NXP UM10580 Mini board Application note

Mini board PCU9669 is a demonstration board for I2C-bus controllers. This demo board enables quick and easy evaluation of PCU9669 and PCA9665 with mbed.

UM10580

Mini board PCU9669

Rev. 1 — 25 February 2014 User manual

Document information

Info	Content
Keywords	I2C, I2C-bus, PCU9669, PCA9665, bus controllers, mbed, Ultra Fast-mode, UFm
Abstract	Mini board PCU9669 is a demonstration board for I^2C-bus controllers. This demo board enables quick and easy evaluation of PCU9669 and PCA9665 with mbed.

Revision history

Rev	Date	Description
v.1	20140225	User manual; initial release

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

The ‘mini board PCU9669’ is an evaluation board kit for I²C-bus controllers, the PCU9669 family and PCA9665. This evaluation board can demonstrate its advanced functionality with easy-to-use software development platform.

The PCU9669 is a new generation I²C-bus controller that supports new ‘Ultra-Fast mode (UFm)’, which is defined in *UM10204, “I²C-bus specification and user manual”* (Ref. 1).

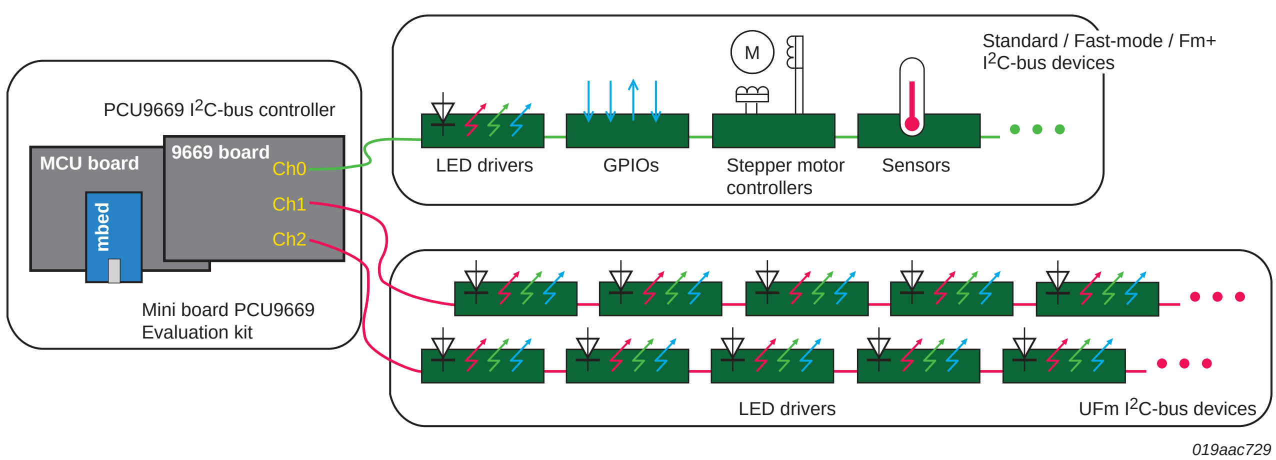
The PCU9669 bridges the MCU parallel bus and 3 channel I²C buses (UFm × 2 ch + Fm+ × 1 ch). The PCU9669 (and its family) has a big 4 kB buffer to manage transfers with ultra low CPU load. All I²C-bus channels can be operated only as a master.

The PCA9665 is a 1 channel I²C bus controller that cam work in the multi-master environment. The PCA9665 can work as both I²C roles of master and slave. I²C-bus transfer can be done byte-by-byte or by using internal buffer up to 68 bytes.

Sample codes are available for this evaluation board kit. They work on an mbed microcontroller that uses NXP LPC1768 as the main MCU. The mbed has an advanced software development environment for the microcontrollers. The Mini board PCU9669 kit provides a quick and easy way to evaluate the I²C-bus controllers with mbed’s cloud compiler and its powerful libraries. The LPC1768 on the mbed board supports building a feature-rich demo with its high capability (ARM Cortex-M3 core runs 96 MHz and 64K-SRAM and 512K internal flash).

This evaluation board kit is populated with the following:

- Mini board PCU9669 (9669 board)
- Mini board PCA9665 (9665 board)
- Mini board mbed (MCU board)
- Sample code



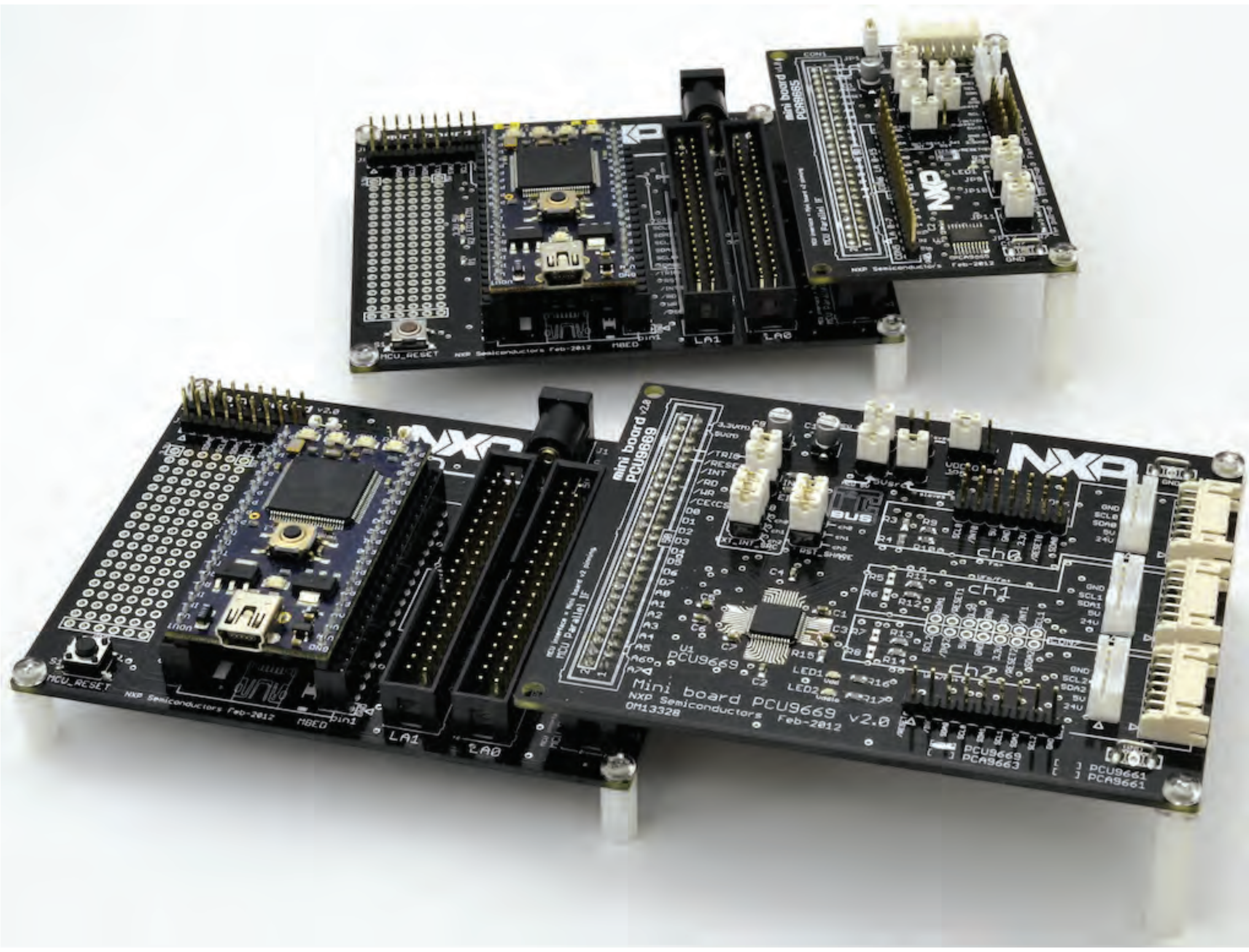
Channel 0 of PCU9669 is a Fm+ channel. Normal I²C-bus devices can be connected on this channel.

Channel 1 and channel 2 of PCU9669 are UFm channels.

Fig 1. Sample allocation diagram using Mini board PCU9669

2. Features

- Complete evaluation platform for the PCU9669 and the PCA9665
- I²C connectors are compatible with I²C-bus slave device demo boards
- Easy software development capability powered by mbed.org
- Complete sample code for PCU9669 and PCA9665 operation
- The sample code built-in layers to abstract levels of the control
- The layered code structure makes code porting easy



019aac730

Upper: PCA9665 board and MCU board with mbed.

Lower: PCU9669 board and MCU board with mbed.

Fig 2. Mini board PCU9669 evaluation kit

3. Getting started

3.1 Assumptions

Familiarity with the I^2C-bus is helpful, but not required.

3.2 Target versions

- This manual is written based on the versions of:
- Mini board MCU – version 2.0
- Mini board PCU9669 – version 2.0
- Mini board PCA9665 – version 1.0
- Sample code – version 1.0

3.3 Static handling requirements

CAUTION

This device is sensitive to ElectroStatic Discharge (ESD). Therefore care should be taken during transport and handling. You must use a ground strap or touch the PC case or other grounded source before unpacking or handling the hardware.

3.4 Ordering

Please contact us at **i2c.support@nxp.com** if you would like a board.

3.5 Minimum requirements

- A PC with internet connection (Windows, Mac or Linux)
- Latest version of web browser (Chrome, Safari, Firefox or Internet Explorer)
- Mini board PCU9669 kit
- mbed NXP LPC1768
- PCU9955 and PCA9955 evaluation boards

3.6 Setup

This section describes how to set up the mini board PCU9669.

Two different setups (for PCU9669 and PCA9665) will be discussed in next sections.

3.6.1 PCU9669 demo setup

3.6.1.1 Preparation (hardware)

Requirements: Prepare the following boards for the demo setup.

- Mini board PCU9669
- Mini board MCU
- mbed NXP LPC1768

mbed NXP LPC11U24 (yellow-mbed) cannot be used because sample code is not compatible to the yellow-mbed main chip (LPC11U24) port configuration.

The following equipment is not mandatory but recommended to see the demo features.

Next boards can be used as I^2C-bus slave devices that enable seeing the PCU9669 working by LED blinks/dimming. If these boards are not available, user can check the operation by checking the I^2C signal with an oscilloscope or a logic analyzer. PCA9955 and PCU9955 are constant current drive 16 channel LED controllers.

- PCA9955 demo board (constant current drive 16 channel LED controller, Fm+ I^2C-bus)
- PCU9955 demo boards (2 boards; constant current drive 16 channel LED controller, UFm I^2C-bus)

5 V power supply is required to power those boards. 5 V >1.5 A AC adapter is suitable. The DC connector is a 2.1 mm/5.5 mm outside diameter standard type with inner 5 V and outer ground.

019aac731

Fig 3. DC 5 V connector

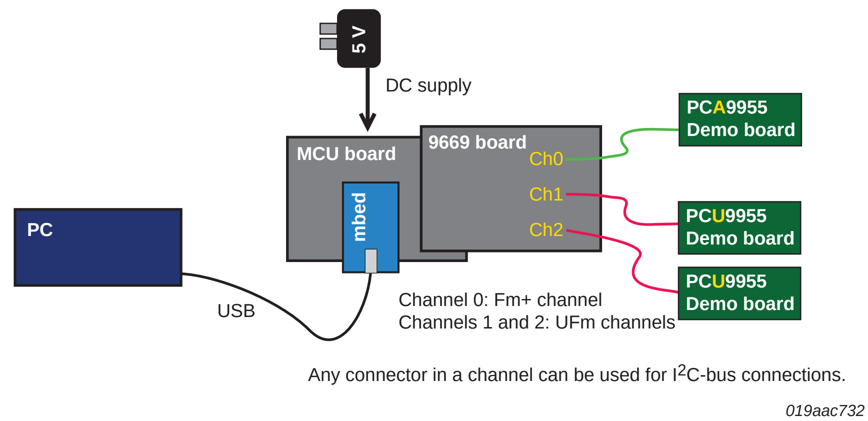
Connections

1. Make sure the all jumper settings are in default position (see Figure 5).
2. Connect MCU-board (mini board MCU) and 9669 board (mini board PCU9669).
3. Put the mbed on MCU board.
4. Connect a PCA9955 board and PCU9955 boards on each I^{2}C channel. Any connector on each channel can be used for slave connections.
    a. Set all **slave devices** (PCA9955 and PCU9955) **I^{2}C address to ‘0xC0’**.
5. Leave the channel open if no slave device is available. The operation can be checked by oscilloscope or logic analyzer on I^{2}C signals.
    a. The UFm bus does not care if the slaves are connected or not.
    b. But the Fm+ bus transfer will be terminated when NACK happened. Just I^{2}C-bus address sending can be seen without slave device.
6. Connect DC power supply to J1 on MCU board (the LED1 and LED2 on both boards will be turned ON).
7. Connect mbed to PC via USB cable.

This PC connection is not required after executable file is copied to mbed. The mbed runs as standalone while it has the file inside.

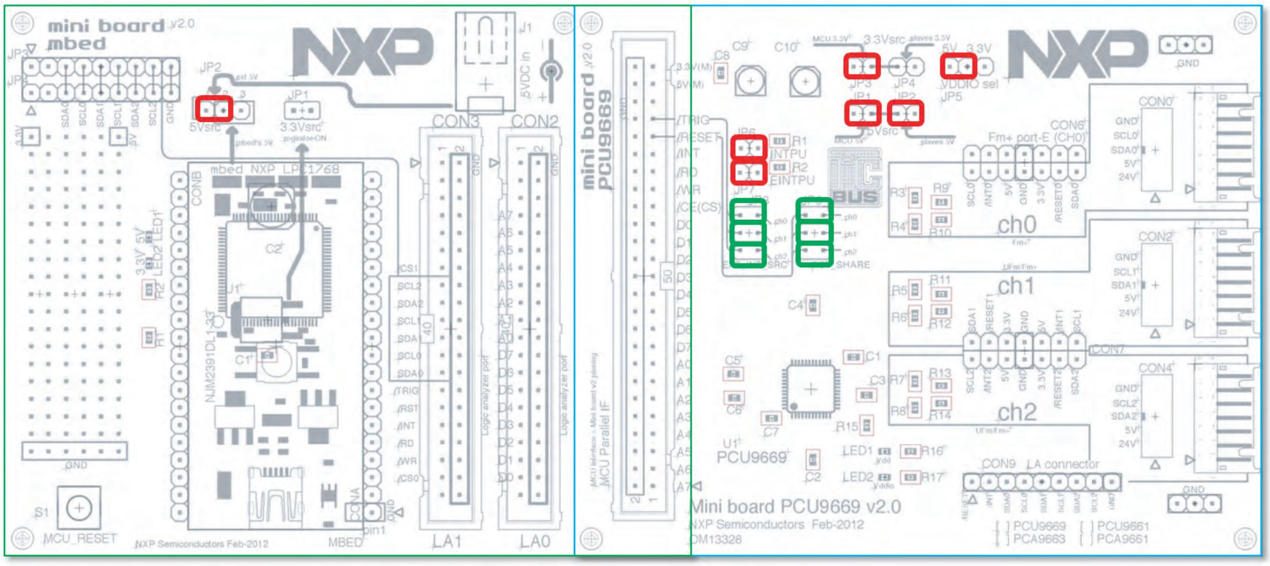
The PC connection can be used for power supply via USB if the slave board does not require much current. The mbed 5 V output (VU pin) can share a few hundred mA of USB bus supply. Short pin2 and pin3 on JP2 of MCU board when the setup uses the mbed 5 V output.

See Section 10 (Appendix D) for the LED controller demo board connection sample.



All slave devices should have I^{2}C-bus address = 0xC0.

Fig 4. Board connection (for 9669)



019aac733

RED jumper positions must be shorted.

Opened jumpers should be left open.

GREEN jumpers are optional.

Fig 5. Default jumper setting

3.6.1.2 Preparation (software)

mbed: This evaluation system uses the mbed software development environment. Obtain an account and become familiarized with the mbed tools before using this kit.

