# AN2424
# Application note

## STMPE2401 - Port expander PWM controller

## Introduction

STMPE2401 is the first in the family of ST port-expander logic products. The principle of a basic expander logic is to provide additional I/Os that can be used by the host processor to implement additional features such as increasing the number of control signals and mixed signal lines, or controlling more number of peripherals.

In addition to the above mentioned basic features, STMPE2401 comes with integrated intelligence to implement advanced features like Keypad scanning, PWM control, and Rotator dial control. These features enable the processor load to be reduced.

There is also a provision for using a single crystal clock to drive multiple STMPE2401 devices by cascading the devices and using the CLKOUT mode to drive the clock of the cascaded devices.

STMPE2401 can be widely used in the fields of Mobile Communications, Portable media players, Game console, Mobile Phones, Smart Phones, Consumer Electronics and computer peripherals like state-of-the-art printers, Advanced embedded systems etc.

This application note explains the setup and programming of the integrated PWM controller in STMPE2401 to do LED backlighting, brightness control, and LED blinking patterns.

# Contents

# List of tables

# 1 Advantages of an integrated PWM controller

- Low CPU utilization
- Lower power consumption
- Simpler driver software
- Simpler connection to CPU (only two $I^2C$ lines)
- Advanced programmable ramping/blinking patterns for lighting effects
- Analog/PWM output
- Application: Mobile phone backlighting, LED brightness control

## 1.1 STMPE2401 default power-up configuration

STMPE2401 operates at a supply voltage of 1.8 V. The clock can be supplied through a 32 KHz crystal connected across XTALIN, XTALOUT pins or through an external oscillator clock on XTALIN pin. The external oscillator can be low accuracy (16 KHz ~ 32 KHz) and the clock frequency for normal operation should not exceed 32 KHz. The Reset pin should be pulled high for the device to come out of reset and operate in the normal operating mode.

The device can be accessed through the $I^2C$ interface and up to four STMPE2401 devices can be connected to the same $I^2C$ bus. STMPE2401 supports 7-bit addressing as per the Philip $I^2C$ specification Ver 2.1. The slave address is selected by the state of two pins (GPIO15 and GPIO23). The state of the pins is latched into STMPE2401 at power on and this address setting is retained until the power is switched off.

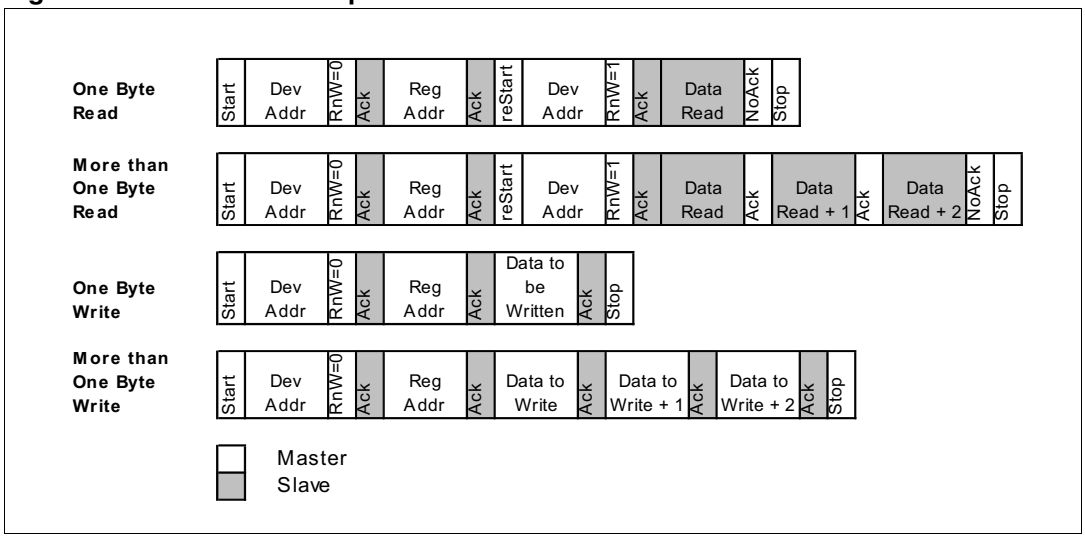
The address can be changed and latched-in again using the soft_reset bit in the SYSCON register.

The $I^2C$ Read/Write is done byte by byte. The R/W bit is added as the LSB to the 7-bit slave address to make up one byte to be sent through the $I^2C$ interface from the Master. Once the slave address is configured and responding correctly, the internal registers can be accessed through $I^2C$ read and write commands. *Table 1* lists the slave addresses that can be used and the $I^2C$ Read/Write protocol to access the device registers.

