AN11593

How to design in and program the PCA9641 I^{2}C arbiter

Rev. 1 — 23 October 2014 Application note

Document information

Info	Content
Keywords	Fast-mode Plus (Fm+) I2C-bus, 2-to-1 I2C-bus multiplexer, recover stuck I2C-bus, I2C-bus collision avoidance, I2C.
Abstract	The PCA9641 is highly integrated and smart design for 2-1 I^{2}C-bus multiplexer. It is used in a system that needs two masters sharing the same slave devices. The internal switch is programmed by the masters but it will not switch in a middle of the task once owned by the master. The PCA9641 sends an interrupt to the master requesting the bus when the downstream bus is available. If the down-stream bus is hung, the initial/recovery function clears the downstream bus and sends status to both masters. Low voltage I^{2}C masters can communicate with higher voltage level of slave devices

Revision history

Rev	Date	Description
1	20141023	Initial version

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

PCA9641 arbiter is a smart 2-to-1 I^{2}C multiplexer. It is a new innovative part taking care of the modern I^{2}C-bus switching without hang up or data loss. PCA9641 has very robust operation and the masters don't need to worry about management scheduling the downstream bus. PCA9641 is very easy to program, and it prevents a master from interrupting the other master until the task is finished. A requested master is notified when the downstream bus is available. With PCA9641, users will simplify their software and not worry about stealing the bus from the other master in the middle of transaction.

PCA9641 can be used to clear the downstream bus should it be hung.

2. Overview of PCA9641 smart 2-to-1 I^{2}C-bus multiplexer

The PCA9641 is a 2-to-1 I^{2}C master multiplexer with an arbiter function. It is designed for high reliability dual master I^{2}C-bus applications where correct system operation is required even when two I^{2}C-bus masters issue their commands at the same time. The arbiter will select a winner and let it work uninterrupted, and the losing master will take control of the I^{2}C-bus after the winner has finished. The arbiter also allows for queued requests where a master requests the downstream bus while the other master has control.

Multiple transactions can be done without interruption. Any master can reserve the downstream bus from 1 ms to 255 ms or forever by programming the reserve time register. During this time, the connection will be protected until the timer expires or the master gives up its ownership.

The pass gates of the switches are constructed such that the VDD pin can be used to limit the maximum high voltage, which will be passed by the PCA9641. This allows the use of different bus voltages on each pair, so that 1.8 V, 2.5 V, or 3.3 V devices can communicate with 3.3 V devices without any additional protection up to 3.6 V.

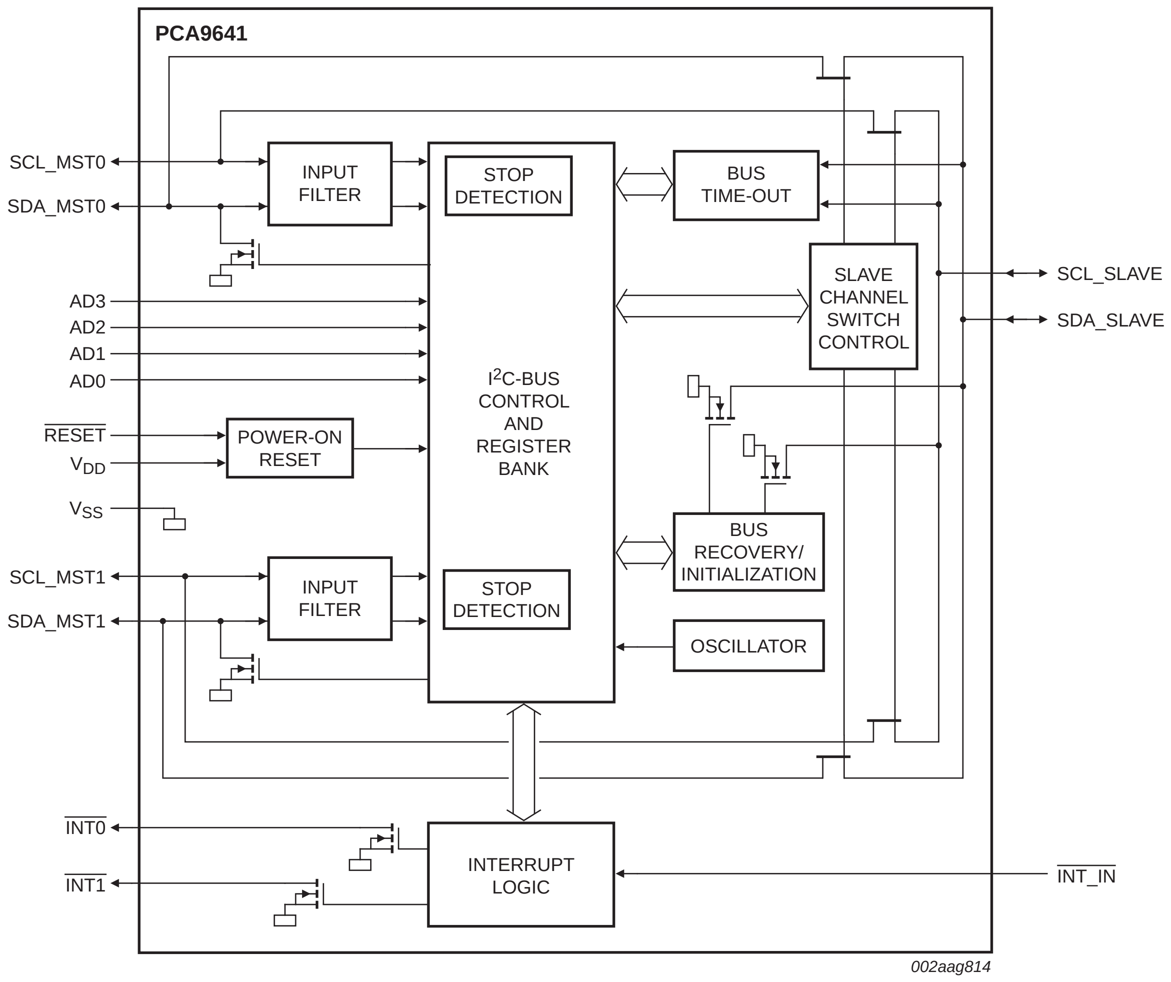

Fig 1. Block diagram of PCA9641

- 2-to-1 bidirectional master selector
- I²C-bus interface logic; compatible with SMBus standards
- Two active LOW interrupt outputs
- Active LOW reset input
- Channel selection via I²C-bus
- Four address pins allowing up to 128 different addresses
- Active arbitration when two masters try to take the downstream I²C-bus at the same time
- The winning master controls the downstream bus until it is done, as long as it is within the reserve time
- Hardware and Software reset
- Bus time-out after 100 ms on an inactive downstream I²C-bus optional
- Readable device ID (manufacturer, device type, and revision)
- Bus initialization/recovery function

