Chapter 14

• Memory Systems
  – Internal Memories
    • FLASH – Program Memory – 512 kB
    • RAM – Data Memory – 14 kB
    • EEPROM – 4 kB
  – External Memory Expansion
  – Memory Timing Analysis
Memory Map

Figure 1-2 MC9S12DP512 Memory Map

* Assuming that all I/O pins were driven, and port 0 drove MCLR during IDL, it is in reset in normal expanded wide or narrow mode.
Module Mapping and Control

• Module Mapping Control (MMC)

![Module Mapping Control Block Diagram](image)

Figure 1-1 Module Mapping Control Block Diagram
MMC Features

- Registers for mapping of address space for on-chip RAM, EEPROM, and FLASH (or ROM) memory blocks and associated registers
- Memory mapping control and selection based upon address decode and system operating mode
- Core address bus control
- Core data bus control and multiplexing
- Core security state decoding
- Emulation chip select signal generation (ECS)
- External chip select signal generation (XCS)
- Internal memory expansion
- External stretch and ROM mapping control functions via the MISC register
- Reserved registers for test purposes
- Configurable system memory options defined at integration of Core into the System-on-a-Chip (SOC).
Internal Resource Remapping (1 of 2)

- The on-chip register block, SRAM, EEPROM, and flash memory have default locations within the 64 KB standard address.
- On-chip register block, SRAM, and EEPROM can be relocated to other places.
- It is advisable to explicitly establish these resource locations during the initialization phase of program execution to protect the inadvertent modification later.
- Writes to resource remapping registers require one cycle to take effect.
- If conflict occurs when mapping resources, the register block will take precedence over other resources.
Internal Resource Remapping (2 of 2)

Table 14.1 Mapping precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BDM firmware or register space</td>
</tr>
<tr>
<td>2</td>
<td>Internal register space</td>
</tr>
<tr>
<td>3</td>
<td>SRAM block</td>
</tr>
<tr>
<td>4</td>
<td>EEPROM</td>
</tr>
<tr>
<td>5</td>
<td>Flash memory</td>
</tr>
<tr>
<td>6</td>
<td>Remaining external memory</td>
</tr>
</tbody>
</table>

Example in the dp512:

- The EEPROM and register space overlap. The register space takes precedence.
Register Block Mapping

- The register block may be 1 KB or 2 KB in size.
  - Register block remapping is controlled by the INITRG register.
  - The register block can be remapped to any 2 KB boundary within the first 32 KB of memory (A15=0).
  - Default Offset 0x0000

Figure 14.1 Contents of the INITRG register

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
SRAM Mapping

- SRAM can be remapped to any 2 KB boundary within the 64 KB memory space.
  - The SRAM remapping is controlled by the INITRM register.
  - RAM15 to RAM11 determine the upper five bits of the base address for the SRAM.
  - Default Offset 0x0800

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM15</td>
<td>RAM14</td>
<td>RAM13</td>
<td>RAM12</td>
<td>RAM11</td>
<td>0</td>
<td>0</td>
<td>RAMHAL</td>
</tr>
</tbody>
</table>

Value after reset:

- RAM15~RAM11: Internal RAM map position
  - These bits determine the upper five bits of the base address for the system's internal RAM array.
- RAMHAL: RAM high-align
  - 0 = aligns the RAM to the lowest address ($0000) of the mappable space
  - 1 = aligns the RAM to the highest address ($FFFF) of the mapping space.

Figure 14.2 RAM initialization register (INITRM)

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
EEPROM Mapping

- The EEPROM can be remapped to any 2 KB boundary within the 64 KB memory space.
  - The remapping of the EEPROM is controlled by the INITEE register.
  - The EEPROM is enabled by the bit 0 of the INITEE register.
  - Default: dp512 is 0x0000

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE15</td>
<td>EE14</td>
<td>EE13</td>
<td>EE12</td>
<td>EE11</td>
<td>0</td>
<td>0</td>
<td>EEOON</td>
</tr>
</tbody>
</table>

- value after reset: 0 0 0 1 0 0 0 1

- EE15~EE11: Internal EEPROM map position
  - These bits specify the upper 5 bits of the 16-bit registers address.
  - These five bits can be written only once in normal modes and can be written many times in special modes. There is no restriction on the reading of this register.

- EEOON: Internal EEPROM On (Enabled reading).
  - 0 = Removes EEPROM from the map.
  - 1 = Places the on-chip EEPROM in the memory map.

Figure 14.3 Contents of the INITEE register

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Miscellaneous System Control Register

- This register enables/disables the on-chip ROM and allows one to stretch the length of the external bus cycle.
  - The value of the ROMONE pin is latched to the ROMON bit of the MISC register after reset.
  - Default:

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>EXSTR1</td>
<td>EXSTR0</td>
<td>ROMHM</td>
<td>ROMON</td>
</tr>
</tbody>
</table>

EXSTR1~EXSTR0: External access stretch bits 1 and 0
00 = no stretch to external bus cycle
01 = stretch the external bus cycle by one E cycle
10 = stretch the external bus cycle by two E cycles
11 = stretch the external bus cycle by three E cycles

ROMHM: Flash EEPROM or ROM only in second half of memory map
0 = The fixed page (s) of flash EEPROM or ROM in the lower half of the memory map can be accessed.
1 = Disable direct access to the flash EEPROM or ROM in the lower half of the memory map. These physical locations of flash memory can still be accessed through the program page window.

ROMON: Enable flash memory or ROM
0 = Disable the flash memory or ROM in the memory map
1 = Enable the flash memory or ROM in the memory map

Note:
1. The reset state of this bit is determined at the chip integration level.

Figure 14.4 Contents of the MISC register
Memory Size Register Zero (MEMSIZ0)

- This register reflects the size of the on-chip I/O register block, EEPROM, and SRAM.
  - Read-only
  - Default: 0x26 – 1 kB reg, 4 kB EEPROM, 14 kB RAM

<table>
<thead>
<tr>
<th>REG_SW0</th>
<th>0</th>
<th>EEP_SW1</th>
<th>EEP_SW0</th>
<th>0</th>
<th>RAM_SW2</th>
<th>RAM_SW1</th>
<th>RAM_SW0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base+$1C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

reset: -- -- -- -- -- -- -- --

REG_SW0: Allocated system register space
0 = Allocated system register space size is 1 KB
1 = Allocated system register space size is 2 KB
EEP_SW1~EEP_SW0: Allocated system EEPROM memory space
00 = 0 KB
01 = 2 KB
10 = 4 KB
11 = 8 KB
RAM_SW2~RAM_SW0: Allocated system RAM memory space
The allocated system RAM space size is as given in Table 14.2.

Figure 14.5 Memory size register zero (MEMSIZ0)
Table 14.2 Allocated RAM memory space

<table>
<thead>
<tr>
<th>RAM_SW2:RAM_SW0</th>
<th>Allocated RAM space</th>
<th>RAM mappable region</th>
<th>INITRM bits used</th>
<th>RAM reset base address$^{(1)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2K bytes</td>
<td>2K bytes</td>
<td>RAM15~RAM11</td>
<td>$0800</td>
</tr>
<tr>
<td>001</td>
<td>4K bytes</td>
<td>4K bytes</td>
<td>RAM15~RAM12</td>
<td>$0000</td>
</tr>
<tr>
<td>010</td>
<td>6K bytes</td>
<td>8K bytes$^{(2)}$</td>
<td>RAM15~RAM13</td>
<td>$0800</td>
</tr>
<tr>
<td>011</td>
<td>8K bytes</td>
<td>8K bytes</td>
<td>RAM15~RAM13</td>
<td>$0000</td>
</tr>
<tr>
<td>100</td>
<td>10K bytes</td>
<td>16K bytes$^{(2)}$</td>
<td>RAM15~RAM14</td>
<td>$1800</td>
</tr>
<tr>
<td>101</td>
<td>12K bytes</td>
<td>16K bytes$^{(2)}$</td>
<td>RAM15~RAM14</td>
<td>$1000</td>
</tr>
<tr>
<td>110</td>
<td>14K bytes</td>
<td>16K bytes$^{(2)}$</td>
<td>RAM15~RAM14</td>
<td>$0800</td>
</tr>
<tr>
<td>111</td>
<td>16K bytes</td>
<td>16K bytes</td>
<td>RAM15~RAM14</td>
<td>$0000</td>
</tr>
</tbody>
</table>