**Table 1. Valid STMPE2401 slave address**

| ADDR1 (GPIO23) | ADDR0 (GPIO15) | 7-bit slave addressing | 8-bit format to be used (including R/W bit in LSB) |
|---|---|---|---|
| 0 | 0 | 42h (1000010b) | 84h |
| 0 | 1 | 43h (1000011b) | 86h |
| 1 | 0 | 44h (1000100b) | 88h |
| 1 | 1 | 45h (1000101b) | 8Ah |

**Figure 1. I²C Read/Write protocol**




At power-up all GPIOs function as inputs and by default the interrupt is configured as an Active Low Level interrupt. The interrupt however remains low irrespective of the settings until the Global Interrupt bit in the ICR register is enabled (set to '1').

# 2 PWM controller

STMPE2401 comes with an integrated PWM controller that can provide three independent PWM outputs. These outputs can be used to generate light effects like rapid blinking or dimming in LEDs. There are three PWM channels and they can be triggered independently. Each PWM channel can be triggered by the other two channels. The unused PWM output pins can be used as GPIO.

Each channel comes with a maximum 64 words x 16-bit command memory. The instructions to be stored in the command memory can be downloaded through the I²C connections. Any attempt to load beyond 64 words causes the internal address pointer to roll-over (0x1f -> 0x00) and the excess instructions overwrite the first address location of the channel and onwards.

# 3 Operation modes and clocking

The PWM controller can be enabled only in the normal operational mode. In operational mode, the PWM controller output is generated using the 32 KHz clock. The instruction fetching and execution is also derived from the 32 KHz clock domain. The internal 5 MHz clock is used only for the $I^2C$ interface. Therefore, if the device enters sleep mode while the PWM is enabled, there is no interruption in the PWM output even though the internal 5 MHz clock is cut-off in the sleep mode. However, the $I^2C$ interface does not function during sleep mode and therefore the PWM should be enabled before entering sleep mode.

During Hibernate mode, the 32 KHz clock is also shut down and all modules are disabled. The device resumes normal operation only after an $I^2C$ wake-up or Reset.

When the PWM function is not in use, power consumption can be reduced by gating off the clock to the PWM Controller. This can be done by setting the 'Enable_PWM' bit in the SYSCON register to zero.

**Table 2. SYSCON register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Soft_Reset | - | Disable_32KHz | Sleep | Enable_GPIO | Enable_PWM | Enable_KPC | Enable_ROT |
| Read/Write (IIC) | W | | RW | RW | RW | RW | RW | RW |
| Read/Write (HW) | RW | | R | RW | R | R | R | R |
| Reset value | 0 | | 0 | 0 | 1 | 1 | 1 | 1 |

**Table 3. SYSCON description**

| Bit | Name | Description |
|---|---|---|
| 2 | Enable_PWM | Writing a ‘0’ to this bit gates off the clock to the PWM Controller module, thus stopping its operation |
| 4 | Sleep | Writing a ‘1’ to this bit puts the device in sleep mode. When in sleep mode, all the units which need to work on clocks synchronous to 32 KHz get the clocks derived from the 32 KHz domain.The internal RC Oscillator shuts down. |
| 5 | Disable_32 KHz | Set this bit to disable the 32 KHz Clk, thus putting the device in hibernate mode. Only a Reset or a wakeup on $I^2C$ resumes normal operation of the device. |
| 7 | Soft_Reset | Writing a ‘1’ to this bit does a soft reset of the device. Once the reset is done, this bit is cleared to ‘0’ by the Hardware. |

# 4 PWM instruction set

The STMPE2401 PWM controller works as a simple MCU, with a program space of 64 instructions and a simple instruction set. The instructions are all 16-bits in length. The three most significant bits are used to identify the command.

- RAMP

This instruction starts the PWM counters and the pwm_output depends on the counter value.

**Table 4. RAMP instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | Prescale | Step time | | | | | | Sign | Increment | | | | | | |

**Table 5. RAMP description**

| Bits | Name | Description |
|---|---|---|
| 6:0 | Increment | Increment size. Takes value between 1-126. ‘0’ setting is not allowed. |
| 7 | Sign | ‘0’ – Step up counter<br>‘1’ – Step Down counter |
| 13:8 | Step Time | ‘0’ – Immediate action<br>1-63 – Step time per increment |
| 14 | Prescale | ‘0’ – Divide clock by 16<br>‘1’ – Divide clock by 512 clock |

Each increment is broken down into the number of steps specified by the 'Step time' parameter and the time taken for each step is based on the prescale clock. (refer *: Example 2:* in section *4.1*)

- SMAX (Set Maximum)

This instruction loads the PWM counter with the maximum value of 0xff and the resulting pwm_output is logic level low.

**Table 6. SMAX instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | $x_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Note: 1 “x“ don‘t care*

- SMIN (Set Minimum)

This instruction loads the PWM counter with the minimum value of 0x0 and the resulting pwm_output is logic level high.

**Table 7. SMIN instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | $x_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Note: 1 “x“ don‘t care*

- GTS (Go To Start)

This instruction branches to internal address 0x0 and executes from 0x0.

