Manual I2C configuration of MPQ4210

martes, 11 de junio de 2024 10:38

MPQ4210 register explanation

Register Map

Address	Register	Type	D7	D6	D5	D4	D3	D2	D1	D0	Reset State
0x00	REF_LSB	R/W	-	-	-	-	-	VREF_L			0000 0100
0x01	REF_MSB	R/W	VREF_H								0011 1110
0x02	Control 1	R/W	SR		DISCHG	Dither	PNG_Latch	Reserved [9]	GO_BIT	ENPWR	0100 0000
0x03	Control 2	R/W	FSW		-	BB_FSW	OCP_MODE		OVP_MODE		1000 0101
0x04	ILIM	R/W	-	-	-	-	Reserved	ILIM			0000 1001
0x05	Interrupt status	R/W	-	-	-	OTP	-	OVP	OCP	PNG	0000 0000
0x06	Interrupt mask	R/W	-	-	-	M_OTP	-	M_OVP	M_OCP	M_PNG	0000 0001

Here you can see the registers which are available on the MPQ4210. Now, what we've got to take into consideration is the following: not every feature has its own register, but rather, they often share the same register. Which means upon changing, let's say, the OCP_MODE bits, we can alter the FSW by mistake or the OVP mode, if the I2C write operation is not executed carefully. How so?

By default we'll start on CONTROL2 = 0b1000 0101, which means that:

Register Name: Control 2, 0x03, *Read/Write*

Name	Bits	Default Value	Description
FSW	D[7:6]	10	Switching frequency setting bit. Writable during both ENPWR = 0 and ENPWR = 1 conditions. The switching frequency changes smoothly after I²C writes these bits.
			FSW bits: 00 / 01 / 10 / 11
			Frequency: 200kHz / 300kHz / 400kHz / 600kHz
BB_FSW	D[4]	0	Buck-boost region switching frequency set bit. See Figure 4. 1 = higher switching frequency in buck-boost region. The higher Buck-Boost switching frequency is 62.5% of the base switching frequency. 0 = lower switching frequency in buck-boost region. The lowers Buck-Boost switching frequency is 37.5% of the base switching frequency.

OCP_MODE	D[3:2]	01	Set OCP protection mode after triggering the cycle-by-cycle switching current limit (valley current limit in buck, or peak current limit in boost). 00 = No hiccup or latch-off protection. Inductor current is limited by cycle-by-cycle current limit 01 = Hiccup protection after triggering the switching current limit and FB < 60% of V_{REF}. Off-period is controlled by SS discharge 10 = Latch-off protection. Must re-power or re-enable for the IC to restart 11 = Reserved
OVP_MODE	D[1:0]	01	Set OVP protection mode after triggering the threshold at 127% of V_{REF}. 00 = No protection after OVP, V_{OUT} is regulated by COMP. No discharge after OVP 01 = Discharge V_{OUT} through an internal resistor and stop switching when V_{FB} exceeds 127% of V_{REF}. Recover when V_{FB} drops to 111% of V_{REF} 10 = Latch-off protection. No discharge after OVP 11 = Reserved

- FSW = 10; the switching frequency is set to 400kHz
- BB_FSW = 0; higher switching frequency for buckboost region is not enabled.
- OCP_MODE = 01; Hiccup protection on OCP
- OVP_MODE = 01; Vout is discharged on OVP condition.

Now, let's say I want to set OCP_MODE to latch-off protection? Which value should I be loading to the register?

Seeing as OCP_MODE occupies the third and fourth bit of the CONTRO2 register it would be logic to think I'd need to write 10 into these bit section as follows:

CONTROL2 <- 0bXXXX 10XX

Now, if we were to convert these Xs into 0s the value we'd have to load would be 0b0000 1000 = 0x08

But what happens if we were to execute the following command

Cmd> i2cset -y 1 0x66 0x03 0x08

With this, not only have we successfully set our OCP_MODE but we have messed up both the FSW and the OVP_MODE

Given that we loaded 0b0000 1000 to the register and that the register is composed as follows:

CONTROL2: FSW|-|BB_FSW|OCP_MODE|OVP_MODE

By writing 0b0000 1000 to the Control2 register

CONTROL2: 0000 1000

We also did set the following

- FSW=00; switching frequency is set to 200kHz
- OVP_MODE=00; no protection after OVP

The obvious solution would be to also write these values to the value they were previously set as follows

CONTROL2: 1000 1001 -> 1000 1001

Thus, the value we load into the CONTROL2 register is no longer 0x08, but 0x89 instead. Now, there's an issue with this approach, we were only able to execute it because we knew the FSW and OVP_MODE values beforehand. **But what happens if we don't?**