Notes:
1. The RAM reset base address is based on the reset value of the INITRM register, $09.
2. Alignment of the allocated RAM space within the RAM mappable region is dependent on the value of RAMHAL.
Memory Size Register One (MEMSIZ1)

- This register is read-only and reflects the state of the flash or ROM physical memory and paging switches at the core boundary.
  - Default: 0x82
    - 48 kB Flash Space, 512 kB Flash

<table>
<thead>
<tr>
<th>ROM_SW1~ROM_SW0</th>
<th>Off-chip space</th>
<th>On-chip space</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>876 KB</td>
<td>128 KB</td>
</tr>
<tr>
<td>01</td>
<td>768 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td>10</td>
<td>512 KB</td>
<td>512 KB</td>
</tr>
<tr>
<td>11</td>
<td>0 KB</td>
<td>1 MB</td>
</tr>
</tbody>
</table>

Figure 14.6 Memory size register one (MEMSIZ1)
Expanded Memory Mapping

• 16 kB Memory blocks can be paged in the memory map
  – The pages appear in the addresses 0x8000 to 0xBFFF
• The PPAGE register defined which 16 kB page is placed in this space using 6 bits, PIX0-5
  – The 32 internal flash pages are addressed as 0x20 to 0x3F
  – External pages are addressed as 0x00 to 0x1F
    When addressing external memory, these 6-bits appear on PTK.0-5
    The lower 14 bits of the external address are on PortB (LSB) and PortA (MSB)

• See the memory map on the following page
FLASH MAP

$FF00 - $FFFF, Flash Protection/Security Field

Note: $20-$3F correspond to the PPAGE register content

Figure 3-1 Flash Memory Map

Software & Hardware
Expanded Memory Mapping

The HCS12 uses the PPAGE register to determine which of the 64 possible 16 KB pages is active in the program window.

- The direct concatenation of the page number and the page offset does not result in linear addresses.

Motorola proposed the following method to address memory above 64 KB:

1. Use $00 to $3F as the page numbers for the 64 16-KB pages.
2. Use higher page numbers to select on-chip flash memory (The rest of the 512 kB available).
   For example, the HCS12 devices with 512KB flash memory assign $20 to $3F as the page numbers of the 32 on-chip 16 KB pages. Use $00 to $1F as page numbers for external memory.
3. Use linear addresses to address external memory and also use linear method to address machine codes for S records.
4. Compute the page number and page address within the $8000~$BFFF page window using the following equations:

\[
\text{PageNum} = \frac{\text{SRecAddr}}{\text{PPAGEWinSize}} \quad (14.1)
\]
\[
\text{PageWinAddr} = (\text{SRecAddr} \% \text{PPAGEWinSize}) + \text{PPAGEWinSTART} \quad (14.2)
\]

where,
- \( \text{SRecAddr} \) is the linear address used in S records
- \( \text{PageWinAddr} \) is a number between $8000 and $BFFF.
- \( \text{PPAGEWinSize} = 16K = $4000 \)
PPAGEWinStart = $8000

5. Compute SRecAddr from PageNum and PageWinAddr as follows:
   \[ \text{SRecAddr} = \text{PageNum} \times \text{PPAGEWinSize} + \text{PageWinAddr} - \text{PPAGEWinStart} \]  

6. When addressing external memory, the highest 6 address bits appear on XADDR19 ~ XADDR14 and the lowest 14 address bits appear on A13 ~ A0.

<table>
<thead>
<tr>
<th>PPAGE value</th>
<th>S-Record Address range</th>
<th>A15~A0</th>
<th>page + A13~A0</th>
<th>Memory type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00~$2F</td>
<td>$00000~$BFFFF</td>
<td>1</td>
<td>$00000~$BFFFF</td>
<td>off-chip memory</td>
</tr>
<tr>
<td>$30</td>
<td>$C0000~$C3FFF</td>
<td></td>
<td>$C0000~$C3FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$31</td>
<td>$C4000~$C7FFF</td>
<td></td>
<td>$C4000~$C7FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$32</td>
<td>$C8000~$CBFFF</td>
<td></td>
<td>$C8000~$CBFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$33</td>
<td>$CC000~$CFFFF</td>
<td></td>
<td>$CC000~$CFFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$34</td>
<td>$D0000~$D3FFF</td>
<td></td>
<td>$D0000~$D3FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$35</td>
<td>$D4000~$D7FFF</td>
<td></td>
<td>$D4000~$D7FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$36</td>
<td>$D8000~$DBFFF</td>
<td></td>
<td>$D8000~$DBFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$37</td>
<td>$DC000~$DFFFF</td>
<td></td>
<td>$DC000~$DFFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$38</td>
<td>$E0000~$E3FFF</td>
<td></td>
<td>$E0000~$E3FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$39</td>
<td>$E4000~$E7FFF</td>
<td></td>
<td>$E4000~$E7FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3A</td>
<td>$E8000~$EBFFF</td>
<td></td>
<td>$E8000~$EBFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3B</td>
<td>$EC000~$EFFFF</td>
<td></td>
<td>$EC000~$EFFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3C</td>
<td>$F0000~$F3FFF</td>
<td></td>
<td>$F0000~$F3FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3D</td>
<td>$F4000~$F7FFF</td>
<td></td>
<td>$F4000~$F7FFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3E</td>
<td>$F8000~$FBFFF</td>
<td></td>
<td>$F8000~$FBFFF</td>
<td>on-chip flash</td>
</tr>
<tr>
<td>$3F</td>
<td>$FC000~$FFFFF</td>
<td></td>
<td>$FC000~$FFFFF</td>
<td>on-chip flash</td>
</tr>
</tbody>
</table>

Note.
1. Repetition of $0000~$3FFF, $4000~$3FFF, $8000~$BFFFF, and $C000~$FFFF for 12 times.
Example 14.2 & 14.3

What are the PageNum and PageWinAddr for the SRecAddr of $E1003?

• Solution: Apply equation 14.1 and 14.2 as follows:
  PageNum = $E1003 / $4000 = $38
  PageWinAddr = ($E1003 % $4000) + $8000 = $1003 + $8000 = $9003

What is the corresponding SRecAddr for the pair of (PageNum, PageWinAddr) equal to ($20, $A003)?