**Table 8. GTS instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | $x_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Note: 1 "x" don't care*

- BRANCH

This instruction branches to an absolute or relative location and executes with looping capability. There are 4 loop counters available and these allow 4 nested loops.

– Absolute Branching: The Absolute branch option jumps to the absolute address (relative to internal address 0x0) specified by the value of the step size. This is in the nature of a JUMP instruction for forward jump and if it is backward jump, it results in a "forever loop" execution.

– Relative Branching: Relative Branch jumps in a backward manner relative to the current address location, that is, a step size of 1 means jump to the previous instruction location and step size 0 means NOP. When looping, once the loop count is reached, the loop counter resets.

**Table 9. BRANCH instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 1 | Loop counter | | Loop count | | | | Absolute/ Relative Step size | Step size | | | | | |

**Table 10. BRANCH description**

| Bits | Name | Description |
|---|---|---|
| 12:11 | Loop counter | 0-3.<br>Four loop counters that can be used as nested loops also. |
| 10:7 | Loop count | 0-15<br>'0' Forever loop. |
| 6 | Absolute/Relative Step size | '0' – Absolute branch<br>'1' – Relative branch |
| 5:0 | Step size | 0-63<br>Absolute address (relative to 0x0) for absolute branch.<br>Relative address (relative to the current location) for relative branch. |

*Note: Each jump step size points to an internal address location of width 16-bits.*

- END

This instruction resets the instruction counter to the starting internal address if Bit 11 is set and generates an interrupt to the host (if Bit 12 is set).

*Note: This instruction does not stop the execution of the PWM channel. The PWM operation can be disabled only through the PWMCS register.*

**Table 11. END instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | Interrupt host | Reset counter | Reserved bits | | | | | | | | | | |

**Table 12. END description**

| Bits | Name | Description |
|---|---|---|
| 12 | Interrupt Host | If this bit is ‘1’ an interrupt is generated to the host. |
| 11 | Reset Counter | If this bit is set to ‘1’, the instruction counter is reset and the output of the pwm is set to level 0. |

- TRIG (Trigger)

This command enables a PWM channel to send a trigger to the other two channels and/or wait for a trigger from other channels to start/resume execution.

This instruction is useful when LEDs have to be triggered in sequence for running displays, for example.

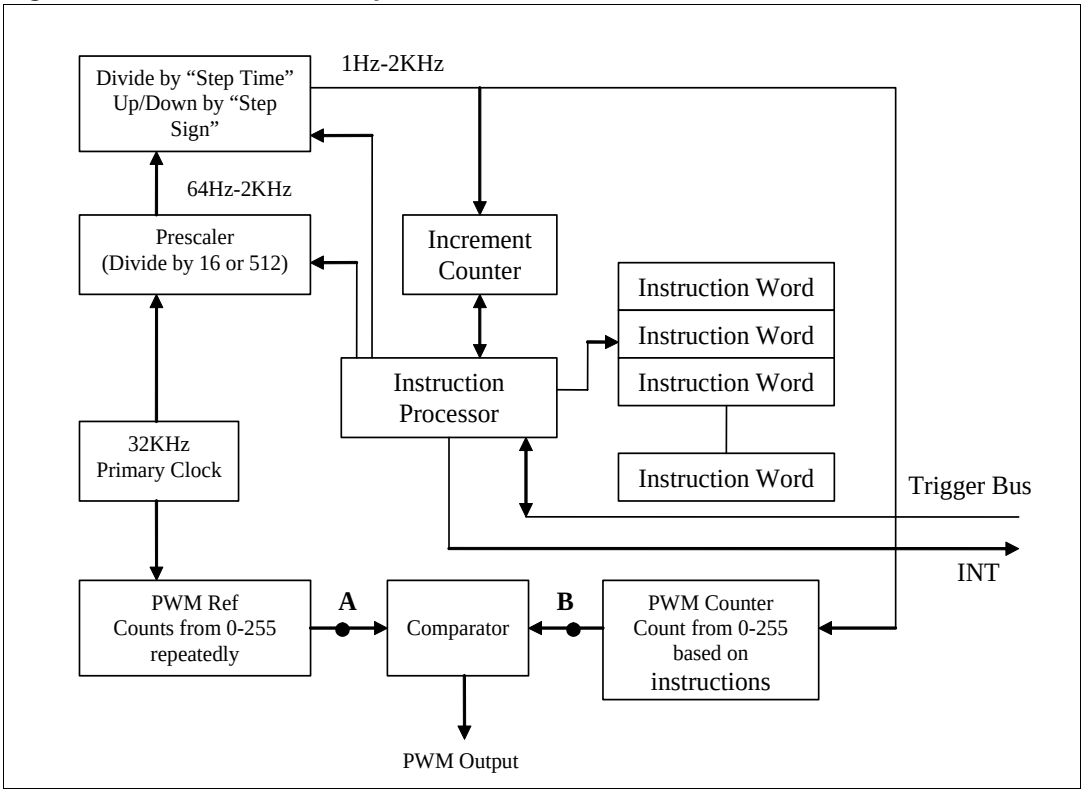
**Table 13. TRIG instruction**

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 0 | 0 | 0 | Wait CH2 | Wait CH1 | Wait CH0 | 0 | 0 | 0 | Send CH2 | Send CH1 | Send CH0 | X |

