Application Report
SNVA715–July 2014

LP8501 Programming Considerations

Sami Kotijarvi

Introduction

This document describes LP8501 programming instructions with examples. Most of the programs are presented with command compiler syntax. Command compiler is described in more detail in Using the LP8501 Evaluation Kit (SNVU451). Compiler software is available with the evaluation kit.

Programs here consist of directives, labels, instructions and comments. The machine code, which is loaded into LP8501 SRAM memory, consists of 16–bit instructions. These instructions are written into registers from 50h to 6Fh. In register 4Fh is a page selector with 6 possible pages to choose from (bits '0' to '101' [0:2]). Instructions must be written to two consecutive addresses, like for example 50h and 51h in page 0. These addresses correspond to SRAM address 00h. The paging of SRAM memory is only for I^2C communication. When developing the code one can treat the whole memory as a whole. This means that the program code can continue to different SRAM pages. The paging needs to be taken into consideration only when the program code is uploaded via I^2C. The paging does not affect program code execution.

Instructions are described also in LP8501 datasheet (SNVS548). In Simple Program Example is an example of a simple program that blinks LED output 1 endlessly. Note that in compiler syntax engines are selected in that order that they appear in the text editor.

```
.segment program1               ;Beginning of a segment.
                 mux_sel 1      ;select LED1
loop1:           set_pwm 255    ;beginning of a loop, set PWM full scale.
                 wait 0.48      ;wait for 0.48 seconds.
                 set_pwm 0      ;set PWM to 0%
                 wait 0.48      ;wait for 0.48 seconds.
                 branch 0, loop1 ;endless loop
```

Simple Program Example

Defining LED outputs

There are two ways of defining which LED outputs the lighting engines use. One way is to use **mux_sel** instruction, which selects directly one and only one LED output. The other way is to use LED mapping table. Mapping table is defined with **mux_ld_start** and **mux_ld_end**.

Mux_sel instruction

This instruction maps one and only one LED output to an engine. In command compiler syntax this instruction has one parameter, which is the selected LED output. In Simple Program Example LED output 1 is selected for engine 1. Instruction ***mux_sel*** *1* in hexadecimal is 9D01h, where 9D is the instruction and 01 means the LED output 1. Parameter 1–9 correspond to LED output 1–9 accordingly, 16 corresponds to GPO.

Defining a mapping table

Creating a mapping table starts with defining a table, where each row defines which LED outputs are mapped at that time. In compiler syntax each row, which is referred later in the program, needs a label. At least mapping table start and end need to be labeled. Labeling is needed especially with **mux_set** instruction, since with this instruction one can select a specific row from the mapping table. In Example of Mapping Table is defined a mapping table, where the starting point and ending point are labeled (begin_mux 1 and end_mux1). Directive **dw** defines which LED outputs are mapped. Here in this example only one LED output at a time is mapped. In the example data is represented in binary, but it can also be

defined with hexadecimal or with decimal numbers. In each row there is a comment telling which LED output is mapped. If GPO would have been mapped, it would have corresponded the MSB bit. In Second Example of Mapping Table each row is labeled. This is needed when using the **mux_set** instruction. Note that the mapping table can be located anywhere in the SRAM memory. In the examples shown in this document, mapping table is located to the beginning of the SRAM memory.

```
begin_mux1:     dw 0000000000000001b ;LED1 on evaluation program, D4 Green
                dw 0000000000000010b ;LED2 on evaluation program, D4 Blue
                dw 0000000001000000b ;LED7 on evaluation program, D4 Red
                dw 0000000000000100b ;LED3 on evaluation program, D5 Green
                dw 0000000000001000b ;LED4 on evaluation program, D5 Blue
                dw 0000000010000000b ;LED8 on evaluation program, D5 Red
                dw 0000000000010000b ;LED5 on evaluation program, D6 Green
                dw 0000000000100000b ;LED6 on evaluation program, D6 Blue
end_mux1:       dw 0000000100000000b ;LED9 on evaluation program, D6 Red
```

Example of Mapping Table

```
row1:           dw 0000000000000001b ;LED1 on evaluation program, D4 Green
row2:           dw 0000000000000010b ;LED2 on evaluation program, D4 Blue
row3:           dw 0000000001000000b ;LED7 on evaluation program, D4 Red
row4:           dw 0000000000000100b ;LED3 on evaluation program, D5 Green
row5:           dw 0000000000001000b ;LED4 on evaluation program, D5 Blue
row6:           dw 0000000010000000b ;LED8 on evaluation program, D5 Red
row7:           dw 0000000000010000b ;LED5 on evaluation program, D6 Green
row8:           dw 0000000000100000b ;LED6 on evaluation program, D6 Blue
row9:           dw 0000000100000000b ;LED9 on evaluation program, D6 Red
```

Second Example of Mapping Table

In machine code labeling is not needed, since the **mux_set**, **mux_ld_start** and **mux_ld_end** instructions refer to a certain address in SRAM. With machine code mapping table is defined by writing 16–bit word telling which LED outputs are mapped into consecutive SRAM addresses. See Machine Code vs Compiler Syntax for example.