The mbed guide is available on these pages:

http://mbed.org/handbook/Setup-guide

http://mbed.org/handbook/Downloading-a-program

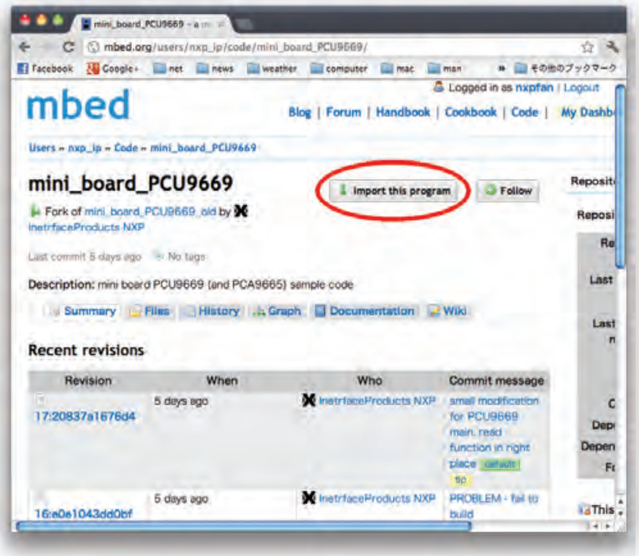
http://mbed.org/handbook/Creating-a-program

3.6.1.3 Importing sample code

A sample code for the mini-board kit is available on mbed.org site. Go to this URL page and import the code into your compiler:

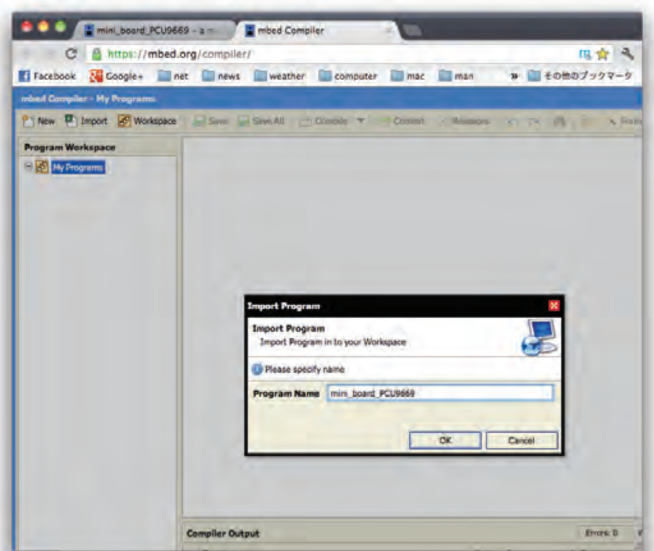
http://mbed.org/users/nxp_ip/code/mini_board_PCU9669/

To import the code, click the ‘import this program’ link. After clicking the link, the program (a project package) will appear on your on-line IDE (Figure 6, Figure 7 and Figure 8).



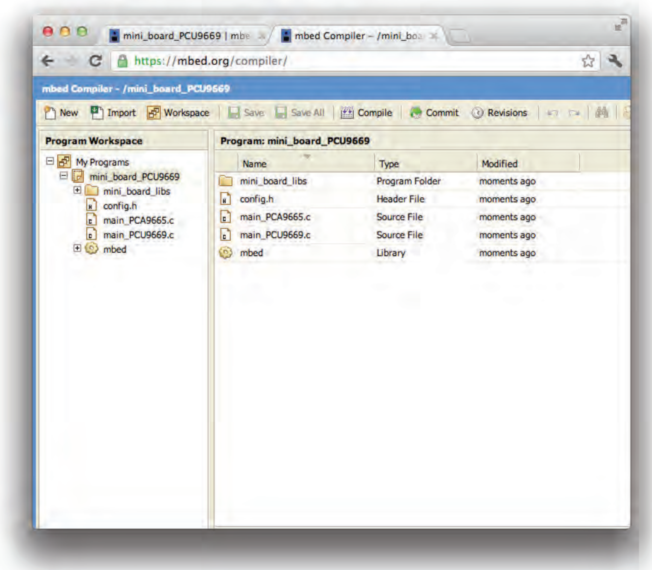
Click the ‘Import this program’ link.

Fig 6. Program import page of mbed.org



Click ‘OK’ to import.

Fig 7. Compiler page will be opened and confirms import in a dialog box



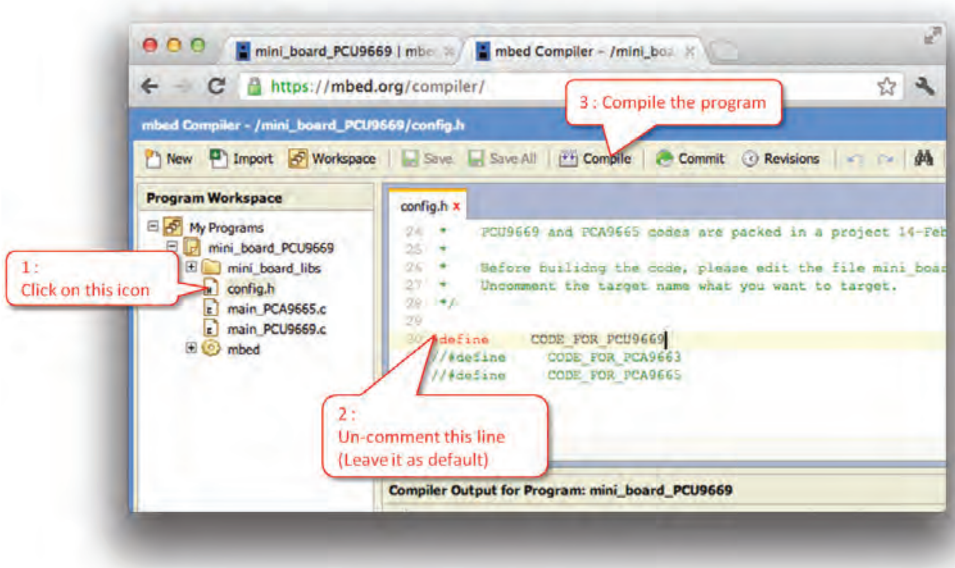
019aac736

The list of files is shown after importing.

Fig 8. Program has been imported into compiler

3.6.1.4 Set the target and compile

Open ‘config.h’ file (click on ‘config.h’ icon in left column) in ‘mini_board_PCU9669’ program and un-comment the line of ‘`#define CODE_FOR_PCU9669`’. All other lines should be commented-out.



019aac737

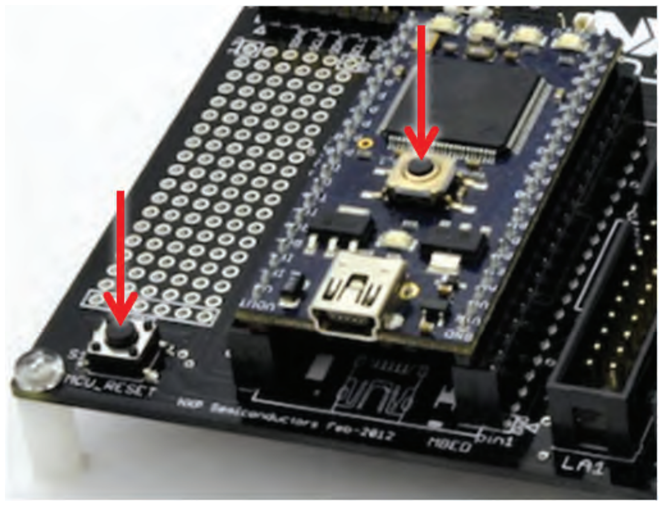
Fig 9. Set the target and compile

3.6.1.5 Copy the executable and let it run

When the compilation has completed successfully, the development tool will let you download an executable file '**mini_board_PCU9669_LPC1768.bin**'.

Download this file and copy into the mbed. (The mbed appears as a USB storage device on the PC.) Just save a file from web browser into mbed or save the file on local storage and copy the file by drag-and-drop into the mbed.

The demo is started after pressing the reset button on the mbed or MCU board.



019aac738

Press one of these buttons to start the program.

Both buttons have the same function.

Fig 10. Reset buttons

3.6.1.6 I²C-bus transfer on PCU9669

The sample code demonstrates I²C-bus transfer on PCU9669. The I²C-bus transfers are repeated in every 10 ms period on each of the three channels.

On the Fm+ channel (channel 0), 42 bytes write, 1 byte write and 41 bytes read are performed. These transactions are doing the PCA9955 registers write from MODE1 to OFFSET and read back. The second and third transactions are started with the RESART condition.

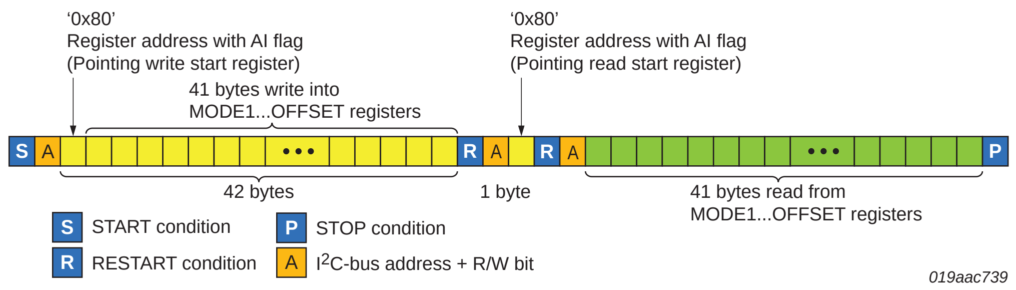


Fig 11. I²C-bus transfer to/from PCA9955

On the UFm channels (channel 0 and 1), 42 bytes write is done for PCU9955 registers (from MODE1 to OFFSET). No read back is performed since this bus is unidirectional.

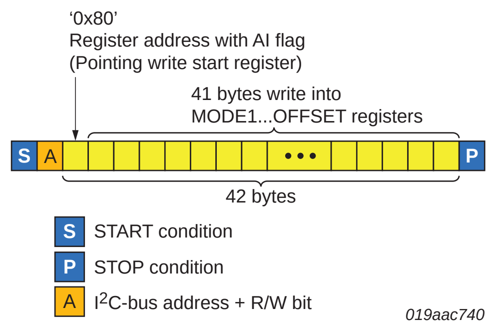


Fig 12. I²C-bus transfer to PCU9955

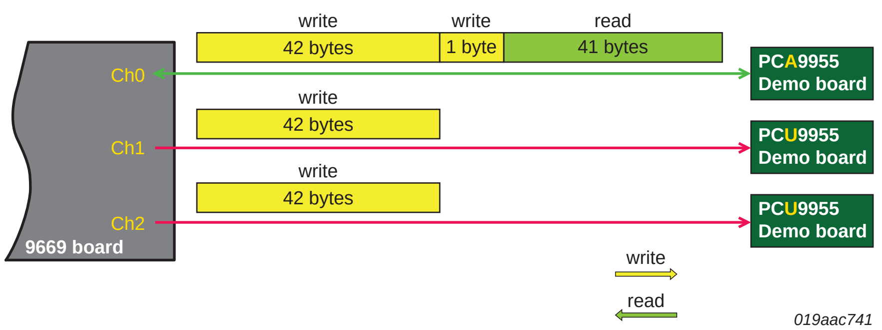
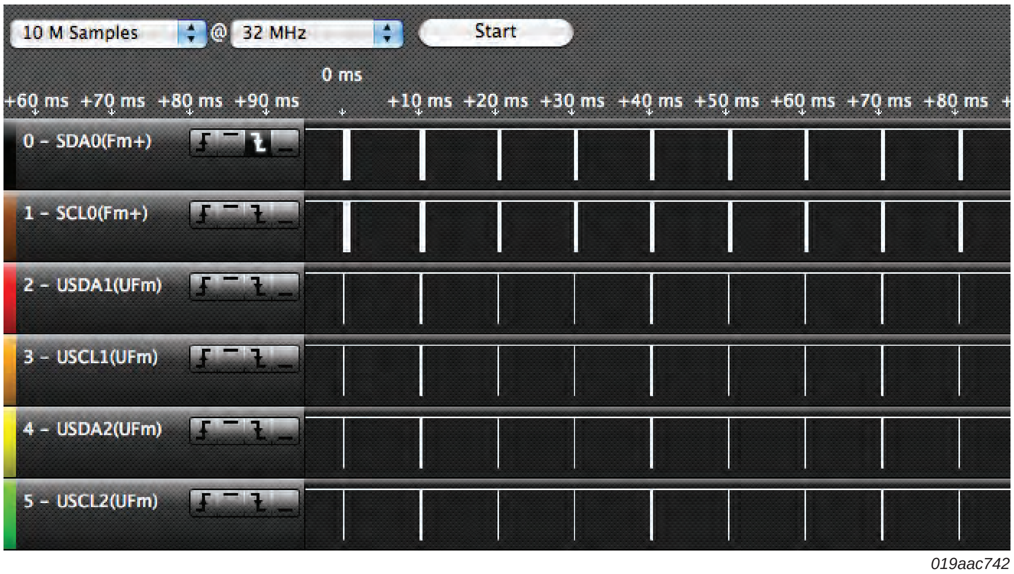
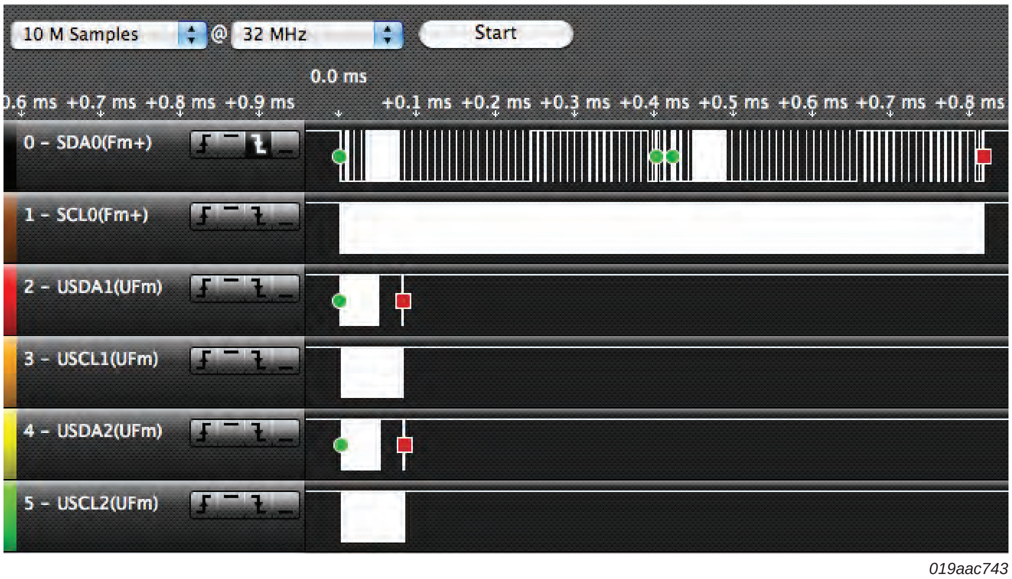


Fig 13. Sample code I²C-bus transfer on PCU9669



All three channels transfer (sequence) executed on 10 ms period.

Fig 14. I²C-bus transfer of sample code (PCU9669)



Waveform with different time span from Figure 14.

I²C-bus transfers on three channels. Green dot shows START/RESTART conditions and red square shows STOP conditions (on SDA lines).

Fig 15. I²C-bus transfer of sample code (PCU9669)

3.6.2 PCU9665 demo setup

3.6.2.1 Preparation (hardware)

Requirements: Prepare the following boards for the demo setup:

- Mini board PCA9665
- Mini board MCU
- mbed NXP LPC1768

The following equipment is not mandatory but recommended to see the demo features. And next, those boards are required as I²C-bus slaves. If these are not available, the user can check the operation by checking the I²C signal with an oscilloscope or a logic analyzer (but the transfer will be terminated by NACK. So only the address transfer can be observed without slave).

- PCA9955 demo board

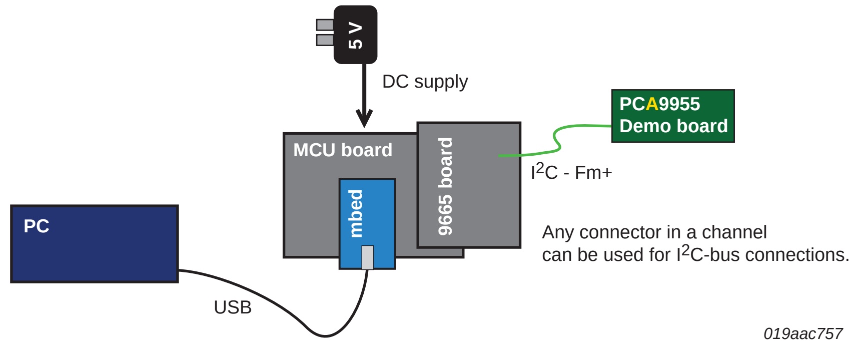
A 5 V power supply is required to power those boards. 5 V >1.5 A AC adapter is suitable. The DC connector is a 2.1 mm/5.5 mm outside diameter standard type with inner 5 V and outer ground.

Connections

1. Make sure the all jumper settings are in default position (see Figure 17).
2. Connect MCU board (mini board MCU) and 9665 board (mini board PCA9665).
3. Put the mbed on MCU board.
4. Connect PCA9955 board to one of the I²C connectors on 9665 board. Any I²C connector can be used for slave connections.
   a. Set **slave I²C-bus address** to **0xC0**.
   b. Just I²C address sending can be seen if no slave device is connected.
5. Connect DC power supply to J1 on MCU board (the LED1 and LED2 on the MCU board and LED1 on 9665 will be turned ON).
6. Connect mbed to PC via USB cable.

