SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

SYMBOLS.INC
ST9 REGISTER ADDRESS AND CONTENT NAMES

Pierre Guillemin

INTRODUCTION

This document has been written in order to provide, to the ST9 software programmer, a suggested guide and a clear notation of ST9 register and bit names for standardisation across software modules.

The **SYMBOLS.INC** files give a symbolic definition for:

- each group within the Register File
- each peripheral page
- each ST9 register pair
- each ST9 peripheral or core register
- each ST9 system and peripheral control bit with its associated mask.

This document assumes a previous knowledge of the ST9 architecture and software tools. Please refer to the ST9 Technical Manual, ST9 Programming Manual and ST9 Software Tools manuals for an understanding of the terms used.

INVOCATION

The **SYMBOLS.INC** file or a part it (depending on the application and the peripherals used) must be assembled with each ST9 software module in order to use the symbolic names.

The ST9 Macro Assembler (AST9) provides two methods for using include files:

1) directly in the invocation line of **AST9** (by giving a list of all include files). In this case, an example of the invocation line of **AST9**, implemented within an MS-DOS batch file and with other options, could be the following:

```
AST9 -v -g -r -o %1.obj -l %1.lst c:\ST9\INC\SYMBOLS.INC %1.ST9
```

where the source file name **%1 (.ST9)** is passed as the first batch parameter, and the **SYMBOLS.INC** file is located in the sub-directory **c:\ST9\INC.** Listing **(%1.lst)** and object code **(%1.obj)** files are produced.

2) inside each ST9 module by using the .include pseudo-instruction. In this case the syntax is:

```
.include "c:\ST9\INC\SYMBOLS.INC"
```

This method of using include files from within the ST9 software module has been expanded to allow the use only of the symbols applicable to the target software module or ST9 device. In this case, the **SYMBOLS.INC** file has been split into several include files related to each peripheral (e.g. A/D converter, MFTimer), or to specific features (e.g. the Bank Switch registers for ST905x family or the security register for ST904x family).

These peripheral include files are also associated with four include files named **ST90xx.inc** (ST902x, ST903x, ST904x, ST905x). These list all the related files applicable to the ST9 family member.

example:

```
; ST904x family description:
; Include file for the definition of the registers and bits for the
; ST904X family
 .include    “c:\st9\inc\system.inc”       ; System register
 .include    “c:\st9\inc\page_0.inc”       ; Page 0 register
 .include    “c:\st9\inc\eeprom.inc”       ; EEPROM register
 .include    “c:\st9\inc\sec_reg.inc”      ; Security register
 .include    “c:\st9\inc\io_port.inc”      ; I/O port register
 .include    “c:\st9\inc\mftimer.inc”      ; MF Timer register
 .include    “c:\st9\inc\ad_c.inc”         ; A/D converter register
 .include    “c:\st9\inc\sci.inc”          ; SCI register
```

The ST9 software programmer may use these include files in the two following methods:

1) use the include file corresponding to the target device within each ST9 software module.

2) directly use the include files corresponding to the ST9 peripheral programmed or used in a specific module: for example, inside a module using I/O ports and MFTimer, only the two include files related to MFTimer and I/O port could be used.

DEFINITIONS AND USAGE

Registers

A name is given for each ST9 register pair or register in upper case letters, corresponding to an "absolute" addressing mode inside the Register File, and in lower case letters, corresponding to an addressing mode inside a working register group (defined by the working register pointer pair **RP0** and **RP1**). This choice follows the notation used in the ST9 Macro Assembler (ie R or RR indicate any 8 or 16 bit register within the Register File, r or rr indicate a register within the working register group.):

```
FCW     :=      RR230     ; Flags and control word: Absolute address
fcw      =      rr6       ; Working Register address, Group E
CICR    :=      R230      ; Central interrupt control register
cicr     =      r6        ; Working Register address, Group E
```

Bits

A name for each bit and its associated mask for each control register has been defined:

1) the bit name is defined using the .defstr pseudo-instruction and the bit location within the working register. This name can be used directly with the boolean instructions:

```
.defstr  gcen  "cicr.7"    ; Global counter enable bit definition
```

2) the mask name is defined by a "one" shifted left by the bit location value inside the associated register. This allows the setting or masking of named bits within a working register:

```
gcenm    :=    (1 <- 7 )   ; Global counter enable masl
```

Bit symbol names are given in lower case letters.
Further examples of the use of these definitions follow.

Two other kinds of names are provided in this file:

- names for register groups within the Register File: one name for 8-working register groups, used with the SRP0 and SRP1 instructions, and one name for 16-working register groups (for system registers and page registers), used with the SRP instruction:

```
srp      #BK_F             ; select working register group F
```

- names for peripheral pages (Group F):

```
spp  #SCI1_PG          ; select SCI1 register page
```

Example 1: SCI initialization using working register addressing mode

The following example, extracted from an initialization routine of the Serial Communications Interface (SCI), shows how to use the **SYMBOLS.INC** file with the working register addressing mode. In the case of initialization of several peripherals (e.g. SCI and TIMER), this method allows the user to save bytes and execution time, due to the shorter instructions and execution times of the working register addressing modes.

```
sppSCI1_PG              ; select SCI1 register page
srp#BK_F                ; select working register group F
ld s_brghr,#0           ; initialize SCI baud rate generator
                        ; register in group F.
ld   s_chcr,#(wl8 | pen | ep | 20)
                        ; initialize character
                        ; configuration register using mask
                        ; bit definitions OR'd together
```

Example 2: using mask and complemented mask

This example shows how to use the mask and complemented mask. In this example, the enable receiver error interrupt and type of parity are set.

```
srp#BK_0                ; working register in group 0
spp#SCI1_PG             ; select SCI1 register page
or   S_IMR,#rxe         ; enable Rx error INTERRUPT using absolute
                        ; addressing in register file (R)
ld data,S_RXBR          ; load data with SCI receiver reg. (R)
if   [data == #0] {     ; if data is 00h...
   and S_CHCR,#~ep      ; ... enable odd parity detection
}                       ; (~ indicates 1's complement)
```

Example 3: bit manipulation

The SYMBOLS.INC file shows definition of the bits in a working register using the AST9 .defstr pseudo-instruction.

```
.defstr S_txdi "s_imr.0" ; transmitter data interrupt
btjz  S_txdi, checkrx    ; poll on tx data interrupt
bres  S_txdi             ; reset if active
```

An alternative method of defininition and usage of bits consists of giving a name to a bit location and associating this name to a register.

```
STATUS_SCI     =      R3       ; SCI status register (Absolute Reg.)
status_sci     =      r3       ; SCI status register (working Reg.)
P_er           =      0        ; parity error detected
Oe_er          =      1        ; overrun error detected
Fe_er          =      2        ; framing error detected
Tx_go          =      3        ; Tx on going
Tx_err         =      4        ; Tx error detected

bset   status_sci.Tx_go        ; set Tx ongoing bit
bres   status_sci.Tx_err       ; reset Tx error detection flag
```

SYMBOL TABLE OVERLOAD PROTECTION

SYMBOLS.INC contains more than 500 symbols and should be included with each ST9 software module which uses the symbols. In order not to overload the symbol table produced by the ST9 linker **LST9**, an **AST9** option **(-r)**, is available, which does not preserve register symbols in its output. When assembling several ST9 modules (each one containing **SYMBOLS.INC**), the first module (usually the main module) must be assembled without the **-r** option, all other modules must be assembled with the -r option. (Refer to the **AST9** User Manual for further information).

INCLUDE FILES

The name of the Include files related to the ST9 Core and Peripherals are listed below. These files are extracted from the SYMBOLS.INC file which is shown in Appendix B, and thus are not shown independently.