I2C masked writing

With the **i2cset** command we are given a masking option, which can help on these scenarios.

```
SYNOPSIS
       i2cset [-f] [-y] [-m mask] [-r] [-a] i2cbus chip-address data-address [value] ...  [mode]
```

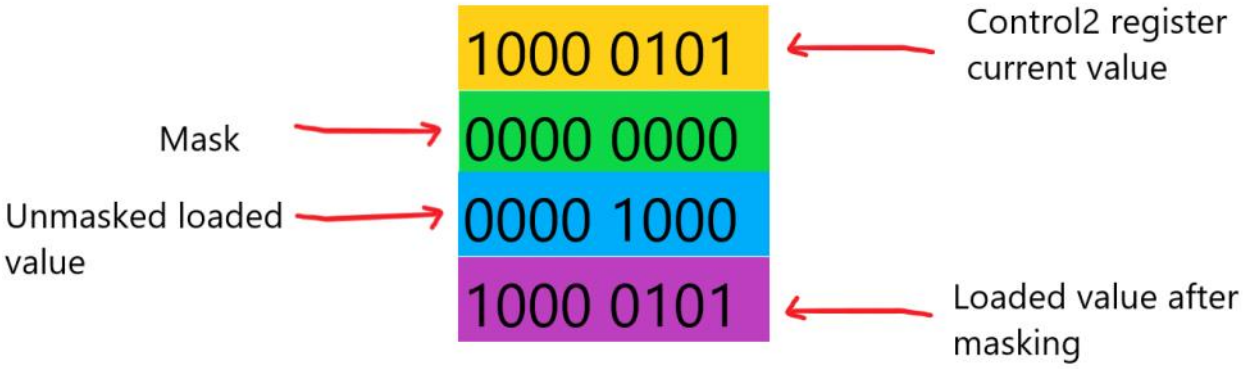
From the i2cset manual we know it works as follows

```
-m mask
       The mask parameter, if specified, describes which bits of value will be actually  written  to  data-ad-
       dress.  Bits  set to 1 in the mask are taken from value, while bits set to 0 will be read from data-ad-
       dress and thus preserved by the operation. Please note that this parameter assumes that  the  read  and
       write  operations  for the specified mode are symmetrical for the device you are accessing. This may or
       may not be the case, as neither I2C nor SMBus guarantees this.
```

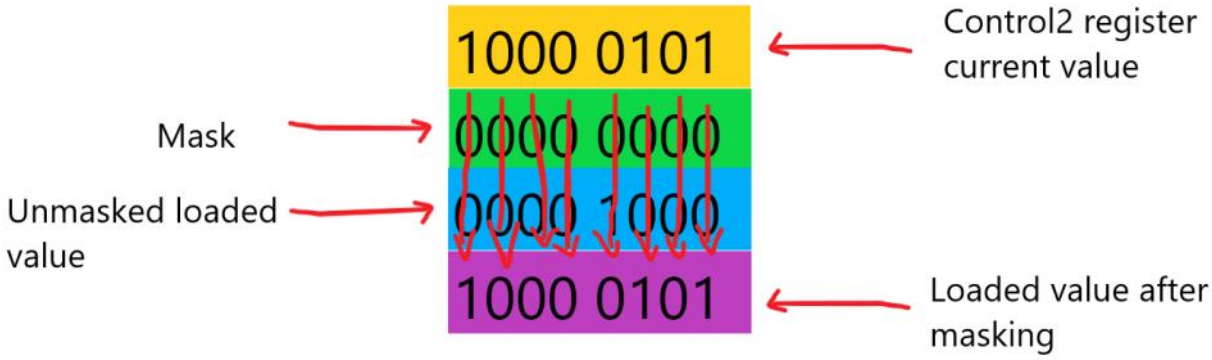
But what does this even mean? The easiest way to understand this would be to see what a 0x00 and a 0xFF mask would do

Mask is 0x00 example

Let's say we attempt to do the same write operation as the first we had proposed, which is to write 0x08 into the CONTRO2 register.