- Bus traffic sensor
- Low R_{on} switches
- Allows voltage level translation between 1.8V, 2.3 V, 2.5 V and 3.3 V buses
- No glitch on power-up
- Supports hot insertion
- Software identical for both masters
- Low standby current
- Operating power supply voltage range of 2.3 V to 3.6 V
- 20 Hz to 1 MHz clock frequency
- Packages offered: TSSOP16, HVQFN16

3. Application design examples for PCA9641 arbiter

3.1 Principle of PCA9641 arbiter

PCA9641 arbiter is used in a system where two I^{2}C masters want to share the same slave devices on the downstream bus:

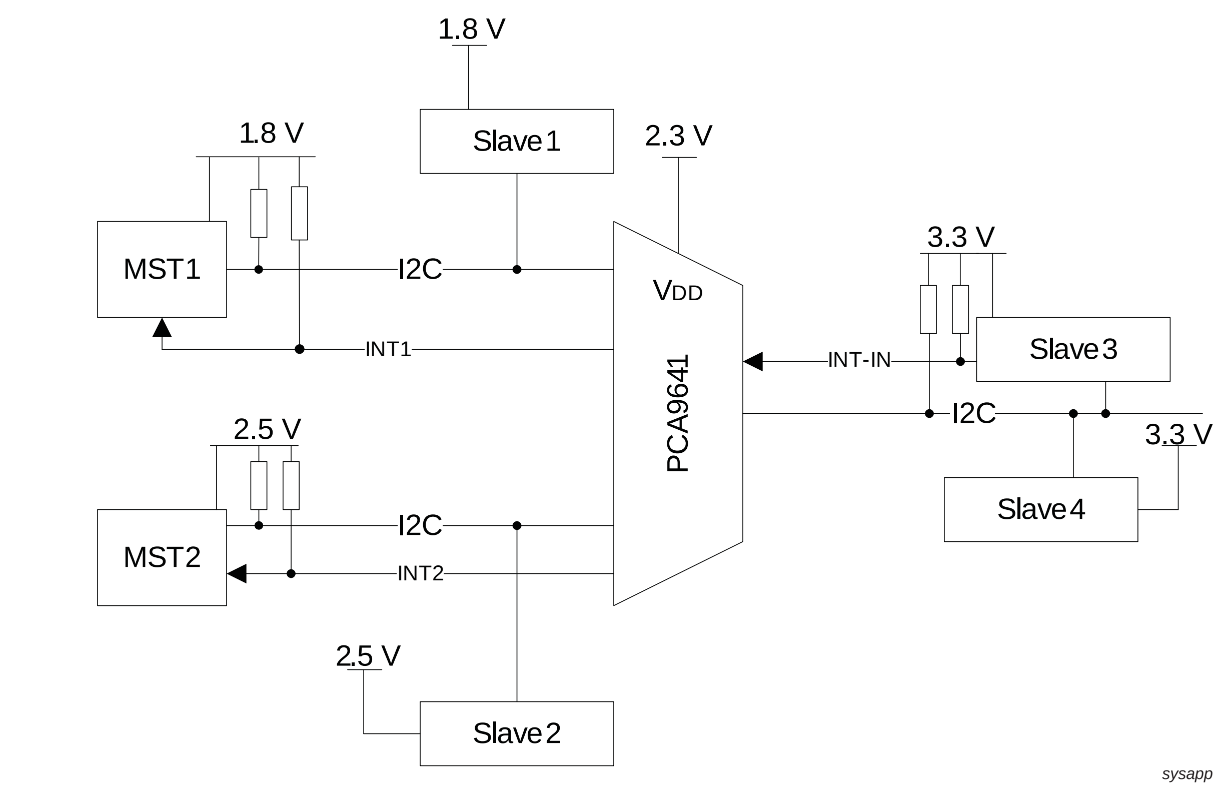

Fig 2. PCA9641 system application

When MST1 has its grant (owns the downstream bus), it can talk to slave1, slave3 and slave4, but it cannot see slave2 (see Fig 1). All its transactions will be protected without any interruption from MST2 until the reserve time is expired or the master gives up its grant. MST2 can request the downstream bus while MST1 owns the bus. After MST1 gives up the bus and MST2 will have its grant. MST2 will be notified by an interrupt from the PCA9641 or LOCK_GRANT register status.

3.2 Difference between PCA9541A de-mux and PCA9641 arbiter

PCA9541A I^{2}C de-mux operation: any master can take control of the downstream bus at any time and starts communicate with downstream slave devices. This action can cause many collisions, data lost and even more serious I^{2}C-bus hung.

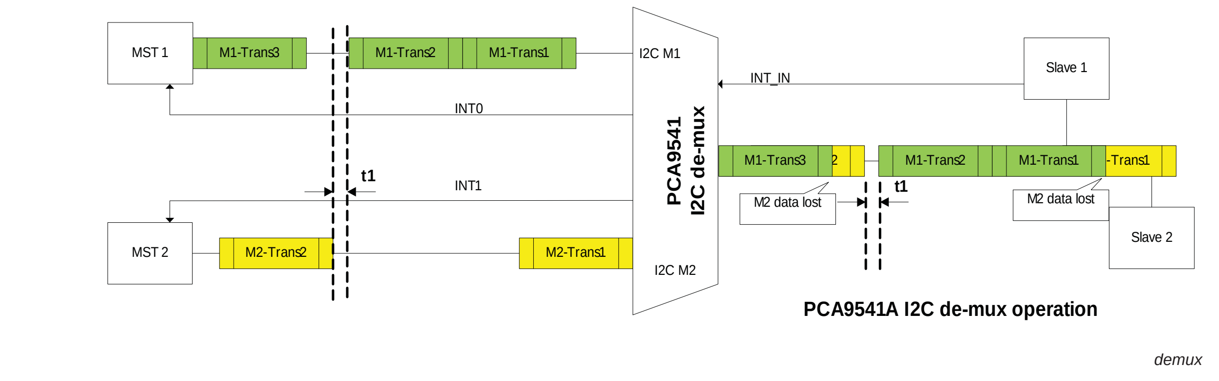

Fig 3. PCA9541A de-mux operation

PCA9641 I^{2}C arbiter operation: Any master can request the downstream bus at any time, but only one wins the bus, the losing master will take control of the bus after the winner finishes its task or gives up its ownership.