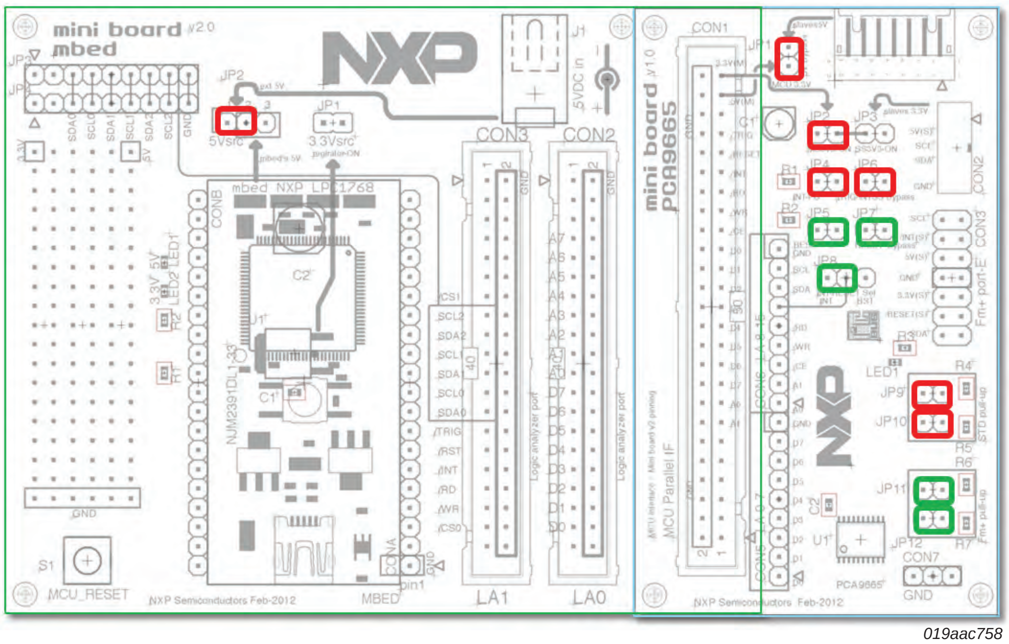
This PC connection is not required after executable file copied to mbed. The mbed runs as standalone while it has the file inside.

The PC connection (USB) can be used for power supply if the slave board does not require much current. The mbed 5 V output (VU pin) can share a few hundred mA of USB bus supply. Short pin2 and pin3 on JP2 of MCU board when the setup uses the mbed 5 V output.



PCA9665 should have I²C-bus address = 0xC0.

Fig 16. Board connection (for 9665)



RED jumper positions must be shorted.

Opened jumpers should be left open.

GREEN jumpers are optional.

Fig 17. Default jumper settings

3.6.2.2 Preparation (software)

Software setup is same as setup of 9669. Follow section Section 3.6.1.2, but be sure the configuration header file ‘config.h’ file has right target setting. Un-comment the line of ‘`#define CODE_FOR_PCA9665`’ and comment-out all other lines.

3.6.2.3 I²C-bus transfer on PCA9665

I²C-bus transfer will be repeated with 10 ms interval on PCA9665.

On the I²C, 42 bytes write, 1 byte write and 41 bytes read will be performed. These transactions are doing the PCA9955 registers write from MODE1 to OFFSET and read back (see Figure 11). The second and third transaction will be started with RESART condition.

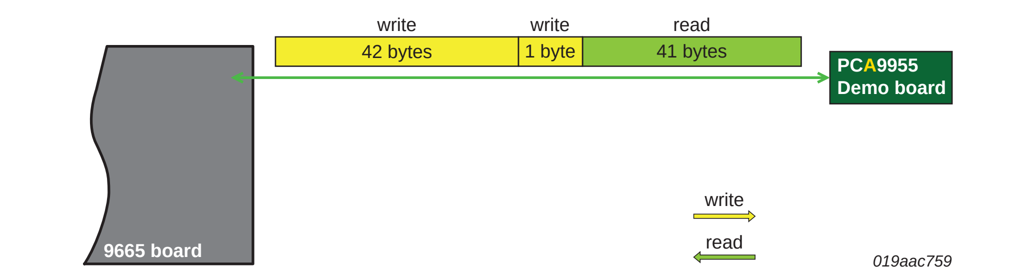
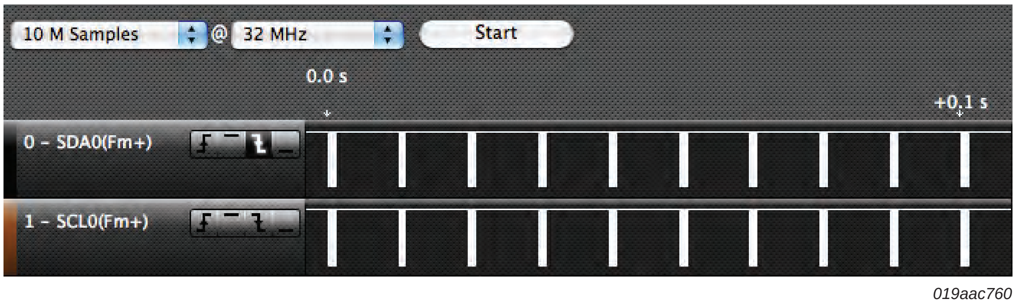
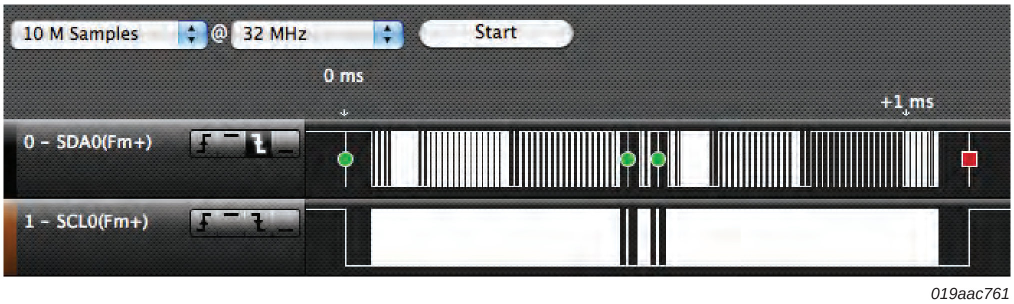


Fig 18. Sample code I²C-bus transfer on PCA9665



Transfer executed with 10 ms interval.

Fig 19. I²C-bus transfer of sample code (PCA9665)



Waveform with different time span from Figure 19.

Green dot shows START/RESTART conditions.

Red square shows STOP conditions.

Fig 20. I²C-bus transfer of sample code (PCA9665)

4. Hardware

4.1 Overview

The mini board PCU9669 evaluation board kit is populated with the following three boards:

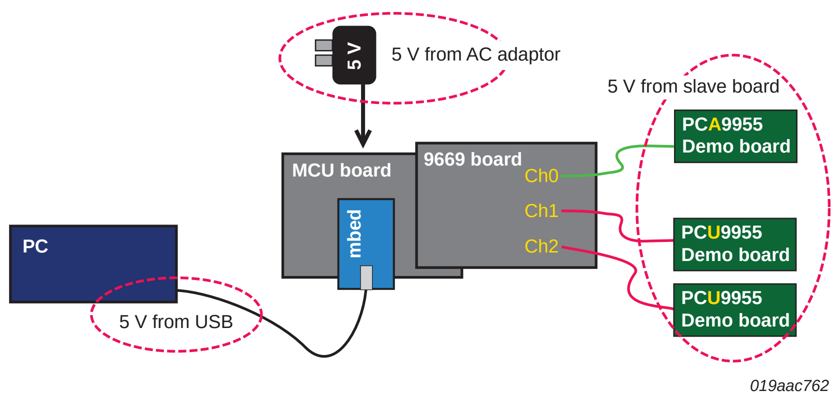
- Mini board PCU9669 (9669 board)
- Mini board PCA9665 (9665 board)
- Mini board mbed (MCU board)

Bus controller boards (9669 board and 9665 board) work with the MCU board. All of these boards have MCU interface port to connect each other.

For evaluation of PCU9669, use ‘mini board PCU9669’ and ‘mini board mbed’ (setup described in Section 3.6.1). For 9665, use ‘mini board PCU9669’ and ‘mini board mbed’ (see Section 3.6.2) for the setup.

4.1.1 Power supply

Mini board kit can work with single 5 V supply. The power can be taken from MCU module, DC connector on MCU board or I^{2}C connector on bus controller board. 3.3 V supply for the bus controller is provided from MCU board.



Three sources from which the 5 V supply can be taken.

Fig 21. Single 5 V supply required

The power supply configuration should be set properly by jumpers on MCU board and bus controller board.

12 V and 24 V supplies on the I^{2}C-bus connectors are not connected to the bus controller circuit. Those high-voltage lines are interconnected between the I^{2}C-bus connectors.

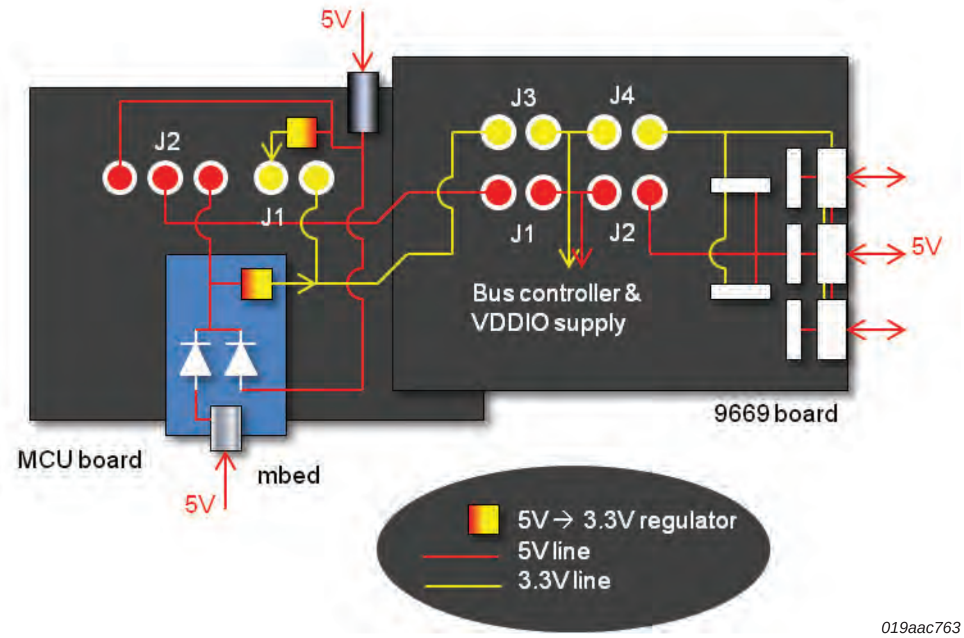


Fig 22. 5 V, 3.3 V lines and jumpers on MCU and 9669 boards

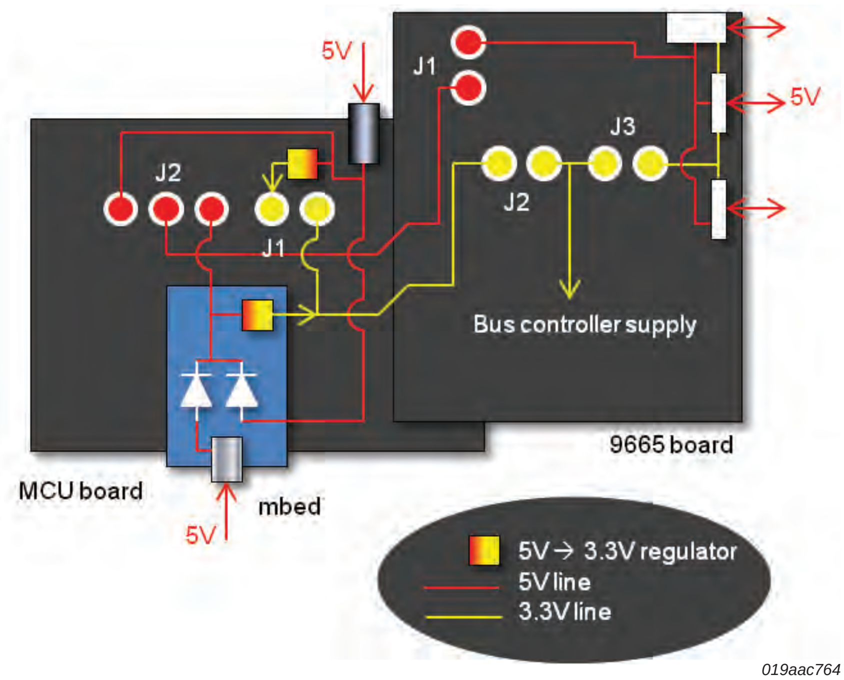


Fig 23. 5 V, 3.3 V lines and jumpers on MCU and 9665 boards

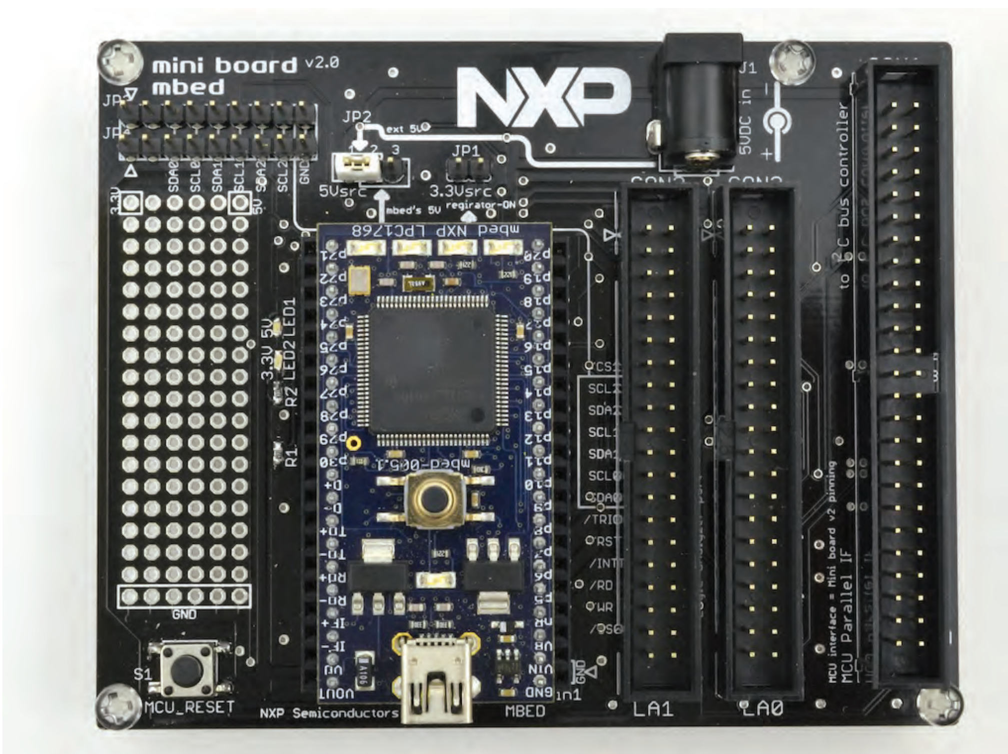
Table 1. Jumper settings for 5 V source

Board	Jumper	DC connector (J1, MCU board)	Slave device board	USB on MCU module
MCU board	JP1	open[1]	open[1]	open[1]
	JP2	pin 2-3 short	pin 2-3 short	pin 1-2 short
9669 board	JP1	short	short	short
	JP2	if necessary[2]	short	if necessary[2]
	JP3	short	short	
	JP4	open	open	
9665 board	JP1	if necessary[2]	short	if necessary[2]
	JP2	short	short	short
	JP3	if necessary[2]	short	if necessary[2]

[1] The jumper pin JP1 on the MCU board should be always open while the MCU module is supplying 3.3 V. 3.3 V is not available when the LPCXpresso is used without LPC-link (JTAG debugger portion). In this case, short the JP1 (on the MCU board) to supply the 3.3 V from the regulator on the MCU board.

[2] Be careful to prevent supply conflict. Also, when supplying to slave boards be sure that there is enough power capacity for all the slave boards that are attached.

4.2 Mini board mbed (MCU board)



019aac765

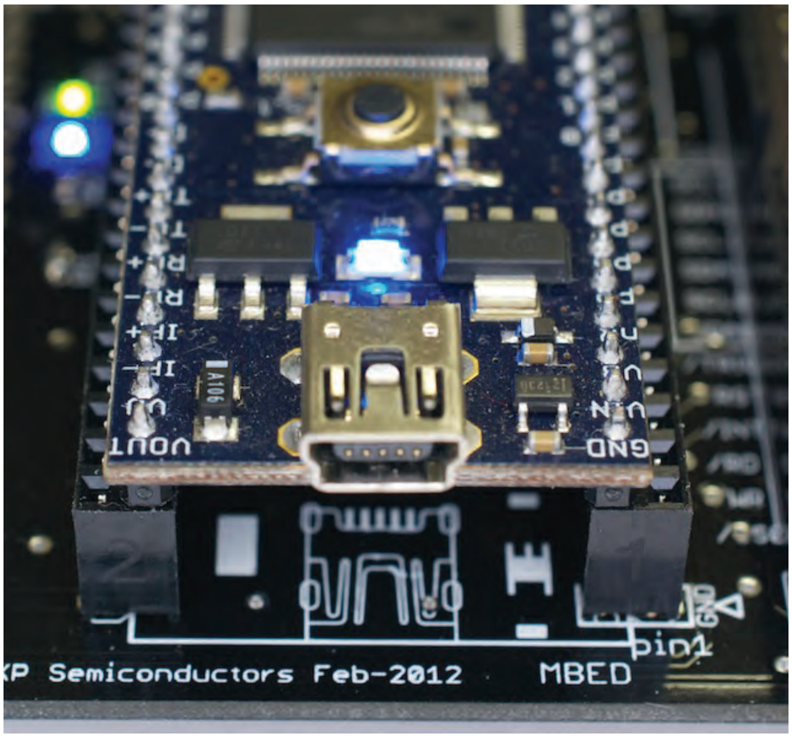
MCU board with mbed.