Machine Code vs Compiler Syntax

Machine Code (in hex)	Corresponding Data In Compiler Syntax	Description
0001h	**dw** 0000000000000001b	Map LED output 1
0002h	**dw** 0000000000000010b	Map LED output 2
0040h	**dw** 0000000001000000b	Map LED output 7
8101h	**dw** 1000000100000001b	Map LED outputs 1, 9 and GPO
00FFh	**dw** 0000000011111111b	Map LED outputs 1–8

Declaring mapping table for engine

For the engines mapping table start address is declared with **mux_ld_start** instruction. In compiler syntax this instruction needs the labeled address from the mapping table. For example ***mux_ld_start*** *begin_mux1* (referring to Example of Mapping Table). In machine code 7 LSB bits define the SRAM address and 9 MSB bits define the instruction. For example if mapping table starts from SRAM address 01, the instruction is 9C01h. The ending of the mapping table is declared likewise with **mux_ld_end**. For example ***mux_ld_end*** *row9* (referring to Second Example of Mapping Table). Example of machine code ending to SRAM address 08, the instruction is 9C88h.

Different engines can refer to same mapping table, partly or totally. If different engines use same mapping table and they use same LED output at the same time, engine 1 has the highest priority to control the LED outputs over other engines. Engine 2 has higher priority than engine 3.

Moving through mapping table

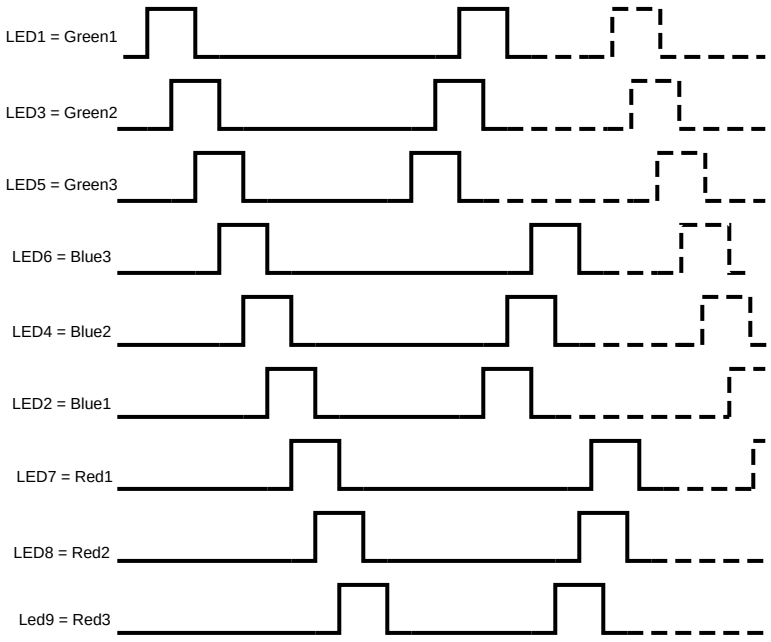
Going through the mapping table is managed with **mux_inc**, **mux_dec** and **mux_set** instructions. **Mux_inc** instruction sets the next row active in the mapping table. **Mux_dec** instruction on the other hand sets the previous row active in the mapping table. In the compiler syntax, these instructions are written as is without any parameters. In machine code **mux_inc** is 9D80 and **mux_dec** 9DC0. If the mapping table end is reached, activation will roll to the start address next time **mux_inc** instruction is called. If the mapping table start is reached, activation will roll to the end address next time **mux_dec** is reached.

Mux_set instruction has the address of the mapping table as a parameter. For example in compiler syntax **mux_set** row6 sets the mapping row labeled with row6 active, like in Second Example of Mapping Table). In machine code 7 LSB bits define the SRAM address and 9 MSB bits the instruction. For example to refer to SRAM address 06 the instruction would be 9F86h.

Below are two longer examples of using **mux_inc**, **mux_dec** and **mux_set**. These examples are created for the RGB LEDs in the evaluation board. The lighting sequence goes back and forth through RGB LEDs changing the color at each end. The sequence is as follows: G1 → G2 → G3 → B3 → B2 → B1 → R1 → R2 → R3 → G3 → G2 → G1 → B1 → B2 → B3 → R3 → R2 → R1 → G1 → ... See Sequence for Mux_inc, Mux_dec and Mux_set Examples for graphical illustration. First example describes the sequence with one engine and the second example with two engines. Note that when using two engines, you need to have the **mux_clr** in the first engine. Otherwise when the sequence goes to engine two, R2 (LED8) is mapped to engine one, which has higher priority and controls the R2.

Notes

One must note with these mapping instructions engines will not push a new PWM value to the LED output before **set_PWM** or **ramp** instruction is executed. If the mapping has been released from a LED output, the value in the PWM register will still control the LED brightness. If mapping is released from the GPO pin, serial bus control takes over the GPO state. One way to release mapping is to use **mux_clr** instruction. In compiler syntax instruction is given as it is, without any parameters. In machine code instruction is 9D00h. The other way to release mapping is to disable engines.