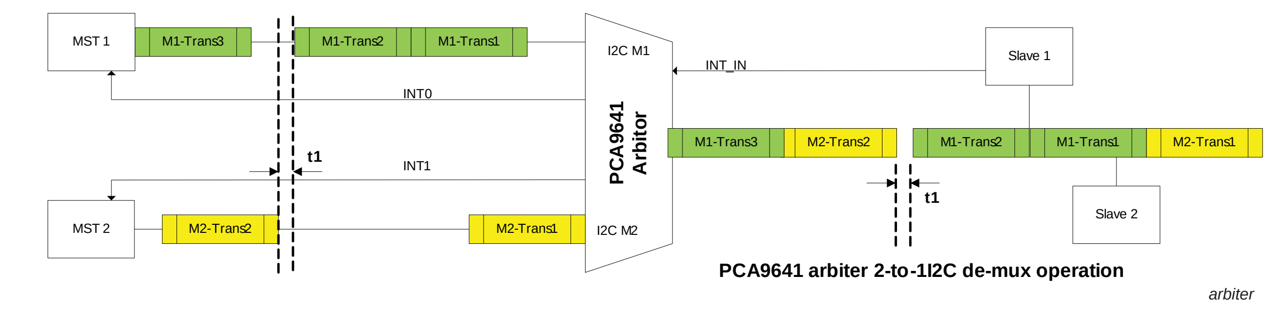

Fig 4. PCA9641 arbiter operation

A transaction includes many I^{2}C start conditions and stop conditions and idle times. Note: idle time should not be longer than 100 ms.

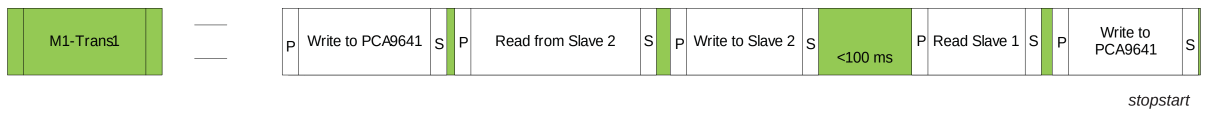

Fig 5. A transaction is included multiple "STOP" and "START"

4. How to program PCA9641 and control the downstream bus

4.1 How to request downstream bus

Any master can request the downstream bus at any time, but the requested master will not have its control until PCA9641 sends back a response that the downstream bus is ready to use. PCA9641 always monitors the two masters and the downstream bus. If the downstream bus is idle and no master is controlling it, the PCA9641 will give the downstream bus to the requested master.

There are two kinds of requests; request with time period and request with disable timer.

1. Request with time period - when the master knows exactly how much time is required it can program the timer register and send a request to PCA9641.
2. Request with disable timer - the master programs the timer with zero value, that means the timer is disable. The master can own the downstream bus forever until it writes to the control register to disable the controller (give up the downstream bus).

4.1.1 Request with time period

Request with time period - when the master knows exactly how much time is required it can program the timer register and send a request to PCA9641.

1. To reserve the downstream bus from 1 ms to 255 ms without any interruption by writing to RT register. Write 0x01 for 1 ms and 0xFF for 255 ms to RT register.
2. Request for the downstream bus by writing '1' to CONTR register bit 0 (LOCK_REQ = 1).

Simple code: A master wants to have 100 ms to talk to the downstream bus through PCA9641with INT (interrupt) pin connects microcontroller (MCU).

```
Step #0 // Enable interrupt service to MCU when LOCK_GRANT status register is active
      (or when the downstream bus is available INT pin will be active LOW)
| S | 9641 addr + W | ACK | 0x05 | ACK | 0x7B | ACK | P |
Step #1 // Master writes to reserve time = 100 ms or set RT = 0x64 (hex)
| S | 9641 addr + W | ACK | 0x03 | ACK | 0x64 | ACK | P |
Step #2 // Master writes '1' to CONTR register bit 0 to request the downstream bus
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x01 | ACK | P |
```

Note: Reserve time register cannot be changed after the request master gets grant

4.1.2 Request with disable reserve time

When a master does not know how much time it needs for the downstream bus. It can own the downstream bus forever until it writes to the control register to disable the controller (give up the downstream bus).

1. Disable the reserve time (reserve the downstream bus without knowing how much time needed) by writing 0x00 to RT (reserve time) register.
2. Request for the downstream bus by writing '1' to CONTR register bit 0 (LOCK_REQ = 1).

Simple code: A master wants to talk to the downstream bus, but it does not know how much time its need. System has setup with hardware interruption, INT pin of PCA9641connects to interrupt input pin of MCU.

```
Step #0 // Enable interrupt service to MCU when LOCK_GRANT status register is active
     (or when the downstream bus is available INT pin will be active LOW)
| S | 9641 addr + W | ACK | 0x05 | ACK | 0x7B | ACK | P |
Step #1 // Master writes '0' to reserve time to disable the timer (RT = 0x00)
| S | 9641 addr + W | ACK | 0x03 | ACK | 0x00 | ACK | P |
Step #2 // Master writes '1' to CONTR register bit 0 to request the downstream bus
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x01 | ACK | P |
```

Note: Reserve time register cannot be changed after the request master gets grant

4.2 Take control of downstream bus

After request command is done, master needs to monitor the interrupt line (INTx). If the interrupt line goes LOW, or LOCK_GRANT bit in CONTR register is set, the requested master has owned the downstream bus. The requested master needs to write to BUS_CONNECT bit in CONTR register to connect I^{2}C-bus from master to downstream bus.

Simple code: Wait for INT is LOW then connect I^{2}C-bus from master to downstream bus

```
// Wait for interrupt (INTx = 0)
While (INTx ); // INTx = 1 do nothing
// Connect I2C-bus from a master to downstream bus (BUS_CONNECT = 1) and make sure
     LOCK_REQ bit is '1'.
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x5 | ACK | P |
```

Now, the master can communicate with any slave devices on the downstream bus.

4.3 How to give up the downstream bus

4.3.1 Give up the downstream bus with reserve time not zero

After LOCK_GRANT bit is set in CONTR register, PCA9641 will start counting down in milliseconds, until the time becomes zero, the LOCK_GRANT will be reset with condition; PCA9641 must see the "STOP" condition and the bus is idle (SCL and SDA are high).

The grant master can write '0' to LOCK_REQ bit in CONTR register to give up the downstream bus before reserve time is expired.

Simple code: give up the ownership of the downstream bus

```
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x00 | ACK | P |
```

4.3.2 Give up the downstream bus with disable reserve time

If a master requests the downstream bus with zero on reserve time register, the master has to write '0' to LOCK_REQ bit in CONTR register to give up its ownership of the downstream bus. Otherwise, the granted master will keep the bus forever.

Simple code: Give up the ownership of the downstream bus

```
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x00 | ACK | P |
```

5. Avoid hogging the bus (e.g. keeping the downstream bus forever)

To avoid a granted master keeping the downstream bus forever, the granted master should enable the IDLE_TIMER_DIS bit in CONTR register. If the bus is idle for more than 100 ms, the LOCK_GRANT will be reset and the ownership of the downstream bus will expire.

Simple code: avoid keeping the bus forever

```
// enable 100 ms idle timer disconnect and request the downstream bus as the same time.
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x21 | ACK | P |
```

Note: Idle means: when SCL and SDA are not toggling after a stop condition.

