ACT82120 LED Programming Guide

Single Cell Battery Charger PMIC

1. Scope

The ACT82120 single-cell charger has many features and capabilities. Among these are sophisticated LED controllers and drivers. The purpose of this user guide is to explain in detail how to utilize the LED control and programming features to enable innovative and feature-rich end applications. This user guide supplements the ACT82120 datasheet, so to make full use of this user guide, it is recommended to refer first to the ACT82120 datasheet.

2. Controlling the LED Drivers

The LED drivers are referred to as B (blue) for LED0, G (green) for LED1, R (red) for LED2, and W (white) for LED3. Each LED driver is controlled either by its own PWM register or by one of three LED program execution engines. The selection of direct PWM control or automated engine control is by multiplexers, which are configured primarily by the LED_MAP register at address F0h. As evident in Fig. 2-1, any of the three LED program execution engines can control any of the four LED drivers, including multiple LED drivers. Similarly, any LED driver can be directly controlled by PWM values written via I²C interface with an external MCU.

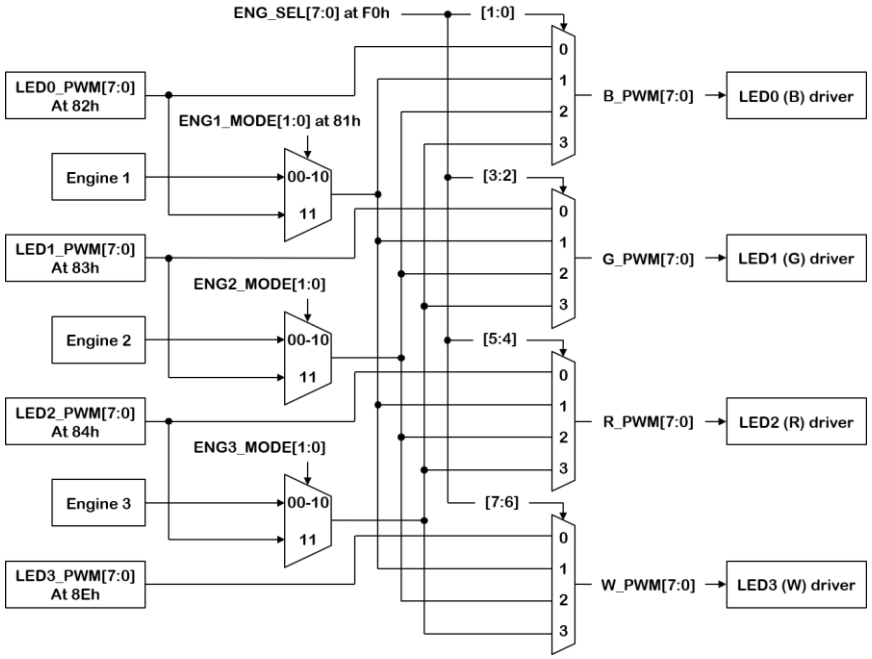

Fig. 2-1. LED MUX schematic

2.1. Direct PWM Control

Follow these steps for direct PWM control by I²C writes. We note that PWM values can be set while other PWM counters and/or program execution engines continue to run by leaving the LED_EN bit set (80h, bit 1).

Table 2-1. PWM control sequence

Step #	Task	Register	Bits
1	Make sure the EN_LED pin is high.	N/A	N/A
2	Clear the engine select bits in the LED_MAP register, which sets mux paths to LED drivers.	F0h	00b for each LED to be PWM controlled
3	Set LEDx output current level for LEDs to be PWM controlled. This can be changed at any time.	85 – 87h, 8Fh	8 bits, 0 to 25 mA in 100 µA steps
4	Set PWM value. This can be changed at any time.	82 – 84h, 8Eh	8 bits, 0 to 100% duty
5	Make sure LED_EN is set in the ENABLE register.	80h	Set bit 1

Direct control of LED brightness is by two registers: x_PWM register value and x_CURRENT register value, where x is B, G, R, or W. Each of these registers is 8 bits wide, allowing values from 0 to 255 (FFh). If either of these registers contains 00h, then the corresponding LED is off. If each contains FFh, then the corresponding LED is at full brightness. A typical approach is to use a constant value in the x_CURRENT registers and adjust LED brightness with the x_PWM registers. The ability to regulate each LED current with the ACT82120 eliminates the need for external resistors in series with the LEDs.

PWM clock speed can be changed from the default 256 Hz to 512 Hz (typical values) by setting bit 7 of the Configuration Control register at address 88h.