Sequence for Mux_inc, Mux_dec and Mux_set Examples

Example of using mux_inc, mux_dec and mux_set with one engine

```
0001    row1:     dw 0000000000000001b ;LED1 on evaluation program, D4 Green
0004    row2:     dw 0000000000000100b ;LED3 on evaluation program, D5 Green
0010    row3:     dw 0000000000010000b ;LED5 on evaluation program, D6 Green
0020    row4:     dw 0000000000100000b ;LED6 on evaluation program, D6 Blue
```

```
0008    row5:     dw 0000000000001000b ;LED4 on evaluation program, D5 Blue
0002    row6:     dw 0000000000000010b ;LED2 on evaluation program, D4 Blue
0040    row7:     dw 0000000001000000b ;LED7 on evaluation program, D4 Red
0080    row8:     dw 0000000010000000b ;LED8 on evaluation program, D5 Red
0100    row9:     dw 0000000100000000b ;LED9 on evaluation program, D6 Red.
segment engine1
9C00    mux_ld_start    row1         ;load mapping table
9C88    mux_ld_end      row9
9F80    loop1:  mux_set row1        ;select mapped LED from row1, LED1
04FF    ramp 0.2,255                       ;ramp LED1 up
9D80    mux_inc                      ;move to next mapped LED, LED3
04FF    ramp 0.2,255              ;ramp LED3 up
9DC0    mux_dec                   ;move back to previous LED, LED1
05FF    ramp 0.2, -255            ;ramp LED1 down
9F82    mux_set row3              ;select mapped LED from row3, LED5
04FF    ramp 0.2,255              ;ramp up LED5
9DC0    mux_dec                   ;move back to previous LED, LED3
05FF    ramp 0.2, -255            ;ramp LED3 down
9F83    mux_set row4              ;select mapped LED from row4, LED6
04FF    ramp 0.2,255              ;ramp up LED6
9DC0    mux_dec                   ;move to previous LED, LED5
05FF    ramp 0.2, -255            ;ramp down LED5
9F84    mux_set row5              ;select mapped LED from row5, LED4
04FF    ramp 0.2, 255             ;ramp up LED4
9DC0    mux_dec                   ;move back to previous LED, LED6
05FF    ramp 0.2, -255            ;ramp down LED6
9F85    mux_set row6              ;select mapped LED from row6, LED2
04FF    ramp 0.2, 255             ;ramp up LED2
9DC0    mux_dec                   ;move back to previous LED, LED4
05FF    ramp 0.2 -255             ;ramp down LED4
9F86    mux_set row7              ;select mapped LED from row7, LED7
04FF    ramp 0.2, 255             ;ramp up LED7
9DC0    mux_dec                   ;move back to previous LED, LED2
05FF    ramp 0.2, -255            ;ramp down LED2
9F87    mux_set row8              ;select mapped LED from row8, LED8
04FF    ramp 0.2, 255             ;ramp up LED8
9DC0    mux_dec                   ;move back to previous LED, LED7
05FF    ramp 0.2, -255            ;ramp down LED7
9F88    mux_set row9              ;select mapped LED from row9, LED9
04FF    ramp 0.2, 255             ;ramp up LED9
9DC0    mux_dec                   ;move back to previous LED, LED8
05FF    ramp 0.2, -255            ;ramp down LED8
9F86    mux_set row7              ;select mapped LED from row7, LED7
05FF    ramp 0.2, -255            ;ramp down LED7
9F88    mux_set row3              ;select mapped LED from row3, LED5
04FF    ramp 0.2,255              ;ramp up LED5
9F88    mux_set row9              ;select mapped LED from row9, LED9
05FF    ramp 0.2, -255            ;ramp down LED9
9F81    mux_set row2              ;select mapped LED from row2, LED3
04FF    ramp 0.2,255              ;ramp up LED3
9D80    mux_inc                   ;move to next LED, LED5
05FF    ramp 0.2, -255            ;ramp down LED5
9F80    mux_set row1              ;select mapped LED from row1, LED1
04FF    ramp 0.2,255              ;ramp up LED1
9D80    mux_inc                   ;move to next LED, LED3
05FF    ramp 0.2, -255            ;ramp down LED3
9F85    mux_set row6              ;select mapped LED from row6, LED2
04FF    ramp 0.2,255              ;ramp up LED2
9F80    mux_set row1              ;select mapped LED from row1, LED1
05FF    ramp 0.2, -255            ;ramp down LED1
9F84    mux_set row5              ;select mapped LED from row5, LED4
04FF    ramp 0.2, 255             ;ramp up LED4
9D80    mux_inc                   ;select next LED, LED2
05FF    ramp 0.2, -255            ;ramp down LED2
9F83    mux_set row4              ;select mapped LED from row4, LED6
04FF    ramp 0.2, 255             ;ramp up LED6
```

```
9D80    mux_inc                 ;select next LED, LED4
05FF    ramp 0.2 -255           ;ramp down LED4
9F88    mux_set row9            ;select mapped LED from row9, LED9
04FF    ramp 0.2, 255           ;ramp up LED9
9F83    mux_set row4            ;select mapped LED from row4, LED6
05FF    ramp 0.2, -255          ;ramp down LED6
9F87    mux_set row8            ;select mapped LED from row8, LED8
04FF    ramp 0.2, 255           ;ramp up LED8
9F88    mux_set row9            ;select next LED, LED9
05FF    ramp 0.2, -255          ;ramp down LED9
9F86    mux_set row7            ;select mapped LED from row7, LED7
04FF    ramp 0.2, 255           ;ramp up LED7
9D80    mux_inc                 ;select next LED, LED8
05FF    ramp 0.2, -255          ;ramp down LED8
9DC0    mux_dec                 ;select previous LED, LED7
05FF    ramp 0.2,-255           ;ramp down LED7
A002    branch 0, loop1         ;loop endlessly
C000    end
```