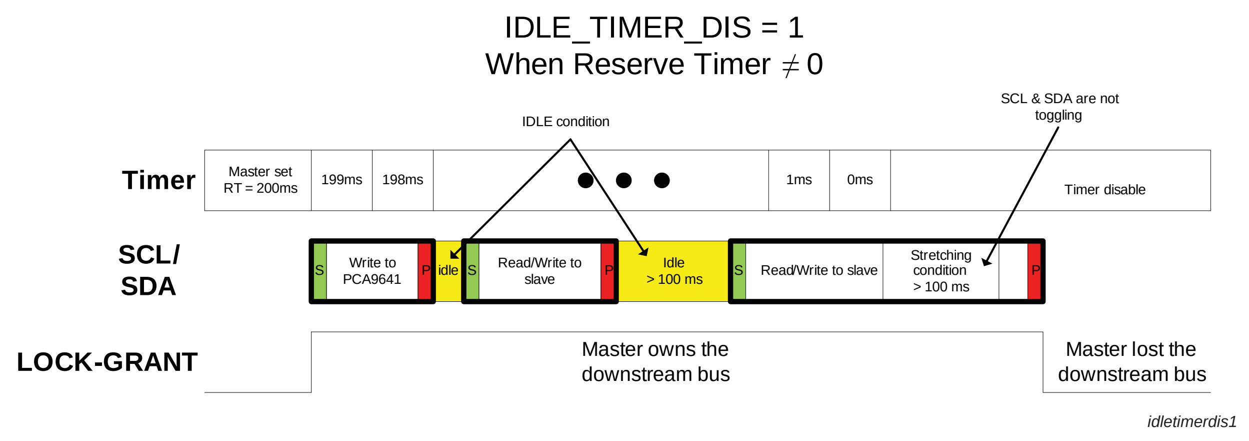

Fig 6. IDLE_TIMER_DIS = 1 when master uses reserve timer ≠ 0

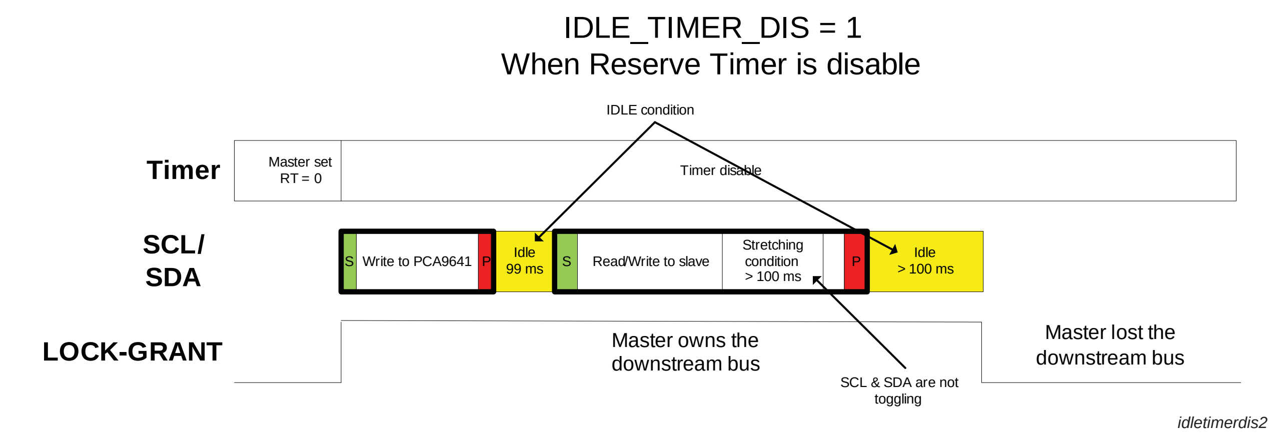

Fig 7. IDLE_TIMER_DIS = 1 when master uses reserve timer = 0

6. Bus initialization and bus recovery

Bus initialization - The granted master should enable this function before make connection with the downstream bus. This function will make sure the downstream bus in idle condition before it makes connection; otherwise, PCA9641 will report to master that the bus is hung.

Bus recovery - If a granted master knows the downstream bus is hung, the master can use this function to clear the downstream bus. If the bus cannot be clear, PCA9641 will send an interrupt to both master.

Here is how Bus init/recovery works: PCA9641 will send clocks out until SDA high, and then send a stop to complete its function. If SDA is still stuck low, PCA9641 reports BUS_INIT_FAIL in STATUS register.

Simple code: Enable BUS_INIT while connecting the I²C-bus to downstream bus

```
// enable: IDLE_TIMER_DIS, BUS_INIT, BUS_CONNECT and LOCK_REQ
| S | 9641 addr + W | ACK | 0x01 | ACK | 0x2D | ACK | P |
```

7. Reset options

There are two ways to reset the PCA9641, hardware reset and software reset

7.1 Hardware reset

PCA9641 has a hardware pin to reset all internal logic and registers to the power-up reset. All internal switches are open and no master owns the downstream bus.

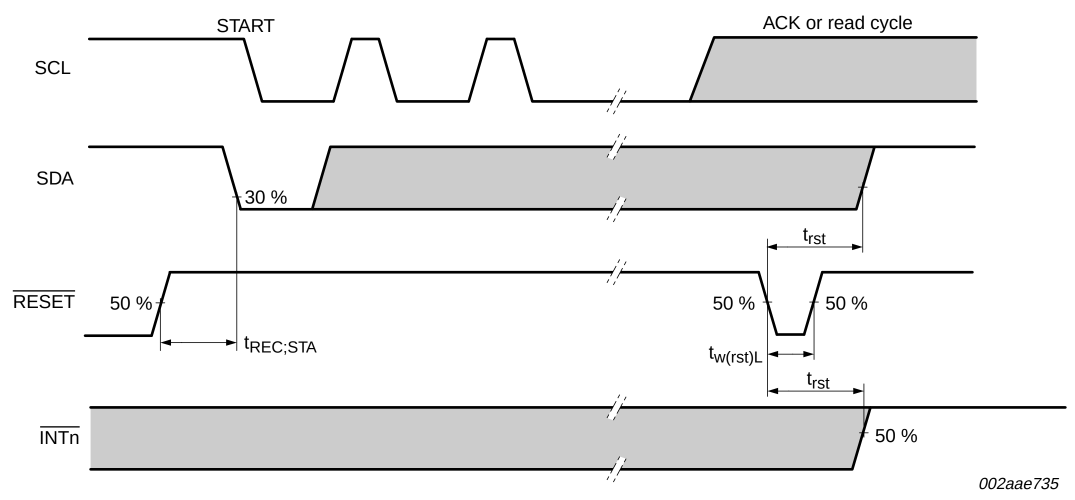

Fig 8. Definition of RESET timing

7.2 Software reset (general call software reset)

7.2.1 SW reset for system with all I²C devices (no SMBus device)

Any master can send a general call software reset to reset internal I²C logic and registers of all slave devices on the I²C-bus.