AD_C.INC	Analog to Digital Converter Registers
BS_REG.INC	Bankswitch Registers (ST905X only)
EEPROM.INC	EEPROM control Registers (ST904X only)
IO_PORT.INC	I/O Ports Registers (All ST9 devices)
MFTIMER.INC	Multifunction Timer Registers
PAGE_0.INC	Page 0 Registers (All ST9 devices)
RW_REG.INC	R/W Control Registers (ST905X only)
SCI.INC	Serial Communications Interface
SEC_REG.INC	Security Register (ST904X only)
SYSTEM.INC	System Registers (Group E, All ST9 devices)
ST902X.INC	ST902X Include File
ST903X.INC	ST903X Include File
ST904X.INC	ST904X Include File
ST905X.INC	ST905X Include File
SYMBOL36.INC	ALL Registers Include File

APPENDIX A: ST90XX Family Definition include files

ST902X family Description **(ST902X.INC)**:

```
; Include file for the definition of the registers and bits for the
; ST902x family
     .include "     c:\st9\inc\system.inc"      ; System register
     .include "     c:\st9\inc\page_0.inc"      ; Page 0 register
     .include "     c:\st9\inc\io_port.inc"     ; I/O port register
     .include "     c:\st9\inc\mftimer.inc"     ; MF Timer register
     .include "     c:\st9\inc\sci.inc"         ; SCI register
```

ST903x family description **(ST903X.INC):**

```
; Include file for the definition of the registers and bits for the
; ST903x family
     .include       "c:\st9\inc\system.inc"     ; System register
     .include       "c:\st9\inc\page_0.inc"     ; Page 0 register
     .include       "c:\st9\inc\io_port.inc"    ; I/O port register
     .include       "c:\st9\inc\mftimer.inc"    ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"       ; A/D converter register
     .include       "c:\st9\inc\sci.inc"        ; SCI register
```

ST904x family description **(ST904X.INC):**

```
; Include file for the definition of the registers and bits for the
; ST904x family
     .include       "c:\st9\inc\system.inc"     ; System register
     .include       "c:\st9\inc\page_0.inc"     ; Page 0 register
     .include       "c:\st9\inc\eeprom.inc"     ; EEPROM register
     .include       "c:\st9\inc\sec_reg.inc"    ; Security register
     .include       "c:\st9\inc\io_port.inc"    ; I/O port register
     .include       "c:\st9\inc\mftimer.inc"    ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"       ; A/D converter register
     .include       "c:\st9\inc\sci.inc"        ; SCI register
```

ST905x family description **(ST905X.INC):**

```
; Include file for the definition of the registers and bits for the
; ST905x family
     .include       "c:\st9\inc\system.inc"     ; System register
     .include       "c:\st9\inc\page_0.inc"     ; Page 0 register
     .include       "c:\st9\inc\rw_reg.inc"     ; R/W signal register
     .include       "c:\st9\inc\io_port.inc"    ; I/O port register
     .include       "c:\st9\inc\bs_reg.inc"     ; Bank switching register
     .include       "c:\st9\inc\mftimer.inc"    ; MF Timer register
     .include       "c:\st9\inc\ad_c.inc"       ; A/D converter register
     .include       "c:\st9\inc\sci.inc"        ; SCI register
```

APPENDIX B: Symbols.inc listing

```
.sbttl“ ST9 family registers and register-bits ”
        .pl     66              ; Number of lines per page
;       .list
;       .list   me              ; Enable macro expansion control
;       .list   bex             ; Enable continuation of code on next
                                ; Line
;       .nlist  line            ; Disable source line number control
;       .nlist  loc             ; Disable current location counter
                                ; control
;       .nlist  code            ; Disable binary code control
;       .nlist  src             ; Disable source line control
;       .nlist  com             ; Disable comment control
;       .nlist  md              ; Disable macro definition control
;       .nlist  mc              ; Disable macro call control
        .nlist

;*************************************************************************
;               ****************************************
;               *   Revision 3.6       MARCH, 04th 1991    *
;               ****************************************

;              ST9 family registers addresses and contents.
;     This file contains the symbolic definitions for the ST9 CPU
;     and Peripherals registers and bits.
;     There is a symbol for each register and for each flag used in
;     an ST9 family device.
;  - Lowercase letters refer to addressing using working registers ( r )
;  - Uppercase letters refer to addressing using direct registers  ( R )

;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************
;
;  ST9 family:  Core, Timer Watch-dog, SPI and EEPROM Control Register
;
;*************************************************************************

;*********************************
;*REGISTER FILE GROUPS DEFINITION*
;*********************************
BK00     =     0          ; r0 to r7         in group 0
BK01     =     1          ; r8 to r15        in group 0
BK10     =     2          ; r0 to r7         in group 1
BK11     =     3          ; r8 to r15        in group 1
BK20     =     4          ; r0 to r7         in group 2
BK21     =     5          ; r8 to r15        in group 2
BK30     =     6          ; r0 to r7         in group 3
BK31     =     7          ; r8 to r15        in group 3
BX40     =     8          ; r0 to r7         in group 4
BK41     =     9          ; r8 to r15        in group 4
BK50     =     10         ; r0 to r7         in group 5
BK51     =     11         ; r8 to r15        in group 5
BK60     =     12         ; r0 to r7         in group 6
BK61     =     13         ; r8 to r15        in group 6
BK70     =     14         ; r0 to r7         in group 7
BK71     =     15         ; r8 to r15        in group 7
BK80     =     16         ; r0 to r7         in group 8
BK81     =     17         ; r8 to r15        in group 8
BK90     =     18         ; r0 to r7         in group 9
BK91     =     19         ; r8 to r15        in group 9
BKA0     =     20         ; r0 to r7         in group A
BKA1     =     21         ; r8 to r15        in group A
BKB0     =     22         ; r0 to r7         in group B
BKB1     =     23         ; r8 to r15        in group B
BKC0     =     24         ; r0 to r7         in group C
BKC1     =     25         ; r8 to r15        in group C
BKD0     =     26         ; r0 to r7         in group D
BKD1     =     27         ; r8 to r15        in group D
BKE0     =     28         ; r0 to r7         in group E
BKE1     =     29         ; r8 to r15        in group E
BKF0     =     30         ; r0 to r7         in group F
BKF1     =     31         ; r8 to r15        in group F
BK_SYS   =     BKE0       ; Group system definition
BK_F     =     BKF0       ; page register definition
```

Appendix B: Symbols.inc listing

```
;******************
;*SYSTEM REGISTERS*
;******************
FCW      := RR230                    ; Flags and control word.
fcw       = rr6

CICR     := R230                     ; Central interrupt control register.
cicr      = r6

         .defstr gcen      “cicr.7”   ; Global counter enable.
         .defstr tlip      ”cicr.6”   ; Top level interrupt pending bit
         .defstr tli       “cicr.5”   ; Top level interrupt bit.
         .defstr ien       ”cicr.4”   ; Interrupt enable flag.
         .defstr iam       “cicr.3”   ; Interrupt arbitration mode.
         .defstr cpl2      ”cicr.2”   ; Current priority level bit 2.
         .defstr cpl1      “cicr.1”   ; Current priority level bit 1.
         .defstr cpl0      ”cicr.0”   ; Current priority level bit 0.

gcenm    := ( 1 <- 7 )               ; Global counter enable bit mask
tlipm    := ( 1 <- 6 )               ; Top level interrupt pending mask.
tlim     := ( 1 <- 5 )               ; Top level interrupt mask.
ienm     := ( 1 <- 4 )               ; Interrupt enabla flag mask.
iamm     := ( 1 <- 3 )               ; Interrupt arbitration mode mask.
cpl2m    := ( 1 <- 2 )               ; Current priority level bit 2 mask.
cpl1m    := ( 1 <- 1 )               ; Current priority level bit 1 mask.
cpl0m    := ( 1 <- 0 )               ; Current priority level bit 0 mask.
cplm     := ( cpl2m|cpl1m|cpl0 )     ; Current priority level


FLAGR    := R231                     ; Flags register.
flagr     = r7

         .defstr c         “flagr.7”  ; Carry flag.
         .defstr z         ”flagr.6”  ; Zero flag.
         .defstr s         “flagr.5”  ; Sign flag.
         .defstr v         ”flagr.4”  ; Overflow flag.
         .defstr d         “flagr.3”  ; Decimal adjust flag.
         .defstr h         ”flagr.2”  ; Half carry flag.
         .defstr uf        “flagr.1”  ; User flag 1.
         .defstr dp        ”flagr.0”  ; Data/program memory flag.

cm       := ( 1 <- 7 )               ; Carry flag mask.
zm       := ( 1 <- 6 )               ; Zero flag mask.
sm       := ( 1 <- 5 )               ; Sign flag mask.
vm       := ( 1 <- 4 )               ; Overflow flag mask.
dm       := ( 1 <- 3 )               ; Decimal adjust flag mask.
hm       := ( 1 <- 2 )               ; Half carry flag mask.
ufm      := ( 1 <- 1 )               ; User flag 1 mask.
dpm      := ( 1 <- 0 )               ; Data/program memory mask.

RPP      := RR232                    ; Register pointer pair.
rpp       = rr8

RP0R     := R232                     ; Register pointer # 0.
rp0r      = r8

         .defstr rp0s      “rp0r.2”   ; Register pointer selector
rp0sm    := ( 1 <- 2 )               ; Register pointer selector mask
```