Example of using mux_inc, mux_dec and mux_set with two engines

```
0001    row1:     dw 0000000000000001b ;LED1 on evaluation program, D4 Green
0004    row2:     dw 0000000000000100b ;LED3 on evaluation program, D5 Green
0010    row3:     dw 0000000000010000b ;LED5 on evaluation program, D6 Green
0020    row4:     dw 0000000000100000b ;LED6 on evaluation program, D6 Blue
0008    row5:     dw 0000000000001000b ;LED4 on evaluation program, D5 Blue
0002    row6:     dw 0000000000000010b ;LED2 on evaluation program, D4 Blue
0040    row7:     dw 0000000001000000b ;LED7 on evaluation program, D4 Red
0080    row8:     dw 0000000010000000b ;LED8 on evaluation program, D5 Red
0100    row9:     dw 0000000100000000b ;LED9 on evaluation program, D6 Red
.segment engine1
9C00    mux_ld_start    row1          ;load mapping table
9C88    mux_ld_end      row9
9F80    loop1:  mux_set row1        ;select mapped LED from row1, LED1
04FF    ramp 0.2,255                        ;ramp LED1 up
9D80    mux_inc                                ;move to next mapped LED, LED3
04FF    ramp 0.2,255            ;ramp LED3 up
9DC0    mux_dec                 ;move back to previous LED, LED1
05FF    ramp 0.2, -255          ;ramp LED1 down
9F82    mux_set row3            ;select mapped LED from row3, LED5
04FF    ramp 0.2,255            ;ramp up LED5
9DC0    mux_dec                 ;move back to previous LED, LED3
05FF    ramp 0.2, -255          ;ramp LED3 down
9F83    mux_set row4            ;select mapped LED from row4, LED6
04FF    ramp 0.2,255            ;ramp up LED6
9DC0    mux_dec                 ;move to previous LED, LED5
05FF    ramp 0.2, -255          ;ramp down LED5
9F84    mux_set row5            ;select mapped LED from row5, LED4
04FF    ramp 0.2, 255           ;ramp up LED4
9DC0    mux_dec                 ;move back to previous LED, LED6
05FF    ramp 0.2, -255          ;ramp down LED6
9F85    mux_set row6            ;select mapped LED from row6, LED2
04FF    ramp 0.2, 255           ;ramp up LED2
9DC0    mux_dec                 ;move back to previous LED, LED4
05FF    ramp 0.2 -255           ;ramp down LED4
9F86    mux_set row7            ;select mapped LED from row7, LED7
04FF    ramp 0.2, 255           ;ramp up LED7
9DC0    mux_dec                 ;move back to previous LED, LED2
05FF    ramp 0.2, -255          ;ramp down LED2
9F87    mux_set row8            ;select mapped LED from row8, LED8
04FF    ramp 0.2, 255           ;ramp up LED8
9DC0    mux_dec                 ;move back to previous LED, LED7
05FF    ramp 0.2, -255          ;ramp down LED7
9F88    mux_set row9            ;select mapped LED from row9, LED9
04FF    ramp 0.2, 255           ;ramp up LED9
9DC0    mux_dec                 ;move back to previous LED, LED8
05FF    ramp 0.2, -255          ;ramp down LED8
```

```
9D00    mux_clr                         ;clear mapped LEDs
E004    trigger s{2}                         ;send trigger to engine2
E100    trigger w{2}                         ;wait for trigger from engine2
9F86    mux_set row7                         ;select mapped LED from row7, LED7
05FF    ramp 0.2, -255                    ;ramp down LED7
A002    branch 0, loop1                 :loop endlessly
C000    end
.segment engine2
9C00    mux_ld_start    row1                 ;define LED mapping
9C88    mux_ld_end    row9
E080    loop2: trigger w{1}           ;wait trigger from engine 1
9F82    mux_set row3              ;select mapped LED from row3, LED5
04FF    ramp 0.2,255              ;ramp up LED5
9F88    mux_set row9              ;select mapped LED from row9, LED9
05FF    ramp 0.2, -255            ;ramp down LED9
9F81    mux_set row2              ;select mapped LED from row2, LED3
04FF    ramp 0.2,255              ;ramp up LED3
9D80    mux_inc                   ;move to next LED, LED5
05FF    ramp 0.2, -255            ;ramp down LED5
9F80    mux_set row1              ;select mapped LED from row1, LED1
04FF    ramp 0.2,255              ;ramp up LED1
9D80    mux_inc                   ;move to next LED, LED3
05FF    ramp 0.2, -255            ;ramp down LED3
9F85    mux_set row6              ;select mapped LED from row6, LED2
04FF    ramp 0.2,255              ;ramp up LED2
9F80    mux_set row1              ;select mapped LED from row1, LED1
05FF    ramp 0.2, -255            ;ramp down LED1
9F84    mux_set row5              ;select mapped LED from row5, LED4
04FF    ramp 0.2, 255             ;ramp up LED4
9D80    mux_inc                   ;select next LED, LED2
05FF    ramp 0.2, -255            ;ramp down LED2
9F83    mux_set row4              ;select mapped LED from row4, LED6
04FF    ramp 0.2, 255             ;ramp up LED6
9D80    mux_inc                   ;select next LED, LED4
05FF    ramp 0.2 -255             ;ramp down LED4
9F88    mux_set row9              ;select mapped LED from row9, LED9
04FF    ramp 0.2, 255             ;ramp up LED9
9F83    mux_set row4              ;select mapped LED from row4, LED6
05FF    ramp 0.2, -255            ;ramp down LED6
9F87    mux_set row8              ;select mapped LED from row8, LED8
04FF    ramp 0.2, 255             ;ramp up LED8
9F88    mux_set row9              ;select next LED, LED9
05FF    ramp 0.2, -255            ;ramp down LED9
9F86    mux_set row7              ;select mapped LED from row7, LED7
04FF    ramp 0.2, 255             ;ramp up LED7
9D80    mux_inc                   ;select next LED, LED8
05FF    ramp 0.2, -255            ;ramp down LED8
E002    trigger s{1}              ;send trigger to engine 1
A002    branch 0, loop2              ;loop endlessly
C000    end                                  ;end program
```