EX: If MST1 connects to downstream bus and sends general call software reset, all the blue devices on I²C-bus will be affected

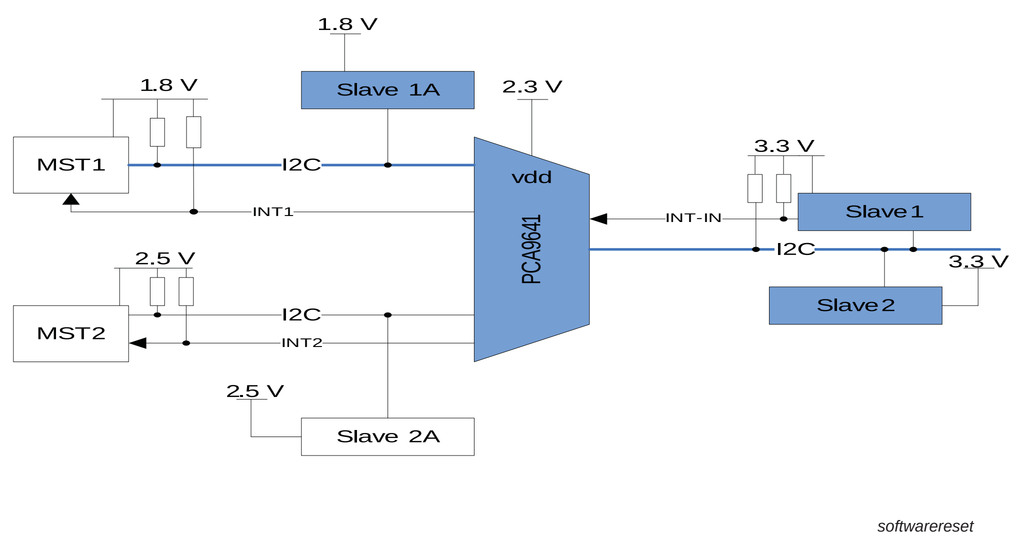

Fig 9. Software reset with all I²C slave devices (no SMBus device)

Simple code: general call software reset

```
| S | 0x00 + W | ACK | 0x06 | ACK | P |
```

7.2.2 SW reset for system with mixed I²C devices and SMBus devices

If a system has any SMBus device on I²C-bus, SMBus device would not understand the general call software reset command. Therefore, PCA9641 has an option to generate SMBus time-out to reset SMBus devices on the I²C-bus when it receives a general call software reset.

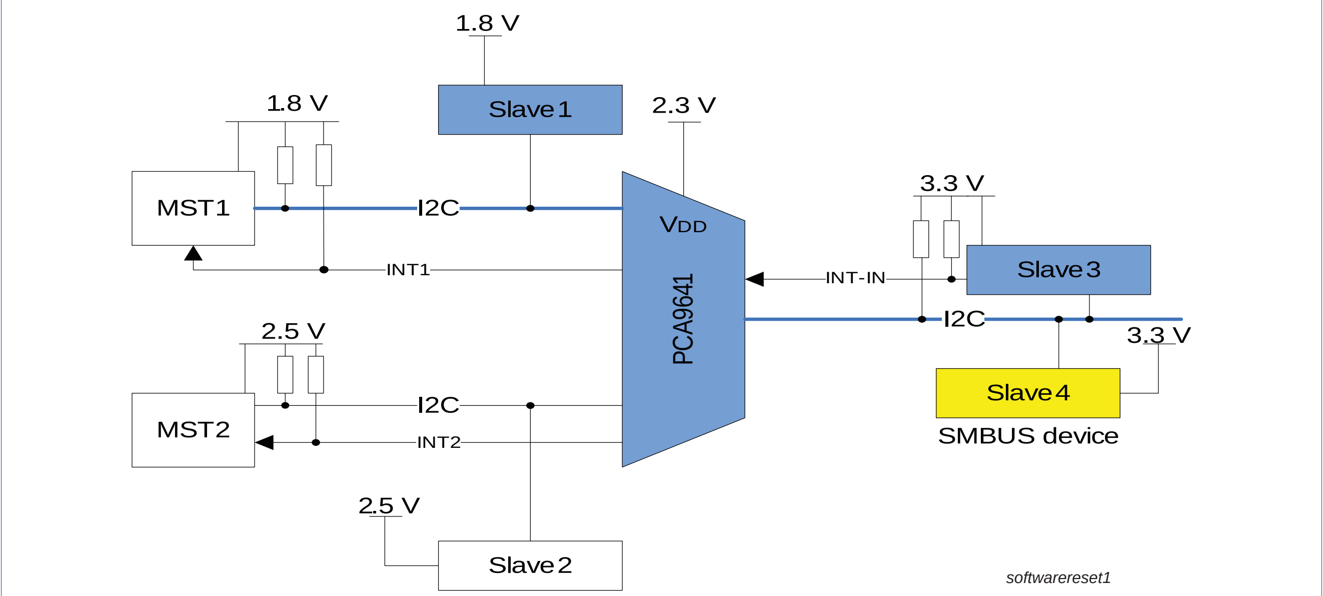

Fig 10. Software reset for a system has I²C devices and SMBus devices

Slave4 is a SMBus slave device, it will not reset when general call software reset is sent. If SMBUS_WSRST is set, PCA9641 will send SMBus time-out (35 ms SCL low) and then reset PCA9641.

```
Simple code: Turn on SMBUS_WSRST (CONTR[4] = 1)
| S | 9641 addr + W | ACK | 0x01 | ACK | (0xXX || 0x10) | ACK | P |
```

8. How to take care of a hung downstream bus

PCA9641 monitors the downstream bus and two master buses. If any SCL or SDA is stuck low for more than 500 ms, the PCA9641 will generate an interrupt to the master if BUS_HUNG_MSK is set in INT_MSK register.

8.1 How to separate the master I²C-bus from the hung downstream bus

There are two ways for a granted master to get off (disconnect) the hung downstream bus: (Note - user needs to set these bits before the event happens)

1. Set IDLE_TIMER_DIS - if there is any reason the I²C-bus has not toggled for more than 100 ms after reserve time is expired or disable, the PCA9641 will disconnect the master bus from the downstream bus and takes away its grant.

Simple code: how to enable IDLE_TIMER_DIS function

```
| S | 9641 addr + W | ACK | 0x01 | ACK | (0xXX || 0x20) | ACK | P |
```

2. Hardware reset pin - if the grant master pulls the reset pin LOW, then the PCA9641 will reset all internal registers and open the switch between masters and the downstream bus.

Note: A master can push other master, who owns the downstream bus, out by sending software reset or hardware reset to PCA9641 (not recommended)

8.2 How to clear the stuck downstream bus once requested from the bus

There are two methods to clear the downstream bus after the master has disconnected itself (e.g. Section 8.1)

1. Make sure the master owns the downstream bus and is not connect to the downstream bus. The granted master writes a command to enable the BUS_INIT function and then writes a command to connect to the downstream bus. PCA9641 will send clocks out to recover the downstream bus. If the bus cannot recover, PCA9641 will send an interrupt to both masters and also update BUS_HUNG_INT bit.

