Accessing the functions provided in the MAXQ utility ROM

All MAXQ utility ROMs include routines for accessing data and tables stored in the program space.

Using lookup tables within the application code is a common programming practice when working with microcontrollers. Because of the single-cycle nature of the MAXQ core, application software cannot read directly from code space and, therefore, cannot directly access any tables defined within the application code. To alleviate this issue, all MAXQ utility ROMs include routines for accessing data and tables stored in the program space. In addition to these core functions, the ROM for each MAXQ variation may have routines specific to that part. Because these functions might be located anywhere within the ROM, and could move with each revision of a ROM, a standard technique was developed for accessing the routines. This allows code written for one version of the ROM to be reused with all subsequent revisions without needing to rewrite or recompile the code.

For all variants of the MAXQ processor, the utility ROM has a table with addresses for each of its supported functions. The location of this table can vary from part to part, so a pointer to this table is always stored at address 800Dh. The addresses for the supported functions can then be found by indexing into the table. This table always maintains the same order for the functions throughout all revisions of a particular ROM. **Table 1** lists the MAXQ2000 functions and their entry point within the table.

Table 1. MAXQ2000 Utility ROM User-Function Table

FUNCTION NUMBER	FUNCTION NAME	ENTRY POINT (USERTABLE = ROM[800Dh])
0	Reserved	ROM[userTable + 0]
1	Reserved	ROM[userTable + 1]
2	Reserved	ROM[userTable + 2]
3	moveDP0	ROM[userTable + 3]
4	moveDP0inc	ROM[userTable + 4]
5	moveDP0dec	ROM[userTable + 5]
6	moveDP1	ROM[userTable + 6]
7	moveDP1inc	ROM[userTable + 7]
8	moveDP1dec	ROM[userTable + 8]
9	moveFP	ROM[userTable + 9]
10	moveFPinc	ROM[userTable + 10]
11	moveFPdec	ROM[userTable + 11]
12	copyBuffer	ROM[userTable + 12]

Executing a utility ROM function requires four steps. Firstly, retrieve the location of the function table from address 800Dh. Secondly, add the offset for the desired function. Thirdly, retrieve the address of the utility function by reading from the computed location. Finally, execute the function by performing a call to the location found in the table. The following MAXQ assembly function demonstrates these four steps, using the `moveDP1inc` function of the MAXQ2000 as an example.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function:      ReadDataAtDP1
;; Description:   This function uses the utility ROM function "moveDP1inc"
;;                to read from program memory the data stored at the
;;                address in DP[1].  If DP[1] is in word mode two
;;                bytes will be read.  If DP[1] is in byte mode only
;;                one byte is read.  DP[1] is then post incremented.
;; Returns:       The result is returned in GR.
;; Destroys:      ACC and DP[0]
;; Notes:         This function assumes that DP[0] is set to word
;;                mode and the device has 16-bit accumulators.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ReadDataAtDP1:
  move  DP[0], #0800Dh ; This is where the address of the table is stored.
  move  ACC, @DP[0]    ; Get the location of the function table.
  add   #7             ; Add the index to the moveDP1inc function.
  move  DP[0], ACC     ; Point to where the address of moveDP1 is stored.
  move  ACC, @DP[0]    ; Retrieve the address of the function.
  call  ACC            ; Execute the function.
  ret
```

Because future ROM versions of a particular MAXQ variant might place the utility functions in a different location, using a routine similar to the `ReadDataAtDP1` function guarantees forward compatibility. The "cost" of this compatibility is larger code size and longer execution times. In some cases, these tradeoffs might be unacceptable, making it worthwhile to call the utility ROM functions directly. To call a utility function directly, simply determine the location of the desired function and use this location as the destination of a `call`.

Reading a string defined in code space illustrates a common situation requiring the use of utility functions. A programmer might store error strings, informational strings, or even debug strings that get displayed during execution of an application. The code segment below shows one way of achieving this using the `ReadDataAtDP1` function as previously described.

```
Text:
  DB  "Hello World!",0                ; Define a string in code space.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function:      PrintText
;; Description:   Prints the string stored at the "Text" label.
;; Returns:       N/A
;; Destroys:      ACC, DP[1], DP[0], and GR.
;; Notes:         This function assumes that DP[0] is set to word mode,
;;                DP[1] is in byte mode, and the device has 16-bit
;;                accumulators.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PrintText:
  move  DP[1], #Text      ; Point to the string to display.
  move  ACC, DP[1]        ; "Text" is a word address and we need a
  sla                     ; byte address, so shift left 1 bit.
  or    #08000h           ; Code space is mapped to 8000h when running
  move  DP[1], ACC        ; from the ROM, so the address must be masked.
PrintText_Loop:
  call  ReadDataAtDP1     ; Fetch the byte from code space.
  move  ACC, GR
  jump  Z, PrintText_Done ; Reached the null terminator.
  call  PrintChar         ; Call a routine to output the char in ACC
  jump  PrintText_Loop    ; Process the next byte.
PrintText_Done:
  ret
```

Having a common way of accessing utility ROM routines also allows developers to write code that will work with all variants of a particular MAXQ processor.

Conclusion

Utility functions give developers an easy way to read data stored in program memory. Having a common way of accessing utility ROM routines also allows developers to write code that will work with all variants of a particular MAXQ processor. Libraries can be constructed once and then reused, eliminating concern that future ROM versions will not be compatible.