Controlling LED outputs

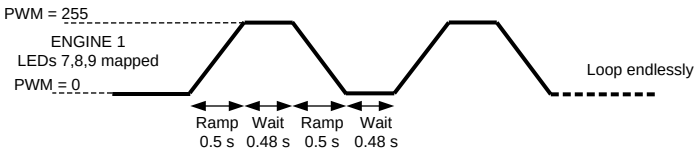
Set PWM instruction

Set_pwm instruction adjusts PWM level with 8-bit control from 0 to 255. PWM level is adjusted to new value in 0.488 ms (typ.). In compiler syntax **set_pwm** instruction has one parameter, which is the PWM value. Parameter can be set in decimal or hexadecimal. For example ***set_pwm*** *127* (or ***set_pwm*** *7F*) sets the PWM to 50% (in linear mode i.e. log not enabled). In machine code 8 LSB bits define the PWM value. For example 407Fh .

Ramp instruction

Ramp instruction generates either increasing or decreasing PWM ramp, which execution time and number of steps can be defined. In one **ramp** instruction PWM value can be incremented or decremented up to 255 steps from the present PWM value. Maximum PWM value is 255 which can be interpreted, that channel's current source is constantly active. In compiler syntax **ramp** instruction has two parameters, time and PWM step number. The maximum time is 31(step time)*15.6ms (prescale)*255(maximum PWM steps) = 123 s, although the compiler allows to feed maximum time of 127s. When using the compiler user does not need to calculate step times and prescales. For example ***ramp*** *0.5, 255*, which ramps up the mapped LED output(s) to full PWM value in 0.5 seconds in 255 steps (see ... if PWM value is different than 0 in this case). In PWM parameter, there can be minus sign to state that the ramp is decreasing. For example ***ramp*** *0.5, —127*, which ramps down the mapped LED output(s) 127 steps. With machine code, user has to decide prescale value (0 = 0.49 ms cycle time, 1 = 15.6 ms cycle time) and step time (maximum step time is 31). So maximum step time span would be 15.6 ms * 31 = 484 ms/step. The whole ramp time consists of this step time span times the number of PWM increment/decrement steps. For example 08FF has prescale value 0 –> 0.49 ms cycle time, step time 4, which leads to that step time span 4*0.49 is 1.96 ms, which then gets multiplied by 255, which results 499,8 ms so the whole ramp time 0.5 seconds in compiler syntax. For example 09FF would do the same as previous example with the exception that the ramp is decreasing. Note that if all the step time bits are set to zero, instruction is considered as **set_PWM** instruction.

Example below (Ramp Instruction Example) shows how LED outputs 7–9 are mapped to engine 1 and their PWM values are ramped up and down in 0.5 seconds. Here also is used **wait** instruction with maximum wait time 0.48s. In compiler syntax maximum wait time is 0.48 seconds. Also with machine code this is maximum, since with **wait** instruction there is available prescale and time. Prescale value 0 = 0.49 ms and 1 = 15.6 ms cycle time. Maximum time is 31. This results to 15.6 ms * 31 = 484 ms.