Simple code:

```
| S | 9641 addr + W | ACK | 0x01 | ACK | (0xXX || 0x08) | ACK | P |
```

2. Toggle SCL signal by programming the SCL_IO bit in the status register. The granted master can remote toggle SCL_SLAVE then check SDA_SLAVE by reading the SDA_IO in the status register.

Simple code: How to toggle SCL_SLAVE pin

```
// set SCL_SLAVE = 0 (low), clear bit 6 of status register
| S | 9641 addr + W | ACK | 0x02 | ACK | (0xXX & 0xDF) | ACK | P |
Wait period
// set SCL_SLAVE = 1 (low), set bit 7 of status register
| S | 9641 addr + W | ACK | 0x02 | ACK | (0xXX || 0x40) | ACK | P |
Wait period
```

9. Software flowchart how to program PCA9641

Here is the flowchart for how to get control of downstream bus. Two methods are presented; interrupt and polling. Best results are from using the interrupt.

9.1 Interrupt method

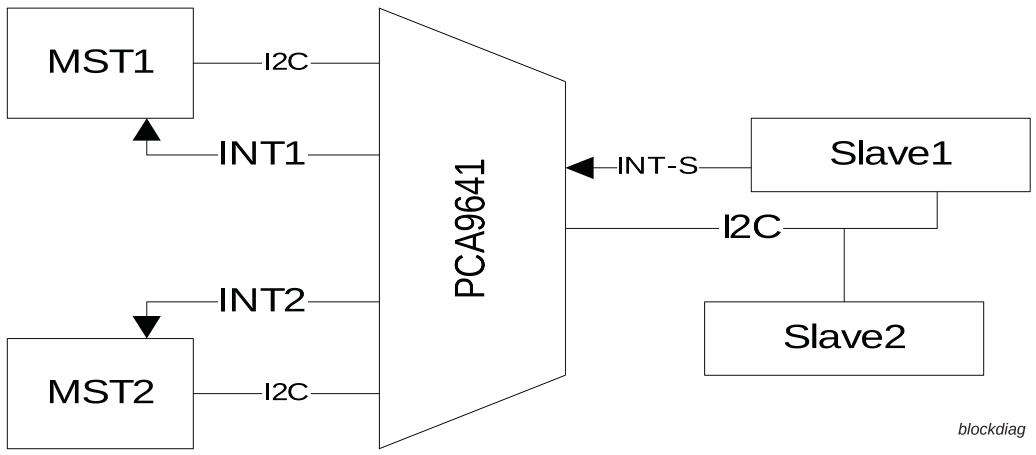

Fig 11. Block diagram of a system with interrupt connection

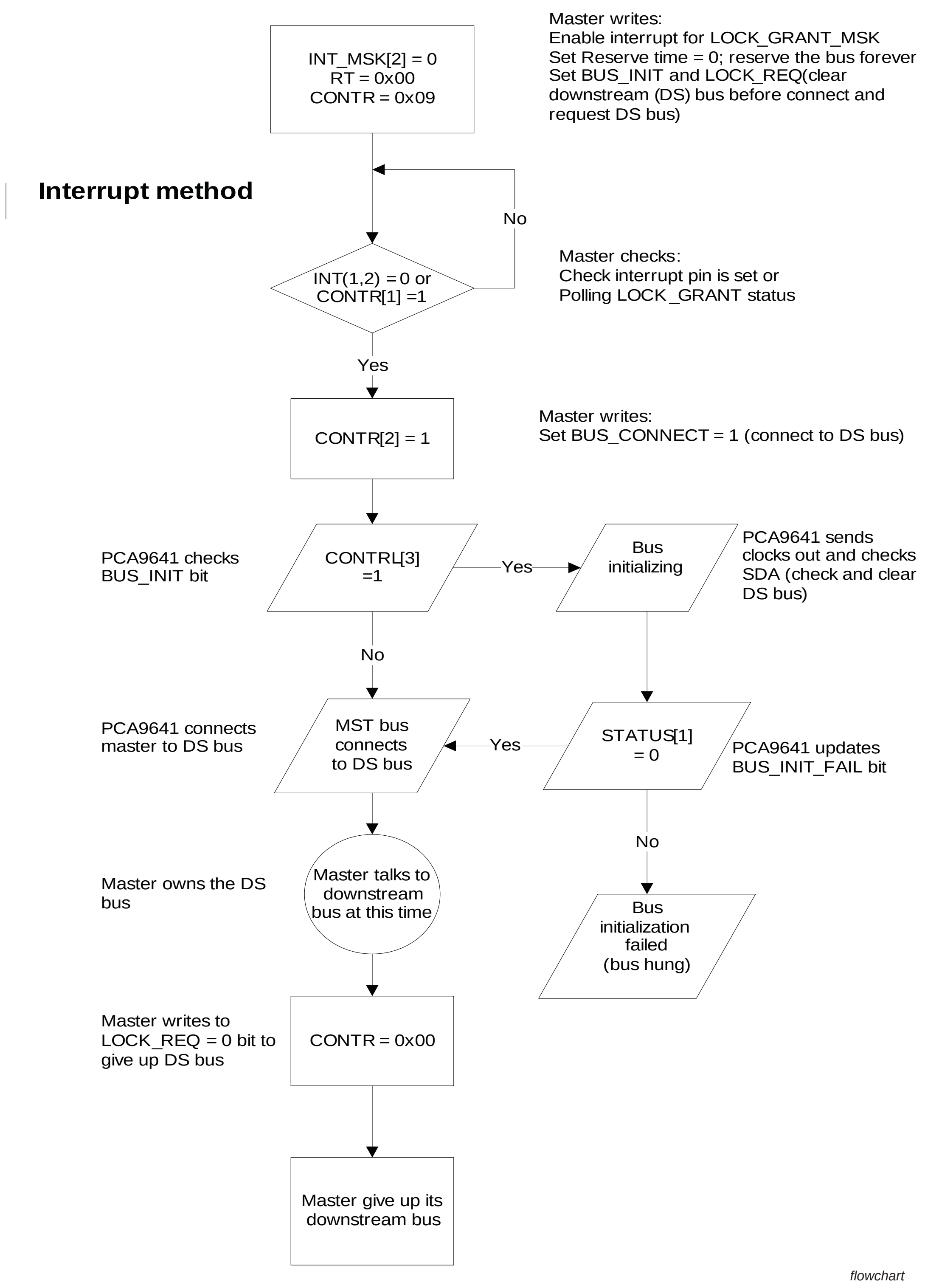
INT_MSK[2] = 0
RT = 0x00
CONTR = 0x09
Master writes:
Enable interrupt for LOCK_GRANT_MSK
Set Reserve time = 0; reserve the bus forever
Set BUS_INIT and LOCK_REQ(clear downstream (DS) bus before connect and request DS bus)
Interrupt method
No
INT(1,2) = 0 or CONTR[1] =1
Master checks:
Check interrupt pin is set or Polling LOCK_GRANT status
Yes
CONTR[2] = 1
Master writes:
Set BUS_CONNECT = 1 (connect to DS bus)
PCA9641 checks BUS_INIT bit
CONTRL[3] =1
Yes
Bus initializing
PCA9641 sends clocks out and checks SDA (check and clear DS bus)
No
PCA9641 connects master to DS bus
MST bus connects to DS bus
Yes
STATUS[1] = 0
PCA9641 updates BUS_INIT_FAIL bit
No
Master owns the DS bus
Master talks to downstream bus at this time
Bus initialization failed (bus hung)
Master writes to LOCK_REQ = 0 bit to give up DS bus
CONTR = 0x00
Master give up its downstream bus
flowchart