2.2. Programming the LED Engines

The LED engines are effectively small microprocessors with 6 programming instruction words: Ramp/Wait, Set PWM, Go to Start, Branch, End, and Trigger. This allows LED functionality to be highly automated, greatly reducing computational and communication burdens for the control MCU. Each programming engine automatically updates the PWM registers corresponding to which LEDs are controlled by each engine, as determined by the LED_MAP register at address F0h. The programming instructions (also called commands) and bit assignments are shown in Table 2-2. Each of the instruction words are described in detail below.

Table 2-2. Program engine instructions

Bit / Instr.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ramp/Wait	0	Pre-scale	Period Multiplier						Sign	Increment Count (number of steps)						
Set PWM	0	1	0						PWM value							
Go to Start	0	0	0						0	0	0	0	0	0	0	0
Branch	1	0	1	Loop Count						x	Program Counter					
End	1	1	0	Int	Reset	x										
Trigger	1	1	1	x	x	x	Wait for trigger from			x	x	x	Send trigger to			x
							Eng. 3	Eng. 2	Eng. 1				Eng. 3	Eng. 2	Eng. 1	

2.2.1. Ramp/Wait

The Ramp/Wait instruction ramps LED brightness up or down or waits without changing the LED drive. Each LED program engine is synchronized to a 32 kHz clock. This is a nominal value that can vary somewhat. The LED engine clock frequency is 32/16 = 2 kHz, and the clock period is 0.5 ms. Step time is the LED engine clock period multiplied by Period Multiplier, which is the value in bits 8 through 13. For example, a Period Multiplier of 1 would result in a step time of 0.5 ms. Because Period Multiplier is 6 bits wide, its maximum value is 63, corresponding to a maximum step time of 31.5 ms. This is often too short, so the Pre-scale bit is provided. Setting Pre-scale divides the Ramp/Wait instruction clock by 32. Therefore, maximum step time is 32 · 0.5 ms · 63 ≈ 1 second. Increment Count is the number of times the corresponding PWM register is updated during execution of the Ramp/Wait instruction, with a step time between each PWM update. The total time to execute the Ramp/Wait instruction is the step time multiplied by 1 added to Increment Count, as shown in Table 2-3. Setting Increment Count to zero results in a wait (without incrementing any PWM registers) equal to a single step time as calculated in Table 2-3.

Table 2-3. Ramp/Wait instruction execution time calculation

Pre-scale	Step Time	Total Execution Time
0	0.5 ms · Period Multiplier	Step time · (Increment Count + 1)
1	32 · 0.5 ms · Period Multiplier	

Increment Count is 7 bits wide with a maximum value of 127. Each PWM register is 8 bits wide, so its maximum value is twice as large (plus 1). With each step time, the PWM register value increments (or decrements if the Sign bit is set) by 1 if the corresponding ENGx_STEP_SIZE bit is 0 in the CONFIG register at 88h. The PWM register increments/decrements by 2 if the ENGx_STEP_SIZE bit is 1. The step time is unaffected. For example, if Engine 1 Increment Count is 127 and ENG1_STEP_SIZE in the CONFIG register is set, the PWM register is incremented by 2 for 127 times. Setting the ENGx_STEP_SIZE bit therefore allows a PWM ramp from 0 to 254 with a single Ramp/Wait instruction. Continuing the example, if Period Multiplier is kept the same, execution time is half as much as for

two consecutive instructions, each incrementing the PWM register by 1 for 127 times. If the PWM register already contains its maximum or minimum value (255 or 0), further attempts by the Ramp/Wait instruction to respectively increment or decrement the PWM register value are ignored.

2.2.2. Set PWM

The Set PWM instruction is simply the opcode 40h followed by the 8-bit PWM value.

2.2.3. Go to Start

An instruction word of 0000h causes program execution to loop continuously from the beginning of the program. Clearing the ENGx_MODE bits in the OP_MODE register at address 81h stops program execution.

2.2.4. Branch

The branch instruction is really a jump instruction. It loops the number of times specified by the Loop Count value. Execution jumps to the instruction indicated by the value in the Program Counter field. If the Loop Count value is zero, program execution loops continuously until stopped by clearing the ENGx_MODE bits in the OP_MODE register at address 81h.

2.2.5. End

The End instruction stops program execution, puts engine program execution in Hold mode by clearing both ENGx_EXEC bits in the ENABLE register at address 80h, and it resets the program counter (PC) to zero. If the Reset bit is set, then the PWM register also resets to 00h, otherwise it retains its value. If the Int bit is set, then the corresponding ENGx_INT interrupt flag bit is set in the INTERRUPT register at address 8Ch, regardless of the state of the ENGx_INT_MSK bits in the CONFIG register at 88h. Reading the INTERRUPT register clears interrupt flags. I^2C polling the ENGx_INT flag bits reveals when an Engine program ends. An LED interrupt will not set the nIRQ pin low, regardless of the state of the ENGx_INT_MSK bits.