**Table 14. TRIG description**

| Bits | Name | Description |
|---|---|---|
| 9 | Wait CH2 | Makes the PWM channel-x wait for a trigger from Channel-2 before starting execution. Continues execution if all selected triggers are present. |
| 8 | Wait CH1 | Makes the PWM channel-x wait for a trigger from Channel-1before starting execution. Continues execution if all selected triggers are present. |
| 7 | Wait CH0 | Makes the PWM channel-x wait for a trigger from Channel-0 before starting execution. Continues execution if all selected triggers are present. |
| 3 | Send CH2 | Sends a trigger to CH2. Continues if there is no “Wait for Trigger” in this instruction. |
| 2 | Send CH1 | Sends a trigger to CH1. Continues if there is no “Wait for Trigger” in this instruction. |
| 1 | Send CH0 | Sends a trigger to CH0. Continues if there is no “Wait for Trigger” in this instruction. |
| 0 | X | Don’t care |

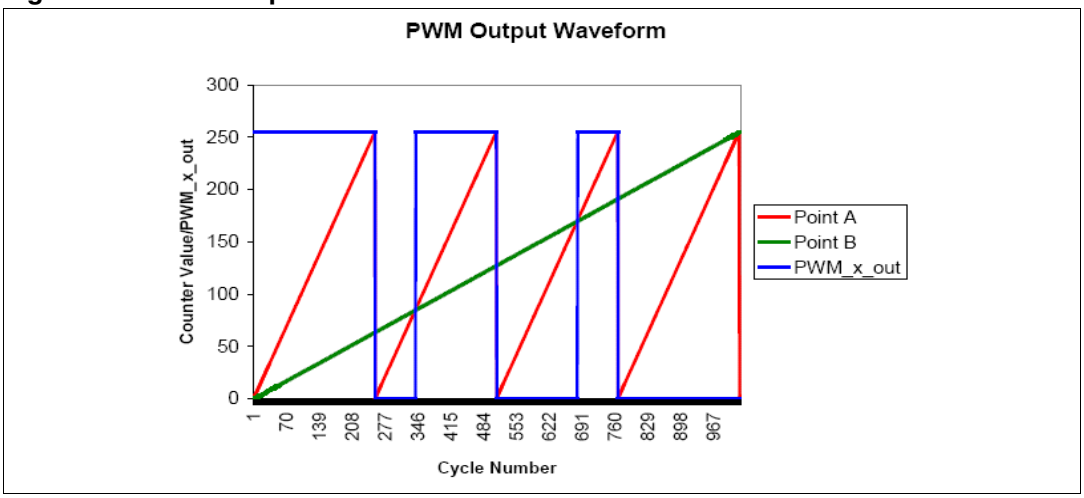
**Figure 2. PWM controller operation**




## 4.1 PWM Controller operation

*Figure 2.* depicts the operation of the STMPE2401 PWM Controller and *Figure 3.* shows the PWM-output for a given RAMP instruction set in the command memory. There is a Reference PWM counter that continuously counts from 0-255 independently (Point A output). The instructions written into the command memory determine the slope of the Ramp at point B. A comparator is used to get the PWM_output based on the A and B point signals as shown in *Figure 3*.

**Figure 3. PWM output waveform**



- PWM_out = HIGH if A > B
- PWM_out = LOW if A < B

For instance, the instruction SMAX will load the counter at B to 255 giving a low at the PWM output. And similarly for SMIN instruction the counter will be set to 0, giving a HIGH at the PWM_output. A suitable filter at the PWM output can be used to get a signal of a specified frequency by varying the increment size in the RAMP instruction.

If there is no succeeding instruction after the RAMP instruction, the PWM counter continues to run until the average PWM output becomes 0V or 1.8 V.

### Example 1:

Prescale = '0' (Divide by 16 => 32 KHz/16 = 2 KHz)

Increment cycles = 126.

Step time = '0' (immediate action)

Two RAMP_UP instructions

Two RAMP_DN instructions

Frequency at the end of these set of instructions can be calculated as:

- Period to complete the instructions = (1/Prescale freq) x Increment cycles per RAMP instruction x step time x No.of. RAMP_UP instructions x No.of RAMP_DN instructions
- Period = (1/2000) x 126 x 1x 2 x 2 = 0.252 secs.

The frequency of the signal after the filter would be 1/0.252 = 3.97 Hz.