• Solution: Apply equation 14.3 as follows:
  SRecAddr = $20 \times $4000 + $A003 - $8000 = $82003
On-Chip Flash Memory (1 of 3)

- The on-chip flash memory size can be 32 KB, 64 KB, 128 KB, 256KB, and 512 KB for the HCS12 devices.
- A flash memory larger than 64 KB is divided into 64-KB blocks (128-KB blocks for dp512).
- Programming and erasure of flash memory are performed by sending commands to the command register.
- The flash memory has a flexible protection scheme against accidental programming and erasure.
- The flash memory also implements security measures to prevent the application code from being pirated.
- The flash memory map is shown in Figure 14.8.
On-Chip Flash Memory (3 of 3)

• Default Conditions:
  – The memory space from $0000 to $3FFF has been occupied by I/O registers, EEPROM, and SRAM and is not available to the flash memory.
  – The first 128 KB of the flash memory is referred to as block 0 and is assigned with page numbers $38 to $3F.
  – The pages with addresses from $4000 to $7FFF, $8000 to $BFFF, and $C000 to $FFFF are assigned with the page numbers of $3E, $3D, and $3F.
Flash Block Diagram

Figure 1-1 Module Block Diagram

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Flash Memory Protection (1 of 3)

- Flash memory protection is provided to prevent against accidental erasure or programming.
- Flash protection is controlled by a flash protection register (FPROT).
- For HCS12 devices with multiple flash blocks, there is a separate flash protection register for each flash block.
- Flash protection registers share a common address, with the active register selected by the bank select bits of the flash configuration register (FCNFG).
- During the HCS12 reset sequence (execution of reset start up routine), the flash protection registers for each flash block are loaded from the programmed bytes within a flash block.
- For the MC9S12DP512, the locations $FF0A,$FF0B, $FF0C, and $FF0D store the protection information of block three, two, one, and zero, respectively (4 x 128 kB blocks).
Flash Protection/Option Fields

Table 14.5 Flash protection/options field

<table>
<thead>
<tr>
<th>Address</th>
<th>Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FF00~$FF07</td>
<td>8</td>
<td>Backdoor comparison keys</td>
</tr>
<tr>
<td>$FF08~$FF09</td>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>$FF0A</td>
<td>1</td>
<td>Block 3 flash protection byte</td>
</tr>
<tr>
<td>$FF0B</td>
<td>1</td>
<td>Block 2 flash protection byte</td>
</tr>
<tr>
<td>$FF0C</td>
<td>1</td>
<td>Block 1 flash protection byte</td>
</tr>
<tr>
<td>$FF0D</td>
<td>1</td>
<td>Block 0 flash protection byte</td>
</tr>
<tr>
<td>$FF0E</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>$FF0F</td>
<td>1</td>
<td>Flash options/security byte</td>
</tr>
</tbody>
</table>

- Special addresses used for Flash protection
Flash Memory Protection (2 of 3)

• The contents of each FPROT register determine whether the entire block or just a subsection is protected from being accidentally erased or programmed.

• Each flash block (128 KB) can be entirely protected, or can have one or two separate protected areas.
  – One area to be protected is the lower protected block starting at a point 32 KB below the maximum flash block address and is extensible toward higher addresses.
  – The other area is the upper protected area that ends at the top of the flash block and is extended toward lower addresses.

• The lower protected area and upper protected area do not meet up.

• To change the flash protection that will be loaded on reset, the upper sector of the flash memory must be unprotected, then the flash protect/security byte located as in Table 14.5 must be written into.
Flash Memory Protection (3 of 3)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPOPEN</td>
<td>NV6</td>
<td>FPHDIS</td>
<td>FPHS1</td>
<td>FPHS0</td>
<td>FPLDIS</td>
<td>FPLS1</td>
<td>FPLS0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

reset: $FFFF_FFFFFF$

FPOPEN: Opens the flash for program or erase
0 = The whole flash block is protected. In this case, bits 5 to 0 have no effect.
1 = The flash sectors not protected are enabled for program or erase.

NV6: Not volatile flag bit.
This bit is available for nonvolatile flag usage

FPHDIS: Flash protection higher address range disable
0 = higher address range protection enabled
1 = higher address range protection disabled

FPHS1~FPHS0: Flash protection higher address size
00 = 2 KB
01 = 4 KB
10 = 8 KB
11 = 16 KB

FPLDIS: Flash protection lower address range disable
0 = lower address range protection enabled
1 = lower address range protection disabled

FPLS1~FPLS0: Flash protection lower address size
00 = 512 bytes
01 = 1 KB
10 = 2 KB
11 = 4 KB

Figure 14.9 FPROT register
Flash Related Registers

- FCLKDIV
- FSEC
- FTSTMOD
- FCNFG
- FSTAT
- FCMD
FCLKDIV Register

- The flash programming and erasure timing is controlled by this register.
- The only configuration to be done on the clock signal is to set the prescaler to the bus clock.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDIVLD</td>
<td>PRDIV8</td>
<td>FDIV5</td>
<td>FDIV4</td>
<td>FDIV3</td>
<td>FDIV2</td>
<td>FDIV1</td>
<td>FDIV0</td>
</tr>
</tbody>
</table>

- **FDIVLD**: Clock divider loaded
  - 0 = Register has not been written.
  - 1 = Register has been written to since the last reset.
- **PRDIV8**: Enable prescaler by 8
  - 0 = The input oscillator clock is directly fed into the FCLKDIV divider.
  - 1 = Enables a divide-by-8 prescaler, to divide the flash module input oscillator clock before feeding into the CLKDIV divider.
- **FDIV[5:0]**: Clock divider bits
  - The combination of PRDIV8 and FDIV[5:0] effectively divides the flash module input oscillator clock down to a frequency of 150KHz~200KHz. The maximum divide ratio is 512.

Figure 14.10 Flash clock divider register (FCLKDIV)
Flash Security Register (FSEC)

- All the bits of this register are readable, but not writable.
- This register has no reset state and must be loaded from flash memory and, hence, its state after reset is ‘F’.
- If the flash is unsecured using the backdoor key access, the SEC bits are forced to 10.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYEN</td>
<td>NV6</td>
<td>NV5</td>
<td>NV4</td>
<td>NV3</td>
<td>NV2</td>
<td>SEC1</td>
<td>SEC0</td>
</tr>
<tr>
<td>reset:</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- **KEYEN**: Enable backdoor key to security
  - 0 = backdoor to flash is disabled
  - 1 = backdoor to flash is enabled

- **NV6~NV2**: Non volatile flag bits
  - These 5 bits are available to the user as non-volatile flags

- **SEC[1:0]**: Memory security bits
  - 00 = secured
  - 01 = secured
  - 10 = unsecured
  - 11 = secured

Figure 14.11 Flash security register (FSEC)
Flash Test Mode Register (FTSTMOD)

- This register is not banked and is mainly used to control the flash memory test modes.
- The WRALL bit allows the user to launch a command on all blocks in parallel.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>WRALL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
</tbody>
</table>

reset: 0 0 0 0 0 0 0 0

WRALL: Write to all register banks
0 = Write only to the bank selected via BKSEL
1 = Write to all register banks

Figure 14.12 Flash test mode register (FTSTMOD)

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Flash Configuration Register (FCNFG)

- This register enables the flash interrupts, gates the security backdoor writes, and selects the register bank to be operated on.
- For a HCS12 device with multiple flash memory banks, the BKSEL1 and BKSEL0 bits select the set of control registers to be active to be accessed.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>CBEIE: Command buffer empty interrupt enable</td>
</tr>
<tr>
<td></td>
<td>0 = Command buffer empty interrupts disabled</td>
</tr>
<tr>
<td></td>
<td>1 = An interrupt will be requested whenever the CBEIF flag is set.</td>
</tr>
<tr>
<td>5-4</td>
<td>CCIE: Command completion interrupt enable</td>
</tr>
<tr>
<td></td>
<td>0 = Command complete interrupts disabled</td>
</tr>
<tr>
<td></td>
<td>1 = An interrupt will be requested whenever the CCIF flag is set.</td>
</tr>
<tr>
<td>3-2</td>
<td>KEYACC: Enable security key writing</td>
</tr>
<tr>
<td></td>
<td>0 = Flash writes are interpreted as the start of a program or erase sequence.</td>
</tr>
<tr>
<td></td>
<td>1 = Writes to flash array are interpreted as keys to open the backdoor. Reads of the flash array return invalid data.</td>
</tr>
<tr>
<td>1-0</td>
<td>BKSEL[1:0]: Register bank select</td>
</tr>
<tr>
<td></td>
<td>00 = flash 0</td>
</tr>
<tr>
<td></td>
<td>01 = flash 1</td>
</tr>
<tr>
<td></td>
<td>10 = flash 2</td>
</tr>
<tr>
<td></td>
<td>11 = flash 3</td>
</tr>
</tbody>
</table>

Figure 14.13 Flash configuration register (FCNFG)