2.2.6. Trigger

The Trigger command provides a way to link program execution from one engine to another. Set bits corresponding to which engine(s) to trigger from/to. See example 3 below.

2.2.7. Programming Sequence

Follow these steps to program each LED program engine. Engines can be programmed separately while other engines and/or PWM counters continue to run by leaving the LED_EN bit set (80h, bit 1).

Table 2-4. Programming sequence

Step #	Task	Register	Bits
1	Make sure EN_LED pin is high. Set PU_EN_LED to use internal pullup, or use external drive or pullup resistor.	05h	Bit 0
2	Stop program by clearing ENGx_MODE bits	81h	00b for each engine being programmed
3	Optional: Stop program by clearing ENGx_EXEC bits. Clearing LED_EN (bit 1) at the same time is optional.	80h	00b for each engine programmed
4	Set engine select bits in LED_MAP register, which sets mux paths to LED drivers	F0h	2 bits per engine, per datasheet as needed. Also see Fig. 2-1.
5	Set LEDx output current level for LEDs to be controlled by a program engine. This can be changed at any time.	85 – 87h, 8Fh	8 bits, 0 to 25 mA in 100 µA steps
6	Set program words in program engines.	90 – EFh	See examples
7	Load program and reset PC to zero by setting ENGx_MODE bits.	81h	01b for each engine programmed
8	Set PC value if needed. Must first clear ENGx_EXEC bits in ENABLE register at 80h.	89 – 8Bh	4 bits for each engine
9	Prepare to run program by setting ENGx_MODE bits.	81h	10b for each engine programmed
10	Run program by setting ENGx_EXEC bits. Make sure LED_EN (bit 1) is set.	80h	10b for each engine programmed

2.2.8. Example 1, Using Ramp/Wait and Branch

In this example, LED0 PWM value is ramped from 0 to 254 and back to zero three times with about a one second pause after each LED pulse. The timing chart is shown in Fig. 2-2. In this and all remaining examples, the contents of the LED_MAP register at address F0h is 39h, corresponding to LED0 (B) controlled by Engine 1, LED1 (G) controlled by Engine 2, LED2 (R) controlled by Engine 3, and LED3 (W) directly controlled I²C writes to its PWM register. Furthermore, the pseudocode Ramp and Wait are really the same Ramp/Wait instruction but with non-zero versus zero value for Increment Count. Refer to Table 2-2.

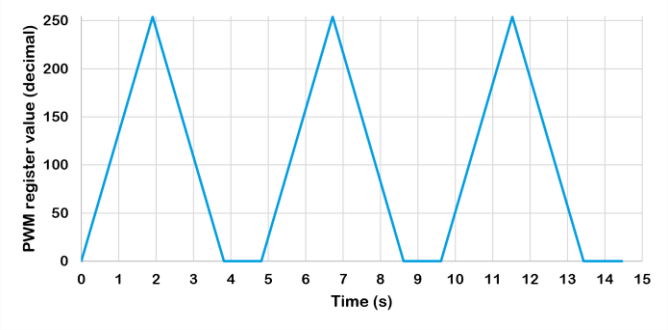

Fig. 2-2. Output of Engine 1 in examples 1 and 2

In Example 1, the value in the CONFIG register at address 88h is zero, corresponding to the default PWM frequency, ENGx_STEP_SIZE = 0, and ENGx_INT_MSK = 0. The PWM register therefore increments by 1 with each Increment Count in the Ramp instructions.

Table 2-5. Example 1 pseudo code and instruction word values in binary and hex

Engine 1 Instruction #	PC	Instruction	Binary	Hex
1	0	Ramp 15, 127	0000 1111 0111 1111	0F 7F
2	1	Ramp 15, 127	0000 1111 0111 1111	0F 7F
3	2	Ramp 15, -127	0000 1111 1111 1111	0F FF
4	3	Ramp 15, -127	0000 1111 1111 1111	0F FF
5	4	Wait 50	0011 0010 0000 0000	32 00
6	5	Branch 40, 4	1011 0100 0000 0100	B4 04
7	6	Branch 2, 0	1010 0001 0000 0000	A1 00
8	7	End with reset	1100 1000 0000 0000	C8 00

The first two instructions ramp the PWM register value from 0 to 254. This requires two consecutive instructions because the Ramp instruction Increment Count field is 7 bits wide, thus its maximum value is 127. The next two instructions ramp the PWM register value back down from 254 to 0 by setting the Sign bit. The Wait 50 instruction step time is 0.5 ms · 50 = 25 ms (see Table 2-3), and we note that it does not increment the PWM register value because its Increment Count field is zero. Instruction 6 (at PC = 5) loops to PC = 4, Wait 50 instruction, 40 times, resulting in a 1 second delay. Instruction 7 (at PC = 6) loops twice to the beginning of the program, causing 3 total LED pulses, as in Fig. 2-2. The End instruction ensures that the final PWM register value is zero because the Reset bit is set.