### Example 2:

For the PWM output to RAMP for 1 second:

Prescale = '0' (Divide by 16 => 32 KHz/16 = 2 KHz)

Increment cycles = 126

Step time = 16

RAMP instruction time can be calculated as:

Period = (1/prescale freq) x increment cycles x step time per increment

Period = (1/2000) x 126 x 16 = 1 sec.

Therefore, to get a PWM output ramp of 1 sec, we would need 1 RAMP instruction of period 1 secs.

# 5 Registers in the PWM controller

The main system registers to configure the PWM are as given in *Table 15*.

**Table 15. PWM controller registers**

| Address | Register name | Description | Auto-Increment (during Read/Write) |
|---|---|---|---|
| 0x30 | PWMCS | PWM Control and Status register | Yes |
| 0x38 | PWMIC0 | PWM instructions for channel-0 are initialized through this data port. | No |
| 0x39 | PWMIC1 | PWM instructions for channel-1 are initialized through this data port. | No |
| 0x3A | PWMIC2 | PWM instructions for channel-2 are initialized through this data port. | No |

## 5.1 PWM control and status register (PWMCS)

PWMCS register is used to control the three PWM channels and also display the status of an illegal instruction if any are given during the PWM execution. The instructions can be written into the command memory only when the respective PWM channel is in reset state.

**Figure 4. PWMCS register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Reserved | | II2 | II1 | II0 | EN2 | EN1 | EN0 |
| Read/Write | R | R | R | R | R | RW | RW | RW |
| Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16. PWMCS description**

| Bits | Name | Description |
|---|---|---|
| 0 | EN0 | PWM Channel 0 Enable bit.<br>'1' - Enable the PWM Channel 0<br>'0' - Reset the PWM Channel 0. |
| 1 | EN1 | PWM Channel 1 Enable bit.<br>'1' - Enable the PWM Channel 1<br>'0' - Reset the PWM Channel 1. |
| 2 | EN2 | PWM Channel 2 Enable bit.<br>'1' - Enable the PWM Channel 2<br>'0' - Reset the PWM Channel 2. |

**Table 16. PWMCS description (continued)**

| Bits | Name | Description |
|---|---|---|
| 3 | II0 | PWM Invalid Instruction Status bit for PWM Channel 0<br>'0' - No invalid command encountered during the instruction execution.<br>'1' - Invalid command encountered and this puts the PWM Channel 0 into reset state. |
| 4 | II1 | PWM Invalid Instruction Status bit for PWM Channel 1<br>'0' - No invalid command encountered during the instruction execution.<br>'1' - Invalid command encountered and this puts the PWM Channel 1 into reset state. |
| 5 | II2 | PWM Invalid Instruction Status bit for PWM Channel 2<br>'0' - No invalid command encountered during the instruction execution.<br>'1' - Invalid command encountered and this puts the PWM Channel 2 into reset state. |

## 5.2 PWM instruction channel_x (PWMICx)

This PWMICx is the dataport that allows the instructions to be loaded into the PWM channel. The three PWM channels have unique dataport addresses and each have their own 64 x 16 command memory.

The 'x' refers to the particular PWM Instruction channel and takes value 0,1 or 2 corresponding to the PWM channel to be used.

**Figure 5. PWM instruction**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | IB7 | IB6 | IB5 | IB4 | IB3 | IB2 | IB1 | IB0 |
| Read/Write | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17. PWM instruction description**

| Bits | Name | Description |
|---|---|---|
| 7:0 | IB[x] | As an instruction is 16-bits wide, writing the instruction into this 8-bit PWMICx dataport requires two 8-bit data write. The most significant byte of the 16-bit instruction is to be written first, followed by the least significant byte of the instruction. The same order applies to the read operation. |

Using $I^2C$ to load the Command memory:

The $I^2C$ interface is used to load the instructions into the PWM command memory. This is done by writing continuously to the respective PWM dataport. As this dataport address falls in the non-auto increment region, continuous write operation on $I^2C$ writes into the same dataport address.

To access these dataports, the corresponding ENx in the PWMCS register must be set to 0 first to put the PWM channel in reset state.

Only when the PWM channel is in reset state, can the stream of commands be written into its dataport.

## 5.3 Alternate function register (GPAFR_U_msb)

The three PWM channels are the Alternate Functions of GPIO23-21. So to enable the particular PWM channel, the corresponding GPAFR_U_msb bits should be set to '01'.

**Figure 6. GPAFR register [Address: 0x9B]**

GPAFR_U_msb

| Bit | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| | GPIO23 | | GPIO22 | | GPIO21 | | GPIO20 | |
| R/W | RW | RW | RW | RW | RW | RW | RW | RW |

**Table 18. GPAFR description**

| Bits | Name | Description |
|---|---|---|
| 23:16 | AF[x] | GPIO Pin 'x' Alternate Function Select (where x = 23 to 20).<br>'00' - The corresponding GPIO pin (GPIO[x]) is configured to Primary Function.<br>'01' - The corresponding GPIO pin (GPIO[x]) is configured to Alternate Function 1.<br>'10' - The corresponding GPIO pin (GPIO[x]) is configured to Alternate Function 2.<br>'11' - The corresponding GPIO pin (GPIO[x]) is configured to Alternate Function 3. |

## 5.4 Interrupt control register (ICR)

ICR register is used to configure the Interrupt Controller. It has a global enable interrupt mask bit (IC0) that controls the interruption to the host. This bit should be set to '1' to enable interrupts to the host. The type of interrupt and polarity can be set with the IC1 and IC2 bits.

