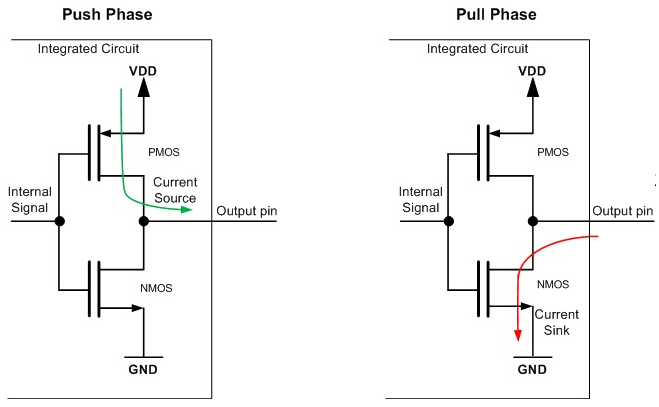
# **Push-Pull vs Open Drain**

*(Content is taken from open Internet)*

**Push-Pull**

Microcontrollers use pins for interfacing with the outside world. In general, the pins are the physical points on the package of an integrated circuit (IC)where a connection can be made to the printed circuit board.  Behind each pin (inside the IC) there is a special circuitry used for driving it. This circuitry (usually called a pad) can be configured to allow the pin to interface with different types of digital and analog circuits.

Push-pull is the most common output configuration. Just as its name suggests, push-pull output is capable of driving two output levels. One is pull to ground (pull/sink current from the load) and the other is push to power supply voltage (push/source current to the load). The push-pull output can be implemented using a pair of switches. The practical implementation in an integrated circuit involves the use of transistors.



* **Push phase** – When the Internal Signal connected to the gates of the transistors (see the figure above) is set to a low logic level (logic 0), the PMOS transistor is activated and current flows through it from the VDD to the output pin. NMOS transistor is inactive (open) and not conducting.
* **Pull phase** – When the Internal Signal connected to the gates of the transistors is set to a high logic level (logic 1), the NMOS transistor is activated (closed) and current starts to flow through it from the output pin to the GND. At the same time, the PMOS transistor is inactive (open) and is not conducting current.

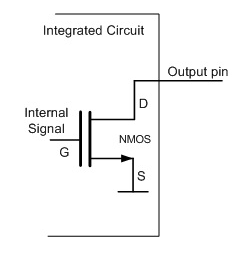
This type of output doesn’t allow connecting multiple devices together in a bus configuration, like the open drain output. Push-pull configuration is most commonly used in interfaces that have unidirectional lines (transmission on the line is only in a single direction – SPI, UART etc.).

As push-pull outputs are constantly driven (high or low), they provide better performance when it comes to the slopes of the generated output digital signals.

**Open Drain**

In open drain configuration, the logic behind the pin can drive it only to ground (logic 0). The other possible state is **high impedance** **(Hi-Z)**. The implementation involves the use of a single transistor. If its drain terminal is open (the device is off) the pin is left floating to Hi-Z state. Driving it to high logic level requires the use of an additional circuit or component. In most cases, an external pull-up resistor is used (there are microcontrollers that provide internal pull-up resistors for open drain configurations).

On the picture below we can see an open drain output. It is implemented using an N-channel MOS transistor that pulls the output pin to ground when the transistor is on and leaves it floating when the transistor is off.



Open drain outputs are most commonly used in communication interfaces where multiple devices are connected on the same line (e.g I2C, One-Wire etc.). When all of the outputs of the devices connected to the line are in Hi-Z state, the line is driven to a default logic 1 level by a pull-up. Any device can pull the line to logic 0 using its open drain output and all devices can see this level.

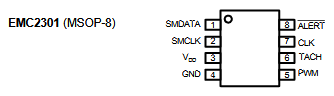
**Summary**

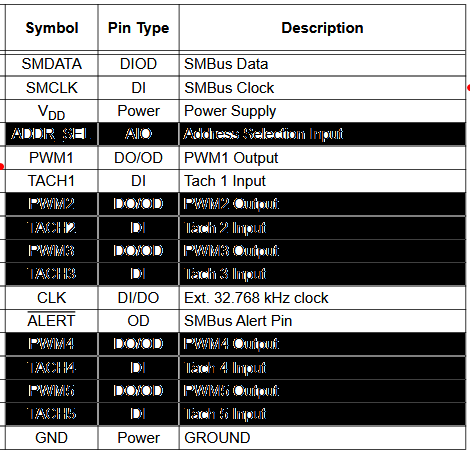
* Push-pull output is best suited for communication interfaces that have single direction lines (e.g SPI, UART etc.). Open drain is commonly used for bidirectional single line communication interfaces, where more than two devices are connected on the same line(e.g I2C, One-Wire etc.)
* Open drain output has higher power consumption during active transfers due to the pull-up resistors that are used.
* In general, the push-pull output has faster slopes than the open drain output.

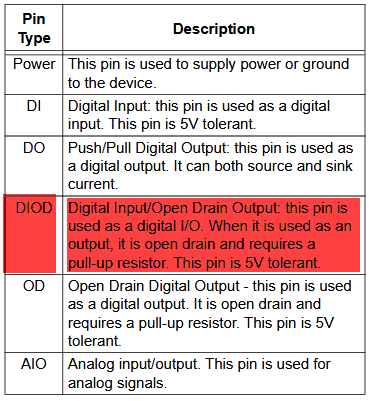
# **EMC2301 PWM Controller (CM4 IO and Seaberry Boards)**

*(Content is taken from EMC2301 Data Sheet)*

The EMC2301 is an SMBus compliant fan controller with a single controlled PWM fan driver. Fan driver is controlled by a programmable frequency PWM driver and Fan Speed Control algorithm that operates in either a closed loop fashion or as a directly PWM-controlled device.