Flash Status Register (FSTAT)

- The programming and erase of flash memory is controlled by a finite state machine.
- The FSTAT register defines the flash state machine command status and flash array access, protection, and bank verify status. This register is banked.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>CBEIF: Command buffer empty interrupt flag</td>
<td>0 = Command buffers are full, 1 = Command buffers are ready to accept a new command</td>
</tr>
<tr>
<td>6</td>
<td>CCIF: Command completion interrupt flag</td>
<td>0 = Command in progress, 1 = All commands are completed</td>
</tr>
<tr>
<td>5</td>
<td>PVOL: Protection violation</td>
<td>0 = No protection violation has occurred, 1 = A protection violation has occurred</td>
</tr>
<tr>
<td>4</td>
<td>ACCERR: Flash access error</td>
<td>0 = No failure, 1 = Access error has occurred</td>
</tr>
<tr>
<td>3</td>
<td>BLANK: Array has been verified as erased</td>
<td>0 = If an erase and verify command has been requested, and the CCIF flag is set, then a zero in BLANK indicates that the block is not erased, 1 = Flash block verifies as erased</td>
</tr>
</tbody>
</table>

Figure 14.14 Flash status register (FSTAT)
Flash Command Register (FCMD)

- This register defines the flash commands.
- This register is banked.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CMDB6</td>
<td>CMDB5</td>
<td>0</td>
<td>0</td>
<td>CMDB2</td>
<td>0</td>
<td>CMDB0</td>
</tr>
</tbody>
</table>

reset: 1 1 0 0 0 0 0 0

CMDB6, CMDB5, CMDB2, and CMDB0: Command bits
Valid commands includes the following:
$05 = Erase and verify
$20 = Program a word (two bytes)
$40 = Sector erase
$41 = Bulk erase

Figure 14.15 Flash command register (FCMD)
Procedure to Execute the Flash Command

- **Step 1**
  - Configure the FCLKDIV register properly.
- **Step 2**
  - Make sure the sector to be erased is not protected.
- **Step 3**
  - Erase the word to be programmed and make sure it is not protected.
- **Step 4**
  - Make sure the flash address to be programmed or erased is word-aligned.
- **Step 5**
  - If the flash address is in the range of $8000 to $BFFF, then make sure to write into the PPAGE register to select the desired page.
- **Step 6**
  - Clear the ACCERR and PVIOL bits in all flash blocks.
Secure the Flash Memory  (1 of 2)

- The flash memory contents are secured by programming the security bits within the flash options/security byte at the address $FF0F.
- On devices that have a memory page window, the flash options/security byte is also available at the address $BF0F by setting the value of the PPAGE register to $3F.
- The contents of the $FF0F (or $BF0F) are copied into the flash security register (FSEC) during the reset sequence.
- The flash sector $FE00 to $FFFF must be erased before the flash options/security byte is programmed.
- Secured operation takes effect on the next reset after programming the security bits of the FSEC register to a secure value.
- The effects that the secured operation has on the HCS12 microcontroller are listed in Table 14.6.
### Secure the Flash Memory (2 of 2)

Table 14.6 Effects of secured operations on the HCS12 operation modes

<table>
<thead>
<tr>
<th>Operation mode</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal single chip mode</td>
<td>1. Background debug module operation is completely disabled.</td>
</tr>
<tr>
<td></td>
<td>2. Flash and EEPROM commands PROG, bulk erase, sector erase,</td>
</tr>
<tr>
<td></td>
<td>erase and verify, and sector modify remain enabled.</td>
</tr>
<tr>
<td>Special single chip mode</td>
<td>1. BDM firmware commands are disabled.</td>
</tr>
<tr>
<td></td>
<td>2. BDM hardware commands are restricted to register space.</td>
</tr>
<tr>
<td></td>
<td>3. Flash and EEPROM commands are limited to <strong>bulk erase</strong> only.</td>
</tr>
<tr>
<td>Expanded modes</td>
<td>1. BDM operation is completely disabled.</td>
</tr>
<tr>
<td></td>
<td>2. External access to internal flash and EEPROM is disabled.</td>
</tr>
<tr>
<td></td>
<td>3. Internal visibility (IVIS) and CPU pipe (IPIPE) information is</td>
</tr>
<tr>
<td></td>
<td>disabled.</td>
</tr>
<tr>
<td></td>
<td>4. Flash and EEPROM commands cannot be executed from external</td>
</tr>
<tr>
<td></td>
<td>memory in normal expanded mode.</td>
</tr>
</tbody>
</table>
Unsecuring the MCU with Backdoor Key Access

• In single chip and expanded modes, security can be temporarily disabled by means of the backdoor key access method.

• Backdoor key access requires three actions to be taken:
  – Program the backdoor key at $FF00−$FF07 to a valid value.
  – Set the KEYEN1 and KEYEN0 bits of the flash option/security byte to “10”.
  – In single chip mode, design applications to have the capability to write to the backdoor key locations.

• Backdoor key is not allowed to have the value of $0000$ or $FFFF$.

• The backdoor key access sequence includes:
  – Set the KEYACC bit in the flash configuration register (FCNFG).
  – Write the first 16-bit word of the backdoor key to $FF00$.
  – Write the second 16-bit word of the backdoor key to $FF02$.
  – Write the third 16-bit word of the backdoor key to $FF04$.
  – Write the fourth 16-bit word of the backdoor key to $FF06$.
  – Clear the KEYACC bit in the flash configuration register FCNFG.

• If all four 16-bit words match the flash contents at $FF00$ to $FF07$, the MCU will be forced to unsecured state “10”.

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Reprogramming the Security Bits

• This method can be used only if the options/security bit is unprotected.

• In normal single chip mode, security can also be disabled by means of erasing and reprogramming the security bits within the flash option/security byte to the unsecured value.

• The erase operation will erase the entire sector from $FE00 to $FFFF, the backdoor key and interrupt vectors will also be erased.
Complete Memory Erase

- The microcontroller can be unsecured in special single chip modes by erasing the entire EEPROM and flash memory. This can be done by a BDM kit.
Configuring the FCLKDIV Register

- The FCLKDIV register controls the timing of programming and erasing of flash memory.
- It is necessary to divide the oscillator frequency down to within the 150 KHz and 200 KHz range.
- The algorithm for determining the flash clock dividing factor is illustrated in Figure 14.16.
- The following notations are used in Figure 14.16:
  - FCLK refers to the flash timing clock
  - TBUS refers to the period of the bus clock.
  - INT(x) takes integer part of x.
Flash Clock Dividing Factor Algorithm

Figure 14.16 PRDIV8 and FDIV bits determination procedure
Flash Memory Programming and Erasing Algorithms

• One needs to verify that there is no pending access error or protection violation in any flash blocks. This initial set up include the following three steps:
  – Verify that all ACCERR and PVIOL flags in the FSTAT register are cleared in all banks.
  – Write to bits BKSEL in the FCNFG register to select the bank of registers corresponding to the flash block to be programmed or erased.
  – Write to the PPAGE register to select one of the pages to be programmed if programming is to be done in the $8000~$BFFFF address space.
Example 14.5 ACCERR and PVIOL

- Write a function to clear the ACCERR and PVIOL flags in all four blocks in the HCS12 devices with 256 KB of on-chip flash memory

```c
void clearflags (void)
{
    FCNFG &= ~0x03;       /* select bank 0 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
    FCNFG |= 0x01;        /* select bank 2 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
    FCNFG &= 0xFE;        /* select bank 0 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
    FCNFG |= 0x01;        /* select bank 1 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
    FCNFG &= 0xFE;        /* select bank 3 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
    FCNFG |= 0x01;        /* select bank 2 */
    FSTAT = ACCERR+PVIOL; /* clear the ACCERR and PVIOL flags */
}
```
Three-Step Procedure

• Step 1
  – Write the aligned data word to be programmed to the valid flash address space.