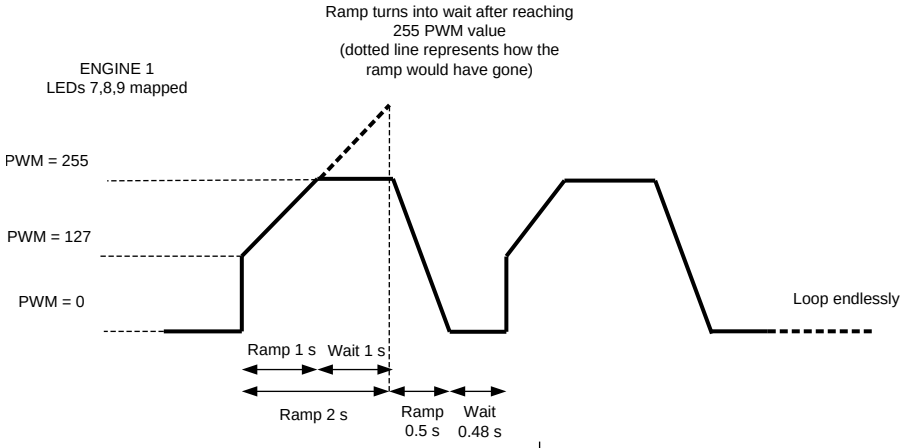


Program code in compiler syntax:		Program code in machine code:
mapping_leds:	dw 0000000111000000b	01C0
.segment example1		
	mux_ld_start mapping_leds	9C00
loop1:	ramp 0.5, 255	08FF
	wait 0.48	7E00
	ramp 0.5, -255	09FF
	wait 0.48	7E00
	branch 0, loop1	A001
	end	C000

Ramp Instruction Example

In case **ramp** instruction reaches the full or zero PWM value before all the ramp time has passed, the rest of the ramp time will saturate to wait time. In example Ramp and Wait Combined the program first sets the PWM value of the mapped LED outputs to 127 and after that starts to ramp up. When the maximum PWM value is reached, after 128 steps, the rest of the ramp will saturate to wait time. Used by this way the **ramp** instruction can be used as a wait also, reducing the need of extra **wait** instructions. In case where PWM value is already full or zero, **ramp** instruction produces wait for the ramp time period.

Ramp Instruction Example (continued)



Program code in compiler syntax:		Program code in machine code:
mapping_leds:	dw 0000000111000000b	01C0
.segment example1		
	mux_ld_start mapping_leds	9C00
loop1:	set_pwm 127	407F
	ramp 2, 255	20FF
	ramp 0.5, -255	09FF
	wait 0.48	7E00
	branch 0, loop1	A001
	end	C000

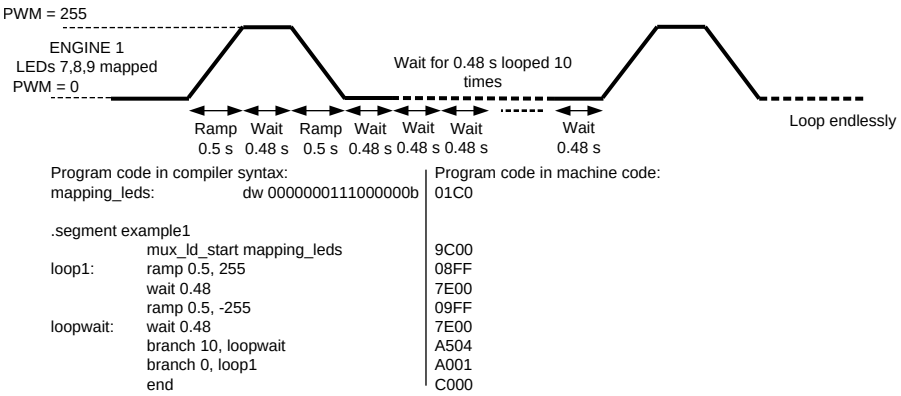
Ramp and Wait Combined

Looping

Branch Instruction

Branch instruction can be used to loop certain sequences in program. **Branch** instruction has two parameters, the loop count and the step number to be loaded into program counter. In compiler syntax the starting point of the loop must be labeled. For example *loop1: ramp 0.5, 255* (labeling the loop start address) and later on in the code ***branch*** *10, loop1*, which executes the code starting from loop1 labeled row to the branch instruction 10 times. One must notice that the program executes the sequence first time as normally and then do the 10 loops, so basically the code is executed 11 times. 0 in loop count parameter means endless loop. The maximum loop count is 63 in one branch command, but LP8501 supports loop inside loop i.e. nested looping.

In machine code 7 LSB bits are for defining the loop step count. The step count defines the steps needed from engine Start Address to the start of the loop. Loop count is defined with bits 7–12. For example A504h set to loop count bit 1010b, which is 10 in decimals. The program counter is set to start 4 steps from engine start address. See Example of Nested Loop, which also shows the nested loop example. Example of Nested Loop is almost the as Ramp Instruction Example with the exception that the **wait** instruction is now inside a loop allowing longer waiting period. In example Internal Trigger Example one can see the step number is the same (01) for all of the loops in different engines. This means that the start address of the loop is one step from the engine start address. Also in External Trigger Example example one can see how the steps change when loop is later on in the program.