APPENDIX B: Symbols.inc listing

```
RP1R     := R233                      ; Register pointer # 1.
 rp1r     = r9
         .defstr rp1s      “rp1r.2”   ; Register pointer selector
rp1sm    := ( 1 <- 2 )                ; Register pointer selector mask
PPR      := R234                      ; Page pointer register.
ppr       = r10
MODER    := R235                      ; Mode register.
moder     = r11
         .defstr ssp       “moder.7”  ; System stack pointer flag (Int/Ext).
         .defstr usp       ”moder.6”  ; User stack pointer flag (Int/Ext).
         .defstr div2      “moder.5”  ; External clock divided by 2.
         .defstr prs2      ”moder.4”  ; Internal clock prescaling bit 2.
         .defstr prs1      “moder.3”  ; Internal clock prescaling bit 1.
         .defstr prs0      ”moder.2”  ; Internal clock prescaling bit 0.
         .defstr brqen     “moder.1”  ; Bus request enable.
         .defstr himp      ”moder.0”  ; High impedance enable.
sspm     := ( 1 <- 7 )                ; System stack pointer mask (Int/Ext).
uspm     := ( 1 <- 6 )                ; User stack pointer mask (Int/Ext).
div2m    := ( 1 <- 5 )                ; External clock divided by 2 mask.
prs2m    := ( 1 <- 4 )                ; Internal clock prescaling bit 2 mask.
prs1m    := ( 1 <- 3 )                ; Internal clock prescaling bit 1 mask.
prs0m    := ( 1 <- 2 )                ; Internal clock prescaling bit 0 mask.
prsm     := ( prs2m|prs1m|prs0m )     ; Internal clock prescaler
brqenm   := ( 1 <- 1 )                ; Bus request enable mask.
himpm    := ( 1 <- 0 )                ; High impedence enable mask.
USPR     := RR236                     ; User stack pointer.
uspr      = rr12
USPHR    := R236                      ; User stack pointer, msb.
usphr     = r12
USPLR    := R237                      ; User stack pointer, lsb.
usplr     = r13
SSPR     := RR238                     ; System stack pointer.
sspr      = rr14
SSPHR    := R238                      ; System stack pointer, msb.
ssphr     = r14
SSPLR    := R239                      ; System stack pointer, lsb.
ssplr     = r15
;.page
```

APPENDIX B: Symbols.inc listing

```
;****************
;*PAGE REGISTERS*
;****************
EEP_PG   := 0                          ; EEPROM register page

EECR     := R241                       ; EEprom control register
eecr      = r1

         .defstr verify    “eecr.6”   ; EEPROM verify mode
         .defstr EEstby    ”eecr.5”   ; EEPROM stand-by
         .defstr EEien     “eecr.4”   ; EEPROM interrupt enable
         .defstr pllst     ”eecr.3”   ; Parallel write start
         .defstr pllen     “eecr.2”   ; Parallel write enable
         .defstr EEbusy    ”eecr.1”   ; EEPROM busy
         .defstr EEwen     “eecr.0”   ; EEPROM write enable

verify         := ( 1 <- 6 )           ; EEPROM verify mode mask
eestby         := ( 1 <- 5 )           ; EEPROM stand-by mask
eeien := ( 1 <- 4 )                    ; EEPROM interrupt enable mask
pllst    := ( 1 <- 3 )                 ; Parallel write start mask
pllen    := ( 1 <- 2 )                 ; Parallel write enable mask
eebusy   := ( 1 <- 1 )                 ; EEPROM busy mask
eewen    := ( 1 <- 0 )                 ; EEPROM Write enable mask

EXINT_PG := 0                          ; EXTERNAL interrupt register page

EITR     := R242                       ; External interrupt trigger level register
eitr      = r2

         .defstr tea0      “eitr.0”   ; Trigger Event A0 bit
         .defstr tea1      ”eitr.1”   ; Trigger Event A1 bit
         .defstr teb0      “eitr.2”   ; Trigger Event B0 bit
         .defstr teb1      ”eitr.3”   ; Trigger Event B1 bit
         .defstr tec0      “eitr.4”   ; Trigger Event C0 bit
         .defstr tec1      ”eitr.5”   ; Trigger Event C1 bit
         .defstr ted0      “eitr.6”   ; Trigger Event D0 bit
         .defstr ted1      ”eitr.7”   ; Trigger Event D1 bit

tea0m    := ( 1 <- 0 )                 ; Trigger Event A0 mask
tea1m    := ( 1 <- 1 )                 ; Trigger Event A1 mask
teb0m    := ( 1 <- 2 )                 ; Trigger Event B0 mask
teb1m    := ( 1 <- 3 )                 ; Trigger Event B1 mask
tec0m    := ( 1 <- 4 )                 ; Trigger Event C0 mask
tec1m    := ( 1 <- 5 )                 ; Trigger Event C1 mask
ted0m    := ( 1 <- 6 )                 ; Trigger Event D0 mask
ted1m    := ( 1 <- 7 )                 ; Trigger Event D1 mask

EIPR     := R243                       ; External interrupt pending register
eipr      = r3

         .defstr ipa0      “eipr.0”   ; Interrupt Pending bit Channel A0
         .defstr ipa1      ”eipr.1”   ; Interrupt Pending bit      “   A1
         .defstr ipb0      ”eipr.2”   ; Interrupt Pending bit      “   B0
         .defstr ipb1      ”eipr.3”   ; Interrupt Pending bit      “   B1
         .defstr ipc0      ”eipr.4”   ; Interrupt Pending bit      “   C0
         .defstr ipc1      ”eipr.5”   ; Interrupt Pending bit      “   C1
         .defstr ipd0      ”eipr.6”   ; Interrupt Pending bit      “   D0
         .defstr ipd1      ”eipr.7”   ; Interrupt Pending bit      “   D1
```