**Figure 7. ICR Register [ICR_lsb Address: 0x11]**

| | ICR_msb | | | | | | | | | | | | | ICR_lsb | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | | | | | | | | | | | | IC2 | IC1 | IC0 |
| R/W | R | R | R | R | R | R | R | R | R | R | R | R | R | RW | RW | RW |
| Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 19. ICR description**

| Bits | Name | Description |
|---|---|---|
| 0 | IC[0] | Global Interrupt Mask bit<br>When this bit is set to '1', it will allow interruption to the host. If it is set to '0', it disables all interruption to the host. Writing to this bit does not affect the IER value. |
| 1 | IC[1] | Output Interrupt Type<br>'0' = Level interrupt<br>'1' = Edge interrupt |
| 2 | IC[2] | Output Interrupt Polarity<br>'0' = Active Low / Falling Edge<br>'1' = Active High / Rising Edge |

## 5.5 Interrupt enable mask register (IER)

IER register should be used to enable the interruption from a particular interrupt source to the host. In this case the PWM controller interrupt mask (IE5, IE6 and IE7) should be set to '1' to detect the END interrupt for each PWM channel separately.

**Figure 8. IER register [IER_lsb Address: 0x13]**

| | IER_msb | | | | | | | IER_lsb | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | | | | | | IE8 | IE7 | IE6 | IE5 | IE4 | IE3 | IE2 | IE1 | IE0 |
| R/W | R | R | R | R | R | R | R | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 20. IER description**

| Bits | Name | Description |
|---|---|---|
| 8:0 | IE[x] | Interrupt Enable Mask (where x = 8 to 0)<br>IE0 = Wake-up Interrupt Mask<br>IE1 = Keypad Controller Interrupt Mask<br>IE2 = Keypad Controller FIFO Overflow Interrupt Mask<br>IE3 = Rotator Controller Interrupt Mask<br>IE4 = Rotator Controller Buffer Overflow Interrupt Mask<br>IE5 = PWM Channel 0 Interrupt Mask<br>IE6 = PWM Channel 1 Interrupt Mask<br>IE7 = PWM Channel 2 Interrupt Mask<br>IE8 = GPIO Controller Interrupt Mask<br>Writing a '1' to the IE[x] bit enables the interruption to the host. |

## 5.6 Interrupt status register (ISR)

ISR register monitors the status of the interruption from a particular interrupt source to the host. Regardless of whether the IER bits are enabled, the corresponding ISR bits are updated. Writing a '1' clears the corresponding interrupt.

**Figure 9. ISR register [ISR_lsb Address: 0x15]**

| | ISR_msb | | | | | | | ISR_lsb | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | | | | | | IS8 | IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 |
| R/W | R | R | R | R | R | R | R | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 21. ISR description**

| Bits | Name | Description |
|---|---|---|
| 8:0 | IS[x] | Interrupt Status (where x = 8 to 0)<br>Read: IS0 = Wake-up Interrupt Status<br>IS1 = Keypad Controller Interrupt Status<br>IS2 = Keypad Controller FIFO Overflow Interrupt Status<br>IS3 = Rotator Controller Interrupt Status<br>IS4 = Rotator Controller Buffer Overflow Interrupt Status<br>IS5 = PWM Channel 0 Interrupt Status<br>IS6 = PWM Channel 1 Interrupt Status<br>IS7= PWM Channel 2 Interrupt Status<br>IS8 = GPIO Controller Interrupt Status<br>Write:<br>A write to a IS[x] bit with a value of '1' clears the interrupt and a write with a value of '0' has no effect on the IS[x] bit. |

## 5.7 Programming sequence - example

Before enabling the PWM controller operation, proper setup should be done by configuring the clock, input and output ports involved. This is achieved by programming the following registers.

- The Enable_PWM bit in the SYSCON register should be set to '1' to provide the required clock signals.
- The GPIO alternate function register (GPAFR_U_msb) bits for the respective PWM channel should be set to alternate function '01'. The unused PWM outputs can be used as GPIO and in this case, the alternate function should be set to '00'.
- Load the instructions into the PWM channel x by writing into the corresponding PWMICx. The ENx of the PWMCS register should be kept '0' while loading. By default, it has a value of '0'.
- The PWM channel x has a 64-word depth (16-bit width). Any instructions of size less than or equal to 64 words can be loaded into the channel. Any attempt to load beyond

64 words results in internal address pointer roll-over (0x1f →0x00) and the excess instructions over-write the first address location of the channel and onwards.

