## Embedded System Design Memory Map Details
### 24KB EPROM, 32KB SRAM, 8KB Peripherals

### Basic Memory Map

<table>
<thead>
<tr>
<th>Address in Binary</th>
<th>Hex</th>
<th>Address Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111xxxxxxxxxxxxx</td>
<td>FFFFh</td>
<td>Peripherals (8KB)</td>
</tr>
<tr>
<td>1110xxxxxxxxxxxxx</td>
<td>E000h</td>
<td>(8192 bytes)</td>
</tr>
<tr>
<td>1101xxxxxxxxxxxxx</td>
<td>DFFFh</td>
<td></td>
</tr>
<tr>
<td>1100xxxxxxxxxxxxx</td>
<td></td>
<td>SRAM (32KB)</td>
</tr>
<tr>
<td>1011xxxxxxxxxxxxx</td>
<td></td>
<td>(32768 bytes)</td>
</tr>
<tr>
<td>1010xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110xxxxxxxxxxxxx</td>
<td>6000h</td>
<td></td>
</tr>
<tr>
<td>0101xxxxxxxxxxxxx</td>
<td>5FFFh</td>
<td></td>
</tr>
<tr>
<td>0100xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001xxxxxxxxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000xxxxxxxxxxxxx</td>
<td>0000h</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The upper address bits are used to decode the address space and to generate chip select signals for the various chips. The lower address lines in that address space are used to access each memory element within the range.

**Note:** Although the EPROM contains 32KB of physical memory, only 24KB will actually be accessible when using this memory map.
Memory Map When MON51 Is Used

- **$FFFF**: Peripherals (8192 bytes)
- **$E000 - $DFFF**: MON51 Variables (80 bytes)
- **$DFA0 - $DFAF**: Available for User Code and Variables (32688 bytes)
- **$6000 - $5FFF**: Available for User Code and Constants if MON51 EPROM is Reprogrammed (20779 bytes)
- **$0ED6 - $0ED5**: MON51 Code (3797 bytes)
- **$0000**: Reset Vector is fixed at $0000. All Interrupt Vectors except Timer 1 are re-vectored to the beginning of SRAM.

**Note:** MON51 requires 80 bytes of external SRAM storage in order to support monitor features such as breakpoints. If you have your own copy of MON51 and are programming your own EPROM, you should adjust the memory map so that the MON51 variables are located at the highest addresses in your SRAM. This will provide more contiguous SRAM for your program variables.
Memory Map Using MICRO-C (Compact Model) Without Monitor (MON51/PAULMON)

Equates at the beginning of C:\MC\LIB51\8051RLPC.ASM:

```assembler
?RAM EQU $6000          External DATA (RAM) Starts here
?RAMEND EQU $DFFF       External DATA (RAM) Ends here
ORG $0000               CODE Starts here in ROM
```

Peripherals (8192 bytes)

Available for User Variables (32768 bytes)

External Stack. Used for local variables. Referenced by the `?stack MICRO-C` variable. Starts at top of SRAM and grows downward.


Uninitialized MICRO-C global variables.

Available for User Code and Constants (24576 bytes)

Note: With this memory map, user code would be stored in EPROM, starting at address $0000.

Note: The usage of memory space by the MICRO-C compiler depends on the MICRO-C memory model chosen and the addresses chosen in the corresponding memory model configuration file.
Memory Map Using MICRO-C (Compact Model) With MON51

Equates at the beginning of C:\MC\LIB51\8051RLPC.ASM:

\[\text{?RAM} \quad \text{EQU} \quad \$B000 \quad \text{External DATA (RAM) Starts here}\]
\[\text{?RAMEND} \quad \text{EQU} \quad \$DFAF \quad \text{External DATA (RAM) Ends here}\]
\[\text{ORG} \quad \$6000 \quad \text{CODE Starts here in RAM (normally in ROM)}\]

---

**Peripherals** (8192 bytes)

$FFFF

**MON51 Variables** (80 bytes)

$DFB0

Available for User Variables

(12208 bytes)

$DFAF

Available for User Code

(20480 bytes)

$B000

$AFFF

Available for User Code and Constants

if MON51 EPROM is Reprogrammed

(20779 bytes)

$6000

$5FFF

MON51 Code (3797 bytes)

$0000

$0ED5

$0ED6

Note: With the above memory map, if user code ever exceeds 20480 bytes, problems will result at run time, since temporary values stored in RAM will overwrite any code which is present in RAM addresses higher than $AFFF. Symptoms of this problem include erratic program behavior which disappears if you comment out a function or section of code and then run the recompiled code. If code size exceeds 20480 bytes, ?RAM will need to be redefined to some higher value, such as $C000. The choice of the value for ?RAM depends on your particular application and its requirements for code space versus data space. If external stack space usage ever exceeds 20480 bytes, then code could be potentially overwritten, and any heap values or uninitialized MICRO-C global variables will be overwritten. If the heap ever grows past $DFAF, the MON51 variables will be overwritten and MON51 will stop working correctly.

Note: This particular memory map is historical in nature. There exist many MON51 EPROMs which have been programmed with the memory bounds set at $DFB0 and $DFFF. If you have your own copy of MON51 and are programming your own EPROM, you can adjust the memory map as you like.
Memory Map Using MICRO-C (Compact Model) With PAULMON2

Equates at the beginning of C:\MC\LIB51\8051RLPC.ASM:

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$6000</td>
<td>CODE Starts here in RAM (normally in ROM)</td>
</tr>
<tr>
<td>$AFFF</td>
<td>External DATA (RAM) Starts here</td>
</tr>
<tr>
<td>$DFFF</td>
<td>External DATA (RAM) Ends here</td>
</tr>
</tbody>
</table>

**Peripherals (8192 bytes)**

**Available for User Variables (12288 bytes)**

**Available for User Code (20480 bytes)**

**Available for User Code and Constants or for PAULMON2 extensions if EPROM is Reprogrammed (~20635 bytes)**

**PAULMON2 Core Code (~3941 bytes)**

**Note:** With the above memory map, if user code ever exceeds 20480 bytes, problems will result at run time, since temporary values stored in RAM will overwrite any code which is present in RAM addresses higher than $AFFF. Symptoms of this problem include erratic program behavior which disappears if you comment out a function or section of code and then run the recompiled code. If code size exceeds 20480 bytes, ?RAM will need to be redefined to some higher value, such as $C000. The choice of the value for ?RAM depends on your particular application and its requirements for code space versus data space. If more space is required for user variables, ?RAM can be redefined to some lower value, such as $A000. You can examine the listing (.LST) file to see the addresses used by your program.