• Step 2
  – Write the program or erase command to the command buffer.

• Step 3
  – Clear the CBEIF flag by writing a “1” to it to launch the command. The clearing of the CBEIF flag indicates that the command was successfully launched.
C Function for Erasing a Flash Sector

```c
int EraseFSector(int *pt)
{
    if (!(FSTAT & CBEIF))
        return 1; /* command buffer not empty, erase prohibited */
    *pt = 0x00; /* write any data to the sector */
    FCMD = SectorErase; /* write sector erase command */
    FSTAT = CBEIF; /* launch the erase command */
    if (FSTAT & (ACCERR+PVIOL))
        return 1; /* return error code 1 */
    while(!(FSTAT & CCIF)); /* wait until erase command is completed */
    return 0; /* return normal code */
}
```

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
C Function to Perform Flash Bulk Erase

```c
int bulkeraseF(int *ptr)
{
    if(FPROT&(FPOPEN | FPHDIS | FPLDIS)!= 0xA4)
        return 1;    /* can't bulk erase */
    if(!(FSTAT & CBEIF))
        return 1;    /* command buffer isn't empty, bulk erase not allowed */
    else {
        *ptr = 0x00; /* write anything to flash block location */
        FCMD = BulkErase; /* write bulk erase command */
        FSTAT = CBEIF; /* launch bulk erase command */
        if (FSTAT & (ACCERR | PVIOL))
            return 1;      /* error flag is set, command failed */
        while(!(FSTAT & CCIF)); /* wait until command completion */
        return 0;
    }
}
```

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
C Function that Programs a Block of Words to Flash Memory

```c
int feProgBlok(char cnt, int *destptr, int *srcptr)
{
    if(cnt == 0) return 0;    /* if word count is 0, do nothing */
    while(cnt){
        if(FSTAT & CBEIF){ /* if command buffer is not empty, do nothing */
            *destptr++ = *srcptr++; /*write data word to flash location*/
            FCMD = Program;         /* write program command */
            FSTAT = CBEIF;           /* launch program command */
            if(FSTAT & (ACCERR+PVIOL))
                return 1;           /* program error? */
            cnt--;
        }
    }
    while(!(FSTAT&CCIF));       /* wait for the last command to complete*/
    return 0;
}
```

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
C Function that Performs Erase and Verify

```c
int feraseverify(int *ptr)
{
    if(!(FSTAT & CBEIF))
        return 1;    /* command buffer not empty, returns */
    *ptr = 0x00;      /* write data to flash sector address */
    FCMD = EraseVerify;   /* write erase verify command */
    FSTAT = CBEIF;       /* launch the command */
    if(FSTAT&(ACCERR | PVIOL))
        return 1;       /* errors have occurred */
    while(!(FSTAT & CCIF));   /* wait until command is completed */
    if(FSTAT & BLANK)
        return 0;       /* command completed successfully */
    else
        return 1;
}
```

On-Chip EEPROM

- An HCS12 device may have 1 KB, 2 KB, or **4 KB** of on-chip EEPROM.
  - The MC9S12DP256/512 has 4 KB of EEPROM

- The EEPROM is organized as an array of two-byte words.
  - The erase sector size is two rows or two words.

- The whole EEPROM can be protected by setting the EPOOPEN bit of the EPROT register.
  - The protected EEPROM block can be sized from 64 to 512 bytes.

- A 16-byte field is reserved inside the EEPROM module from the address $_{FF0}$ to $_{FFF}$.
EEPROM BD and MAP

Figure 1-1 Module Block Diagram

Figure 3-1 EEPROM Memory Map

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning
## EEPROM Registers

### Table 3-2  EEPROM Register Memory Map

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Use</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0x00</td>
<td>EEPROM Clock Divider Register (ECLKDIV)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x01</td>
<td>RESERVED1&lt;sup&gt;1&lt;/sup&gt;</td>
<td>R</td>
</tr>
<tr>
<td>$0x02</td>
<td>RESERVED2&lt;sup&gt;1&lt;/sup&gt;</td>
<td>R</td>
</tr>
<tr>
<td>$0x03</td>
<td>EEPROM Configuration Register (ECNFG)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x04</td>
<td>EEPROM Protection Register (EPROT)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x05</td>
<td>EEPROM Status Register (ESTAT)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x06</td>
<td>EEPROM Command Register (ECMD)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x07</td>
<td>RESERVED3&lt;sup&gt;1&lt;/sup&gt;</td>
<td>R</td>
</tr>
<tr>
<td>$0x08</td>
<td>EEPROM High Address Register (EADDRHI)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x09</td>
<td>EEPROM Low Address Register (EADDRLO)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x0A</td>
<td>EEPROM High Data Register (EDATAHI)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0x0B</td>
<td>EEPROM Low Data Register (EDATALO)</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**NOTES:**
1. Intended for factory test purposes only.

### 0x0110-0x011B
EEPROM Clock Divide Register (ECLKDIV)

- This register controls the timing of EEPROM programming and erasing.
- Bits 0-6 can be written once after reset.

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIVLD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PRDIV8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

EDIVLD: Clock divider loaded
0 = Register has not been written.
1 = Register has been written to since the last reset.

PRDIV8: Enable prescaler by 8
0 = The input oscillator clock is directly fed into the FCLKDIV divider.
1 = Enables a divide-by-8 prescaler, to divide the flash module input oscillator clock before feeding into the CLKDIV divider

EDIV[5:0]: Clock divider bits
The combination of PRDIV8 and FDIV[5:0] effectively divides the flash module input oscillator clock down to a frequency of 150KHz~200KHz. The maximum divide ratio is 512.

Figure 14.19 EEPROM clock divider register (ECLKDIV)

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
EEPROM Configuration Register (ECNFG)

CBEIE: Command buffer empty interrupt enable
0 = Command buffer empty interrupts disabled
1 = An interrupt will be requested whenever the CBEIF flag is set.

CCIE: Command completion interrupt enable
0 = Command complete interrupts disabled
1 = An interrupt will be requested whenever the CCIF flag is set.

Figure 14.20 EEPROM configuration register (ECNFG)
EEPROM Protection Register (EPROT) (1 of 2)

- This register defines which EEPROM sectors are protected against program or erase.
- This register is loaded from EEPROM at $\_FFD in the reset sequence.
- The EP[2:0] bits can be written any time until the EPDIS bit is cleared.
- The contents of this register are shown in Figure 14.21.
## EEPROM Protection Register (EPROT) (2 of 2)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOPEN</td>
<td>NV6</td>
<td>NV5</td>
<td>NV4</td>
<td>EPDIS</td>
<td>EP2</td>
<td>EP1</td>
<td>EP0</td>
</tr>
</tbody>
</table>

**Reset:**
- **EOPEN**: F
- NV6: F
- NV5: F
- NV4: F
- EPDIS: F
- EP2: F
- EP1: F
- EP0: F

**EOPEN:** Opens the EEPROM for programming or erasure
- 0 = The whole EEPROM array is protected. In this case, EPDIS and EP bits within the protection register have no effect.
- 1 = The EEPROM sectors not protected are enabled for program or erase.