Fig 24. Mini board MCU (MCU board)

4.2.1 MCU module

The ‘mini board mbed’ (this will be called ‘MCU board’ in following sections) is a baseboard for an MCU module. The mini board kit uses an ‘mbed NXP LPC1768’ or LPCXpresso-LPC1768/LPC1769 (see Section 5.1 for how the LPCXpresso can be used).

The MCU board has MCU module socket: 2 times 20 $\times$ 2 header sockets. Inner rows of each of the 20 pins are used to connect the MCU module. The outer side of this slot is available for probing the signal.



019aac766

Use inner row to connect the MCU module.

Fig 25. MCU module socket

The MCU is powered from the USB connector on its module or from the DC connector on this MCU board.

The mini board kit can be operated by supply from the MCU module because the mbed can provide 5 V and 3.3 V supplies. However, those supply capacities are limited (a few hundred mA for each), so they are not capable of supplying directly high current LED slave boards. If high current is required, the supply should be taken from external DC supply through DC jack (J1) or slave side through bus controller boards.

4.2.2 Connectors

The MCU board has several connectors to interface to the bus controller board and measurement equipment.

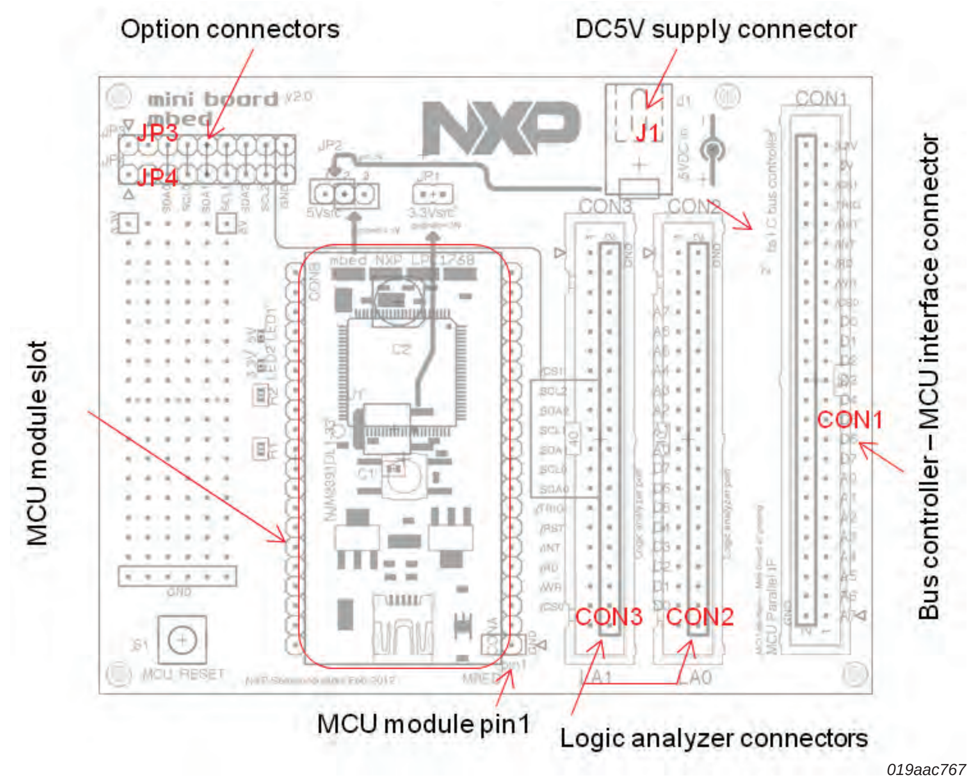


Fig 26. MCU board connectors

4.2.2.1 Bus controller - MCU interface connector

A MIL-50pin connector (CON1) is available to connect bus controller board. This connector contains parallel bus, reset, interrupt, trigger/external interrupt signals and 5 V and 3.3 V supply lines.

All logic signals are at 3.3 V level.

Table 2. Bus controller - MCU interface connector pinning

Pin number[1]	Signal name	Description	Direction
1	A7	Address bus (8 bits)	From MCU to bus controller
3	A6		
5	A5		
7	A4		
9	A3		
11	A2		
13	A1		
15	A0		
17	D7	Data bus (8 bits)	Bidirectional
19	D6		
21	D5		
23	D4		
25	D3		
27	D2		
29	D1		
31	D0		
33	$\overline{\text{CS}}$	Chip select (active LOW)	From MCU to bus controller
35	$\overline{\text{WR}}$	Write strobe (active LOW)	From MCU to bus controller
37	$\overline{\text{RD}}$	Read strobe (active LOW)	From MCU to bus controller
39	$\overline{\text{INT}}$	Interrupt (active LOW)	From bus controller to MCU
41	$\overline{\text{RESET}}$	Reset (active LOW)	Bidirectional
43	TRIG	Trigger signal output / External interrupt (EINT) input	Bidirectional
45	$\overline{\text{CS1}}$	Auxiliary chip select signal (not used in this version)	
47	+3V3	3.3 V power supply line	
49	+5V	5 V power supply line	

[1] All even-numbered pins are GND.

4.2.2.2 Optional connectors

CON2 and CON3 are the Logic Analyzer (LA) ports. This allows logic analyzer direct connect if the interface is matched.

JP3 and JP4 are optional connectors to connect bus controller board. Since the MCU interface connector does not include the I^{2}C-bus signal from bus controller, JP3 or JP4 connectors can be used to receive those signals from the 9669 board. These connector signals are connected to LA1 port (CON3). With jump wires from CON9 of the PCU9669 board to one of these connectors, the logic analyzer will have I^{2}C-bus signals on its port.

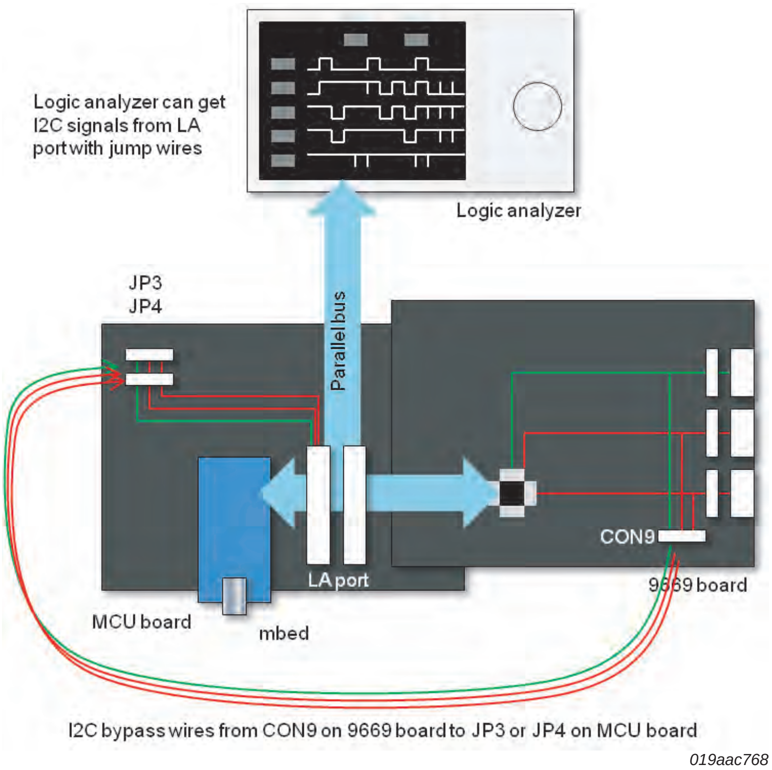


Fig 27. Putting I^{2}C-bus signals on logic analyzer ports

4.2.3 Jumpers and switch

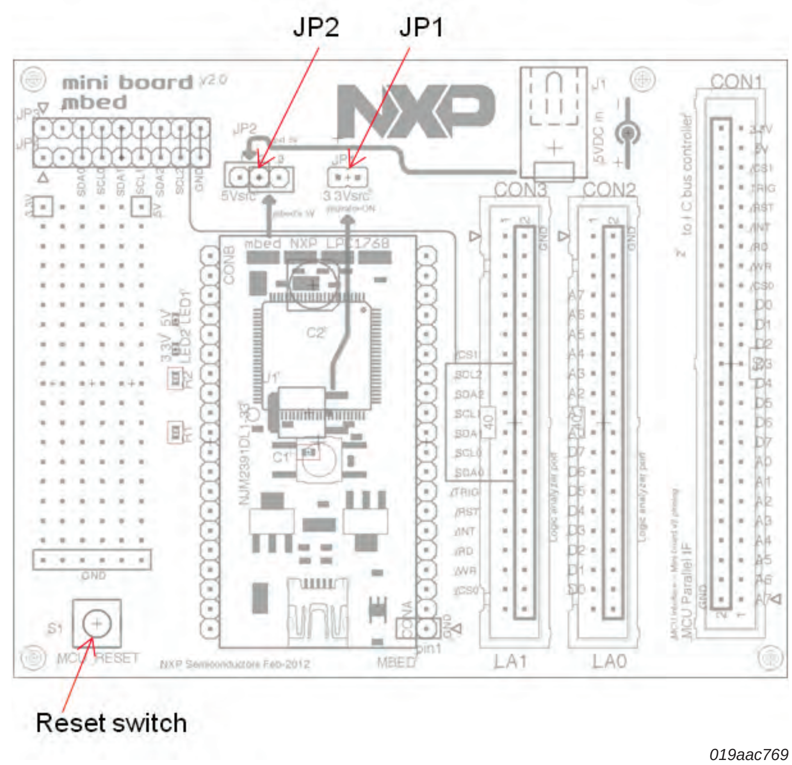


Fig 28. Jumpers and switch on the MCU board

Supply configuration can be changed by jumper pins (Table 3). Power setting summary is available in Table 1 (in Section 4.1.1 "Power supply").

Table 3. MCU board jumper settings

MCU board power supply setting.

Jumper	State	Description
JP1	open (default)	3.3 V is from MCU module
	short	3.3 V is from regulator[1]
JP2	short pin 1 and pin 2 (default)	Use 5 V supply from DC jack (J1) or bus controller board
	short pin 2 and pin 3	Use 5 V supply from MCU board

[1] mbed and LPCXpresso (with LPC-Link) have a 3.3 V regulator on that module, but if the LPCXpresso as target itself (without LPC-Link), short this JP1 to use regulator on MCU board.

4.2.3.1 5 V supply

Short pin 1 and pin 2 on JP2 when the 5 V supply is from the DC jack or bus controller board. Shorting pin 2 and pin 3 is a setting to use 5 V supply from the MCU module (supplied from the PC through USB).

When the 5 V supply is available, LED1 (green LED) on the MCU board will be turned ON.

4.2.3.2 3.3 V supply

The MCU board does not require 3.3 V supply. MCU works with 5 V only, but the bus controller boards require the 3.3 V supply. The MCU modules or the MCU board can supply this 3.3 V for the bus controller board.

JP1 is an option setting for this 3.3 V output. The mbed and LPCXpresso with LPC-Link can feed 3.3 V. However, if LPCXpresso without LPC-Link (target only), short the JP1 to use the regulator output on the MCU board.

The LED2 (blue LED) will be turned ON while the 3.3 V supply is ON.

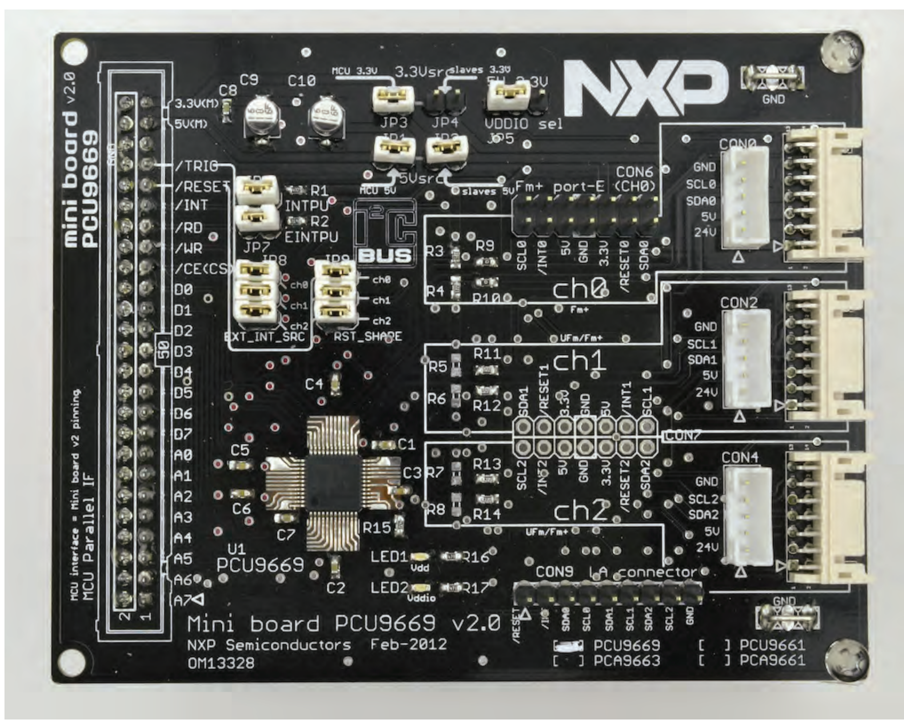
4.2.3.3 Reset switch

A switch is available to reset the MCU. This has the same function of the switch which is on the mbed. Since the LPCXpresso does not have reset button on the board, this external switch can be used to reset it without having to power cycle the LPCXpresso.

4.3 Mini board PCU9669 (9669 board)

Mini board PCU9669 (9669 board) has a PCU9669 as an I^{2}C-bus controller that bridges MCU interface parallel port to 3-channel I^{2}C-bus (channel 0 is Fm+, channel 1 and channel 2 are UFm). Those I^{2}C-bus signals are available in 5-pin, 14-pin connectors and 14-pin header with supply lines (header pin connector is available on Fm+ channel only).

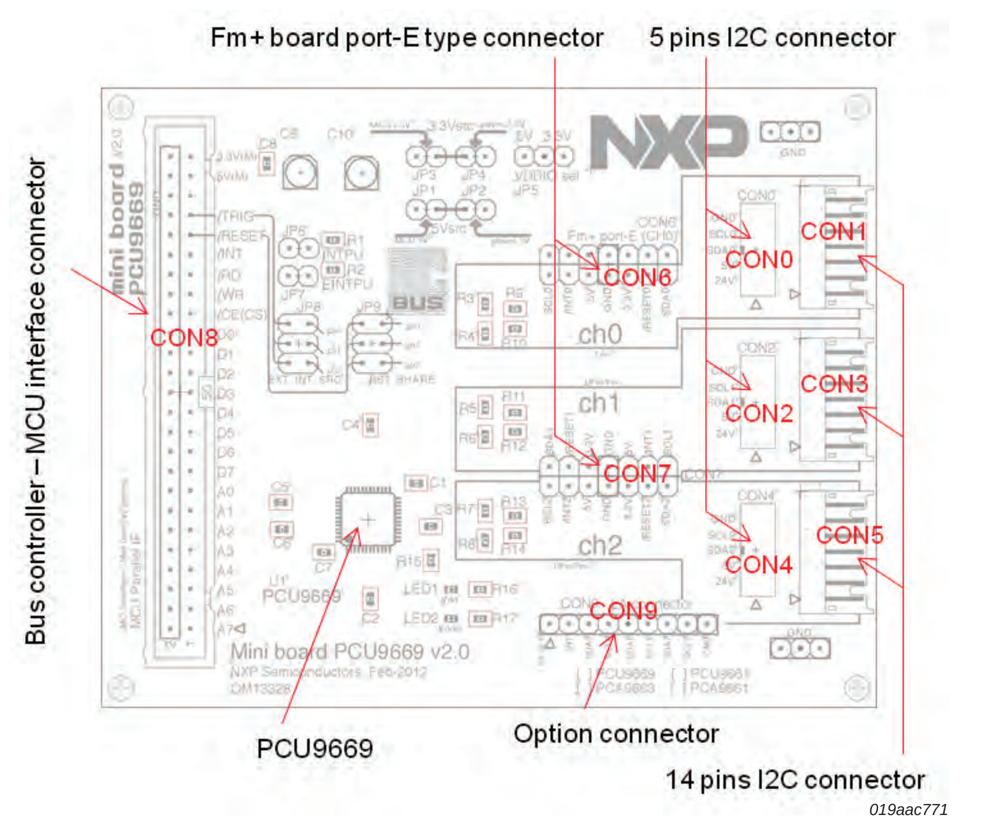
The PCA9663 board can be made with different component options. The PCA9663 has 3-channel Fm+ I^{2}C-bus, so the components on channel 1 and channel 2 have the same components as channel 0 (will be discussed in Section 4.3.3.2).