APPENDIX B: Symbols.inc listing

```
ipa0m    := ( 1 <- 0 )              ; Interrupt Pending A0 mask
ipa1m    := ( 1 <- 1 )              ; Interrupt Pending A1 mask
ipb0m    := ( 1 <- 2 )              ; Interrupt Pending B0 mask
ipb1m    := ( 1 <- 3 )              ; Interrupt Pending B1 mask
ipc0m    := ( 1 <- 4 )              ; Interrupt Pending C0 mask
ipc1m    := ( 1 <- 5 )              ; Interrupt Pending C1 mask
ipd0m    := ( 1 <- 6 )              ; Interrupt Pending D0 mask
ipd1m    := ( 1 <- 7 )              ; Interrupt Pending D1 mask
EIMR     := R244                    ; External interrupt mask register
eimr      = r4
         .defstr ima0 “eimr.0”      ; Int. A0 bit
         .defstr ima1 ”eimr.1”      ; Int. A1 bit
         .defstr imb0 “eimr.2”      ; Int. B0 bit
         .defstr imb1 ”eimr.3”      ; Int. B1 bit
         .defstr imc0 “eimr.4”      ; Int. C0 bit
         .defstr imc1 ”eimr.5”      ; Int. C1 bit
         .defstr imd0 “eimr.6”      ; Int. D0 bit
         .defstr imd1 ”eimr.7”      ; Int. D1 bit
ia0m     := ( 1 <- 0 )              ; Int. A0 mask
ia1m     := ( 1 <- 1 )              ; Int. A1 mask
ib0m     := ( 1 <- 2 )              ; Int. B0 mask
ib1m     := ( 1 <- 3 )              ; Int. B1 mask
ic0m     := ( 1 <- 4 )              ; Int. C0 mask
ic1m     := ( 1 <- 5 )              ; Int. C1 mask
id0m     := ( 1 <- 6 )              ; Int. D0 mask
id1m     := ( 1 <- 7 )              ; Int. D1 mask
EIPLR    := R245                    ; Ext. interrupt priority level register
eiplr     = r5
EIVR     := R246                    ; External interrupt vector register
eivr      = r6
         .defstr ewen  “eivr.0”     ; External wait enable
         .defstr ia0s  ”eivr.1”     ; Interrupt A0 selection
         .defstr tlis  “eivr.2”     ; Top level input selection
         .defstr tltev ”eivr.3”     ; Top level trigger event
ewenm    := ( 1 <- 0 )              ; External wait enable mask
iaosm    := ( 1 <- 1 )              ; Interrupt A0 selection mask
tlism    := ( 1 <- 2 )              ; Top level Input selection mask
tltevm   := ( 1 <- 3 )              ; Top level trigger event mask

NICR     := R247                    ; Nested interrupt control register
nicr      = r7
         .defstr tlnm     “nicr.7”  ; Top level not maskable
tlnmm    := ( 1 <- 7 )              ; Top level not maskable mask
```

APPENDIX B: Symbols.inc listing

```
WDT_PG   := 0                          ; Timer Watchdog page
WDTR     := RR248                      ; TWD timer constant register.
wdtr      = rr8
WDTHR    := R248                       ; TWD timer high constant register
wdthr     = r8
WDTLR    := R249                       ; TWD timer low constant register
wdtlr     = r9
WDTPR    := R250                       ; TWD timer prescaler constant register
wdtpr     = r10
WDTCR    := R251                       ; TWD timer control register
wdtcr     = r11
         .defstr WD_stsp   “wdtcr.7”   ; TWD start stop.
         .defstr WD_sc     ”wdtcr.6”   ; TWD single continuous mode.
         .defstr WD_inmd1  “wdtcr.5”   ; Input mode 1
         .defstr WD_inmd2  ”wdtcr.4”   ; Input mode 2
         .defstr WD_inen   “wdtcr.3”   ; TWD input section enable/disable.
         .defstr WD_outmd  ”wdtcr.2”   ; TWD output mode.
         .defstr WD_wrout  “wdtcr.1”   ; TWD output bit.
         .defstr WD_outen  ”wdtcr.0”   ; TWD output enable.
stsp     := ( 1 <- 7 )                 ; TWD start stop mask
sc       := ( 1 <- 6 )                 ; TWD single continuous mode mask
inen     := ( 1 <- 3 )                 ; TWD input section enable/disable mask
outmd    := ( 1 <- 2 )                 ; TWD output mode mask
wrout    := ( 1 <- 1 )                 ; TWD output bit mask
outen    := ( 1 <- 0 )                 ; TWD output enable mask
inm_evc  := 0                          ; TWD input mode event counter.
inm_g    := 010h                       ; TWD input mode gated.
inm_t    := 020h                       ; TWD input mode triggerable.
inm_r    := 030h                       ; TWD input mode retriggerable.
```

APPENDIX B: Symbols.inc listing

```
WCR      :=    R252                  ; Wait control register
wcr       =    r12
         .defstr WD_wden   “wcr.6”    ; TWD timer enable.
wdgen    := ( 1 <- 6 )                ; TWD timer enable mask
wdm2     := ( 1 <- 5 )                ; Data Memory Wait Cycle
wdm1     := ( 1 <- 4 )
wdm0     := ( 1 <- 3 )
wpm2     := ( 1 <- 2 )                ; Program Memory Wait Cycle
wpm1     := ( 1 <- 1 )
wpm0     := ( 1 <- 0 )
dmwc1    := wdm0                      ; 1 wait cycle  on Data M.
dmwc2    := wdm1                      ; 2 wait cycles on Data M.
dmwc3    := ( wdm1 | wdm0 )           ; 3 wait cycles on Data M.
dmwc4    := wdm2                      ; 4 wait cycles on Data M.
dmwc5    := ( wdm2 | wdm0 )           ; 5 wait cycles on Data M.
dmwc6    := ( wdm2 | wdm1 )           ; 6 wait cycles on Data M.
dmwc7    := (wdm2 | wdm1 | wdm0)      ; 7 wait cycles on Data M.
pmwc1    := wpm0                      ; 1 wait cycle  on Prog M.
pmwc2    := wpm1                      ; 2 wait cycles on Prog M.
pmwc3    := ( wpm1 | wpm0 )           ; 3 wait cycles on Prog M.
pmwc4    := wpm2                      ; 4 wait cycles on Prog M.
pmwc5    := ( wpm2 | wpm0 )           ; 5 wait cycles on Prog M.
pmwc6    := ( wpm2 | wpm1 )           ; 6 wait cycles on Prog M.
pmwc7    := (wpm2 | wpm1 | wpm0 )     ; 7 wait cycles on Prog M.
```