**NV6~NV4:** Not volatile flag bit. These bits are available for nonvolatile flag usage

**EPDIS:** EEPROM protection address range disable
- 0 = protection enabled
- 1 = protection disabled

**EP2~EP0:** EEPROM protection address size
- 000 = 64 bytes ($\_FC0~\_FFF$)
- 001 = 128 bytes ($\_F80~\_FFF$)
- 010 = 192 bytes ($\_F40~\_FFF$)
- 011 = 256 bytes ($\_F00~\_FFF$)
- 100 = 320 bytes ($\_EC0~\_FFF$)
- 101 = 384 bytes ($\_E80~\_FFF$)
- 110 = 448 bytes ($\_E40~\_FFF$)
- 111 = 512 bytes ($\_E00~\_FFF$)

**Figure 14.21** EPROT register

---

EEPROM Status Register (ESTAT)

- The bits CBEIF, PVIOL, and ACCERR are readable and writable, bits CCIF and BLANK are readable but not writable.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>CBEIF: Command buffer empty interrupt flag</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>CBEIF: Command buffer empty interrupt flag</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CCIF: Command completion interrupt flag</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ACCERR: Flash access error</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>BLANK: Array has been verified as erased</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>PVIOL: Protection violation</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>ACCERR: Flash access error</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>BLANK: Array has been verified as erased</td>
<td>0</td>
</tr>
</tbody>
</table>

Reset: 1 1 0 0 0 0 0 0

CBEIF: Command buffer empty interrupt flag
0 = Command buffers are full
1 = Command buffers are ready to accept a new command

CCIF: Command completion interrupt flag
0 = Command in progress
1 = All commands are completed

PVIOL: Protection violation
0 = No failure
1 = A protection violation has occurred

ACCERR: Flash access error
0 = No failure
1 = Access error has occurred.

BLANK: Array has been verified as erased
0 = If an erase and verify command has been requested, and the CCIF flag is set, then a zero in BLANK indicates that the block is not erased.
1 = Flash block verifies as erased.

Figure 14.22 EEPROM status register (ESTAT)
CMDB6, CMDB5, CMDB2, and CMDB0: Command bits
Valid commands includes the following:
$05 = \text{Erase and verify}
$20 = \text{Program a word (two bytes)}
$40 = \text{Sector erase}
$41 = \text{Bulk erase}
$60 = \text{Sector modify}

Figure 14.23 EEPROM command buffer and register (ECMD)
Configuring the ECLKDIV Register

- The configuration method of this register is identical to that of the FCLKDIV register.

![Diagram](Image)

Figure 14.16 PRDIV8 and FDIV bits determination procedure
Programming and Erasure of EEPROM

- The programming algorithm for EEPROM is almost identical to that of the flash memory.
int eraseEEsector (int *ptr)
{
    ESTAT = ACCERR | PVIOL; /* clear error flags */
    if(!(ESTAT & CBEIF))
        return 1; /* command buffer not empty, can't issue new command*/
    *ptr = 0x00;   /* write any data to EEPROM sector location */
    ECMD = SectorErase; /* write sector erase command */
    ESTAT = CBEIF;      /* launch the command */
    if(ESTAT & (ACCERR | PVIOL))
        return 1;        /* error occurred */
    while(!(ESTAT&CCIF)); /* wait for command completion */
    return 0;             /* command completed correctly */
}
C Function to Bulk Erase an EEPROM

```c
int eraseEEsector (int *ptr)
{
    ESTAT = ACCERR | PVIOL; /* clear error flags */
    if(!(ESTAT & CBEIF))
        return 1; /* command buffer not empty, can't issue new command*/
    *ptr = 0x00; /* write any data to EEPROM sector location */
    ECMD = BulkErase; /* write bulk erase command */
    ESTAT = CBEIF; /* launch the command */
    if(ESTAT & (ACCERR | PVIOL))
        return 1; /* error occurred */
    while(!(ESTAT&CCIF)); /* wait for command completion */
    return 0; /* command completed correctly */
}
```

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
int eeprogram (unsigned int data, unsigned int *ptr)
{
    ESTAT = ACCERR+PVIOL;   /* clear error flag */
    if(!(ESTAT&CBEIF))
        return 1;
    *ptr = data;      /* write data to word-aligned address */
    ECMD = Program;   /* write program command */
    ESTAT = CBEIF;    /* launch command */
    if(ESTAT &(ACCERR|PVIOL))
        return 1;
    while(!(ESTAT&CCIF)); /* wait for command to complete */
    return 0;
}
Example 14.14

• Write a function that executes the sector-modify command to change the contents of an EEPROM sector. The index register X holds the word-aligned address of the first EEPROM word to be modified. The index register Y holds the word-aligned address of the first word of data to be programmed.

• Solution:

```assembly
  eesectormodify
    movb    #ACCERR+PVIOL,ESTAT    ; clear error flags
    brclr   ESTAT,CBEIF,cantmod    ; command buffer not empty
    movw    0,Y,0,X                 ; write data to EEPROM address
    movb    #SectorModify,ECMD     ; write sector modify command
    movb    #CBEIF,ESTAT           ; launch the erase and check command
    brclr   ESTAT,ACCERR+PVIOL,EEModOK
    cantmod ldab #1                 ; return error code 1
    rts

  EEModOK
    brclr   ESTAT,CBEIF,EEModOK    ; wait for command buffer to empty
    movw    2,Y,2,X                 ; write second data word to EEPROM
    movb    #Program,ECMD          ; write program command
    movb    #CBEIF,ESTAT           ; launch the program command
    brclr   ESTAT,ACCERR+PVIOL,EEPR2OK
    ldab #1
    rts
```

int EESectorModify(unsigned int *src, unsigned int *dest) 
{
    ESTAT = ACCERR | PVIOL; /* clear error flags */
    if(!(ESTAT&CBEIF))
        return 1; /* command buffer not empty is error */
    *dest = *src; /* write first data word */
    ECMD = SectorModify; /* write sector modify command */
    ESTAT = CBEIF; /* launch the sector modify command */
    if(ESTAT&(ACCERR | PVIOL))
        return 1; /* command failed */
    while(!(ESTAT&CCIF)); /* wait for command buffer becomes empty */
    *(dest+1) = *(src+1); /* write second data word */
    ECMD = Program; /* write the program command */
    ESTAT = CBEIF; /* launch the program command */
    if(ESTAT&(ACCERR | PVIOL))
        return 1; /* command failed */
    while(!(ESTAT&CCIF)); /* wait for command buffer becomes empty */
    return 0;
}
Multiplexed External Bus Interface

Figure 1-1 MEBI Block Diagram
HCS12 External Interface (1 of 3)

- External memory interface is implemented across four I/O ports (A, B, E, and K).
  - External memory signal pins are listed in Table 14.7.
- Signals ADDR0…ADDR15 and XADDR14…XADDR19 are used to select a memory location to access and are referred to as the address bus.
- DATA0…DATA15 are used to carry data and hence are referred to as the data bus.
- When the external memory is not paged, only the lower 16 address/data pins are used.
- If expanded memory is enabled, then ADDR14 and ADDR15 are not used.
  - A0…A15 are used instead of ADDR0…ADDR15.
  - D0…D15 are used instead of DATA0…DATA15.
  - XA14…XA19 are used instead of XADDR14…XADDR19.
- R/W signal is used to indicate the direction of data transfer.
- LSTRB signal is used to indicate whether the lower data bus (D7…D0) carries valid data.
- In expanded narrow mode, external memory data pins are connected to D15…D8.
- LSTRB, R/W, and A0 interface the type of bus access that is taking place.
## HCS12 External Interface (2 of 3)