2.2.9. Example 2, Using Pre-scale and ENGx_STEP_SIZE

The code in Example 2 has the same result as in Example 1 but with fewer instructions by using Pre-scale and ENGx_STEP_SIZE. Specifically, the ENG1_STEP_SIZE bit is set in the CONFIG register at 88h (not shown in the example tables). The first instruction therefore can ramp the PWM register value from 0 to 254 with its Increment Count field set to 127 because the PWM register increments twice for each Increment Count. Similarly, the second instruction ramps the PWM register value from 254 back to 0 by setting the Sign bit. There is no need for duplicate instructions.

The third instruction sets the Pre-scale bit, resulting in a step time of 32 · 0.5 ms · 63 ≈ 1 second. This eliminates the need for a subsequent Branch instruction to loop enough times to create the desired 1 second delay.

Table 2-6. Example 2 pseudo code and instruction word values in binary and hex

Engine 1 Instruction #	PC	Instruction	Binary	Hex
1	0	Ramp 30, 127	0001 1110 0111 1111	1E 7F
2	1	Ramp 30, (-127)	0001 1110 1111 1111	1E FF
3	2	Wait 32 · 63	0111 1111 0000 0000	7F 00
4	3	Branch 2, 0	1010 0001 0000 0000	A1 00
5	4	End	1100 1000 0000 0000	C8 00

The code in Table 2-6 results in the same output as that in Table 2-5, and the timing chart in Fig. 2-2. Again, the instruction count reduces due to ENG1_STEP_SIZE being set in the CONFIG register, and Pre-scale (bit 14, the 32x clock divider) being set in the third instruction at PC = 2.

2.2.10. Example 3, Using Set PWM, Interrupt, and Trigger

Example 3 continues from Example 2 by having Engine 1 trigger Engine 2 to commence its program execution. This example therefore has two sets of code, one for Engine 1 and the other for Engine 2.

Table 2-7. Example 3, Engine 1 pseudo code and instruction word values in binary and hex

Engine 1 Instruction #	PC	Instruction	Binary	Hex
1	0	Ramp 30, 127	0001 1110 0111 1111	1E 7F
2	1	Ramp 30, (-127)	0001 1110 1111 1111	1E FF
3	2	Wait 32 · 63	0111 1111 0000 0000	7F 00
4	3	Branch 2, 0	1010 0001 0000 0000	A1 00
5	4	Set PWM 170	0100 0000 1010 1010	40 AA
6	5	Trigger engine 2	1110 0000 0000 0100	E0 04
7	6	End with interrupt	1101 0000 0000 0000	D0 00

The Engine 1 program adds two new instructions: Set PWM and Trigger. The Set PWM instruction is straightforward, allowing any value from 0 to 255 to be written directly to the PWM register that Engine 1 is set to control. (See Fig. 2-1). We note that for the LED to remain lit, the End instruction Reset bit must be clear. The End instruction has the Int (interrupt) bit set, but this has no effect on the PWM register. It simply sets the ENG1_INT bit in the INTERRUPT register at 8Ch.

The Trigger instruction sets a trigger for Engine 2. For this to have the desired effect, the Engine 2 program must also have a Trigger instruction with its ‘Wait for trigger from Engine 1’ bit set, as shown in Table 2-8.

Table 2-8. Example 3, Engine 2 pseudo code and instruction word values in binary and hex

Engine 2 Instruction #	PC	Instruction	Binary	Hex
1	0	Trigger on engine 1	1110 0000 1000 0000	E0 80
2	1	Ramp 30, 127	0001 1110 0111 1111	1E 7F
3	2	Ramp 30, (-127)	0001 1110 1111 1111	1E FF
4	3	Branch 2, 1	1010 0001 0000 0001	A1 01
5	4	Ramp 15, 63	0000 1111 0011 1111	0F 3F
6	5	Ramp 15, (-63)	0000 1111 1011 1111	0F BF
7	6	Branch 0, 4	1010 0000 0000 0100	A0 04
8	7	End with reset	1100 1000 0000 0000	C8 00

The Engine 2 program waits at its Trigger from Engine 1 instruction until the Engine 1 Trigger command executes. Both Engine 1 and Engine 2 then continue executing instructions following their respective Trigger instructions, so they run the remainder of their programs simultaneously.