Example of Nested Loop

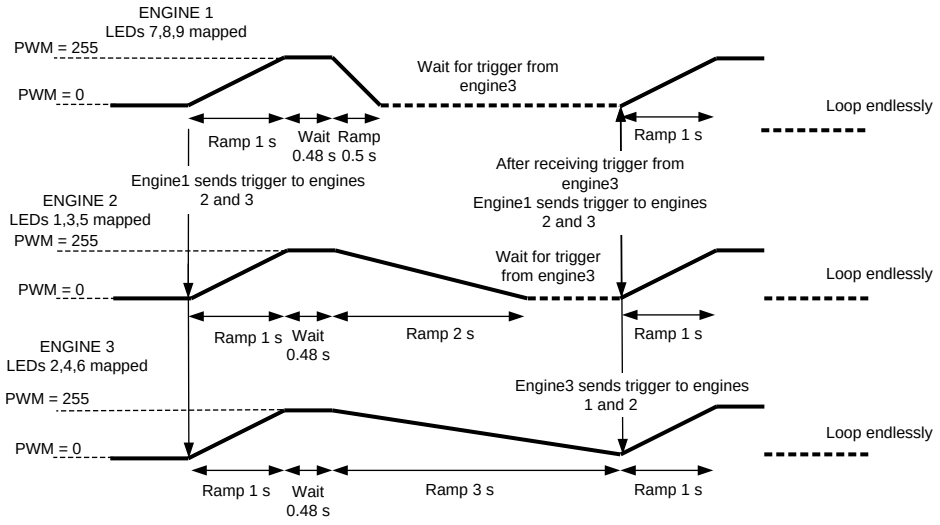
Go to Start Instruction

Go to start instruction resets program counter and program execution will be started from the beginning of the program. Go to start can be interpreted as infinite loop. By default all program memory locations are reset to zeros which implies to Go to start instruction. In command compiler syntax this instruction is **rst**. If program memory is fully occupied, and last instruction is **ramp**, **wait**, **set_pwm** or **trigger**, program execution will be continued from the beginning of the program.

Triggering

Triggering is an efficient way of controlling program execution between LP8501 engines or getting an external trigger to start program execution. Trigger signal can also be connected to processor. All engines can send and wait for trigger from other engines or from external trigger. In compiler syntax **trigger** has as a parameter **s**{x}, for sending a trigger, **w**{x}, for waiting a trigger, where x is value from 1–3 (engine number) or e (external trigger). The parameter value can consist also from multiple values separated by point. For example ***triggerss****{2.3}* instruction can be with engine 1, which sends trigger to engines 2 and 3. For example ***triggerw****{e}* can be set to engine to wait external trigger. In machine code bit 1–6 define sending trigger, bits 7–12 define wait for trigger. For example E008h sends a trigger to engine 3.

See Internal Trigger Example for internal triggering example. In this example engine sends trigger to engines 2 and 3. LED outputs 7–9 are mapped to engine 1, LED outputs 1,3 and 5 to engine 2 and LED outputs 2,4 and 6 to engine 3. With triggering all LED outputs are set to full PWM at the same time, since all have ramps up of 1 second. On the other hand ramping down is not done in same time and with triggering ramping up the LED outputs again can be set to start simultaneously.

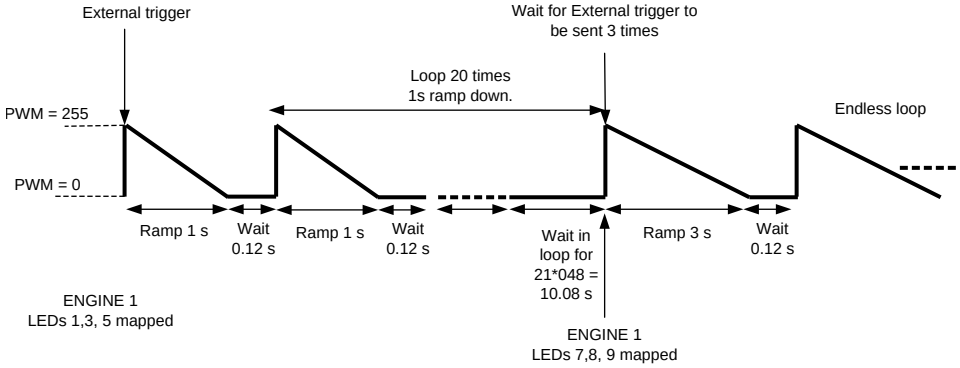



```
Program in compiler syntax:                                                    Program in machine code:
;Three LEDs muxed into three engines
begin_mux1:   dw 0000000111000000b   ;map LED7,LED8 and LED9 to first mux      01C0
begin_mux2:   dw 0000000000010101b   ;map LED1,LED3 and LED5 to second mux     0015
begin_mux3:   dw 0000000000101010b   ;map LED2,LED4 and LED6 to third mux      002A
.segment program1                ;Beginning of the segment 1.
        mux_ld_start begin_mux1 ;load first mux                                9C00
loop1:  trigger s{2.3}                                                         E00C
        ramp 1, 255              ;beginning of a ramp, in 1 seconds ramp to full scale  10FF
        wait 0.48                ;wait for 0.48 seconds.                       7E00
        ramp 0.5, -255           ;ramp PWM down to zero in 0.5 seconds         09FF
        trigger w{2.3}                                                         E300
        branch 0, loop1           ;jump to the beginning loop1, repeat endlessly  A001
.segment program2                ;beginning of the segment 2
        mux_ld_start begin_mux2  ;load second mux                              9C01
loop2:  trigger w{1}                                                           E080
        ramp 1, 255              ;ramp to full PWM in 1 second                 10FF
        wait 0.48                ;wait for 0.48 seconds                        7E00
        ramp 2, -255             ;ramp PWM down to zero in 1 second            21FF
        trigger s{1}                                                           E002
        branch 0, loop2          ;jump to the beginning of loop2, repeat endlessly  A001
.segment program3                ;beginning of the segment 3, same as segment 2
        mux_ld_start begin_mux3                                                9C02
loop3:  trigger w{1}                                                           E080
        ramp 1, 255                                                            10FF
        wait 0.48                                                              7E00
        ramp 3, -255                                                           31FF
        trigger s{1}                                                           E002
        branch 0, loop3                                                        A001
```