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR0/DATA0</td>
<td>EMI address bit 0 or data bit 0</td>
</tr>
<tr>
<td>ADDR1/DATA1</td>
<td>EMI address bit 1 or data bit 1</td>
</tr>
<tr>
<td>ADDR2/DATA2</td>
<td>EMI address bit 2 or data bit 2</td>
</tr>
<tr>
<td>ADDT3/DATA3</td>
<td>EMI address bit 3 or data bit 3</td>
</tr>
<tr>
<td>ADDT4/DATA4</td>
<td>EMI address bit 4 or data bit 4</td>
</tr>
<tr>
<td>ADDT5/DATA5</td>
<td>EMI address bit 5 or data bit 5</td>
</tr>
<tr>
<td>ADDT6/DATA6</td>
<td>EMI address bit 6 or data bit 6</td>
</tr>
<tr>
<td>ADDT7/DATA7</td>
<td>EMI address bit 7 or data bit 7</td>
</tr>
<tr>
<td>ADDR8/DATA8</td>
<td>EMI address bit 8 or data bit 8</td>
</tr>
<tr>
<td>ADDR9/DATA9</td>
<td>EMI address bit 9 or data bit 9</td>
</tr>
<tr>
<td>ADDR10/DATA10</td>
<td>EMI address bit 10 or data bit 10</td>
</tr>
<tr>
<td>ADDT11/DATA11</td>
<td>EMI address bit 11 or data bit 11</td>
</tr>
<tr>
<td>ADDT12/DATA12</td>
<td>EMI address bit 12 or data bit 12</td>
</tr>
<tr>
<td>ADDT13/DATA13</td>
<td>EMI address bit 13 or data bit 13</td>
</tr>
<tr>
<td>ADDT14/DATA14</td>
<td>EMI address bit 14 or data bit 14</td>
</tr>
<tr>
<td>ADDT15/DATA15</td>
<td>EMI address bit 15 or data bit 15</td>
</tr>
<tr>
<td>XADDR14</td>
<td>EMI extended address bit 14</td>
</tr>
<tr>
<td>XADDR15</td>
<td>EMI extended address bit 15</td>
</tr>
<tr>
<td>XADDR16</td>
<td>EMI extended address bit 16</td>
</tr>
<tr>
<td>XADDR17</td>
<td>EMI extended address bit 17</td>
</tr>
<tr>
<td>XADDR18</td>
<td>EMI extended address bit 18</td>
</tr>
<tr>
<td>XADDR19</td>
<td>EMI extended address bit 19</td>
</tr>
<tr>
<td>R/W</td>
<td>Read/write</td>
</tr>
<tr>
<td>LSTRB</td>
<td>lower byte strobe</td>
</tr>
<tr>
<td>ECLK</td>
<td>E clock</td>
</tr>
<tr>
<td>ECS/ROMONE</td>
<td>Emulated chip select/on-chip ROM enable</td>
</tr>
<tr>
<td>XCS</td>
<td>External data chip select</td>
</tr>
</tbody>
</table>

Note. EMI stands for external memory interface

---

Table 14.7 HCS12 external memory interface signal pins

HCS12 External Interface (3 of 3)

- E clock (ECLK) is used as the timing reference for external access.
  - Aligned 16-bit access takes one E clock cycle to complete.
  - Misaligned external 16-bit access takes two E clock cycles to complete.
  - Misaligned internal 16-bit access takes one E clock cycle to complete.

- The ECS signal is used as the chip select signal for external memory chips. This signal can be used to latch address signals so that they stay valid throughout the whole access cycle.

<table>
<thead>
<tr>
<th>LSTRB</th>
<th>A0</th>
<th>R/W</th>
<th>Type of access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8-bit read of an even address</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-bit read of an odd address</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8-bit write of an even address</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8-bit write of an odd address</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>16-bit read of an even address</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16-bit read of an odd address</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>16-bit write of an even address (low/high bytes swapped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16-bit write of an odd address (low/high bytes swapped)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>16-bit write of an odd address (low/high bytes swapped)</td>
</tr>
</tbody>
</table>

Table 14.8 Access type vs. bus control signals

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Waveforms of Bus Signals (1 of 3)

- The waveform of a typical digital signal is shown.

![Figure 14.25 A typical digital waveform](image)

- The rise time ($t_r$) of a signal is the time needed for a signal to rise from 10% of the power supply voltage to 90% of the power supply voltage.

- The fall time ($t_f$) of a signal is the time needed for a signal to drop from 90% of the power supply voltage to 10% of the power supply voltage.
Waveforms of Bus Signals (2 of 3)

- **Single bus signal waveform**
  
  ![Figure 14.26 Single signal waveform](Image)

- **Multiple-signal waveform**
  
  ![Figure 14.27 Multiple-signal waveform](Image)
Waveforms of Bus Signals (3 of 3)

- Unknown signal

- Floating signals

Figure 14.28 Unknown signals. (a) single signal. (b) multiple signal.

Figure 14.29 Floating signals. (a) Single signal. (b) Multiple signals.

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.
Categories of Bus Signals

- Address
- Data
- Control
Bus Transactions

- Includes sending the address and receiving or sending the data
  - A read transaction (also called read bus cycle) transfers data from memory to either the CPU or the I/O device.
  - A write transaction (also called write bus cycle) writes data to the memory.

- In a read transaction, the address is first sent down the bus to the memory, together with the appropriate control signals indicating a read.
- The memory responds by placing the data on the bus and asserting a control signal to indicate that the data is valid.
- A read bus transaction is shown in Figure 14.30.
- In Figure 14.30, a read cycle takes one clock cycle to complete. If necessary, the memory system can delay the assertion of the Ready signal.
Read Transaction

Figure 14.30 A typical bus read transaction

Terminology

• Bus master
  – A device that can generate address and control signals during a bus transaction
• Bus slave
  – A device such as a memory chip that can only respond to the bus transaction
• Synchronous bus
  – A bus that uses a clock signal to synchronize bus transactions
• Asynchronous bus
  – A bus that does not use a clock signal to synchronize the bus transaction

Figure 14.31 Asynchronous read bus transaction.
Bus Multiplexing

- It is used to minimize the number of pins used in a chip.
- Address bus and data bus are often multiplexed.
- Address signals must be stable throughout the whole bus transaction cycle.
- In a multiplexed bus system, external circuit must make a copy of the address signals to make them stable throughout the whole bus cycle.
The HCS12 Bus Cycles

• The timing diagrams of a read and a write bus cycle are combined in Figure 14.32.
• Figure 14.32 shows that a bus cycle starts with the MCU driving the address signals onto the address bus.
• The events that occur during a read cycle are illustrated in Figure 14.33.
• The events that occur during a write a cycle are illustrate in Figure 14.34.
Read/Write Bus Cycle

Numbers refer to timing specifications

Figure 14.32 The HCS12 Read/Write bus cycle timing diagram
### Timing Specifications