APPENDIX B: Symbols.inc listing

```
SPI_PG   := 0                          ; SPI register page
SPIDR    := R253                       ; SPI Data register
spidr     = r13
SPICR    := R254                       ; SPI Control register
spicr     = r14
         .defstr SP_spen   “spicr.7”   ; Serial Peripheral Enable.
         .defstr SP_bms    ”spicr.6”   ; SBUS/I2C bus Mode Selector.
         .defstr SP_arb    “spicr.5”   ; Arbitration flag bit.
         .defstr SP_busy   ”spicr.4”   ; SPI busy flag.
         .defstr SP_cpol   “spicr.3”   ; SPI transmission clock polarity
         .defstr SP_cpha   ”spicr.2”   ; SPI transmission clock phase
         .defstr SP_spr1   “spicr.1”   ; SPI rate bit 1
         .defstr SP_spr0   ”spicr.0”   ; SPI rate bit 0
spen     := ( 1 <- 7 )                 ; Serial Peripheral Enable mask
bms      := ( 1 <- 6 )                 ; SBUS/I2C bus selector mask
arb      := ( 1 <- 5 )                 ; Arbitration mask
sp_busy  := ( 1 <- 4 )                 ; SPI busy mask
cpol     := ( 1 <- 3 )                 ; SPI transmission clock polarity mask
cpha     := ( 1 <- 2 )                 ; SPI transmission clock phase
SP_8     := 0                          ; SPI clock divider   8 = 1500 kHz  (12MHz)
SP_16    := 1                          ; SPI clock divider  16 =  750 kHz  (12MHz)
SP_128   := 2                          ; SPI clock divider 128 = 93.75 kHz (12MHz)
SP_256   := 3                          ; SPI clock divider 256 = 46.87 kHz (12MHz)
RW_PG    := 0                          ; R/W signal programming page
RWR      := R255                       ; R/W signal programming register
rwr       = r15
         .defstr RW_rw     “rwr.0”     ; R/W bit
         .defstr RW_bs     ”rwr.1”     ; Bank switch port timing
rw       := ( 1 <- 0 )                 ; R/W mask
bs       := ( 1 <- 1 )                 ; Bank Switch mask
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;              ST9 FAMILY I/O PORTS REGISTER ADDRESSES.

;*************************************************************************

; P0DR, P1DR, P2DR, P3DR, P4DR, P5DR are mapped in the system registers
; BS_DSR, BS_PSR are mapped in the system registers
P0C_PG   := 2                          ; Port 0 control registers page
P0DR     := R224                       ; Port 0 data register
P0C0R    := R240                       ; Port 0 control register 0
P0C1R    := R241                       ; Port 0 control register 1
P0C2R    := R242                       ; Port 0 control register 2
p0dr      = r0
p0c0r     = r0
p0c1r     = r1
p0c2r     = r2
P1C_PG   := 2                          ; Port 1 control registers page
P1DR     := R225                       ; Port 1 data register
P1C0R    := R244                       ; Port 1 control register 0
P1C1R    := R245                       ; Port 1 control register 1
P1C2R    := R246                       ; Port 1 control register 2
p1dr      = r1
p1c0r     = r4
p1c1r     = r5
p1c2r     = r6
P2C_PG   := 2                          ; Port 2 control registers page
P2DR     := R226                       ; Port 2 data register
BS_DSR   := R226                       ; Bank Switch data segment register
P2C0R    := R248                       ; Port 2 control register 0
BS_DDSR  := R248                       ; Bank Switch Data DMA segment register
P2C1R    := R249                       ; Port 2 control register 1
BS_PDSR  := R249                       ; Bank Switch Program DMA segment Register
P2C2R    := R250                       ; Port 2 control register 2
p2dr      = r2
bs_dsr    = r2
p2c0r     = r8
bs_ddsr   = r8
p2c1r     = r9
bs_pdsr   = r9
p2c2r     = r10
```

APPENDIX B: Symbols.inc listing

```
P3C_PG   := 2                          ; Port 3 control registers page
P3DR     := R227                       ; Port 3 data register
BS_PSR   := R227                       ; Bank Switch Program Segment Register
P3C0R    := R252                       ; Port 3 control register 0
P3C1R    := R253                       ; Port 3 control register 1
P3C2R    := R254                       ; Port 3 control register 2
p3dr      = r3
bs_psr    = r3
p3c0r     = r12
p3c1r     = r13
p3c2r     = r14
P4C_PG   := 3                          ; Port 4 control registers page
P4DR     := R228                       ; Port 4 data register
P4C0R    := R240                       ; Port 4 control register 0
P4C1R    := R241                       ; Port 4 control register 1
P4C2R    := R242                       ; Port 4 control register 2
p4dr      = r4
p4c0r     = r0
p4c1r     = r1
p4c2r     = r2
P5C_PG   := 3                          ; Port 5 control registers page
P5DR     := R229                       ; Port 5 data register
P5C0R    := R244                       ; Port 5 control register 0
P5C1R    := R245                       ; Port 5 control register 1
P5C2R    := R246                       ; Port 5 control register 2
p5dr      = r5
p5c0r     = r4
p5c1r     = r5
p5c2r     = r6
P6C_PG   := 3                          ; Port 6 control registers page
P6D_PG   := 3                          ; Port 6 data register page
P6DR     := R251                       ; Port 6 data register
P6C0R    := R248                       ; Port 6 control register 0
P6C1R    := R249                       ; Port 6 control register 1
P6C2R    := R250                       ; Port 6 control register 2
p6dr      = r11
p6c0r     = r8
p6c1r     = r9
p6c2r     = r10
```

APPENDIX B: Symbols.inc listing

```
P7C_PG   := 3                          ; Port 7 control registers page
P7D_PG   := 3                          ; Port 7 data register page
P7DR     := R255                       ; Port 7 data register
P7C0R    := R252                       ; Port 7 control register 0
P7C1R    := R253                       ; Port 7 control register 1
P7C2R    := R254                       ; Port 7 control register 2

p7dr      = r15
p7c0r     = r12
p7c1r     = r13
p7c2r     = r14

P8C_PG   := 43                         ; Port 8 control registers page
P8D_PG   := 43                         ; Port 8 data register page

P8DR     := R251                       ; Port 8 data register
P8C0R    := R248                       ; Port 8 control register 0
P8C1R    := R249                       ; Port 8 control register 1
P8C2R    := R250                       ; Port 8 control register 2

p8dr      = r11
p8c0r     = r8
p8c1r     = r9
p8c2r     = r10

P9C_PG   := 43                         ; Port 9 control registers page
P9D_PG   := 43                         ; Port 9 data register page

P9DR     := R255                       ; Port 9 data register
P9C0R    := R252                       ; Port 9 control register 0
P9C1R    := R253                       ; Port 9 control register 1
P9C2R    := R254                       ; Port 9 control register 2

p9dr      = r15
p9c0r     = r12
p9c1r     = r13
p9c2r     = r14
```

APPENDIX B: Symbols.inc listing

```
HDCTL2R  := R251                        ; Port 2 handshake DMA control register
HDCTL3R  := R255                        ; Port 3 handshake DMA control register
HDCTL4R  := R243                        ; Port 4 handshake DMA control register
HDCTL5R  := R247                        ; Port 5 handshake DMA control register
hdctl2r   = r11
hdctl3r   = r15
hdctl4r   = r3
hdctl5r   = r7


;Handshake DMA control register configuration.
hsdis    := 0E0h                        ; Handshake disabled mask
hso2     := 0C0h                        ; Handshake output 2 lines mask
hso1     := 040h                        ; Handshake output 1 line mask
hsi2     := 0A0h                        ; Handshake input 2 lines mask
hsi1     := 020h                        ; Handshake input 1 line mask
hsb      := 000h                        ; Handshake bidirectional mask
den      := 000h                        ; DMA enable mask
ddi      := 010h                        ; DMA disable mask
ddw      := 000h                        ; Data direction output mask (write)
ddr      := 008h                        ; Data direction input mask (read)
dst      := 004h                        ; DMA strobe on chip event mask
dcp0     := 000h                        ; DMA channel capture0 mask
dcm0     := 002h                        ; DMA channel compare0 mask
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;     ST9 FAMILY MULTI-FUNCTION TIMER DESCRIPTION.

;*************************************************************************

T0D_PG   := 10                          ; MFTimer 0 data registers page
T0C_PG   := 9                           ; MFTimer 0 control registers page
T1D_PG   := 8                           ; MFTimer 1 data registers page
T1C_PG   := 9                           ; MFTimer 1 control registers page
T2D_PG   := 14                          ; MFTimer 2 data registers page
T2C_PG   := 13                          ; MFTimer 2 control registers page
T3D_PG   := 12                          ; MFTimer 3 data registers page
T3C_PG   := 13                          ; MFTimer 3 control registers page


T_REG0R  := RR240                       ; MFTimer REG0 load and capture register.
t_reg0r   = rr0
T_REG0HR := R240                        ; Register 0 high register
t_reg0hr  = r0
T_REG0LR := R241                        ; Register 0 low register
t_reg0lr  = r1
T_REG1R  := RR242                       ; MFTimer REG1 load constant
t_reg1r   = rr2                               ; and capture register.
T_REG1HR := R242                        ; Register 1 high register
t_reg1hr  = r2
T_REG1LR := R243                        ; Register 1 low register
t_reg1lr  = r3
T_CMP0R  := RR244                       ; MFTimer CMP0 store compare constant.
t_cmp0r   = rr4
T_CMP0HR := R244                        ; Compare 0 high register
t_cmp0hr  = r4
T_CMP0LR := R245                        ; Compare 0 low register
t_cmp0lr  = r5
T_CMP1R  := RR246                       ; MFTimer CMP1 store compare constant.
t_cmp1r   = rr6
T_CMP1HR := R246                        ; Compare 1 high register
t_cmp1hr  = r6
T_CMP1LR := R247                        ; Compare 1 low register
t_cmp1lr  = r7
```