Fig 12. PCA9641 software flow chart for a system without interrupt connection

9.2 Polling method

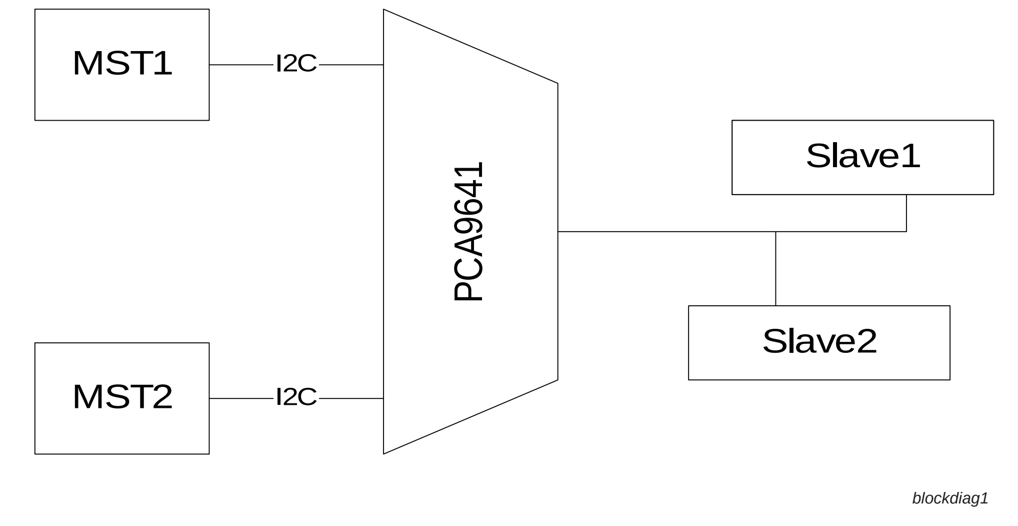

Fig 13. Block diagram of a system without interrupt connection

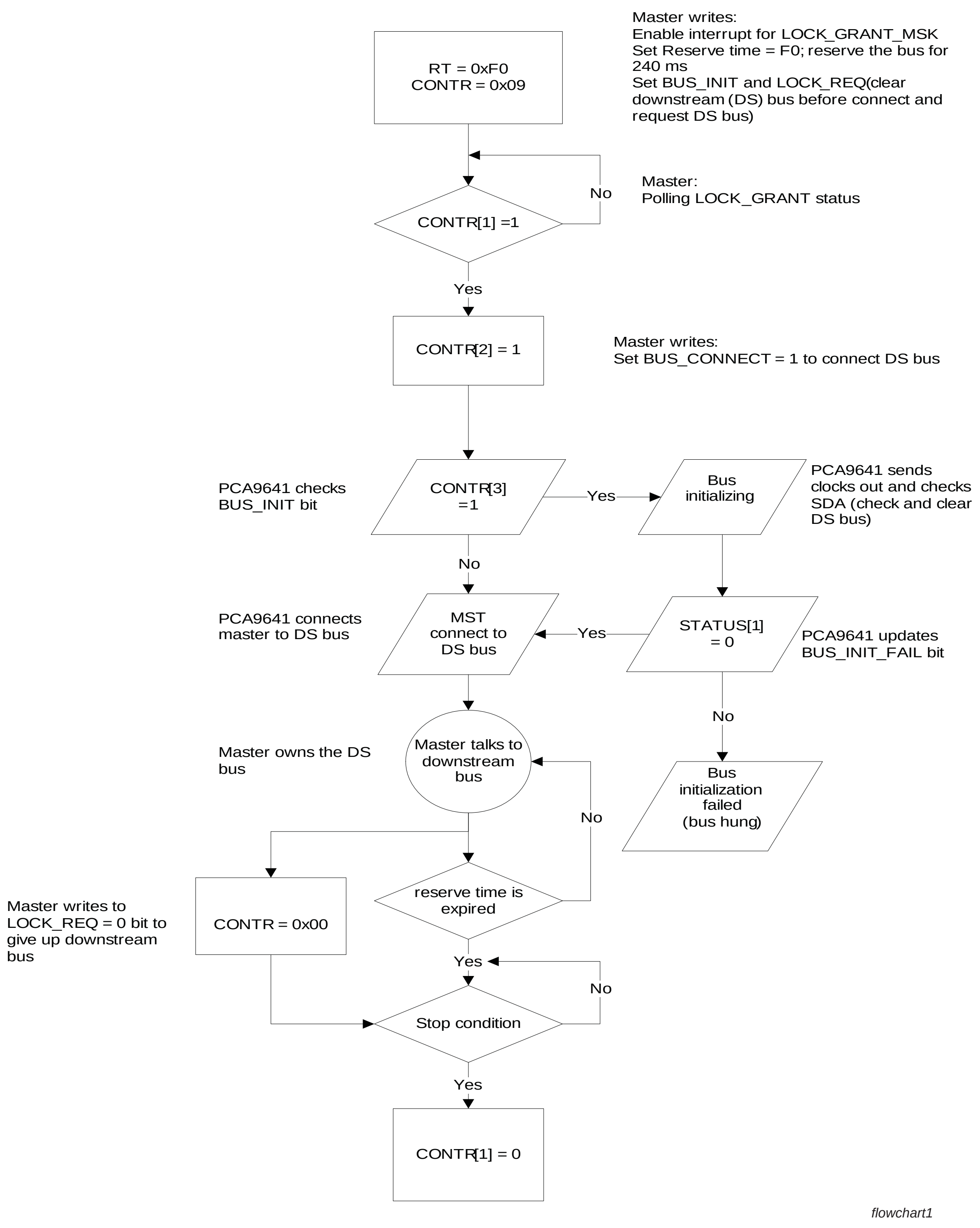
Master writes:
Enable interrupt for LOCK_GRANT_MSK
Set Reserve time = F0; reserve the bus for 240 ms
Set BUS_INIT and LOCK_REQ(clear downstream (DS) bus before connect and request DS bus)
RT = 0xF0
CONTR = 0x09
Master:
Polling LOCK_GRANT status
CONTR[1] =1
No
Yes
CONTR[2] = 1
Master writes:
Set BUS_CONNECT = 1 to connect DS bus
PCA9641 checks BUS_INIT bit
CONTR[3] =1
Yes
Bus initializing
PCA9641 sends clocks out and checks SDA (check and clear DS bus)
No
PCA9641 connects master to DS bus
MST connect to DS bus
Yes
STATUS[1] = 0
PCA9641 updates BUS_INIT_FAIL bit
No
Bus initialization failed (bus hung)
Master owns the DS bus
Master talks to downstream bus
No
Master writes to LOCK_REQ = 0 bit to give up downstream bus
CONTR = 0x00
reserve time is expired
Yes
No
Stop condition
Yes
CONTR[1] = 0
flowchart1

