be used.

On-chip caches are also important for embedded-control applications. For example, real-time applications that perform high-speed context switching typically contain code and data critical to performance. This information must be immediately available to the CPU and, therefore, should reside in the on-chip caches at all times.

To support this requirement, the NS32GX32 provides a cache-locking feature. This allows for the contents of the Instruction Cache and Data Cache to be locked to specific memory locations.

For the Data Cache, data is locked by reading the required locations and then setting the Lock Data Cache (LDC) bit in the configuration (CFG) register.

For the Instruction Cache, code can be locked by implementing the Cache Lock program presented in this application note. This program locks the code without actually executing it.

ABOUT THE CACHE LOCK PROGRAM

The Cache Lock program presented here takes advantage of the pipelined execution and branch prediction features of the NS32GX32 in an unconventional manner:

1. While the execution unit is busy with a long instruction, the branch predictor begins fetching from the area that is to be locked. This prefetch will cause some of the code to be loaded into the Instruction Cache. The code is not actually executed because this prediction is always incorrect.
2. Next, the code doing the loading modifies itself so that the next prediction will be to the next section to be loaded.

The Cache Lock program consists of the two main sections: loadlock and cod__mod.

loadlock

This section initializes the Instruction Cache by unlocking it and invalidating its contents. It then sets up the displacement for the ''bne'' instruction at the address of ''icbr''. This instruction will later be the one that sends the branch predictor off to fetch the code to be locked.

The main loop within ''load__lock'' is labeled ''i__c__next''. It calls ''cod__mod'' once for each double word to be loaded. After the entire Instruction Cache is loaded, it is locked, and the program terminates.

cod__mod

This section modifies the displacement of the ''bne'' instruction at ''icbr''. The displacement points to the next double word to be loaded. This is done one byte at a time for the four byte displacement.

Next, the execution is serialized. This is necessary to allow for the code modification to take place in memory before attempting to execute it.

Finally, a long (''remd'') instruction is executed. While this instruction is being executed in the Execution unit, the branch predictor (within the Loader unit) begins prefetching from the new target, i.e., the next double word to be loaded. Since this prediction is false (''branch not equal'' after ''compare r3 to r3''), the actual execution path is not altered. ''cod__mod'' then returns to the calling procedure, ''loadlock''.

IMPLEMENTING THE CACHE LOCK PROGRAM

The following sections describe the general requirements, interface requirements, and assembly code for the Cache Lock program.

General Requirements

The following requirements must be met for the Cache Lock program to operate correctly:

- The locking program (''loadlock'' and ''cod__mod'') must reside in a non-cacheable area (indicated by activating the CII input).
- The locking program must be executed from RAM.
- The code which is loaded must reside in a cacheable area.
- The code which is loaded must not exceed 512 bytes.
- The code which is loaded must be aligned as double-words on double word boundaries.
- The length of the code to be loaded must be a multiple of four.
- Interrupts must be disabled for the duration of the program.
- HOLD must not be asserted.
- No more than 10 wait states are allowed.

Interface Requirements

The following interface parameters must be supplied before calling ''loadlock'':

- Address of code to be locked must be in r0.
- Number of bytes to be locked must be in r1.

RRD-B30M75/Printed in U. S. A.

Assembly Code

The assembly code for the Cache Lock program is presented below.

```
#
          .set     load_length, 4        # load a double word at a time
          .set     LOC_IC, 0x1000        # lock IC (13th bit in cfg) on

          .text

loadlock::

          sprd     cfg,r4                # Store the current cfg in r4
          andd     $0xfffefff,r4         # unlock instruction cache
          cinv     a,i,$0                # Invalidate the entire IC

          addr     icbr,r4               # icbr is the location of the
                                         # BNE instruction. The displacement
                                         # in this instruction is modified
                                         # by this program as it executes.
          addd     r1,r0
          addqd    -4,r0
          movd     r0,r2
                                         # initialize r2 to the address of last
                                         # double word in code_block.
          subd     r4,r2                 # initialize displacement reg.
                                         # r2 contains the distance between
                                         # the icbr label and the last double word
                                         # in code_block.
          movd     r2,r6                 # save r2

          movqd    0,r5


i_c_next:                                # main loop
          movd     r6,r2                 # restore r2
          jsr      cod_mod               # load next dw/entry
          addd     $load_length, r5      # by load_length
          cmpw     r5,r1                 # to code_length
          blt      i_c_next
# Terminate by locking the IC
          sprd     cfg,r2                # r2 ← save cfg
          ord      $LOC_IC,r2            # r2 ← r2||LOC_IC
          lprd     cfg,r2                # cfg ← r2
          ret      $0                    # Return to calling program
                                         #
```

```
#
# Cod_mod modifies the displacement of the BNE instruction
#
cod_mod:
         subd    r5,r2                   # r2 contains displacement from icbr
                                         # to next double word to be loaded.
                                         # Note that the IC is loaded from higher
                                         # addresses to lower addresses
         ord     $0xc0000000,r2          # enter proper mode for 30 bits
                                         # displacement
         movqd   3,r3
i_nxt_byte:
         movb    r2,1(r4)[r3:b]          # move byte to proper place in
                                         # "bne" instruction.
         lshd    $-8,r2                  # get next byte of the displacement
                                         # to lsbyte of r2

         addqd   -1,r3
         cmpqb   -1,r3
         bne     i_nxt_byte              # If not done, modify next byte,
                                         # else-
         bicpsrw $0                      # serialize the execution to allow
                                         # for the write to update memory.
long_instr:
         remd    r4,r7                   # long instruction. While this is
                                         # being executed in the EXU, the
                                         # next instruction will be executed
                                         # in the Address Unit, and the next
                                         # one in the Loader.
                                         # This will allow enough time for the
                                         # next double word to enter the IC.
         cmpb    r3,r3
icbr:
         bne     cod_modl                # br "false" (always not taken)
         nop                             # the 4 bytes of the displacement are
         nop                             # replaced prior to execution.
         nop                             # this branch is always predicted as
                                         # taken, so the destination is fetched
                                         # into the IC

cod_modl:                                # This is a dummy label for the
                                         # assembler.
         ret     $0                      # return to main loop
```

Lit. # 100608

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998