Internal Trigger Example

See External Trigger Example for external trigger example. Program will start after receiving an external trigger. LED outputs 1,3 and 5 are mapped first and they are set to full PWM value and ramped down in 1 second 20 times. After this loop there is wait in loop that lasts for 10.08 seconds. After this wait there is a loop, where three external triggers are expected. After getting all the triggers, LED outputs 7–9 are mapped and their PWM set to full and ramped down in 3 seconds in endless loop.

Internal Trigger Example (continued)



Program in compiler syntax		Program in machine code
row1:	dw 0000000000010101b	0015
row2:	dw 0000000111000000b	01C0
.segment program1		
	trigger w{e}	F000
	mux_ld_start row1	9C00
loop1:	set_pwm FFH	40FF
	ramp 1, -255	11FF
	wait 0.12	5000
	branch 20, loop1	AA02
loopwait:	wait 0.48	7E00
	branch 20, loopwait	AA06
looptrigger:	trigger w{e}	F000
	branch 2, looptrigger	A108
	mux_ld_start row2	9C01
loop2:	set_pwm 255	40FF
	ramp 3, -255	31FF
	wait 0.12	5000
	branch 0, loop2	A00B

External Trigger Example

Note that if all the engines have external triggering in the beginning, they will start all from one external triggering. External trigger input signal must stay low for at least two 32 kHz clock cycles to be executed. Trigger output signal is three 32 kHz clock cycles long. External trigger signal is active low, i.e. when trigger is send/received the pin is pulled to GND. If send and wait external trigger are used on the same instruction, the send external trigger is executed first, then the wait external trigger. Sent external trigger is masked, i.e. the device which has sent the trigger will not recognize it. If send and wait external trigger are used on the same instruction, the send external trigger is executed first, then the wait external trigger. Note also if engine tries to send a trigger to itself, send trigger alone will not have any effect but with wait trigger the engine will be stuck (waiting for trigger which will not come).

Sending interrupt and ending program

Interrupt

Interrupt instruction can be used to notify the processor. Interrupt pulls INT pin low and status bits in register address 3Ah informs which engine has caused the interrupt. Interrupt pin state and status bits will be cleared when status register 3Ah is read. In compiler syntax simply write **int** without any parameters.

End instruction

End instruction stops program execution. There are two parameters which can be defined with end command: interrupt and reset. Interrupt can be used to notify processor that program execution is at the end. Interrupt pulls INT pin low, and status bits in register address 3Ah informs which engine has caused the interrupt. Interrupt pin state and status bits will be cleared when status register 3Ah is read. Reset parameter resets program counter to 0 of the mapped LED outputs, changes channel to hold from run mode, and sets PWM output to 0. If no parameters are defined, channel will be changed to hold mode and PWM value will remain. It is preferred that every program ends with **end** instruction. In compiler syntax **end** instruction has optional parameter i (for interrupt) or r (reset). In machine code Int corresponds to bit 12 and reset to bit 11, for example D000h correspond to **end** instruction with interrupt.

Instruction Tables

LED DRIVER INSTRUCTIONS

Inst.	Bit[15]	Bit[14]	Bit[13]	Bit[12]	Bit[11]	Bit[10]	Bit[9]	Bit[8]	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
Ramp	0	prescale	Step time					Sign	# of increments							
Set PWM	0	1	0	0	0	0	0	0	PWM value							
Wait	0	prescale	Time					0	0	0	0	0	0	0	0	0

LED MAPPING INSTRUCTIONS

Inst.	Bit[15]	Bit[14]	Bit[13]	Bit[12]	Bit[11]	Bit[10]	Bit[9]	Bit[8]	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
mux_ld_start	1	0	0	1	1	1	0	0	0	SRAM address 0–95						
mux_ld_end	1	0	0	1	1	1	0	0	1	SRAM address 0–95						
mux_sel	1	0	0	1	1	1	0	1	0	LED select						
mux_clr	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0
mux_inc	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
mux_dec	1	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0
mux_set	1	0	0	1	1	1	1	1	1	SRAM address 0–95						

MISCELLANEOUS INSTRUCTIONS

Inst.	Bit[15]	Bit[14]	Bit[13]	Bit[12]	Bit[11]	Bit[10]	Bit[9]	Bit[8]	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
Go to Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Branch	1	0	1	Loop count						Step number						
Int	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
End	1	1	0	Int	Reset	X	X	X	X	X	X	X	X	X	X	X
Trigger	1	1	1	Wait for trigger						Send trigger						X
				Ext. trig	X	X	Engine3	Engine2	Engine1	Ext. trig	X	X	Engine3	Engine2	Engine1	X

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265