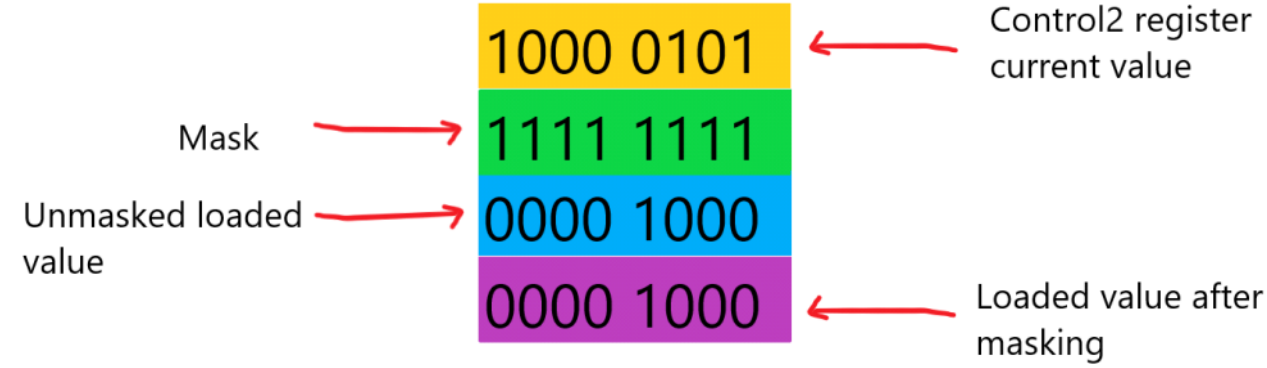


Here none of the bits we wanted to write were taken into account due to the mask being set to all zeros

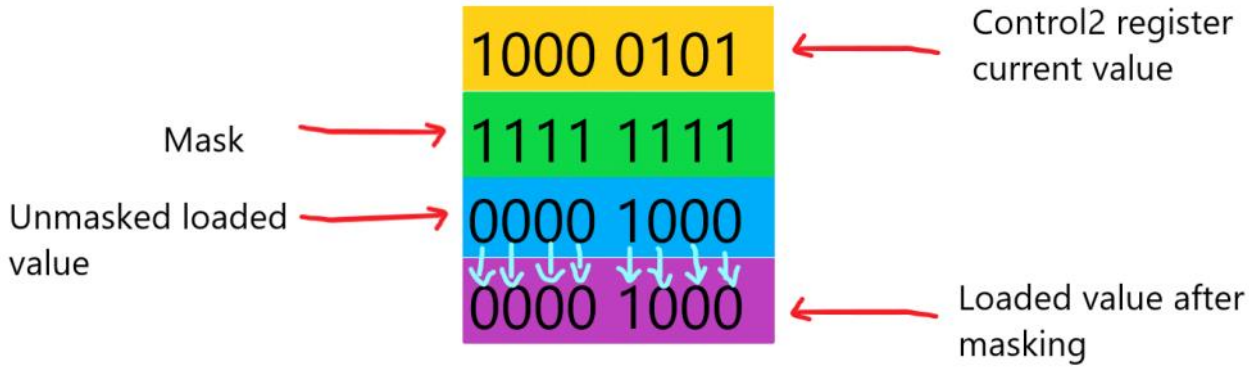


Mask is 0xFF example

Let's say we attempt to do the same write operation as the first we had proposed, which is to write 0x08 into the CONTRO2 register.



Here, none of the bits from the CONTROL2 register were kept, as our masking specified all of the bits from the loaded value were actually being loaded to the register