019aac770

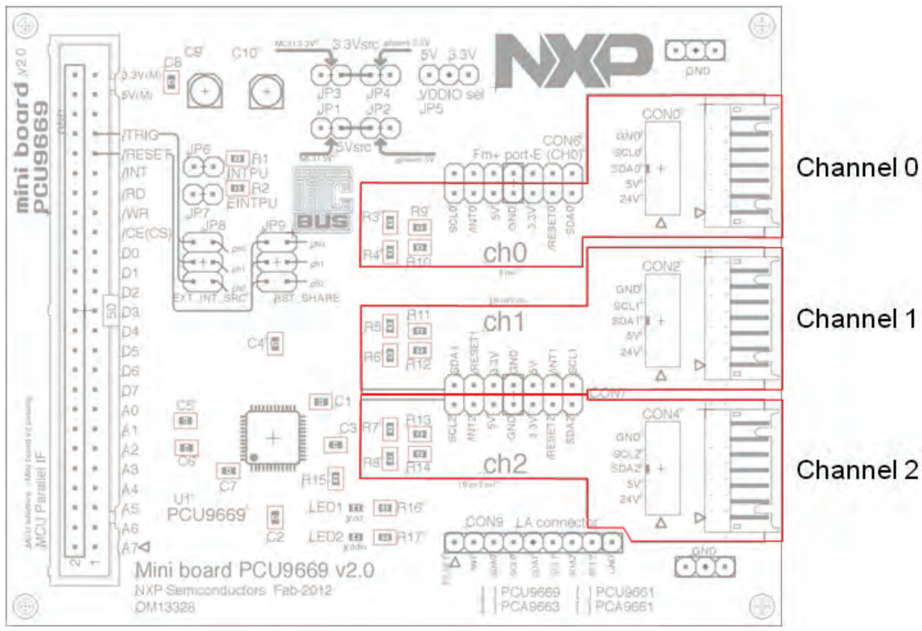
Fig 29. Mini board PCU9669 (9669 board)

4.3.1 Connectors



CON7 is not available on PCU9669.

Fig 30. 9669 board connectors



019aac772

Channel 0: Fm+

Channel 1 and channel 2: UFm

Fig 31. Channels on the 9669 board

4.3.1.1 Bus controller - MCU interface connector

The MCU interface connector is available as CON8. This connector is intended to connect to the MCU board's CON1 or user MCU target board's parallel bus signals. Pin list is shown on Table 2 in section Section 4.2.2.1.

4.3.1.2 I^{2}C-bus interface connectors

All three channels of I^{2}C-bus signals are available on two types of connectors with supply lines.

Each channel has 5 pins (JST PH type) and 14 pins (JST PHD type) connectors. The 5-pin connectors have I^{2}C-bus signals (SDA and SCL) with GND, 5 V and 24 V supply lines.

14-pin connectors have I^{2}C-bus signals with GND, 5 V, 3.3 V, 12 V and 24 V supply lines, as well as /RESET and /INT signals.

The 9669 board can provide 5 V and 3.3 V supplies to slave boards by jumper settings. If user needs 12 V or 24 V at slave boards, that needs to be managed by slave side. These high-voltage supply lines are connected to each other across the channels.

/RESET and /INT signals can be connected to the MCU side with jumper settings.

Fm+ I^{2}C-bus channels (for channel 0 of PCU9669 and all channels of PCA9663) have 'Fm+ board port-E type connectors'. Those connectors can be used when trying to connect the Fm+ version 2 board.

Table 4. Signals on connector pins: 5-pin I²C-bus connector

Connector - JST PH type: B5B-PH-K-S

Pin number	Signal name
1	24 V supply
2	5 V supply
3	SDA
4	SCL
5	GND

Table 5. Signals on connector pins: 14-pin I²C-bus connector

Connector - JST PHD type: S14B-PHDSS

Pin number	Signal name
1	24 V supply
2	24 V supply
3	GND
4	GND
5	5 V supply
6	3.3 V supply
7	SDA
8	/SLAVE_RESET
9	GND
10	/SLAVE_INT
11	SCL
12	GND
13	12 V supply
14	12 V supply

Table 6. Signals on connector pins: Fm+ board port-E type header

14-pin header

Pin number	Pin number	Signal name
1	8	SCL
2	9	/SLAVE_INT
3	10	5 V supply
4	11	GND
5	12	3.3 V supply
6	13	/SLAVE_RESET
7	14	SDA

4.3.1.3 Optional connector

CON9 is a connector for a logic analyzer or oscilloscope probes.

This CON9 can be used to connect JP3 or JP4 of MCU board (see section Section 4.2.2.2).

4.3.2 Jumper settings

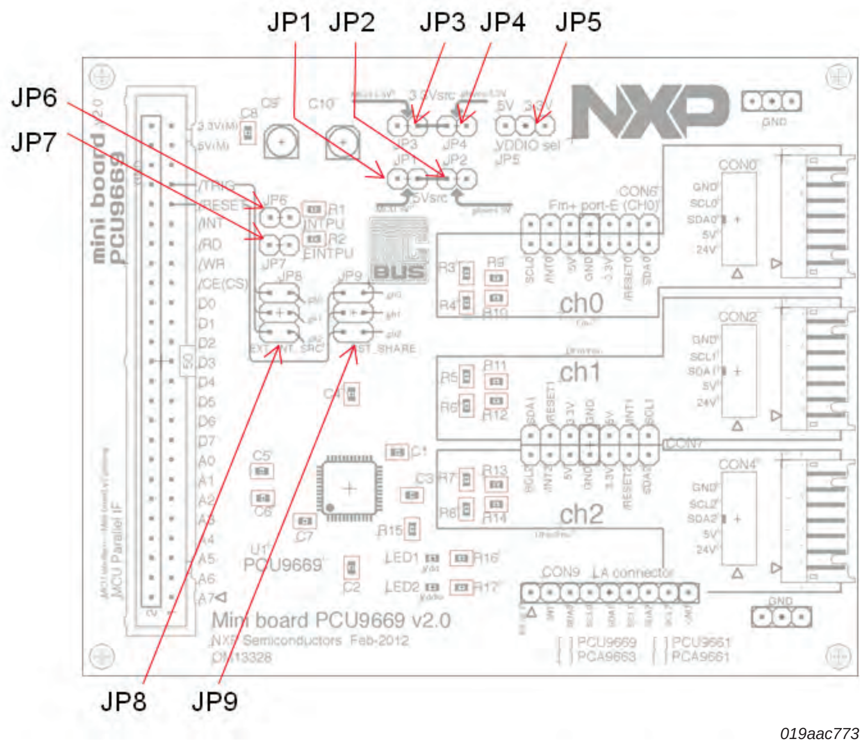


Fig 32. Jumpers on 9669 board

4.3.2.1 Power supply

The PCU9669 board can get power supply from the MCU interface connector or I^{2}C-bus connector (power setting summary is available in Table 1 in section Section 4.1.1)

5 V: JP1 and JP2 are setting for 5 V supply, shorting JP1 connects the 5 V supply to MCU side. Shorting JP2 connects 5 V to I^{2}C-bus connectors.

Default setting of this board is both JP1 and JP2 are shorted. This means the MCU side and I^{2}C-bus connectors' 5 V supplies are tied together. This setting can be used when the MCU provides supply for I^{2}C slaves, or I^{2}C slave supplies 5 V to MCU.

3.3 V: JP1 and JP2 are setting for 3.3 V supply, shorting JP1 connects the 3.3 V supply to MCU side. Shorting JP2 connects 5 V to I^{2}C-bus.

$V_{DD(IO)}$: PCU9669 can have 5 V or 3.3 V interface voltage by setting of its $V_{DD(IO)}$ pin. This interface voltage can be selected by JP5.

Pull-up: For $\overline{\text{INT}}$ and $\overline{\text{TRIG}}$/EINT signals' pull-up resistors (pull-up to 3.3 V) can be set by JP6 and JP7. See following section for $\overline{\text{INT}}$ and $\overline{\text{TRIG}}$/EINT signals.

For pull-up resistors of I^{2}C-bus signals, see Section 4.3.3.2.

Table 7. 9669 board jumper settings: power supply and pull-up

Power supply and pull-up settings.

Jumper	Setting	Description
JP1	open	Disconnect 5 V of MCU side
	short (default)	Connect 5 V of MCU side
JP2	open	Disconnect 5 V of I^2C-bus connectors
	short (default)	Connect 5 V of I^2C-bus connectors
JP3	open	Disconnect 3.3 V of MCU side
	short (default)	Connect 3.3 V of MCU side
JP4	open (default)	Disconnect 3.3 V of I^2C-bus connectors
	short	Connect 3.3 V of I^2C-bus connectors
JP5	short pin 1 and pin 2 (default)	Set $V_{DD(IO)}$ as 5 V
	short pin 2 and pin 3	Set $V_{DD(IO)}$ as 3.3 V
JP6	open	Pull-up resistor for $\overline{\text{INT}}$ signal is OFF
	short (default)	Pull-up resistor for $\overline{\text{INT}}$ signal is ON
JP7	open	Pull-up resistor for EINT signal is OFF
	short (default)	Pull-up resistor for EINT signal is ON

4.3.2.2 $\overline{\text{RESET}}$, $\overline{\text{INT}}$ signals

$\overline{\text{RESET}}$ signals for PCU9669 can be shared with I^2C-bus slave devices. This setting can be done channel-by-channel on JP9.

Table 8. 9669 board jumper settings: $\overline{\text{RESET}}$ and $\overline{\text{INT}}$ signals

$\overline{\text{RESET}}$ and $\overline{\text{INT}}$ signals: default = all pins short

Jumper	Setting	Description
JP8	short pin 1 and pin 2	Connect $\overline{\text{RESET}}$ signal to slaves on channel 0
	short pin 3 and pin 4	Connect $\overline{\text{RESET}}$ signal to slaves on channel 1
	short pin 5 and pin 6	Connect $\overline{\text{RESET}}$ signal to slaves on channel 2
JP9	short pin 1 and pin 2	Connect $\overline{\text{INT}}$ (EINT) signal from slaves on channel 0
	short pin 3 and pin 4	Connect $\overline{\text{INT}}$ (EINT) signal from slaves on channel 1
	short pin 5 and pin 6	Connect $\overline{\text{INT}}$ (EINT) signal from slaves on channel 2

4.3.3 Signals

4.3.3.1 MCU bus signal levels

The PCU9669 parallel interface logic level is +4.6 V maximum. So, make sure that the logic level is compatible if user going to use a different MCU interface to the 9669 board.

4.3.3.2 I^2C-bus

UFm channels: Since the UFm signals are push-pull outputs, those lines have no pull-up resistor.

With PCU9669, the resisters R5, R6, R7 and R8 are left open, because those resisters are pull-ups for channel 1 and channel 2 I^2C-bus lines.

The resistors R11, R12, R13 and R14 are series resistors on I^{2}C-bus lines. Those are impedance-matching resistors for UFm bus lines. The 9669 board has 22 Ω resistors for those.

Fm+ channel(s): Fm+ lines require pull-ups. 220 Ω resistors are used on R3 and R4 at PCU9669 board (if the PCA9663 is mounted, all resistors from R3 to R8 are 220 Ω).

The resistors may be replaced if the user needs different values to adjust I^{2}C-bus driving current.

Series resistors on the I^{2}C-bus lines are 0 Ω to short the lines from the PCU9669 to connectors.

Table 9. Resistor value difference by bus controller

Resistor	Function	PCU9669	PCA9663
R3	SCL pull-up for channel 0	220 Ω	220 Ω
R4	SDA pull-up for channel 0	220 Ω	220 Ω
R5	SCL pull-up for channel 1	open	220 Ω
R6	SDA pull-up for channel 1	open	220 Ω
R7	SCL pull-up for channel 2	open	220 Ω
R8	SDA pull-up for channel 2	open	220 Ω
R9	SCL series resistor for channel 0	0 Ω	0 Ω
R10	SDA series resistor for channel 0	0 Ω	0 Ω
R11	SCL series resistor for channel 1	22 Ω	0 Ω
R12	SDA series resistor for channel 1	22 Ω	0 Ω
R13	SCL series resistor for channel 2	22 Ω	0 Ω
R14	SDA series resistor for channel 2	22 Ω	0 Ω

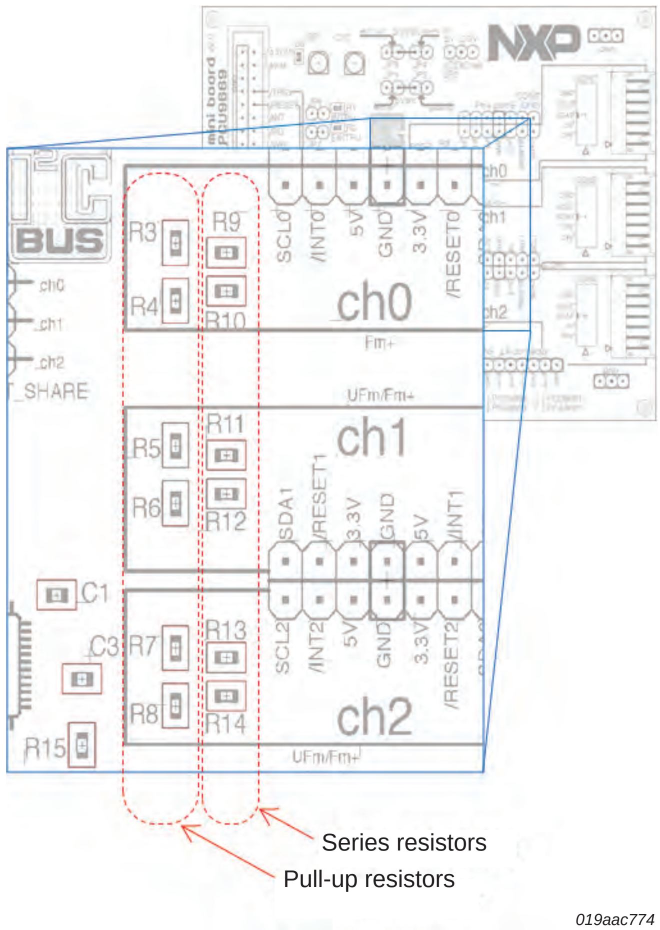


Fig 33. I²C-bus pull-up and series resistors

$\overline{\text{INT}}$: The $\overline{\text{INT}}$ signal from PCU9669 is connected to MCU with 4.7 kΩ pull-up resistor. The pull-up resistor can be disabled by JP6 open.

$\overline{\text{RESET}}$: $\overline{\text{RESET}}$ is given from MCU as open-drain signal with MCU internal pull-up. So the signal can be driven from slave side, too.

This $\overline{\text{RESET}}$ signal does not reset the PCU9669 only, but the slaves also when those are connected by JP9 jumpers.

If the reset signal is shared with slave devices, adjust the reset pulse width and reset recovery time in the software.

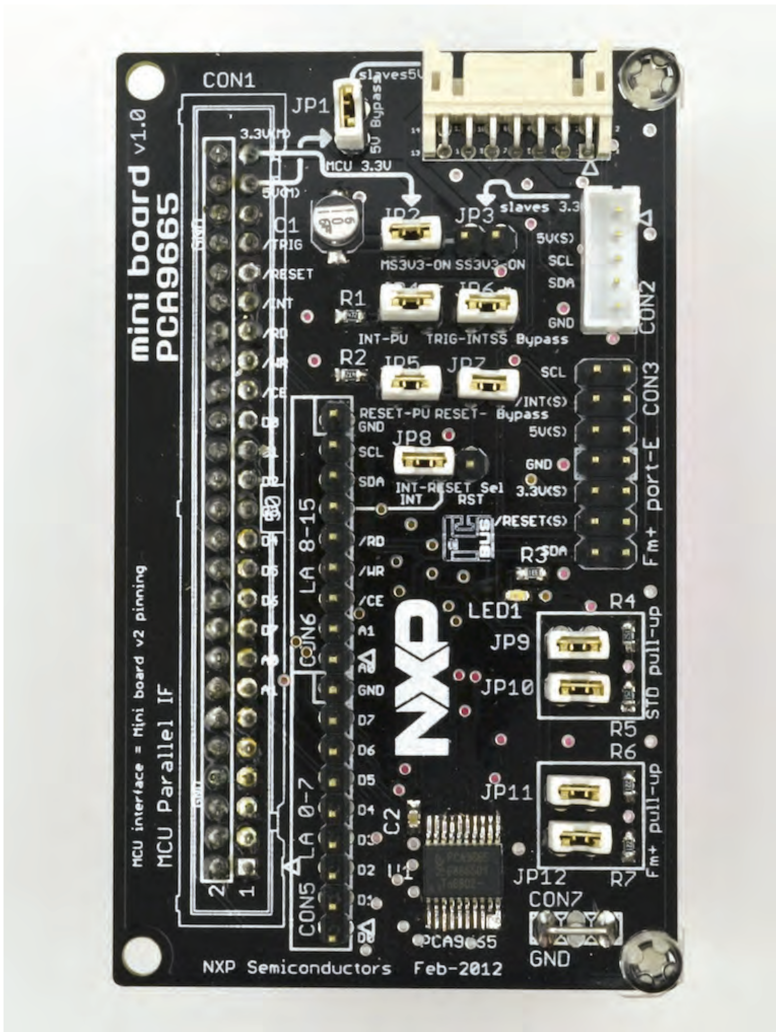
$\overline{\text{TRIG}}$ / EINT: '$\overline{\text{TRIG}}$' is a trigger signal from MCU to PCU9669. This can start the I²C-bus transfer if the channel is set for $\overline{\text{TRIG}}$ signal. The $\overline{\text{TRIG}}$ signal is an advanced feature of PCU9669. This enables the synchronization of the I²C-bus transfer to external timing source (like video-sync signal). The trigger function can be enabled/disabled by (PCU9669's) channel register setting.

The sample code (software) has function to drive this, but disabled in default.

