C. EPROM Programming

This appendix describes the MSP430 EPROM module. The EPROM module is erasable with ultraviolet light, and electrically programmable. Devices with an EPROM module are offered in a windowed package for multiple programming and OTP package, for one time programmable.
C.1 EPROM Operation

The CPU can fetch data and instructions from the EPROM. When the programming voltage is applied to the TDI/VPP pin, the CPU can also write to the EPROM module. Reading the EPROM is an identical process to that with other internal peripheral modules. Both programming and reading can occur on byte or word boundaries.

Erasure

Before programming, the entire EPROM should be erased. Erasing of the EPROM module is achieved by exposing the transparent window to ultraviolet light.

**Note:** EPROM exposed to ambient light

Normal ambient light contains the correct wavelength for erasure. When a device with a transparent window is programmed for use the window should be covered with an opaque label.

Exposing the EPROM module to ultraviolet light will also cause erasure in the EEPROM module, if it is on-chip. Any useful data in the EEPROM module must be reprogrammed after exposure to ultraviolet light.

The data in the EPROM module can be programmed serially via the integrated ‘JTAG’ feature, or via software which is part of the application software. The ‘JTAG’ implementation features an internal mechanism for security purposes. Once the ‘security fuse’ is activated, no accesses to the device via the ‘JTAG’ functions are possible. The ‘JTAG’ is permanently switched to the by-pass mode.

Programming

The application must provide an external voltage supply to the TDI/VPP pin, to provide the necessary voltage and current for programming. The minimum programming time is noted in the electrical characteristics of the device data sheets.

The EPROM control register EPCTL controls the EPROM programming, once the external voltage is supplied. The erase state is a ‘1’. When EPROM bits are programmed, they are read as ‘0’.

The programming of the EPROM module can be done for single bytes, words, blocks of individual length, or with the entire module. All bits that have a final level of ‘1’ must be erased before programming. The programming can be done on single devices or even in-system. The supply voltage should be in the range required by the device datasheet. The levels on the ‘JTAG’ pins are defined in the device datasheet, and are usually CMOS levels.
The execute bit EXE initiates and ends the programming to the EPROM module. The external voltage must be supplied to the TDI/VPP before EXE bit is set. The timing conditions are noted in the datasheets.

Bit 1: When the VPPS bit is set, the external programming voltage is connected to the EPROM module. The VPPS bit must be set before EXE bit is set. It can be reset together with the EXE bit. The VPPS bit must not be cleared between programming operations.

EPROM Protect

The EPROM access via the serial test and programming interface ‘JTAG’ can be inhibited when the ‘security fuse’ is activated. The security fuse is activated via serial instructions shifted into the ‘JTAG’. Activating the fuse is not reversible and any access to the internal system is disrupted. The by-pass function described in the standard IEEE1149.1 is active.
C.2 FAST Programming Algorithm

The FAST programming cycle is normally used to program the data into the EPROM. A programmed logical ‘0’ can be erased only by ultraviolet light. Fast programming uses two types of pulses: prime and final. The length of the prime pulse is typically 100µs (see the latest datasheet). After each prime pulse, the programmed data is verified. If it fails 25 times, the programming operation was false. If correct data is read, the final programming pulse is applied; the final programming pulse is 3 times the number of prime pulses applied.
C3. Program EPROM module via serial data link using ‘JTAG’ feature

The hardware interconnection of the ‘JTAG’ pins is done via four separate pins, plus the ground or VSS reference level. The ‘JTAG’ pins are TMS, TCK, TDI/(VPP) and TDO/(TDI).

Note *: TDI in standard mode, VPP input during programming
Note **: TDO in standard mode, Data input TDI during programming
Note ***: see electrical characteristics in the latest data sheet
Note ****: Optional, fast incrementing of address via PC possible
Switches shown for programming situation
C4. Programming EPROM module via controller’s software

The hardware needed to program an EPROM module is quite simple: connect the required supply to the TDI/VPP pin, and run the proper software algorithm. The software algorithm that controls the EPROM programming cycle can not run in the same EPROM module to which the data should be written. It is impossible to read instructions from the EPROM and write data to it at the same time. The software needs to run from another memory - from a ROM module, a RAM module or another EPROM module.