APPENDIX B: Symbols.inc listing

```
T_TCR    := R248                        ; MFTimer Control Register.
t_tcr     = r8
         .defstr T_cs      “t_tcr.0”   ; Counter status
         .defstr T_of0     ”t_tcr.1”   ; over/underflow on CAP on REG0
         .defstr T_udcs    “t_tcr.2”   ; up/down count status
         .defstr T_udc     ”t_tcr.3”   ; up/down count
         .defstr T_ccl     “t_tcr.4”   ; Counter clear
         .defstr T_ccmp0   ”t_tcr.5”   ; Clear on compare 0
         .defstr T_ccp0    “t_tcr.6”   ; Clear on capture
         .defstr T_cen     ”t_tcr.7”   ; Counter enable
cs       := ( 1 <- 0 )                  ; Counter status mask
of0      := ( 1 <- 1 )                  ; over/underflow mask on CAP on REG0
udcs     := ( 1 <- 2 )                  ; up/down count status mask
udc      := ( 1 <- 3 )                  ; up/down count mask
ccl      := ( 1 <- 4 )                  ; Counter clear mask
ccmp0    := ( 1 <- 5 )                  ; Clear on compare mask
ccp0     := ( 1 <- 6 )                  ; Clear on capture mask
cen      := ( 1 <- 7 )                  ; Counter enable mask

T_TMR    := R249                        ; MFTimer Mode Register.
t_tmr     = r9
   .defstr T_co           “t_tmr.0”    ; Continuous/one shot bit
   .defstr T_ren          ”t_tmr.1”    ; retrigger enable bit
   .defstr T_eck          “t_tmr.2”    ; Enable clocking mode bit
   .defstr T_rm0          ”t_tmr.3”    ; register 0 mode bit
   .defstr T_rm1          “t_tmr.4”    ; register 1 mode bit
   .defstr T_bm           ”t_tmr.5”    ; bivalue mode bit
   .defstr T_oe0          “t_tmr.6”    ; output 0 enable bit
   .defstr T_oe1          ”t_tmr.7”    ; output 1 enable bit
co       := ( 1 <- 0 )                  ; Continuous/one shot mask
ren      := ( 1 <- 1 )                  ; retrigger enable mask
eck      := ( 1 <- 2 )                  ; Enable clocking mode mask
rm0      := ( 1 <- 3 )                  ; register 0 mode mask
rm1      := ( 1 <- 4 )                  ; register 1 mode mask
bm       := ( 1 <- 5 )                  ; bivalue mode mask
oe0      := ( 1 <- 6 )                  ; output 0 enable mask
oe1      := ( 1 <- 7 )                  ; output 1 enable mask
```

APPENDIX B: Symbols.inc listing

```
T_ICR    := R250                    ; MFTimer External Input Control Register.
t_icr     = r10
exb_f    := 01h                     ; External B falling edge sensitive mask
exb_r    := 02h                     ; External B rising edge sensitive mask
exb_rf   := 03h                     ; External B falling and rising edge mask
exa_f    := 04h                     ; External A falling edge sensitive mask
exa_r    := 08h                     ; External A rising edge sensitive mask
exa_rf   := 0Ch                     ; External A falling and rising edge mask
ab_ii    := 00h                     ; A I/O B I/O mask
ab_it    := 10h                     ; A I/O B trigger mask
ab_gi    := 20h                     ; A gate B I/O mask
ab_gt    := 30h                     ; A gate B trigger mask
ab_ie    := 40h                     ; A I/O B external clock mask
ab_ti    := 50h                     ; A trigger B I/O mask
ab_ge    := 60h                     ; A gate B external clock mask
ab_tt    := 70h                     ; A trigger B trigger mask
ab_cucd  := 80h                     ; A clock up B clock down mask
ab_ue    := 90h                     ; A clock up/down B external clock mask
ab_tutd  := 0A0h                    ; A trigger up B trigger down mask
ab_ui    := 0B0h                    ; A up/down clock B I/O mask
ab_aa    := 0C0h                    ; A autodiscr. B autodiscr. mask
ab_te    := 0D0h                    ; A trigger B external clock mask
ab_et    := 0E0h                    ; A external clock B trigger mask
ab_tg    := 0F0h                    ; A trigger B gate mask
T_PRSR   := R251                    ; MFTimer prescaler register
t_prsr    = r11
T_OACR   := R252                    ; MFTimer Output A Control Register.
t_oacr    = r12
cev      := 02h                     ; on chip event bit on COMPARE 0 mask
T_OBCR   := R253                    ; MFTimer Output B Control Register.
t_obcr    = r13
op       := 01h                     ; output preset bit mask
oev      := 02h                     ; on chip event bit on OVF/UDF mask
ou_set   := 00h                     ; overflow underflow set mask
ou_tog   := 04h                     ; overflow underflow toggle mask
ou_res   := 08h                     ; overflow underflow reset mask
ou_nop   := 0Ch                     ; overflow underflow nop mask
c1_set   := 00h                     ; Compare 1 set mask
c1_tog   := 10h                     ; Compare 1 toggle mask
c1_res   := 20h                     ; Compare 1 reset mask
c1_nop   := 30h                     ; Compare 1 nop mask
c0_set   := 00h                     ; Compare 0 set mask
c0_tog   := 40h                     ; Compare 0 toggle mask
c0_res   := 80h                     ; Compare 0 reset mask
c0_nop   := 0C0h                    ; Compare 0 nop mask
```

APPENDIX B: Symbols.inc listing

```
T_FLAGR  := R254                    ; MFTimer Flags Register.
t_flagr   = r14

         .defstr T_ao      "t_flagr.0"; and/or on capture interrupt
         .defstr T_ocm0    "t_flagr.1"; overrun compare 0
         .defstr T_ocp0    "t_flagr.2"; overrun capture 0
         .defstr T_ouf     "t_flagr.3"; overflow underflow flag
         .defstr T_cm1     "t_flagr.4"; successful compare 1
         .defstr T_cm0     "t_flagr.5"; successful compare 0
         .defstr T_cp1     "t_flagr.6"; successful capture 1
         .defstr T_cp0     "t_flagr.7"; successful capture 0

ao       := ( 1 <- 0 )              ; and/or on capture interrupt mask
ocm0     := ( 1 <- 1 )              ; overrun compare 0 mask
ocp0     := ( 1 <- 2 )              ; overrun capture 0 mask
ouf      := ( 1 <- 3 )              ; overflow underflow flag mask
cm1      := ( 1 <- 4 )              ; successful compare 1 mask
cm0      := ( 1 <- 5 )              ; successful compare 0 mask
cp1      := ( 1 <- 6 )              ; successful capture 1 mask
cp0      := ( 1 <- 7 )              ; successful capture 0 mask

T_IDMR   := R255                    ; MFTimer Interrupt DMA Mask Register.
t_idmr    = r15

         .defstr T_oui     "t_idmr.0" ; overflow underflow interrupt
         .defstr T_cm1i    "t_idmr.1" ; Compare 1 interrupt
         .defstr T_cm0i    "t_idmr.2" ; Compare 0 interrupt
         .defstr T_cm0d    "t_idmr.3" ; Compare 0 DMA
         .defstr T_cp1i    "t_idmr.4" ; Capture 1 interrupt
         .defstr T_cp0i    "t_idmr.5" ; Capture 0 interrupt
         .defstr T_cp0d    "t_idmr.6" ; Capture 0 DMA
         .defstr T_gtien   "t_idmr.7" ; global timer interrupt enable

oui      := ( 1 <- 0 )              ; overflow underflow interrupt mask
cm1i     := ( 1 <- 1 )              ; Compare 1 interrupt mask
cm0i     := ( 1 <- 2 )              ; Compare 0 interrupt mask
cm0d     := ( 1 <- 3 )              ; Compare 0 DMA mask
cp1i     := ( 1 <- 4 )              ; Capture 1 interrupt mask
cp0i     := ( 1 <- 5 )              ; Capture 0 interrupt mask
cp0d     := ( 1 <- 6 )              ; Capture 0 DMA mask
gtien    := ( 1 <- 7 )              ; global timer interrupt enable mask

T0_DCPR  := R240                    ; MFTimer 0 DMA Counter Pointer Register.
t0_dcpr   = r0

T1_DCPR  := R244                    ; MFTimer 1 DMA Counter Pointer Register.
t1_dcpr   = r4

T0_DAPR  := R241                    ; MFTimer 0 DMA Address Pointer Register.
t0_dapr   = r1

T1_DAPR  := R245                    ; MFTimer 1 DMA Address Pointer Register.
t1_dapr   = r5

T0_IVR   := R242                    ; MFTimer 0 Interrupt Vector Register.
t0_ivr    = r2

T1_IVR   := R246                    ; MFTimer 1 Interrupt Vector Register.
t1_ivr    = r6
```