The remainder of the Engine 2 program implements the PWM output shown by the dashed lines in Fig. 2-3.

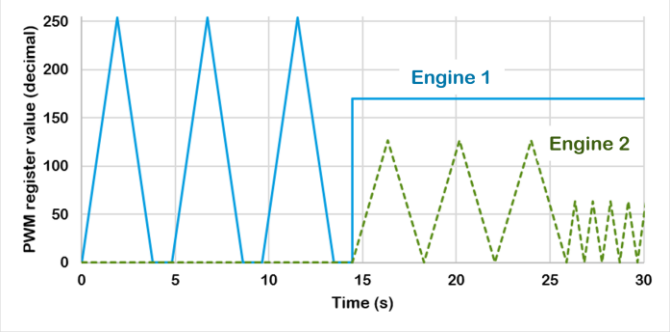

Fig. 2-3. Output of Engine 1 and Engine 2 in example 3

The Engine 1 program generates the same three LED pulses as before but leaves the LED on afterward. Engine 2 ramps the PWM value to 127 and back to 0 three times and then repeats shorter pulses indefinitely. The End instruction in the Engine 2 program is never reached.

Note here that Engine 2 instructions 2 and 3 are identical to Engine 1 instructions 1 and 2, yet the Engine 2 PWM values peak at half the value of those in the Engine 1 program, and they have the same execution time. How is this possible? It is because ENG1_STEP_SIZE (88h, bit 6) is set, whereas ENG2_STEP_SIZE (88h, bit 5) is clear. Therefore, the Engine 1 Ramp instructions increment/decrement by 2, and the Engine 2 Ramp instruction increment/decrement by 1 with each Increment Count.

3. Summary

This application note explains in detail LED control by the ACT82120 single-cell charger. Control of multiple LEDs can be from direct control by I^2C writes to PWM registers, and/or from three individual LED program execution engines. Configuring one or more of the three LED program engines to control one or more of four LEDs is accomplished by writing to two registers using I^2C. There is a sequence of steps that must be followed for both direct PWM control and automated engine control. All six LED engine programming instructions are explained in detail. Three examples demonstrate how to use each programming instruction and include instruction words in binary and hex. This understanding enables the development of highly flexible and reconfigurable applications using the simple LED control features in the ACT82120.

Revision History

Revision	Author	Date	Description
A	Jonathan Dodge, P.E.	10 Sept. 2026	Initial draft

Contact Information

For the latest specifications, additional product information, worldwide sales and distribution locations:

Web: www.qorvo.com

Tel: +1 844-890-8163

Email: customer.support@qorvo.com

Important Notices

The information contained in this Document and any associated documents ("Document Information") is believed to be reliable; however, Qorvo makes no warranties regarding the Document Information and assumes no responsibility or liability whatsoever for the use of or reliance on said information. All Document Information is subject to change without notice. Customers should obtain and verify the latest relevant Document Information before placing orders for Qorvo® products. Information concerning Qorvo's product life cycles is available at https://www.qorvo.com/support/product-lifecycle-information. Document Information or the use thereof does not grant, explicitly, implicitly or otherwise any rights or licenses with respect to patents or any other intellectual property whether with regard to such Document Information itself or anything described by such information.

Qorvo grants you permission to use this Document and any associated resources only to develop an application that uses the Qorvo products described in the Document and any associated resources. Other reproduction and display of this Document and any associated resources is prohibited.

Qorvo's products are provided subject to Qorvo's Terms of Sale or provided in conjunction with such Qorvo products. Qorvo objects to and rejects any additional or different terms customer may have proposed regarding the purchase of Qorvo products.

APPLICATION NOTE INFORMATION DOES NOT CONSTITUTE A WARRANTY WITH RESPECT TO THE PRODUCTS DESCRIBED HEREIN, AND QORVO HEREBY DISCLAIMS ANY AND ALL WARRANTIES WITH RESPECT TO SUCH PRODUCTS WHETHER EXPRESS OR IMPLIED BY LAW, COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE OF TRADE OR OTHERWISE, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Without limiting the generality of the foregoing, Qorvo® products are not warranted or authorized for use as critical components in medical, life-saving, or life-sustaining applications, or other applications where a failure would reasonably be expected to cause severe personal injury or death. Applications described in the Document Information are for illustrative purposes only. Customers are responsible for validating that a particular product described in the Document Information is suitable for use in a particular application.