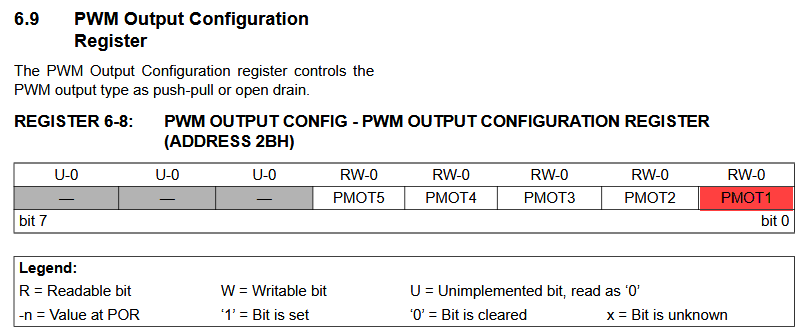


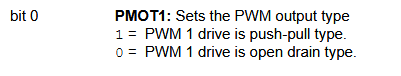
**PWM1** - This is the output control signal from the Fan Speed Control 1 algorithms. **This can be configured as a push-pull PWM output** or an open drain PWM output.

**TACH1** - **Open drain** tachometer input to Fan Speed Control 1 algorithms for feedback. **Requires external pull-up resistor**.

**IMORTANT NOTE:** All data above and below is taken directly from EMC2301 Data Sheet and as you probably already noticed there is an error/discrepancy in pin description table that specifies TACH1 as DI (Digital Input) and detailed pin description later on stating that it is OD (Open Drain). **TACH1 pin is Open Drain**.

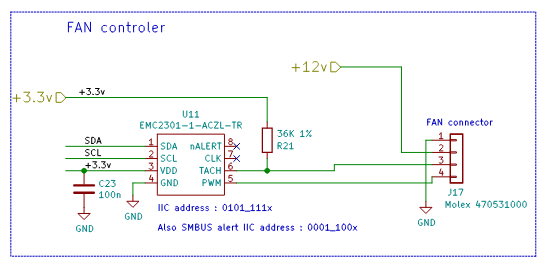
PWM1 output driver can be configured to operate as an open-drain (default) or push-pull driver and can be configured with normal or inverse polarity.





# **Raspberry Pi CM4 IO and Seaberry Boards**

Here is excerpt from CM4 IO board schematic (CM4 Data Sheet – by the very end)

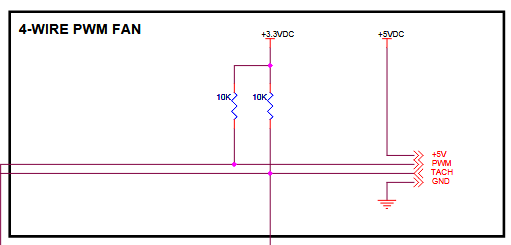


As you can see there are 2 potential problems.

**Problem #1** – R21 resistor value 36 kOhm presents a very weak pull-up that potentially leads to incorrect reading. Usually in such applications with 3.3V rail pull-up supposed to be in 5 kOhm to 10 kOhm range. Soldering an additional 10 kOhm resistor between TACH pin at FAN connector and closest 3.3V rail will implement a sufficient pull-up strength and guarantee correct reading value(s) from Tachometer registers 0x3E (High Byte) and 0x3F (Low Byte).

**Problem #2** – as you already noticed there is no any pull-up resistor at Tachometer connection between EMC2301 and fan connector, which in turn implies that the only way to make it work is to configure 0x2B register as PWM1 push-pull (set bit #0 to 1). Alternatively user can solder an additional 10 kOhm resistor between PWM pin at FAN connector and closest 3.3V rail and leave 0x2B register at its default value 0x00 (default value of 0x2B register on power up is 0x00).

Here is excerpt from Microchip EMC2301 reference design:



It seems that every software utility written by community folks (and we tried a number of them available from GitHub) “blindly” assumes Open Drain function for PWM output from EMC2301 and does not perform any manipulation with 0x2B register to match actual implementation in hardware. It will be wise for SW owners of respective utilities to implement an additional option that defines Open Drain vs Push-Pull operation of PWM driver such as users can distinguish between different boards and HW implementations.

# **Seaberry Pi Board**

Our current implementation of FAN circuitry does follow CM4 IO design except that we have 10 kOhm resistor pull-up at Tachometer line vs 36 kOhm on CM4 IO board. Our PWM line does not have pull-up (same as CM4 IO board) and requires correct 0x2B register setting in order for PWM function correctly. We may re-consider this approach with next board re-spin and implement pull-up at PWM line just to follow Microchip reference design.

Below is quick and dirty scrip to try with your current CM4 IO board in order to confirm that PWM works as it supposed to.

echo "Seaberry and CM4 IO Fan Control"

echo "Initialize EMC2301 chip"

sudo /usr/sbin/i2cset -y 10 0x2f 0x33 0x40

sudo /usr/sbin/i2cset -y 10 0x2f 0x32 0x00

sudo /usr/sbin/i2cset -y 10 0x2f 0x2b 0x01

echo "Turn off fan"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x00

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 50 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x32

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 100 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x64

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 150 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x96

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 200 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0xc8

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 250 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0xFA

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 200 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0xc8

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 150 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x96

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 100 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x64

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Set fan speed to 50 (out of 255)"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x32

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f

echo "Turn off fan"

sudo /usr/sbin/i2cset -y 10 0x02f 0x30 0x00

/usr/bin/sleep 5

echo "Tachometer Reading: "

sudo i2cget -y 10 0x2f 0x3e

sudo i2cget -y 10 0x2f 0x3f