APPENDIX B: Symbols.inc listing

```
T0_IDCR  := R243          ; MFTimer 0 Interrupt/DMA Control Register.
t0_idcr   = r3
T1_IDCR  := R247          ; MFTimer 1 Interrupt/DMA Control Register.
t1_idcr   = r7
T2_DCPR  := R240          ; MFTimer 2 DMA Counter Pointer Register.
t2_dcpr   = r0
T3_DCPR  := R244          ; MFTimer 3 DMA Counter Pointer Register.
t3_dcpr   = r4
T2_DAPR  := R241          ; MFTimer 2 DMA Address Pointer Register.
t2_dapr   = r1
T3_DAPR  := R245          ; MFTimer 3 DMA Address Pointer Register.
t3_dapr   = r5
T2_IVR   := R242          ; MFTimer 2 Interrupt Vector Register.
t2_ivr    = r2
T3_IVR   := R246          ; MFTimer 3 Interrupt Vector Register.
t3_ivr    = r6
T2_IDCR  := R243          ; MFTimer 2 Interrupt/DMA Control Register.
t2_idcr   = r3
T3_IDCR  := R247          ; MFTimer 3 Interrupt/DMA Control Register.
t3_idcr   = r7
plm      := 07h           ; Priority level mask
swen     := 08h           ; Swap function enable mask
dctd     := 10h           ; DMA compare transaction destination mask
dcts     := 20h           ; DMA capture transaction source mask
cme      := 40h           ; Compare 0 end of block mask
cpe      := 80h           ; Capture 0 end of block mask
T_IOCR   := R248          ; MFTimer I/O connection register
t_iocr    = r8
sc0      := 01h           ; TxOUTA and TxINA connection bit
                          ; for even MFTimer
sc1      := 02h           ; TxOUTA and TxINA connection bit
                          ; for odd MFTimer
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;              ST9 FAMILY A/D CONVERTER REGISTERS.

;*************************************************************************

AD0_PG   := 63                          ; A/D converter registers page
AD1_PG   := 62                          ; second A/D unit
AD_D0R   := R240                        ; Channel 0 data register
ad_d0r    = r0                          ; Channel 0 data register
AD_D1R   := R241                        ; Channel 1 data register
ad_d1r    = r1                          ; Channel 1 data register
AD_D2R   := R242                        ; Channel 2 data register
ad_d2r    = r2                          ; Channel 2 data register
AD_D3R   := R243                        ; Channel 3 data register
ad_d3r    = r3                          ; Channel 3 data register
AD_D4R   := R244                        ; Channel 4 data register
ad_d4r    = r4                          ; Channel 4 data register
AD_D5R   := R245                        ; Channel 5 data register
ad_d5r    = r5                          ; Channel 5 data register
AD_D6R   := R246                        ; Channel 6 data register
ad_d6r    = r6                          ; Channel 6 data register
AD_D7R   := R247                        ; Channel 7 data register
ad_d7r    = r7                          ; Channel 7 data register
AD_LT6R  := R248                        ; Channel 6 lower threshold register
ad_lt6r   = r8                          ; Channel 6 lower threshold register
AD_LT7R  := R249                        ; Channel 7 lower threshold register
ad_lt7r   = r9                          ; Channel 7 lower threshold register
AD_UT6R  := R250                        ; Channel 6 upper threshold register
ad_ut6r   = r10                         ; Channel 6 upper threshold register
AD_UT7R  := R251                        ; Channel 7 upper threshold register
ad_ut7r   = r11                         ; Channel 7 upper threshold register
AD_CRR   := R252                        ; Compare result register
ad_crr    = r12                         ; Compare result register
         .defstr AD_c6l    “ad_crr.4” ; Compare channel 6 lower bit
         .defstr AD_c7l    ”ad_crr.5” ; Compare channel 7 lower bit
         .defstr AD_c6u    “ad_crr.6” ; Compare channel 6 upper bit
         .defstr AD_c7u    ”ad_crr.7” ; Compare channel 7 upper bit
c6l      := ( 1 <- 4 )                  ; Compare channel 6 lower mask
c7l      := ( 1 <- 5 )                  ; Compare channel 7 lower mask
c6u      := ( 1 <- 6 )                  ; Compare channel 6 upper mask
c7u      := ( 1 <- 7 )                  ; Compare channel 7 upper mask
```

APPENDIX B: Symbols.inc listing

```
AD_CLR   := R253                      ; Control logic register
ad_clr    = r13                       ; Control logic register
         .defstr AD_st     “ad_clr.0” ; start/stop bit
         .defstr AD_cont   ”ad_clr.1” ; Continuous mode
         .defstr AD_pow    “ad_clr.2” ; power up/down control
         .defstr AD_intg   ”ad_clr.3” ; internal trigger
         .defstr AD_extg   “ad_clr.4” ; External trigger
st       := ( 1 <- 0 )                ; start/stop bit mask
cont     := ( 1 <- 1 )                ; Continuous mode mask
pow      := ( 1 <- 2 )                ; power up/down control mask
intg     := ( 1 <- 3 )                ; internal trigger mask
extg     := ( 1 <- 4 )                ; External trigger mask
sch      := 0E0h                      ; scan channel selection mask
AD_ICR   := R254                      ; interrupt control register
ad_icr    = r14                       ; interrupt control register
         .defstr AD_awdi   “ad_icr.4” ; analog watch-dog interrupt
         .defstr AD_eci    ”ad_icr.5” ; End of count interrupt
         .defstr AD_awd    “ad_icr.6” ; analog watch-dog pending flag
         .defstr AD_ecv    ”ad_icr.7” ; End of conversion pending flag
AD_prl   := 07h                       ; priority level mask
awdi     := ( 1 <- 4 )                ; analog watch-dog interrupt mask
eci      := ( 1 <- 5 )                ; End of count interrupt mask
awd      := ( 1 <- 6 )                ; analog watch-dog pending flag
ecv      := ( 1 <- 7 )                ; End of conversion pending flag
AD_IVR   := R255                      ; interrupt vector register
ad_ivr    = r15                       ; interrupt vector register
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;     ST9 FAMILY SERIAL COMMUNICATION INTERFACE REGISTERS.

;*************************************************************************

SCI1_PG  := 24                          ; SCI1 control registers page
SCI2_PG  := 25                          ; SCI2 control registers page
SCI3_PG  := 26                          ; SCI3 control registers page
SCI4_PG  := 27                          ; SCI4 control registers page


S_RDCPR  := R240                        ; receive DMA counter pointer register
s_rdcpr   = r0                          ; receive DMA counter pointer register
S_RDAPR  := R241                        ; receive DMA address pointer register
s_rdapr   = r1                          ; receive DMA address pointer register
S_TDCPR  := R242                        ; transmit DMA counter pointer register
s_tdcpr   = r2                          ; transmit DMA counter pointer register
S_TDAPR  := R243                        ; transmit DMA address pointer register
s_tdapr   = r3                          ; transmit DMA address pointer register
S_IVR    := R244                        ; interrupt vector register
s_ivr     = r4                          ; interrupt vector register
S_ACR    := R245                        ; address compare register
s_acr     = r5                          ; address compare register
S_IMR    := R246                        ; interrupt mask register
s_imr    = r6                           ; interrupt mask register
         .defstr S_txdi    “s_imr.0”    ; transmitter data interrupt
         .defstr S_rxdi    ”s_imr.1”    ; receiver data interrupt
         .defstr S_rxb     “s_imr.2”    ; receiver break
         .defstr S_rxa     ”s_imr.3”    ; receiver address
         .defstr S_rxe     “s_imr.4”    ; receiver error
         .defstr S_txeob   ”s_imr.5”    ; transmit end of block
         .defstr S_rxeob   “s_imr.6”    ; receive end of block
         .defstr S_hsn     ”s_imr.7”    ; Holding or shift register empty.
txdi     := ( 1 <- 0 )                  ; transmitter data interrupt mask
rxdi     := ( 1 <- 1 )                  ; receiver data interrupt mask
rxb      := ( 1 <- 2 )                  ; receiver break mask
rxa      := ( 1 <- 3 )                  ; receiver address mask
rxe      := ( 1 <- 4 )                  ; receiver error mask
txeob    := ( 1 <- 5 )                  ; transmit end of block mask
rxeob    := ( 1 <- 6 )                  ; receive end of block mask
hsn      := ( 1 <- 7 )                  ; Holding or shift register empty mask.
```