- After loading the command memory, the instruction pointer should be reset to the starting location (0x0) by performing a read on some other register (eg. CHIP_ID register).
- Enable the corresponding interrupt mask bit to allow interruption to the host in IER_lsb register.
- After the instructions are loaded in, the PWM channel x can be enabled by writing a '1' to the ENx bit in the PWMCS register.

The pseudo-code for configuring the PWM channels is given below.

The code below triggers each PWM channel successively and the LEDs connected to each PWM channel blinks successively one after another to create a rolling effect and this cycle is repeated indefinitely.

```
/* Function prototypes */
write_data(reg_addr, number_words, data_source)

read_reg(reg_addr);


/* Address initialization */

PWM0_Addr = 0x38;

PWM1_Addr = 0x39;

PWM2_Addr = 0x3A;


/* Instructions to be downloaded into command memory in 8-bit
words*/

inst_pwm0[128];

inst_pwm1[128];

inst_pwm3[128];



/* PWM-Ch0 instructions: SMAX RAMP(DN) RAMP(UP) BRANCH SMIN TRIG-
CH1 WAIT-CH2 GTS */

 inst_pwm0[0]=0x0; //SMAX - MSB

 inst_pwm0[1]=0x7F; //SMAX - LSB

 inst_pwm0[2]=0x0; //RAMP (DN) - MSB

 inst_pwm0[3]=0xBF; //RAMP (DN) - LSB

 inst_pwm0[4]=0x0; //RAMP (DN) - MSB

 inst_pwm0[5]=0xBF; //RAMP (DN) - LSB

 inst_pwm0[6]=0x0; //RAMP (DN) - MSB
```

```
 inst_pwm0[7]=0xBF; //RAMP (DN) - LSB
 inst_pwm0[8]=0x0; //RAMP (DN) - MSB
 inst_pwm0[9]=0xBF; //RAMP (DN) - LSB
 inst_pwm0[10]=0x0; //RAMP (UP) - MSB
 inst_pwm0[11]=0x3F; //RAMP (UP) - LSB
 inst_pwm0[12]=0x0; //RAMP (UP) - MSB
 inst_pwm0[13]=0x3F; //RAMP (UP) - LSB
 inst_pwm0[14]=0x00; //RAMP(UP) - MSB
 inst_pwm0[15]=0x3F; //RAMP(UP) - LSB
 inst_pwm0[16]=0x00; //RAMP(UP) - MSB
 inst_pwm0[17]=0x3F; //RAMP(UP) - LSB
 inst_pwm0[18]=0xA7; //Loop 16 times with relative address 0x8 - MSB
 inst_pwm0[19]=0xC8; //Loop 16 times with relative address 0x8 - LSB
 inst_pwm0[20]=0x0; //SMIN - MSB
 inst_pwm0[21]=0xFF; //SMIN - LSB
 inst_pwm0[22]=0xE0; //Send CH1 Trigger - MSB
 inst_pwm0[23]=0x04; //Send CH1 Trigger - LSB
 inst_pwm0[24]=0xE2; //Wait CH2 Trigger - MSB
 inst_pwm0[25]=0x00; //Wait CH2 Trigger - LSB
 inst_pwm0[26]=0x00; //GTS - MSB
 inst_pwm0[27]=0x00; //GTS - LSB


/* PWM-Ch1 instructions: WAIT-CH0 SMAX RAMP(DN) RAMP(UP) BRANCH SMIN
TRIG-CH2 GTS */
inst_pwm1[0]=0xE0; //Wait for CH0 Trigger - MSB
 inst_pwm1[1]=0x80; //Wait for CH0 Trigger - LSB
 inst_pwm1[2]=0x0; //SMAX - MSB
 inst_pwm1[3]=0x7F; //SMAX - LSB
 inst_pwm1[4]=0x0; //RAMP (DN) - MSB
 inst_pwm1[5]=0xBF; //RAMP (DN) - LSB
 inst_pwm1[6]=0x0; //RAMP (DN) - MSB
 inst_pwm1[7]=0xBF; //RAMP (DN) - LSB
 inst_pwm1[8]=0x0; //RAMP (DN) - MSB
 inst_pwm1[9]=0xBF; //RAMP (DN) - LSB
 inst_pwm1[10]=0x0; //RAMP (DN) - MSB
```

```
 inst_pwm1[11]=0xBF; //RAMP (DN) - LSB
 inst_pwm1[12]=0x0; //RAMP (UP) - MSB
 inst_pwm1[13]=0x3F; //RAMP (UP) - LSB
 inst_pwm1[14]=0x0; //RAMP (UP) - MSB
 inst_pwm1[15]=0x3F; //RAMP (UP) - LSB
 inst_pwm1[16]=0x0; //RAMP (UP) - MSB
 inst_pwm1[17]=0x3F; //RAMP (UP) - LSB
 inst_pwm1[18]=0x00; //RAMP(UP) - MSB
 inst_pwm1[19]=0x3F; //RAMP(UP) - LSB