While the $\overline{\text{TRIG}}$ signal is not used, this line can be used for forwarding slave's interrupts to MCU. This is the 'EINT' (External INT) signal. The slave's interrupt signals can be connected to MCU via JP8 settings. This is a limitation of this evaluation kit since the signal line is shared by $\overline{\text{TRIG}}$ and external interrupt.

4.4 Mini board PCA9665 (9665 board)

Mini board PCU9665 (9665 board) has a PCU9665 as a bus controller that bridges MCU interface parallel port to an Fm+ I^{2}C-bus. Those I^{2}C-bus are available in 5-pin or 14-pin connectors and 14-pin header with supply lines.



019aac775

Fig 34. Mini board PCA9665 (9665 board)

4.4.1 Connectors

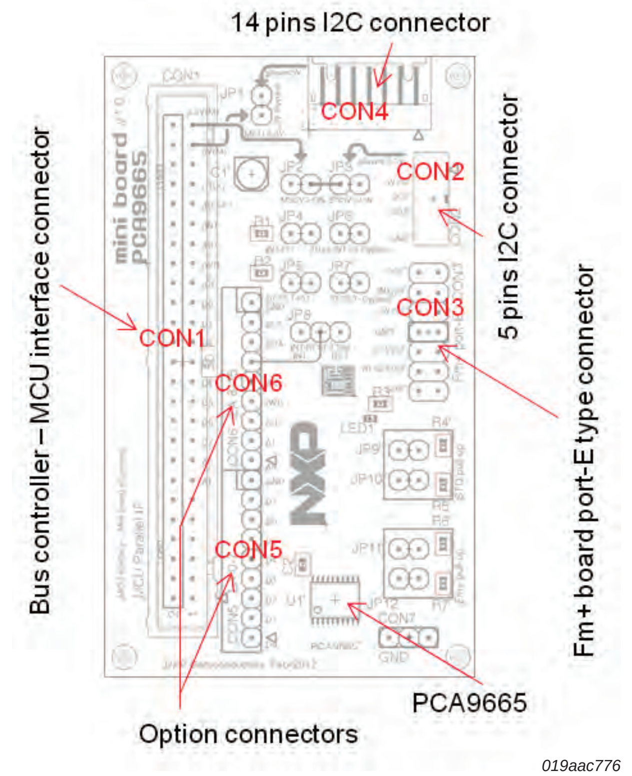


Fig 35. 9665 board connectors

4.4.1.1 Bus controller - MCU interface connector

MCU interface connector is available on CON1. This connector is intended to connect to the MCU board's CON1 or user MCU target board's MCU signals. Pin list is available in Table 2 in Section 4.2.2.1.

4.4.1.2 I^{2}C-bus interface connectors

I^{2}C-bus signals are available on three types of connectors with supply lines.

5-pin (JST PH type) and 14-pin (JST PHD type) connectors. The 5-pin connectors have two I^{2}C-bus signals (SDA and SCL) with GND, 5 V and 24 V supply lines.

Remark: Signal labels SDA and SCL are wrong on CON2 (9665 board version 1.0).

14-pin connectors have I^{2}C-bus signals (SDA and SCL) with GND, 5 V, 3.3 V, 12 V and 24 V supply lines, as well as $\overline{\text{RESET}}$ and $\overline{\text{INT}}$ signals.

Fm+ board port-E type connector is available also. This connector can be used when trying to connect the Fm+ version 2 board.

$\overline{\text{RESET}}$ and $\overline{\text{INT}}$ signals can be connected to the MCU side with jumper settings.

All pinning is compatible with the PCU9669 board (see Table 4, Table 5 and Table 6 in Section 4.3.1.2).

4.4.1.3 Optional connectors

CON5 and CON6 are connectors for a logic analyzer.

4.4.2 Jumper settings

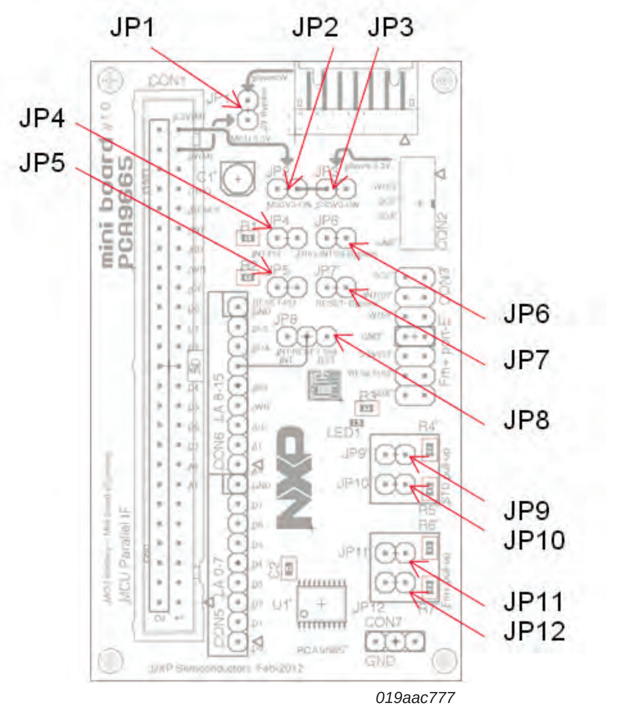


Fig 36. Jumpers on 9665 board

4.4.2.1 Power supply

The 9665 board can get power supply from the MCU interface connector or I^{2}C-bus connector. (Power setting summary is available in Table 1 in Section 4.1.1).

5 V: PCA9665 does not require 5 V supply. To bypass the 5 V line between the MCU connector and I^{2}C-bus connector, short JP1.

3.3 V: JP2 and JP3 are setting for 3.3 V supply. Shorting JP2 connects the 3.3 V supply to the MCU side. Shorting JP3 connects 5 V to the I^{2}C-bus connectors.

$V_{DD(IO)}$: There is no $V_{DD(IO)}$ available because PCA9665 does not support different I/O voltages.

Pull-up for I^{2}C-bus signals: The I^{2}C-bus Standard-mode and Fm+ have different current driving capabilities. For those differences, the 9665 board can select pull-up resistors by jumper pins.

JP9 and JP10 is ON/OFF jumper for Standard-mode pull-up resistors. 1.5 kΩ resistors can be used as SDA and SCL signal lines (on schematic, those are shown as 1.2 kΩ).

JP11 and JP12 are for Fm+. 220 Ω (those are shown as 200 Ω on the schematic) resistors can be used when those jumpers are ON.

When all JP9 to JP12 jumpers are shorted, the pull-up resistor will be 1.5 kΩ // 220 Ω = 192 Ω. This makes the I²C-bus signal current 17 mA at 3.3 V.

Pull-up for $\overline{\text{RESET}}$ and $\overline{\text{INT}}$: Jumper pins are available to choose pull-up resistors ON/OFF for $\overline{\text{RESET}}$ and $\overline{\text{INT}}$ signals.

JP4 and JP5 can enable pull-up resistors for $\overline{\text{INT}}$ and $\overline{\text{RESET}}$ signals.

All jumper settings and descriptions are available in Table 10 and Table 11.

Table 10. Jumper setting for supplies and signals

Jumper	State	Description
JP1	open	Disconnect 5 V line between MCU and I²C-bus connectors
	short (default)	Connect 5 V line between MCU and I²C-bus connectors
JP2	open	Disconnect 3.3 V of MCU side
	short (default)	Connect 3.3 V of MCU side
JP3	open (default)	Disconnect 3.3 V of I²C-bus connectors
	short	Connect 3.3 V of I²C-bus connectors
JP4	open	Pull-up resistor for $\overline{\text{INT}}$ signal is OFF
	short (default)	Pull-up resistor for $\overline{\text{INT}}$ signal is ON
JP5	open	Pull-up resistor for $\overline{\text{RESET}}$ signal is OFF
	short (default)	Pull-up resistor for $\overline{\text{RESET}}$ signal is ON
JP6	open	Disconnect $\overline{\text{INT}}$ signal to I²C-bus connectors
	short (default)	Connect $\overline{\text{INT}}$ signal to I²C-bus connectors
JP7	open	Disconnect $\overline{\text{RESET}}$ signal to I²C-bus connectors
	short (default)	Connect $\overline{\text{RESET}}$ signal to I²C-bus connectors
JP8	short pin left and middle (default)	CON6 pin6 connect to $\overline{\text{INT}}$ signal
	short pin middle and right	CON6 pin6 connect to $\overline{\text{RESET}}$ signal

Table 11. Jumpers for I²C-bus pull-up resistors

Jumper	Description
JP9	Short to enable pull-up 1.5 kΩ for SDA
JP10	Short to enable pull-up 1.5 kΩ for SCL
JP11	Short to enable pull-up 220 Ω for SDA
JP12	Short to enable pull-up 220 Ω for SCL

5. Software (sample code)

5.1 Availability

The latest version of the sample code is available on mbed.org site:

http://mbed.org/users/nxp_ip/programs/mini_board_PCU9669/latest

This is a complete sample code. All code can be browsed on this page. Also the code can be imported to user compiler page by just a few clicks. Code importing was explained in Section 3.6.1.3.

The details of the library interface will not be discussed in this document. If required, please refer to comments in each header file. Descriptions in the files are available in Doxygen format.

The sample code which is built by mbed compiler can run on LPCXpresso-LPC1768/LPC1769 too. Next URL explains how to program the mbed code into LPCXpresso:

http://mbed.org/users/nxpfan/notebook/mbed-led-blink-code-on-lpcxpresso-lpc1768/

019aac778

mbed code can run on both mbed and LPCXpresso-LPC1768/LPC1769

Fig 37. mbec / LPCXpresso-LPC1769 on the MCU board

5.2 Software structure

5.2.1 Overview

The sample code of this evaluation kit has been made to demonstrate functionality of PCU9669 and PCA9665 in a quick and easy way.

The mbed is used for an MCU to control the bus controllers because it has quite unique features to help in building/sharing code. Also, the mbed gives advantage to the evaluation kit by powerful MCU (LPC1768: ARM Cortex-M3 running in 96 MHz, 32KB-SRAM/512K-flash) and libraries.

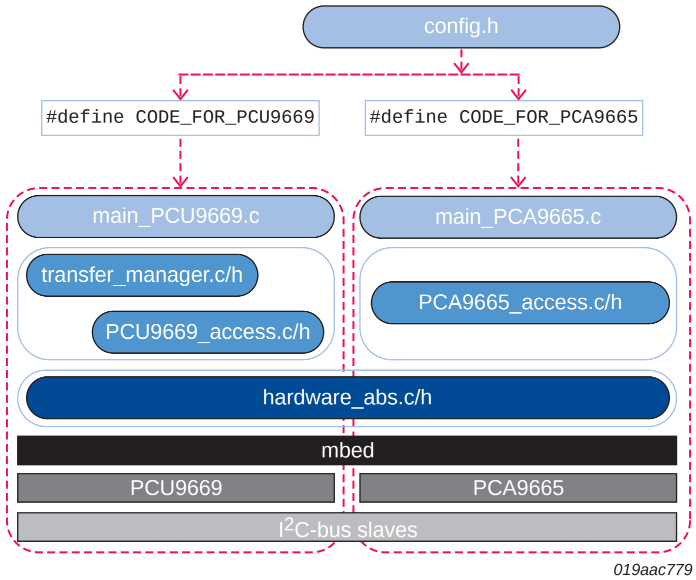
However, unfortunately, the mbed does not have parallel bus to connect the bus controllers. On this evaluation kit, GPIO is used to emulate the parallel bus. The GPIO access is slow compared to normal parallel bus. The emulated bus has about 1 MHz to 3 MHz cycle frequency. It may be a fair speed to have bus controller evaluation. Detailed information is available in Section 8).

The program is composed from several modules (files/libraries) in the program. Each of the modules make software layers. Layers are abstracting lower layers.

This program includes codes for both PCU9669 and PCA9665. Those modules are switched by compile switch in ‘config.h’. The word ‘`CODE_FOR_PCU9669`’ enables code in ‘main_PCU9669.c’. The word ‘`CODE_FOR_PCA9665`’ enables the code in ‘main_PCA9665.c’. All libraries will be compiled together, but un-used code will be un-linked.

All code except hardware abstraction layer is written in ANSI-C standard format. This helps to easily port the code. Only the hardware abstraction layer has hardware and environment-related operations.

The structure is shown in Figure 38.



The program includes code for both targets.

The target is managed by word in ‘config.h’ (switch by compile option).

Two different executable binaries can be built by this compile option.

Fig 38. Target switching and modules

5.2.2 Hardware abstraction layer

Hardware abstraction is done on ‘parallel_bus’ library. This library emulates parallel bus. Abstracting the GPIO access and interrupt service routines.

All ‘mbed’ dependents are encapsulated in this library. This module is the main part of the modification if the code is needed to be ported.

5.2.3 PCU9669 access layer

The PCU9669 hardware access and transaction/sequence can be managed in this library ‘PCU9669’. This library has two modules, one is ‘PCU9669_access’ that abstracts the access of the register and memory access. Another is ‘transfer_manager’ to manage the transactions and sequence transfer.

5.2.4 PCA9665 access layer

PCA9665_access.c in ‘PCA9665’ library is an abstracting layer of the PCA9665 device operation (this version supports I²C-bus master operation only). It provides register access abstraction and I²C-bus transfer function, which is similar to the mbed SDK I2C library functions.

5.2.5 Main modules (application layer)

Two main modules are available in the program. One is for the PCU9669 and another is for the PCA9665. These files are switched by compile option which is defined in 'config.h' file.

Code of 'main_PCU9669.c' is enabled when `CODE_FOR_PCU9669` and code of 'main_PCA9665.c' is enabled when `CODE_FOR_PCA9665`.

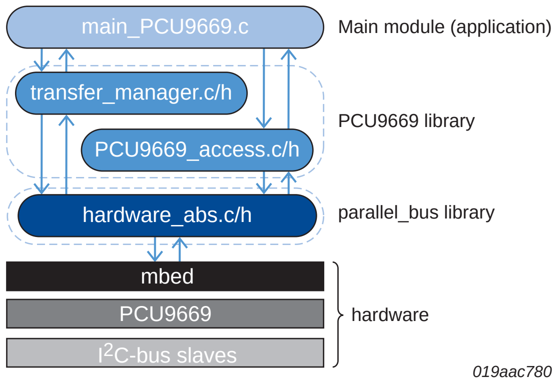


Fig 39. Software structure of PCU9669 program

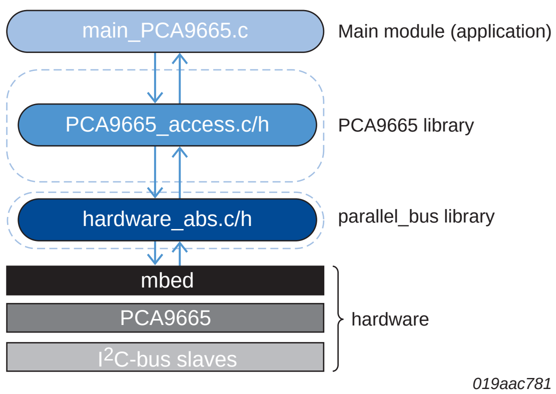


Fig 40. Software structure of PCA9665 program

5.3 Code modifications

This section will show about how to make own transfer on PCU9669 and PCA9665. The transfer management will be discussed in Section 7.

5.3.1 Try a transfer to another I^{2}C-bus slave device

5.3.1.1 Basic setup for 4 byte data transfer

This sample shows 4 byte data transfer for a slave that has I^{2}C-bus address of 0x50.

To make transfer for the slave, prepare an array that contains data for the slave. This sample explains how to make a write transaction for a slave.

```
char data[ 4 ] = { 0x80, 'N', 'X', 'P' };
```

First, some headers needed to be included. The “hardware_abs.h” is required to be included.

```
#include "hardware_abs.h"
```

5.3.1.2 PCU9669 I^{2}C-bus transfer

For the PCU9669, the sample code performs transfer with information of the transfer sequence. The sequence means a group of transactions. The transaction means single data transfer to/from a slave from START or RESTART condition to STOP or next RESTART condition.

This code sample shows single transaction in a sequence (UFm transfer on channel 1).

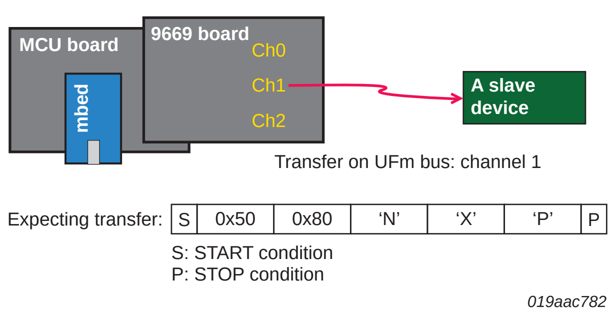


Fig 41. 4 byte transfer sample on PCU9669 channel 1

At beginning of the code, two header files should be included. “PCU9669_access.h” is a file to define the prototypes of PCU9669 register access. “transfer_manager.h” defines abstraction interface of buffer operations.

```
#include "transfer_manager.h"
#include "PCU9669_access.h"
```

In the main function, three functions are required to call before I^{2}C-bus operation. The `hardware_initialize()` function setups/initializes the parallel bus. `reset()` function asserts $\overline{\text{RESET}}$ signal for the PCU9669 and also for the slaves (if the jumper pin (JP9) is set).

`reset()` function takes two arguments. First argument is RESET pulse width in micro-seconds, second argument is RESET recovery time (wait time after RESET de-assertion) in micro-seconds. ‘4’ and ‘650’ are minimum values for PCU9669.