Fig 14. PCA9641 software flow chart for a system without interrupt connection

10. Design consideration

10.1 How to calculate pull up resistors for SCL and SDA

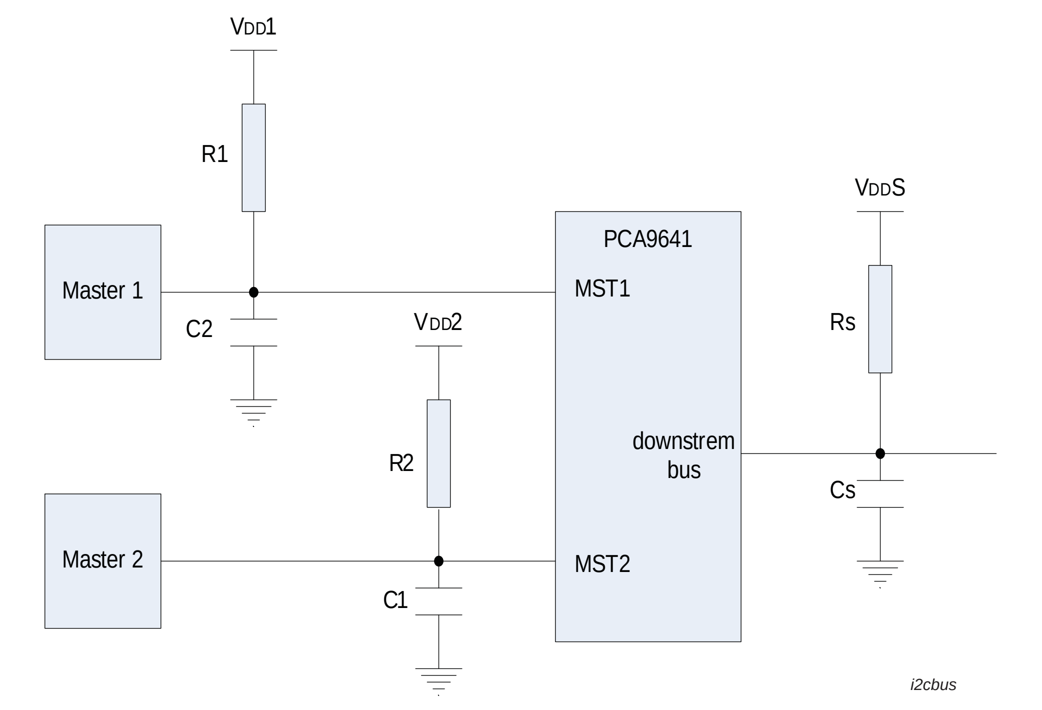

Fig 15. I^{2}C-buses with pull-up resistors

PCA9641 provides isolation between two masters but do not provide any additional drive capability or isolate capacitance between the master buses and downstream bus. Therefore, the overall capacitance of the highest master bus and downstream bus must be taken into consideration when choosing the value of pull-up resistor.

The main considerations in choosing the pull-up resistor are:

1. Ensuring that the current does not exceed the maximum Iol = 3 mA at 0.4 V. This determines the minimum resistor value.
2. Ensuring that the rise time does not exceed 1.0 μs for a standard mode (100 kHz) bus, 300 ns for the Fast-mode (400 kHz) or 120 ns for the Fast-most Plus (1 MHz) (affected by the bus capacitance and pull-up resistor). This determines the maximum resistor value.

When the input voltage to the multiplexer is low, the resistance of the switch is assumed to be negligible in comparison to the pull-up resistors.

For this example of devices operating in the standard mode (100 kHz), the power consumption is not critical since it is operating from the mains, so the maximum 3 mA current is allowed to flow when SDA and SCL are low.

I1 is the current through R1

I2 is the current through R2

Is is the current through Rs

C1 is the total load capacitance of the master 1 bus 300 pF

C2 is the total load capacitance of the master 2 bus 200 pF

Cs is the total load capacitance of the downstream bus 100 pF

VDD1 is voltage of master1 bus 3.3 V

VDD2 is voltage of master2 bus 1.8 V

VDDS is voltage of downstream bus 2.5 V

Since the capacitance of the downstream is ¼ of the total capacitance of the bus when PCA9641 is connected to downstream bus, I1 = 3mA * 1/4 = 0.75 mA and pull up in master 1 and master 2 bus can be set to I1 = I2 = 2.5 mA

R1 = 3.3 V / 2.25 mA = 1.32 kΩ

R2 = 1.8 V / 2.25 mA = 0.8 kΩ

Rs = 2.5 V / 0.75 mA = 3.33 kΩ

Additional verification:

Ensure the rise time specification of 1 μs for standard mode I^{2}C is not exceeded. Consider the VDD -related input threshold of VIH = 0.7 x VDD and VIL = 0.3 x VDD for the purposes of RC time constant calculation.

V(t) = VDD (1-1/e -t /RC) where t is the time since the charging started and RC is the time constant.

V(t1) = 0.3 x VDD = VDD (1-1/e-t 1/RC); then t1 = 0.3566749 x RC

V(t2) = 0.7 x VDD = VDD (1-1/e-t 2/RC); then t2 = 1.2039729 x RC

Trise = t2 - t1 = 0.8472979 x RC

Scenario 1: Master 1 with no downstream channel enabled

Trise = 0.8472979 x R1C1

= 0.8472979 x 1320 x 300e-12

= 0.33 μs

Trise = t2 - t1 = 0.8472979 x RC

Scenario 2: Master 2 with no downstream channel enabled

Trise = 0.8472979 x R2C2

= 0.8472979 x 800 x 200e-12

= 0.14 μs

Scenario 2: Master 1 enabled

Trise = 0.8472979 x (R1// Rs)(C1 // Cs)

= 0.8472979 x (1300x3330/(1300+3330)) x (300 e-12 + 100 e-12)

= 0.32 μs

Scenario 3: Master 2 enabled

Trise = 0.8472979 x (R2// Rs)(C2 // Cs)

= 0.8472979 x (800x3330/(800+3330) x (200 e-12 + 100 e-12)

= 0.16 μs

All rise times are well below the maximum rise time of 1 μs.

11. Summary

PCA9641 is an arbitrator of two I^{2}C-bus masters to avoid collisions and help to recover from hung buses. This application note outlines how to use the PCA9641 and provides software to recover from a hung downstream bus.

12. Legal information

12.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

12.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

12.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

13. Contents

Date of release: 23 October 2014
Document identifier: AN11593