Table 14.9 HCS12 expanded bus timing characteristics

<table>
<thead>
<tr>
<th>Num</th>
<th>Parameter name</th>
<th>Symbol</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frequency of operation (E-clock)</td>
<td>( f_o )</td>
<td>0</td>
<td>-</td>
<td>25.0</td>
<td>ns</td>
</tr>
<tr>
<td>2</td>
<td>Cycle time</td>
<td>( t_{cc} )</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>3</td>
<td>Pulse width, E low</td>
<td>( PW_{EL} )</td>
<td>19</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>4</td>
<td>Pulse width, E high</td>
<td>( PW_{EH} )</td>
<td>19</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>5</td>
<td>Address delay time</td>
<td>( t_{AD} )</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>ns</td>
</tr>
<tr>
<td>6</td>
<td>Address valid time to E rise (( PW_{EL} - t_{AD} ))</td>
<td>( t_{AV} )</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>7</td>
<td>Muxed address hold time</td>
<td>( t_{MACH} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>8</td>
<td>Address hold to data valid</td>
<td>( t_{AHDS} )</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>9</td>
<td>Data hold to address</td>
<td>( t_{DHA} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>10</td>
<td>Read data setup time</td>
<td>( t_{DSR} )</td>
<td>13</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>11</td>
<td>Read data hold time</td>
<td>( t_{DHR} )</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>12</td>
<td>Write data delay time</td>
<td>( t_{DDW} )</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>13</td>
<td>Write data hold time</td>
<td>( t_{DHW} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>14</td>
<td>Write data setup time(^1) (( PW_{EH} - t_{DDW} ))</td>
<td>( t_{DSW} )</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>15</td>
<td>Address access time(^1) (( t_{cc} - t_{AD} - t_{DSR} ))</td>
<td>( t_{ACC} )</td>
<td>19</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>16</td>
<td>E high access time(^1) (( PW_{EH} - t_{DSR} ))</td>
<td>( t_{ACCE} )</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>17</td>
<td>Non-multiplexed address delay time</td>
<td>( t_{NAD} )</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>ns</td>
</tr>
<tr>
<td>18</td>
<td>Non-muxed address valid to E rise (( PW_{EL} - t_{NAD} ))</td>
<td>( t_{NAV} )</td>
<td>15</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>19</td>
<td>Non-multiplexed address hold time</td>
<td>( t_{NAM} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>20</td>
<td>Chip select delay time</td>
<td>( t_{CSD} )</td>
<td>-</td>
<td>-</td>
<td>16</td>
<td>ns</td>
</tr>
<tr>
<td>21</td>
<td>Chip select access time(^1) (( t_{cc} - t_{CSD} - t_{DSR} ))</td>
<td>( t_{ACCS} )</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>22</td>
<td>Chip select hold time</td>
<td>( t_{CSH} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>23</td>
<td>Chip select negated time</td>
<td>( t_{CSN} )</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>24</td>
<td>Read/write delay time</td>
<td>( t_{RWD} )</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>25</td>
<td>Read/write valid time to E rise (( PW_{EL} - t_{RWD} ))</td>
<td>( t_{RWW} )</td>
<td>14</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>26</td>
<td>Read/write hold time</td>
<td>( t_{RWH} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>27</td>
<td>Low strobe delay time</td>
<td>( t_{LSD} )</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>28</td>
<td>Low strobe valid time to E rise (( PW_{EL} - t_{LSD} ))</td>
<td>( t_{LSV} )</td>
<td>14</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>29</td>
<td>Low strobe hold time</td>
<td>( t_{LSH} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>30</td>
<td>NOACC strobe delay time</td>
<td>( t_{NOD} )</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>31</td>
<td>NOACC valid time to E rise (( PW_{EL} - t_{NOD} ))</td>
<td>( t_{NGV} )</td>
<td>14</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>32</td>
<td>NOACC hold time</td>
<td>( t_{NGH} )</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>33</td>
<td>IPIPO[1:0] delay time</td>
<td>( t_{P0D} )</td>
<td>2</td>
<td>-</td>
<td>7</td>
<td>ns</td>
</tr>
<tr>
<td>34</td>
<td>IPIPO[1:0] valid time to E rise (( PW_{EL} - t_{P0D} ))</td>
<td>( t_{P0V} )</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>35</td>
<td>IPIPO[1:0] delay time(^1) (( PW_{EL} - t_{P1V} ))</td>
<td>( t_{P1D} )</td>
<td>2</td>
<td>-</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>36</td>
<td>IPIPO[1:0] valid time to E fall</td>
<td>( t_{P1V} )</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
</tbody>
</table>

Notes. 1. Affected by clock stretch: add \( N \times tcyc \) where \( N = 0, 2, \) or 3, depending on the number of clock stretches.
Read Bus Cycle Sequence

Microcontroller

1. Set R/W to read
2. Place address on bus
3. Assert LSTRB if address is odd
4. Assert ECS and XCS if in range

External memory

Output the data
1. Decode the address
2. Place data on data bus

Acquire the data
1. Latch the data
2. Negate LSTRB if asserted
3. Negate ECS and XCS if asserted

Terminate the read cycle
1. Remove data from data bus and release to HI-Z

Start the next cycle

Figure 14.33 Sequence of events occurred in an HCS12 read bus cycle
Write Bus Cycle Sequence

Figure 14.34 Sequence of events occurred in an HCS12 write bus cycle

Microcontroller

Address the memory
1. Set R/W to write
2. Place address on bus
3. Assert LSTRB if address is odd
4. Assert ECS and XCS if in range
5. Place data on data bus

External memory

Input the data
1. Decode the address
2. Store data from data bus

Terminate write data
1. Negate LSTRB if asserted
2. Negate ECS and XCS if asserted
3. Remove data from data bus

Terminate the write cycle
1. Release data to HI-Z

Start the next cycle
Important Data Read Timing Parameters

- Address set up time: time interval that AD<19:0> valid before the falling edge of ECS. This parameter is $t_{20} - t_5 \geq 8 \text{ ns}$ ($= 16 \text{ ns} - 8 \text{ ns}$).
- Address hold time: amount of time that AD<15:0> remain valid after the falling edge of ECS. This parameter is $t_7 + TE/2 - t_{20} (\geq 5 \text{ ns})$.
- Read data setup time: amount of time that data is valid before the falling edge of ECLK. This parameter is $t_{10} (\geq 13 \text{ ns})$.
- Read data hold time: amount of time that data remains valid after the falling edge of ECLK. This parameter is $t_{11} (\geq 0 \text{ ns})$.
- Address access time: the delay from the moment that address signals are valid until data is driven by memory device. This parameter is $t_{15} (> 19 \text{ ns})$.
- Write data set up time: Write data valid before the W signal goes high. This parameter is $t_{14} + t_{26} \geq 14 \text{ ns}$.
- Write data hold time: Write data stay valid after the W signal goes high. This parameter is $\geq 0 \text{ ns}$.
  - Parameters 1 and 2 are important to make sure that the address signals can be latched correctly by the address latch.
  - Parameters 3, 4, and 5 are required by the MCU during a read access.
  - The actual values for parameters 6 and 7 may be lengthened depending on how the Write signal is derived for the memory chip.
Issues Related to Adding External Memory

- Memory space assignment
- Address decoder and control circuitry design
- Timing verification
Memory Space Assignment

• Equal size assignment
  – The available memory space is divided into blocks of equal size.
  – Each block is assigned to a memory device without regarding for the actual size of each memory-mapped device.
  – Memory space tends to be wasted using this approach because most memory-mapped peripheral chips need only a few bytes to be assigned to their internal registers.

• Demand assignment
  – The user assigns the memory space according to the size of memory devices.
Example 14.2 Design an Embedded Product

- Assigned to design a HCS12DP256-based embedded product that requires 256 KB of external 16-bit SRAM, 256KB of 16-bit EEPROM, and a parallel peripheral interface (PPI) that requires only four bytes of address space.
- The only available SRAM to this designer is the 128K × 8 SRAM chips (This chip has 128K locations with each location containing 8 bits). The only available EEPROM is the 128K × 8 EEPROM chips. Suggest a workable memory space assignment.

- Solution:
  - Two 8-bit wide memory chips are needed to construct a 16-bit memory module.
  - Two 16-bit wide SRAM modules are needed to provide the 512 KB capacity.
  - One 16-bit wide EEPROM module is needed to offer the 256KB capacity.

A possible assignment is as follows:

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Address Range</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>0x00000~0x3FFFF</td>
<td>256KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>0x40000~0x7FFFF</td>
<td>256KB</td>
</tr>
<tr>
<td>PPI</td>
<td>0xBFFFC~0xBFFFF</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

Address Decoder Design

• Address decoder selects one and only one memory device to respond to the memory access request.
• All memory devices have control signals such as chip-enable (CE), chip-select (CS), or output-enable (OE) to control their read and write operations.
• Two address-decoding schemes have been used:
  – Full decoding
  – Partial decoding
• Address decoder design is closely related to memory space assignment.
• The programmable logic devices (PLDs) make full decoding easy and flexible.
• GALs and PALs are often used to implement address decoders.
Timing Verification

• In a read cycle, the most critical timing requirements are the data setup time and data hold time required by the microcontroller.

• The address setup time and hold time requirements for the memory devices are also critical.

• In a write cycle, the most critical timing requirements are the write data setup time and write data hold time required by the memory device.

• The address setup time and hold time must also be satisfied.

• For a write cycle, the write pulse width must be long enough for the write cycle to be successful.
HCS12 External Memory Circuit

Figure 14.46 HCS12DP256B paged external memory example

Material from or based on: The HCS12/9S12: An Introduction to Software & Hardware Interfacing, Thomson Delmar Learning, 2006.