`start_bus_controller()` function checks and waits for the PCU9669 to be ready, then checks the device ID. If it fails, the program may need to care for this error.

```
hardware_initialize();
reset( 4, 650 );
if ( start_bus_controller( PCU9669_ID ) )
    return 1;
```

After those steps, the I^{2}C-bus becomes ready to operate.

The transaction can be set as the next code sample (it is defined as `test_transaction`). The ‘transaction’ is a structure to store the target I^{2}C-bus address, series of data (as an array) and the length of the transfer.

```
transaction  test_transaction  = { 0x50, data, 4 };
```

A sequence is defined as an array of transactions.

```
transaction  sequence[] = { test_transaction };
```

In this way, the transfer is prepared in the MCU memory.

The data of the sequence needs to be copied into the PCU9669 buffer. Use `setup_transfer()` function to setup buffer for the sequence. This function sets the tables of SLATABLE and TRANCONFIG (the PCA9669 registers), too.

```
setup_transfer( 0 /* channel */, sequence, 1 /* number of transactions */ );
```

Now the sequence is ready for transfer in the PCU9669. Call ‘start()’ to let the sequence start.

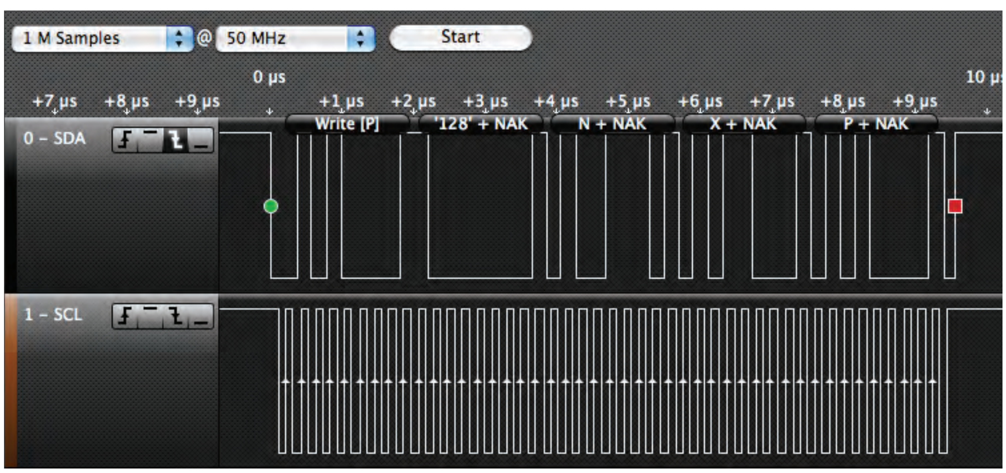
```
start( 0 /* channel */ );
```

Complete sample will be like the following code.

```
#include "transfer_manager.h"   //  abstracting the access of PCU9669 buffer
#include "PCU9669_access.h"     //  PCU9669 chip access interface
#include "hardware_abs.h"      //  to use install_ISR() and wait_sec() functions

void interrupt_handler( void ) {
    //  This ISR sample is doing nothing but clearing INT
    char    global_status;
    char    channel_status;

    global_status  = read_data( CTRLSTATUS );

    if ( global_status & 0x01 ) {  //  ch0
        channel_status  = read_ch_register( 0, CHSTATUS );
    }
    if ( global_status & 0x02 ) {
```

```
            channel_status  = read_ch_register( 1, CHSTATUS );
        }
        if ( global_status & 0x04 ) {
            channel_status  = read_ch_register( 2, CHSTATUS );
        }
    }

    int main() {
        char            data[ 4 ]           = { 0x80, 'N', 'X', 'P' };
        transaction     test_transaction    = { 0x50, data, 4 };
        transaction     sequence[]          = { test_transaction };

        hardware_initialize();          //  initializing bit-banging parallel port
        reset( 4, 650 );                //  assert hardware /RESET signal

        //  wait the bus controller ready and check chip ID
        if ( start_bus_controller( PCU9669_ID ) )
            return 1;

        //  interrupt service routine install
        install_ISR( &interrupt_handler );

        setup_transfer( 1 /* channel */, sequence, 1 /* number of transactions */ );
        start( 1 /* channel */ );

        while ( 1 )
            ;
    }
```

Figure 42 shows the result of the program execution. The transfer shows START condition, address 0x50 (‘P’ in ASCII character), data128 (0x80), N, X, P and STOP condition. All byte transfers have NACK because it is a transfer on UFm channel.



019aac783

Fig 42. PCU9669 transfer sample execution (on channel 1)

5.3.1.3 PCA9665 I²C-bus transfer

PCA9665 library does not have mechanism to manage the sequence like PCU9669. The transfers can be done transaction-by-transaction.

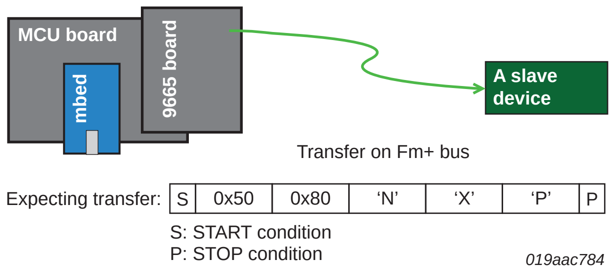


Fig 43. 4 bytes transfer sample on PCA9665

For a transaction, call `I2C_write()` or `I2C_read()` function. It takes 4 parameters: target I²C-bus address, pointer to data array, data length, and I²C condition after transaction. The last parameter can be `STOP` or `NEXT_RESTART`. The `STOP` will generate a STOP condition after transaction. The `NEXT_RESTART` holds the bus and RESTART condition will be generated when next transaction initiated.

In the beginning of the code, a header file 'PCA9665_access.h' needs to be included. That defines prototypes of the PCA9665 related functions.

Initialization will be done in two steps. `hardware_initialize()` prepares the parallel bus, `reset()` generates pulse on the RESET line and `PCA9665_init()` initializes PCA9665.

After those initializations, `I2C_write()` and `I2C_read()` can be called to execute the transfer. Those functions will block the execution of the code.

```
  I2C_write( 0x50, data, 4, STOP );
```

Complete sample will be like the following code.

```
#include "PCA9665_access.h"
#include "hardware_abs.h"

int main() {
    char    data[ 4 ]  = { 0x80, 'N', 'X', 'P' };

    hardware_initialize();          //  initializing bit-banging parallel port
    reset( 10, 10 );
    PCA9665_init();

    set_speed_mode( SPEED_FAST_MODE_PLUS );  //  set I2C as Fm+
    set_buffer_mode( ENABLE );               //  use buffer mode of PCA9665

    I2C_write( 0x50, data, 4, STOP );        //  execute transfer

    while ( 1 )
        ;
}
```

Figure 44 shows a result of the program execution. The transfer shows START condition, address 0x50 (‘P’ in ASCII character), data128 (0x80), N, X, P and STOP condition. All byte transfers have ACK.

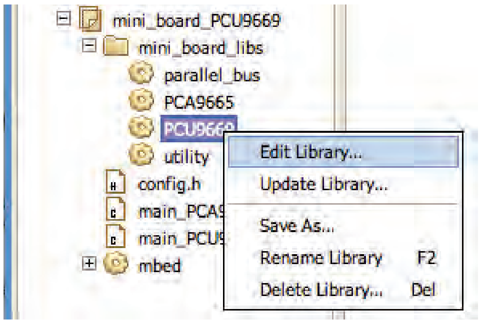
019aac785

Fig 44. Not a figure, but code sample

5.3.2 Modifying library

The code modification in application level was discussed in previous sections. It was using libraries of the mini board PCU9669.

User also can modify the library code. In the mbed compiler page, the libraries are packed in each of the gear icons. They can be opened by right-clicking on the icon and selecting ‘Edit library…’ menu.



019aac786

To see/edit library, right-click on a gear icon.

Choose ‘Edit library...’ from pop-up menu.

Fig 45. ‘Edit library...’ menu

6. Schematics

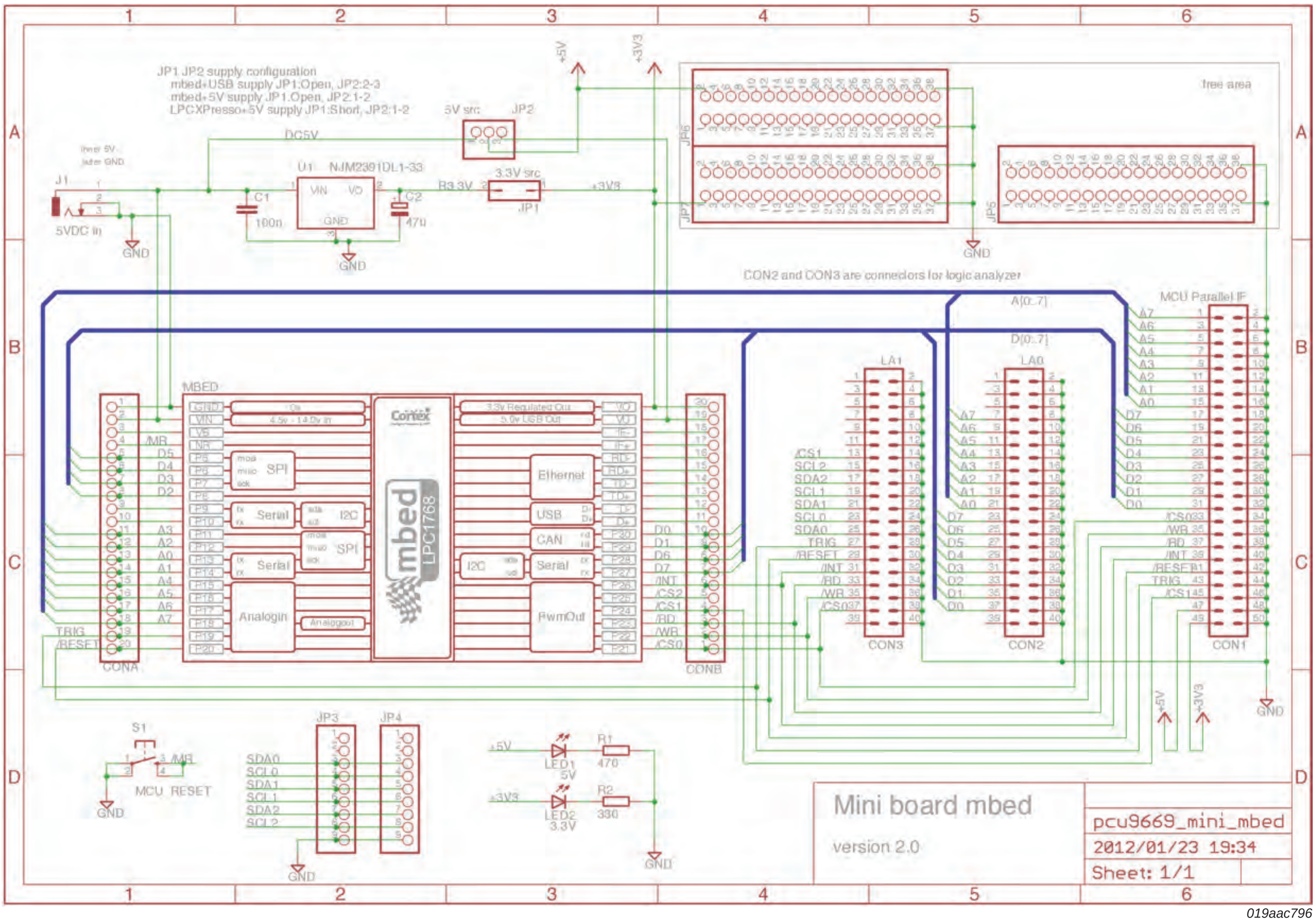


Fig 46. Mini board mbed

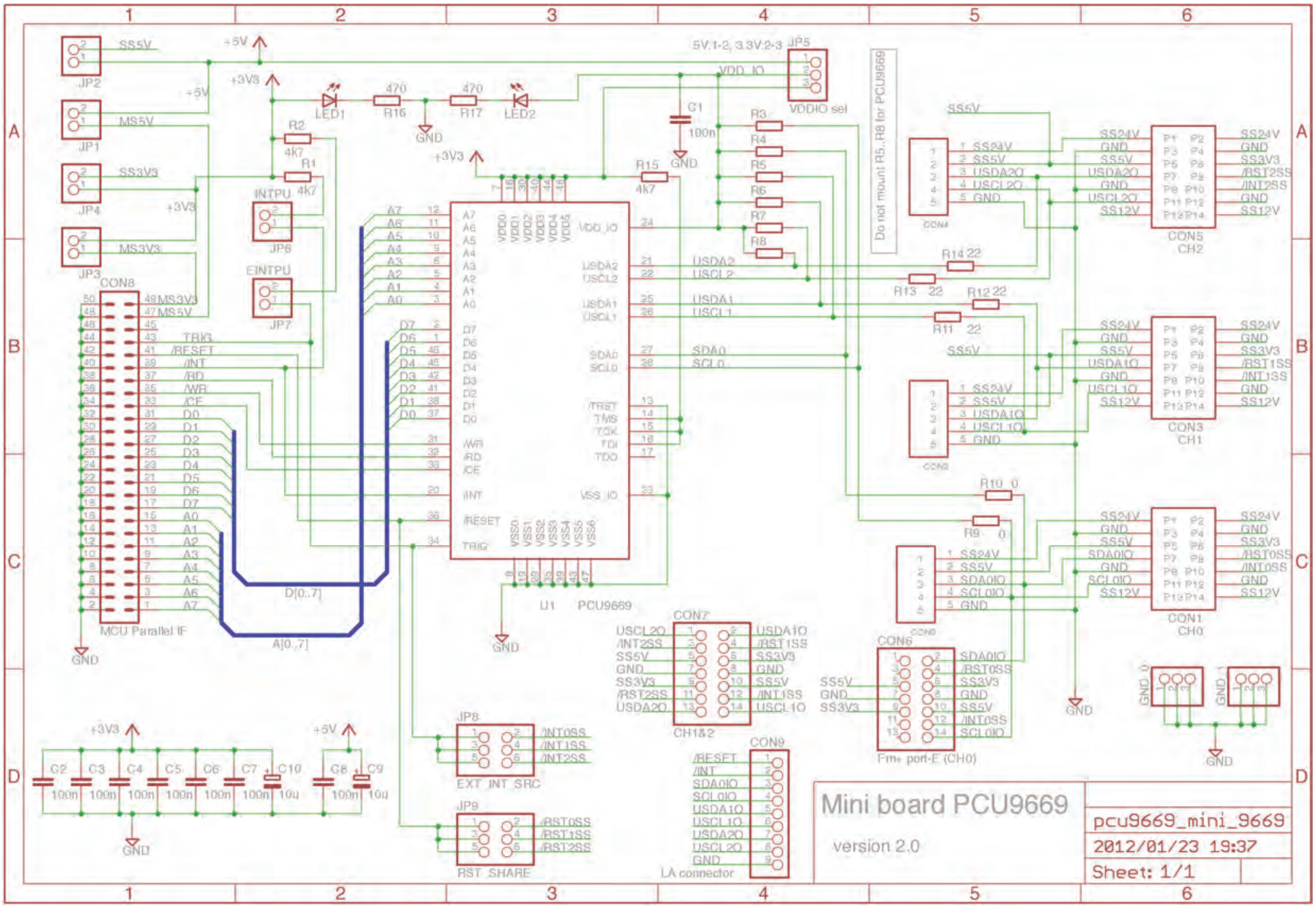


Fig 47. Mini board PCU9669

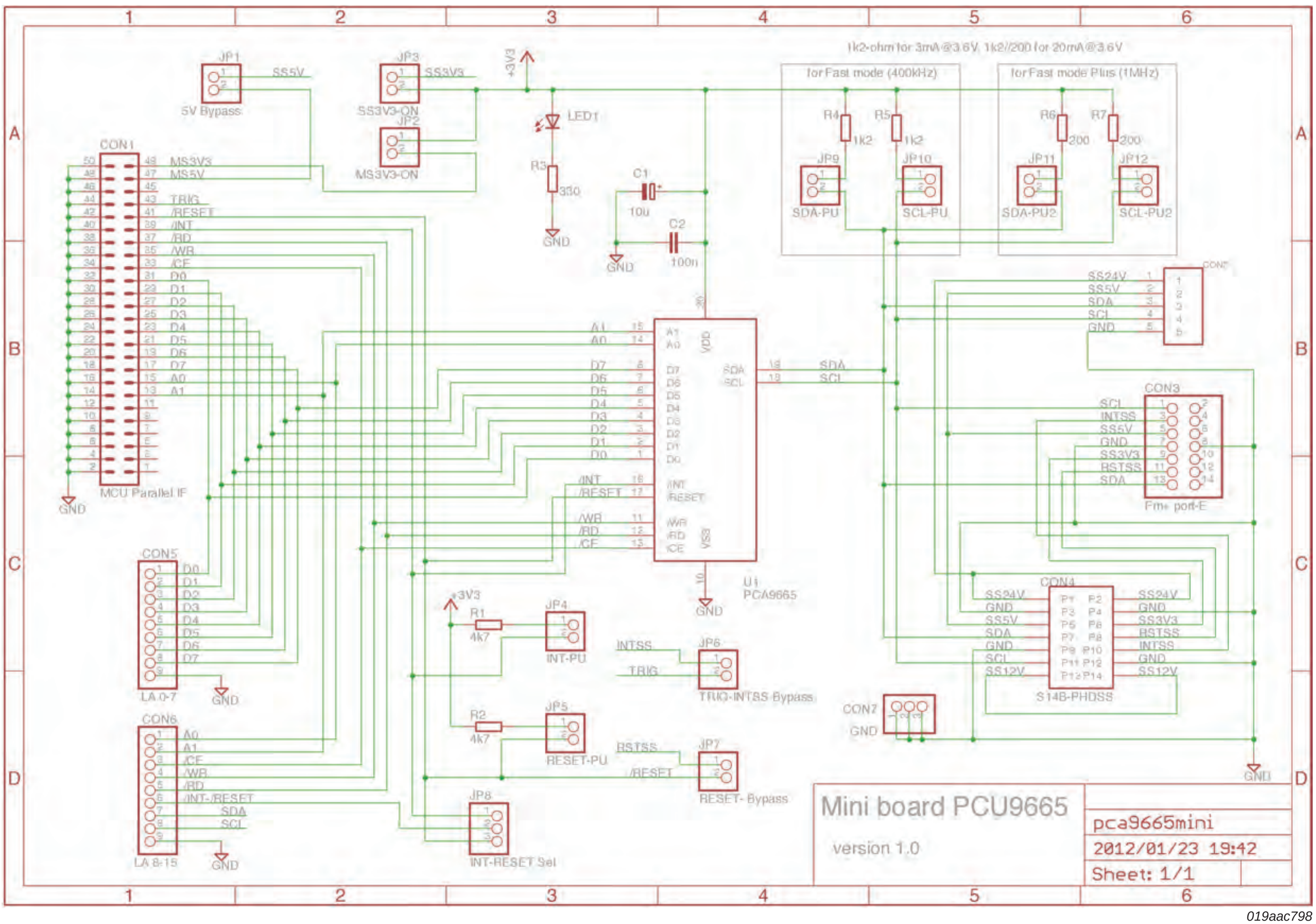


Fig 48. Mini board PCA9665

7. Appendix A — Transfer management of PCU9669

The PCU9669 has a unique feature for the transfers. Each channel that can be operated independently has 4352-byte buffer with table of slave address and transaction length. This enables managing the transfer by 'sequence'.