 inst_pwm1[20]=0xAF; //Loop 16 times starting from address 0x2 - MSB
 inst_pwm1[21]=0xC8; //Loop 16 times starting from address 0x2 - LSB
 inst_pwm1[22]=0x0;  //SMIN - MSB
 inst_pwm1[23]=0xFF; //SMIN - LSB
 inst_pwm1[24]=0xE0; //Send CH2 Trigger - MSB
 inst_pwm1[25]=0x08; //Send CH2 Trigger - LSB
 inst_pwm1[26]=0x0; //GTS - MSB
 inst_pwm1[27]=0x0; //GTS - LSB


/* PWM-Ch2 instructions: WAIT-CH1 SMAX RAMP(DN) RAMP(UP) BRANCH SMIN
TRIG-CH0 GTS */


 inst_pwm2[0]=0xE1; //Wait for CH1 Trigger - MSB
 inst_pwm2[1]=0x00; //Wait for CH1 Trigger - LSB
 inst_pwm2[2]=0x0; //SMAX - MSB
 inst_pwm2[3]=0x7F; //SMAX - LSB
 inst_pwm2[4]=0x0; //RAMP (DN) - MSB
 inst_pwm2[5]=0xBF; //RAMP (DN) - LSB
 inst_pwm2[6]=0x0; //RAMP (DN) - MSB
 inst_pwm2[7]=0xBF; //RAMP (DN) - LSB
 inst_pwm2[8]=0x0; //RAMP (DN) - MSB
 inst_pwm2[9]=0xBF; //RAMP (DN) - LSB
 inst_pwm2[10]=0x0; //RAMP (DN) - MSB
 inst_pwm2[11]=0xBF; //RAMP (DN) - LSB
 inst_pwm2[12]=0x0; //RAMP (UP) - MSB
 inst_pwm2[13]=0x3F; //RAMP (UP) - LSB
```

```
 inst_pwm2[14]=0x0; //RAMP (UP) - MSB
 inst_pwm2[15]=0x3F; //RAMP (UP) - LSB
 inst_pwm2[16]=0x00; //RAMP(UP) - MSB
 inst_pwm2[17]=0x3F; //RAMP(UP) - LSB
 inst_pwm2[18]=0x00; //RAMP(UP) - MSB
 inst_pwm2[19]=0x3F; //RAMP(UP) - LSB
 inst_pwm2[20]=0xB7; //Loop 16 times starting from address 0x2 - MSB
 inst_pwm2[21]=0xC8; //Loop 16 times starting from address 0x2 - LSB
 inst_pwm2[22]=0x0;  //SMIN - MSB
 inst_pwm2[23]=0xFF; //SMIN - LSB
 inst_pwm2[24]=0xE0; //Send CH0 Trigger - MSB
 inst_pwm2[25]=0x02; //Send CH0 Trigger - LSB
 inst_pwm2[26]=0x0; //GTS - MSB
 inst_pwm2[27]=0x0; //GTS - LSB

/* Load instructions into the PWMIC_x */

write_data(PWM0_Addr, 28, inst_pwm0);  //Write into PWM-0 memory
read_reg(CHIP_ID_ADDR);                //To reset the internal
pointer to 0x0

write_data(PWM1_Addr, 28, inst_pwm1);  //Write into PWM-1 register
rad_reg(CHIP_ID_ADDR);   //To reset the internal pointer to 0x0

write_data(PWM2_Addr, 28, inst_pwm2);  //Write into PWM-3 register
read_reg(CHIP_ID_ADDR);   //To reset the internal pointer to 0x0

/* Configure Alternate Function to PWM mode for all 3 PWM channels
*/
write_reg(0x9B,0x54);

/* Enable all three PWM Channels */
write_reg(0x30,0x7);

/* Disable PWM Channels and change Alternate functions back to GPIO
*/
```

```
write_reg(0x30,0x0);
write_reg(0x9B,0x0);
```

To further reduce power consumption, the clock signals to PWM module can be cut off by setting the Enable_PWM bit in the SYSCON register to '0'.

# 6 Conclusion

The PWM controller integrated into the STMPE2401 packs a powerful instruction set which can be used to control LED brightness, Blinking/display patterns and save CPU resources by controlling the display panel independently of the CPU with pre-loaded instructions. The interface to the CPU is also simple through just 2 wires of the $I^2C$ interface. This way STMPE2401 serves as a powerful device to save power and CPU resources in digital engines.

# 7 Reference

- AN2422: STMPE2401 GPIO port expander Hardware Interface guide.

# 8 Revision history

**Table 22. Revision history**

| Date | Revision | Changes |
|---|---|---|
| 04-Apr-2007 | 1 | Initial release. |