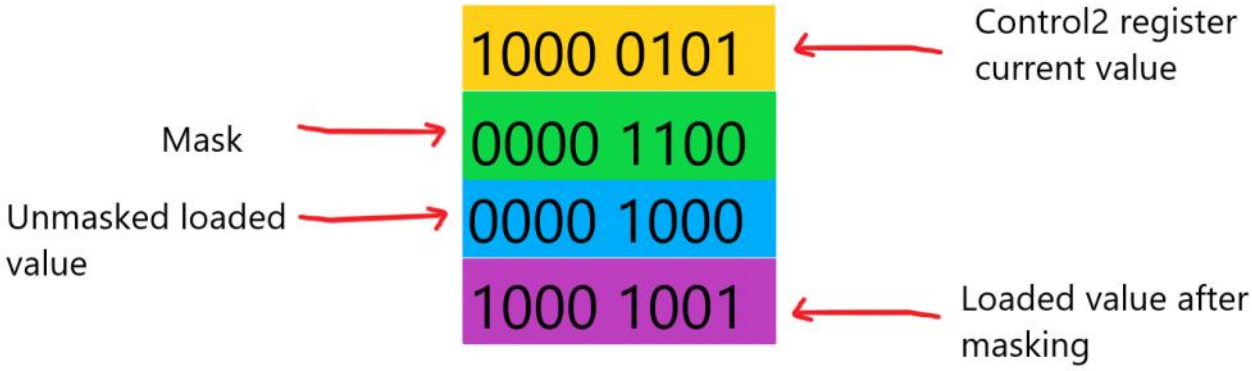


Proper masking for this example

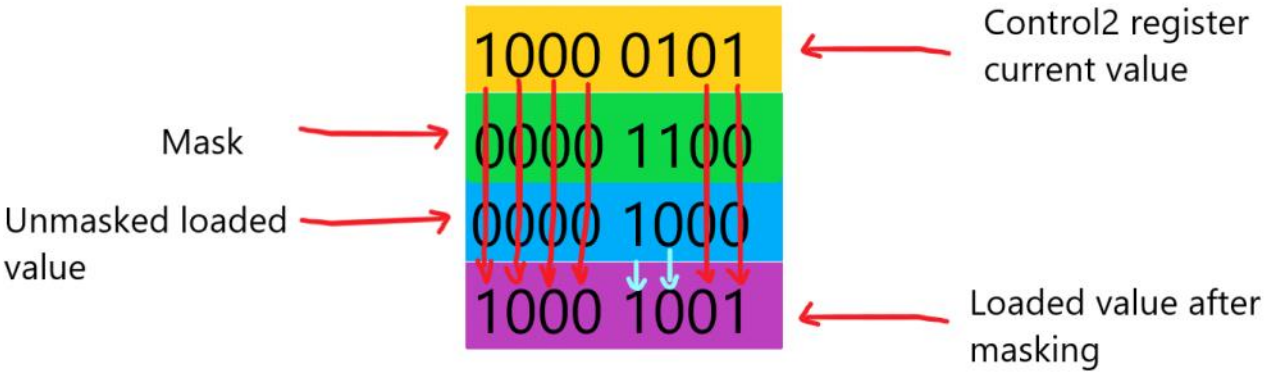
As we saw, none of the latter were appropriate for us, as with a 0x00 mask we didn't change anything, and with a 0xFF mask we messed up the bits which were not related to what we were trying to configure. Though, with proper masking we can achieve what we're looking for. Given that we only want to modify the third and fourth bit of this register, and that only the bits set in our mask are loaded to the register while the other ones are kept from their original value. A proper mask for this example would be the following:

$Mask = 0b0000\ 1100$

Now what we would see is the following



As from our mask, the i2cset would set the written value as follows:



Vref setting

In order to set a new reference voltage in our MPQ4210 evaluation board we have to follow the following steps

1. Get Vref with equation:

   (1) $V_{REF} = \frac{R_2}{R_1 + R_2} V_{out}$

2. Multiply this value by 1000 to get the value to be set in the VREF registers
3. Do the decimal to binary conversion from that number
   a. The three less significant bits have to be loaded to REF_LSB register

      You can load the value with the following command: i2cset -y <I2CBus number> <Device address> 0x00 <hexadecimal representation of the three least significant bits>

   b. The eight bits remaining have to be loaded to REF_MSB register

      You can load the value with the following command: i2cset -y <I2CBus number> <Device address> 0x01 <hexadecimal representation of the eight most significant bits>
4. Set the GO_BIT for the Vref to be reached by the MPQ device you can do this by loading.

NOTE: This process implies that you already have an output previously set by the MP software, as it does not modify the enable bit and the ENPWR to activate or deactivate the output and/or initialize the IC.

Only steps 3 and 4 represent read and write I2C operations with the Rpi you'd have to execute on the cmd. The ones listed on step 3 do not need any masking. While we do have to do proper masking to execute the fourth step, so to not mess up the CONTROL1 register on doing so.

0x02	Control 1	R/W	SR	DISCHG	Dither	PNG_Latch	Reserved [9]	GO_BIT	ENPWR	0100 0000

We want to set the GO_BIT, which is the second bit of the CONTROL1 register. So we need to write 0b0000 0010 = 0x02 into the control register with a masking that allows us not to modify the first bit and from the third to the eight bit. In this particular case our mask would also match the value we want to load, though this will not always be the case, and is given as follows

$GO_{BIT_{MASK}} = 0b0000\ 0010$

With all of this cleared up we can proceed to writing our cmd commands.

1. Load LSB of VREF

   i2cset -y <I2CBus number> <Device address> <Hexadecimal representation of the REF_LSB register = 0x00> <hexadecimal representation of the three least significant bits>

2. Load MSB of VREF

   i2cset -y <I2CBus number> <Device address> <Hexadecimal representation of the REF_MSB register = 0x01> <hexadecimal representation of the eight most significant bits>

3. Set the GO_BIT of the control register

   i2cset -y -m <hexadecimal representation of your mask = 0x02 in this case> <I2CBus number> <Device address> <Hexadecimal value of the CONTROL1 register = 0x02> <Hexadecimal representation of the GO_BIT = 1 value, in this case it's 0x02>