Note *: Internally a pull-up resistor is connected to TMS and TCK
Note **: ROM devices of MSP430 have an internal pull-up resistor at pin TDI/VPP
MSP430Pxxx or MSP430Exxx have no internal pull-up resistor. They should have an ext. pull-down resistor preventing floating input node.
Note ***: The TDO/TDI pin should have an ext. pull-down resistor preventing floating input node for secondary TDI function.
Programming EPROM module via controller’s software, Example;

The software example writes one byte into the EPROM with the fast programming algorithm. The code is written position-independent, and will have been loaded (e.g. to the RAM) before it is used. The programming algorithm runs during the programming sequence in the RAM, thus avoiding conflict when the EPROM is written. The data (byte) which should be written is located in the RAM address ‘BurnByte’, and the target address of the EPROM module is held in the register ‘pointer’ defined with set directive. The timing is adjusted to a cycle time of 1µs. When another cycle time / processor frequency is selected, the software should be adjusted according to the operating conditions.

Example: Write data in yyyy into location xxxx
BurnByte = (yyyy) = (9Ah)
R9 = xxxx

The target EPROM module can not execute the programming code sequence while the data is being written into it. In the example, a subroutine moves the programming code sequence into another memory, e.g. into the RAM.
Start-of-subroutine: load_burn_routine

Source start address of the code sequence >> R7
Destination start address of the code sequence >> R10

Move one word: (R7) >> (R10)
Increment Source and dest. pointer in R7 and R10

End-of-source code?

End-of-subroutine: RET

; Definitions used in Subroutine:
; Move programming code sequence into RAM (load_burn_routine)
; Burn a byte into the EPROM area (RELOC_Burn_EPROM)

EPCTL .set 054h ; EPROM Control Register
VPPS .set 2 ; Program Voltage bit
EXE .set 1 ; Execution bit
BurnByte .set 0220h ; address of data to be written
Burn_orig .set 0222h ; Start address of burn
loops .set 25
r_timer .set r8 ; 1us = 1 cycle
pointer .set r9 ; pointer to the EPROM address
; r9 is saved in the main routine before subroutine call is executed
r_count .set r10
lp .set 3 ; dec r_timer : 1 cycle : loop_t100
; jnz : 2 cycles : loop_t100
ov .set 2 ; mov #(100-ov)/lp, r_timer : 2 cycles

; Load EPROM programming sequence to another location e.g. RAM, Subroutine

;--- Burn subroutine: position independent code!

RAM_Burn_EPROM .set Burn_orig
load_burn_routine
    push r9
    push r10
    mov #Burn_EPROM,R9 ; load pointer source
    mov #RAM_Burn_EPROM,R10 ; load pointer dest.
load_burn1
    mov @(R9,0(R10)) ; move a word
    incd R10 ; dest. pointer + 2
    incd R9 ; source pointer + 2
MSP430 Family

EPROM module

cmp #Burn_end,R9 ; compare to end_of_table
jne load_burn1
pop r9
pop r10
ret

; Program one byte into EPROM, Subroutine

Burn_EPROM
  ; ensure correct burn timing
  dint
  mov.b #VPPS,&EPCTL ; VPPS on
  push r_timer ; save registers
  push r_count ; programming subroutine
  mov #loops,r_count ; 2 cycles = 2 us

Repeat_Burn
  mov.b &BurnByte,0(pointer) ; write to data to EPROM
  ; 6 cycles = 6 us
  bis.b #EXE,&EPCTL ; EXE on
  ; 4 cycles = 4 us
  ; total cycles VPPon to EXE
  ; 12 cycles = 12 us (min.)
  mov #(100-ov)/lp,r_timer ;:programming pulse of
  100us
  wait_100
  102us
    dec r_timer ;:
    jnz wait_100 ;:
    bic.b #EXE,&EPCTL ;:EXE / prog. puls off
    mov #4,r_timer ;:wait min. 10 us
    wait_10
    dec r_timer ;:before verifying
    jnz wait_10 ;:programmed EPROM
    wait_10
    inc r_count ;:location, actual 13+ us
    cmp.b &BurnByte,0(pointer) ; verify data = burned data
    jne Burn_EPROM_bad ; data ≠ burned data > jump
    mov.b &BurnByte,0(pointer) ; write to EPROM again
    bis.b #EXE,&EPCTL ; EXE on
    add #(0xffff-loop),r_count; Number of loops for
    ; successful programming
    final_puls
    mov #(300-ov)/lp,r_timer ;:programming pulse of
    wait_300
    ;:3*100us*N starts
    dec r_timer ;:
    jnz wait_300 ;:
    inc r_count ;:
    jn final_puls ;:
    clr.b &EPCTL ;:EXE off / VPPS off
    jmp Burn_EPROM_end


Burn_EPROM_bad
  dec r_count ; not ok : decrement loop counter
  jnz Repeat_Burn ; loop not ended : do another trial
  inv.b &BurnByte ; return the inverted data to flag
  ; failing the programming attempt
  ; the EPROM address is unchanged

Burn_EPROM_end
  pop r_timer
  pop r_count
  eint
  ret

Burn_end