APPENDIX B: Symbols.inc listing

```
S_ISR    := R247                        ; interrupt status register
s_isr     = r7                          ; interrupt status register

         .defstr S_txsem   "s_isr.0"    ; transmit shift register empty
         .defstr S_txhem   "s_isr.1"    ; transmit hold register empty
         .defstr S_rxdp    "s_isr.2"    ; received data pending bit
         .defstr S_rxbp    "s_isr.3"    ; received break pending bit
         .defstr S_rxap    "s_isr.4"    ; received address pending bit
         .defstr S_pe      "s_isr.5"    ; parity error pending bit
         .defstr S_fe      "s_isr.6"    ; framing error pending bit
         .defstr S_oe      "s_isr.7"    ; overrun error pending bit

txsem    := ( 1 <- 0 )                  ; transmit shift register empty mask
txhem    := ( 1 <- 1 )                  ; transmit hold register empty mask
rxdp     := ( 1 <- 2 )                  ; received data pending mask
rxbp     := ( 1 <- 3 )                  ; received break pending mask
rxap     := ( 1 <- 4 )                  ; received address pending mask
pe       := ( 1 <- 5 )                  ; parity error pending mask
fe       := ( 1 <- 6 )                  ; framing error pending mask
oe       := ( 1 <- 7 )                  ; overrun error pending mask

S_RXBR   := R248                        ; receive buffer register
s_rxbr    = r8                          ; receive buffer register

S_TXBR   := R248                        ; transmit buffer register
s_txbr    = r8                          ; transmit buffer register

S_IDPR   := R249                        ; interrupt/DMA priority register
s_idpr    = r9                          ; interrupt/DMA priority register

         .defstr S_txd     "s_idpr.3"   ; transmitter DMA
         .defstr S_rxd     "s_idpr.4"   ; receiver DMA
         .defstr S_sa      "s_idpr.5"   ; set address
         .defstr S_sb      "s_idpr.6"   ; set break
         .defstr S_amen    "s_idpr.7"   ; address mode enable

S_pri    := 07h                         ; interrupt/DMA priority mask
txd      := ( 1 <- 3 )                  ; transmitter DMA mask
rxd      := ( 1 <- 4 )                  ; receiver DMA mask
sa       := ( 1 <- 5 )                  ; set address mask
sb       := ( 1 <- 6 )                  ; set break mask
amen     := ( 1 <- 7 )                  ; address mode enable mask

S_CHCR   := R250                        ; Character configuration register
s_chcr    = r10                         ; Character configuration register

wl5      := 000h                        ; 5 bits data word mask
wl6      := 001h                        ; 6 bits data word mask
wl7      := 002h                        ; 7 bits data word mask
wl8      := 003h                        ; 8 bits data word mask
sb10     := 000h                        ; 1.0 stop bit mask
sb15     := 004h                        ; 1.5 stop bit mask
sb20     := 008h                        ; 2.0 stop bit mask
sb25     := 00Ch                        ; 2.5 stop bit mask
ab       := 010h                        ; address bit insertion mask
pen      := 020h                        ; parity enable mask
ep       := 040h                        ; Even parity mask
oddp     := 000h                        ; odd parity mask
am       := 080h                        ; address mode mask
```

APPENDIX B: Symbols.inc listing

```
S_CCR    := R251                          ; Clock configuration register
s_ccr     = r11                           ; Clock configuration register
         .defstr S_stpen   “s_ccr.0”      ; stick parity enable
         .defstr S_lben    ”s_ccr.1”      ; loop back enable
         .defstr S_aen     “s_ccr.2”      ; auto echo enable
         .defstr S_cd      ”s_ccr.3”      ; Clock divider
         .defstr S_xbrg    “s_ccr.4”      ; External baud rate generator source
         .defstr S_xrx     ”s_ccr.5”      ; External receiver source
         .defstr S_oclk    “s_ccr.6”      ; output clock selection
         .defstr S_txclk   ”s_ccr.7”      ; transmit clock selection
stpen    := ( 1 <- 0 )                    ; stick parity enable mask
lben     := ( 1 <- 1 )                    ; loop back enable mask
aen      := ( 1 <- 2 )                    ; auto echo enable mask
cd       := ( 1 <- 3 )                    ; Clock divider mask
xbrg     := ( 1 <- 4 )                    ; External baud rate generator source mask
xrx      := ( 1 <- 5 )                    ; External receiver source mask
oclk     := ( 1 <- 6 )                    ; output clock selection mask
txclk    := ( 1 <- 7 )                    ; transmit clock selection mask
S_BRGR   := RR252                         ; baud rate generator register
s_brgr    = rr12                          ; baud rate generator register
S_BRGHR  := R252                          ; baud rate generator reg. high
s_brghr   = r12                           ; baud rate generator reg. high
S_BRGLR  := R253                          ; baud rate generator reg. low
s_brglr   = r13                           ; baud rate generator reg. low
;.page
```

APPENDIX B: Symbols.inc listing

```
;*************************************************************************

;                          ST9040 SECURITY REGISTER.

;*************************************************************************

SEC_PG   := 59                        ; Security register page
SECR      = R255
secr      = r15

         .defstr tlck      “secr.0”   ; test lock bit
         .defstr wf1       ”secr.1”   ; write fuse 1 bit
         .defstr hlck      “secr.2”   ; hardware lock bit
         .defstr wf2       ”secr.3”   ; write fuse 2 bit
         .defstr f2tst     “secr.4”   ; select fuse 2 bit
         .defstr slck      ”secr.7”   ; software lock bit

tlckm    := ( 1 <- 0 )                ; test lock bit mask
wf1m     := ( 1 <- 1 )                ; write fuse 1 bit mask
hlckm    := ( 1 <- 2 )                ; hardware lock bit mask
wf2m     := ( 1 <- 3 )                ; write fuse 2 bit mask
f2tstm   := ( 1 <- 4 )                ; select fuse 2 bit mask
slckm    := ( 1 <- 7 )                ; software lock bit mask

    .list
```