The 'transfer_manager' module in PCU9669_access library provides the mechanism to organize the data as a series of data, transactions and sequence.

7.1 Transfer data preparation

7.1.1 Data array

The data from/to each devices are put in an array. Here is the sample of the data arrays.

```
char data0[] = { 0x00, 0x01, 0x02, 0x03, 0x04,0x05, 0x06, 0x07 };
char data1[] = { 0x08, 0x09, 0x0A, 0x0B };
char data2[] = { 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15 };
char data2[] = { 0x00, 0x00, 0xFF, 0xFF };
```

7.1.2 Transaction

The transaction is a structure that stores the slave address, pointer to the data array and the array length. This sample intended to send each data array contents to I^{2}C-bus address 0xC0, 0xC4, 0x50 and 0x5A.

```
transaction  test_transaction0  = { 0xC0, data0, sizeof( data0 ) };
transaction  test_transaction1  = { 0xC4, data1, sizeof( data1 ) };
transaction  test_transaction2  = { 0x50, data2, sizeof( data2 ) };
transaction  test_transaction3  = { 0x5A, data3, sizeof( data3 ) };
```

7.1.3 Sequence

Sequence is just an array of transactions. The transactions will be done in order of this array.

```
transaction  test_sequence[]  = {
                                  test_transaction0,
                                  test_transaction1,
                                  test_transaction2,
                                  test_transaction3
                                };
```

7.1.4 Defining the sequence with fewer steps

Or, if the sequence has fixed data for each transaction, it can be defined in this next way.

```
char data0[] = { 0x00, 0x01, 0x02, 0x03, 0x04,0x05, 0x06, 0x07 };
char data1[] = { 0x08, 0x09, 0x0A, 0x0B };
char data2[] = { 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15 };
char data2[] = { 0x00, 0x00, 0xFF, 0xFF };

transaction  test_sequence[]  = {
    { 0xC0, data0, sizeof( data0 ) },
    { 0xC4, data1, sizeof( data1 ) },
    { 0x50, data2, sizeof( data2 ) },
    { 0x5A, data3, sizeof( data3 ) },
};
```

7.2 Setup the buffer and tables

The buffer and tables can be set by function call of `setup_transfer()`. This function takes three arguments of I^{2}C-bus channel, pointer to sequence (that is, the pointer to transaction[]) and number of transactions.

```
setup_transfer(
                 2,  /*  channel number  */
                 test_sequence,
                 sizeof( test_sequence ) / sizeof( transaction )
              );
```

7.3 Transfer start

Call `start()` with channel number. The channel will start the transfer.

```
start( 2 );
```

7.4 When the transfer is done

PCU9669 will generate an interrupt to let the MCU know it is done.

The interrupt handler will be called back. In the sample code, `interrupt_handler()` function is installed before the transfer and it will be called when it has been done.

In the PCU9669 interrupt handler, it may need to find the interrupt cause by register information.

The registers can be read by functions `read_data()` and `read_ch_data()`.

The `read_data()` is for reading global registers. It takes single argument of register address. The channel registers can be accessed by `read_ch_data()`. It takes two arguments as channel number and register offset. The register addresses and register address offsets are defined as the register name (same as data sheet) in ‘PCU9669_access.h’. (For register writing functions that are also available, those prototype definitions can be found in PCU9669_access.h.)

```
void interrupt_handler( void ) {
    char    global_status;
    char    channel_status;

    global_status  = read_data( CTRLSTATUS );

    if ( global_status & 0x01 ) {  //  ch0
        channel_status  = read_ch_register( 0, CHSTATUS );
        /*  do something  */
    }
    if ( global_status & 0x02 ) {
        channel_status  = read_ch_register( 1, CHSTATUS );
        /*  do something  */
    }
    if ( global_status & 0x04 ) {
        channel_status  = read_ch_register( 2, CHSTATUS );
        /*  do something  */
    }
}

main()
{
    ...
    install_ISR( &interrupt_handler );  //  ISR installed
    ...
    ...
}
```

8. Appendix B — Performance of emulated parallel bus

As mentioned in an earlier part of this document, this evaluation kit has GPIO emulated parallel bus. This section describes its performance as a reference.

The bus access can be performed if MCU has parallel bus hardware.

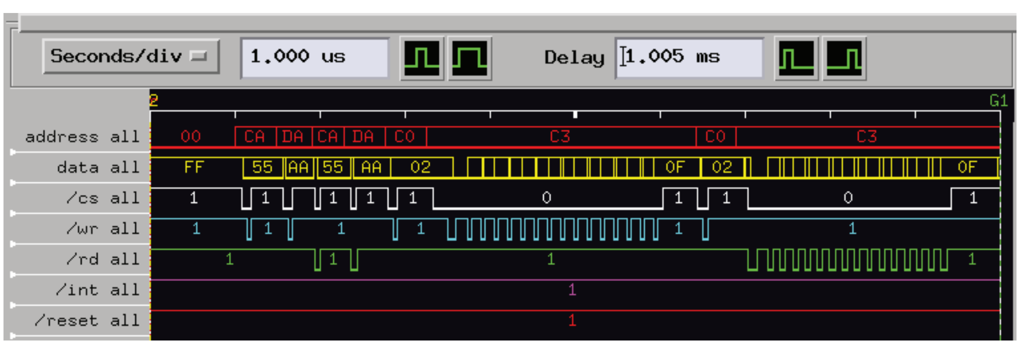


Fig 49. Parallel bus access (register access) sample

8.1 Bus access cycle on parallel bus

8.1.1 Normal access

The mbed GPIO performs parallel bus by software. For the register accesses, it has about 2 MHz bus cycle.

This access can be done in each register access function call.

8.1.2 Burst access

Bus access between an array in software and PCU9669/PCA9665 internal buffer can be optimized. For this purpose, hardware_abs has 'burst access' option interface.

With this interface, the buffer access cycle can be 6 MHz.

This access can be done by call of `write_data_burst()` and `read_data_burst()`.

9. Appendix C — Known problem

9.1 Signal name label on CON2 of 9665 board

The SDA and SCL are labeled incorrectly on CON2 of 9665 board.

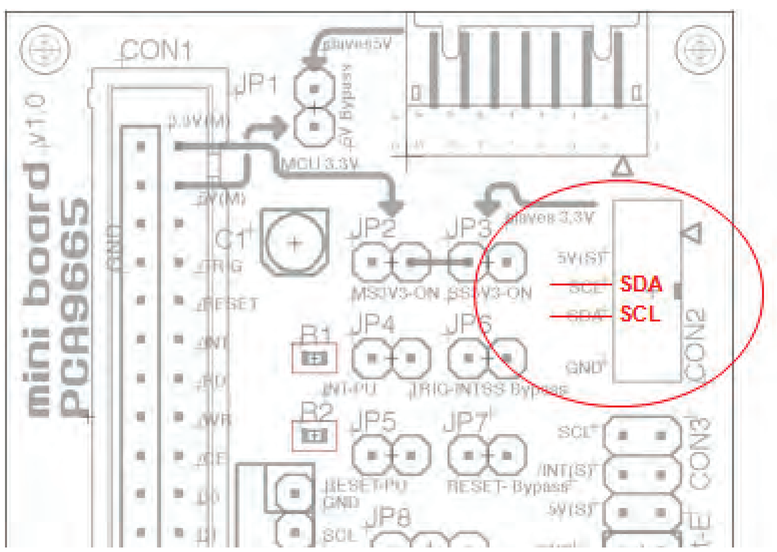


Fig 50. 9665-CON2 SDA and SCL labels should be swapped

10. Appendix D — Setup sample with PCU9955 and PCA9955 boards

The following is a guide for setup sample with PCA9955 and PCU9955 LED controller demo boards.

The PCA9955 demo board (OM13330) and PCU9955 demo board (OM13331) can be connected by CON0, CON2 and CON4 connectors on the mini-board-PCU9669.

The PCA9955 demo board (OM13330) user manual is available at www.nxp.com/documents/user_manual/UM10572.pdf.

Examples of the interconnection of those boards follow.

Since the PCU9955 and PCA9955 demo boards have identical design (there are a few components option differences), here is a sample of the UFm connection. For the PCA9955, use Fm+ channel for the connection.

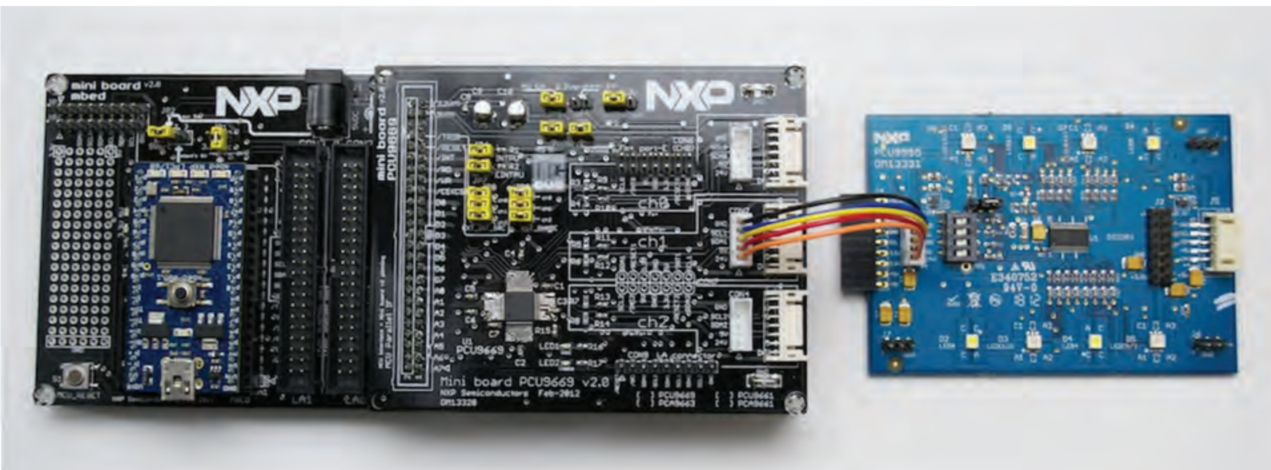
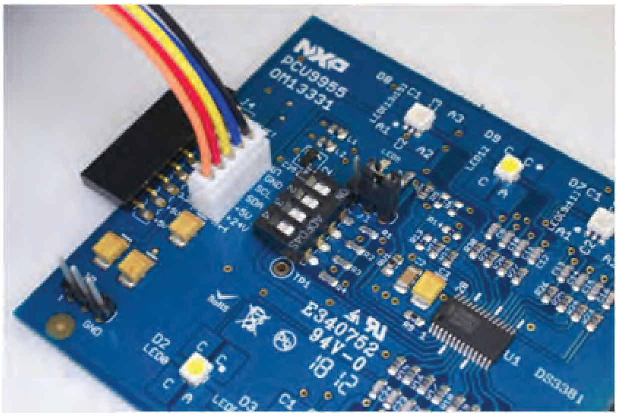


Fig 51. Mini board PCU9669 connected to PCU9955 demo board

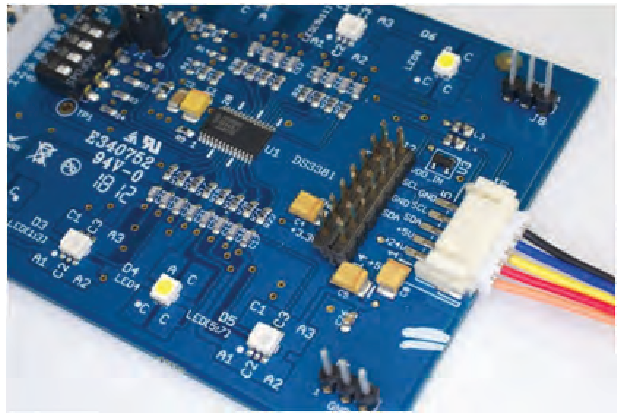
Figure 51 is a connection example of interconnection between the mini board and the PCU9955 demo board using channel 1 of PCU9669 output (channel 2 can be used also).

For PCA9955, use CON0, which is Fm+ port.



019aac793

a. J3 connector

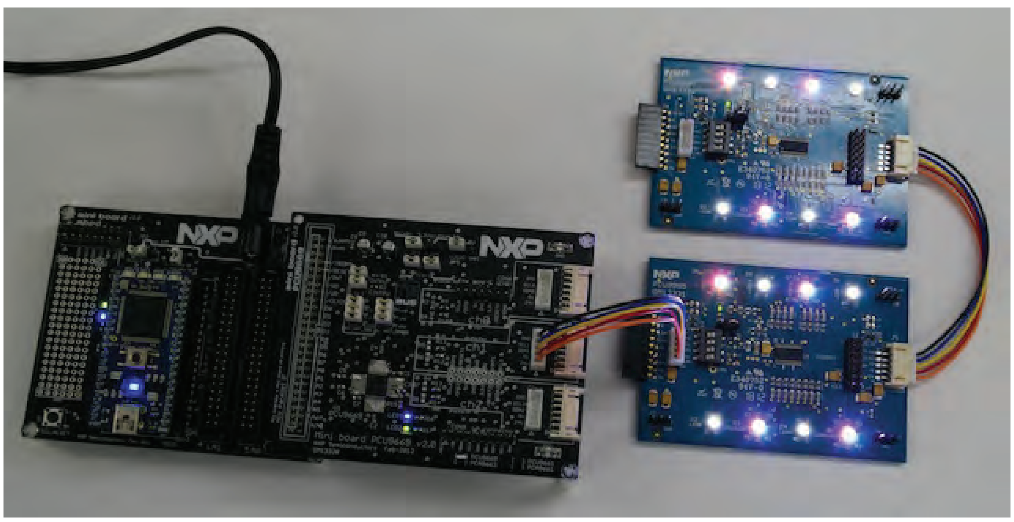


019aac794

b. J5 connector

Fig 52. J3 and J5 connectors can be used for mini board connection

Both J3 and J5 connectors on the LED controller board can be used. The LED boards can be connected in daisy chain using those connectors.



019aac795

With the default setting of mini board, all power supply can be taken from DC connector.

The 5-wire connector including I^2C-bus signals and power supply (5 V).

Two PCU9955 boards are connected in daisy chain.

Fig 53. PCU9955 boards are working

11. Abbreviations

Table 12. Abbreviations

Acronym	Description
AI	Auto Increment
Fm+	Fast-mode Plus
GPIO	General Purpose Input/Output
I^2C-bus	Inter-Integrated Circuit-bus
IDE	Integrated Development Environment
LA	Logic Analyzer
LED	Light-Emitting Diode
MCU	MicroController Unit
PC	Personal Computer
SDK	Software Development Kit
UFm	Ultra Fast-mode
USB	Universal Serial Bus

12. References

[1] **UM10204, “I^2C-bus specification and user manual” —** Rev. 5, 9 October 2012; NXP Semiconductors; www.nxp.com/documents/user_manual/UM10204.pdf

13. Legal information

13.1 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.

13.2 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.

13.3 Trademarks

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

I²C-bus — logo is a trademark of NXP B.V.

14. Contents

Date of release: 25 February 2014
Document identifier: UM10580