Freescale Semiconductor, Inc.
Application Note

Document Number: AN5245
Rev. 0, 01/2016

Using LPI2C in STOP Low Power Mode

1. Introduction

The LPI2C is a low power Inter-Integrated Circuit (I^2C) module that supports an efficient interface to an I^2C bus as a master and/or a slave. The LPI2C continues operating in stop modes if an appropriate clock is available and is designed for low CPU overhead with DMA offloading of FIFO register accesses. The LPI2C implements logic support for standard-mode, fast-mode, fast-mode plus, and ultra-fast modes of operation.

This application note shows how to use the LPI2C in the STOP and VLPS (Very Low Power Stop) low power mode with KS22 SoC. It describes the master and slave cases separately, as LPI2C implements the master and slave in two independent parts and their configurations are different. Both master and slave cases utilize the DMA engine, they setup the DMA channels and the LPI2C to work in STOP mode, they then directly enter VLPS mode. Any valid address match or request on the I^2C bus causes the LPI2C to trigger a DMA request, and perform data transfer without utilizing the CPU. The CPU is then woken up by DMA finish interrupt after the transfer is complete.

Contents

2. Overview

In typical embedded system use cases the designer sets up the whole system or CPU to enter into low power mode for as long as possible so as to save power. This requires some of the IP modules to work in low power mode to deal with non-critical tasks, such as low speed communication, when the CPU or other parts of the SoC have stopped. The LPI2C is designed for this purpose. It works in VLPR, VLPW, STOP, and VLPS low power mode, and transfer data to the I^2C bus without utilizing the CPU.

2.1. Low power mode consideration

LPI2C works in most of the different low power modes, however this application note focuses on how to make LPI2C work in STOP/VLPS mode. VLPS is the lowest power consumption mode that LPI2C can work with. In other modes such as VLPR/VLPW/PSTOP, the LPI2C configuration is very similar to STOP/VLPS.

2.2. Clock source consideration

KS22 SoC has integrated the LPI2C module and provided several functional clock sources for this module, this is shown in Figure 1. These clocks are asynchronous to the bus clock and can remain enabled in low power modes to support LPI2C working in low power modes (except MCGFLLCLK). Among these clocks, only the MCGIRCLK can be enabled in both STOP and VLPS mode. So MCGIRCLK is selected as the clock source for this case.

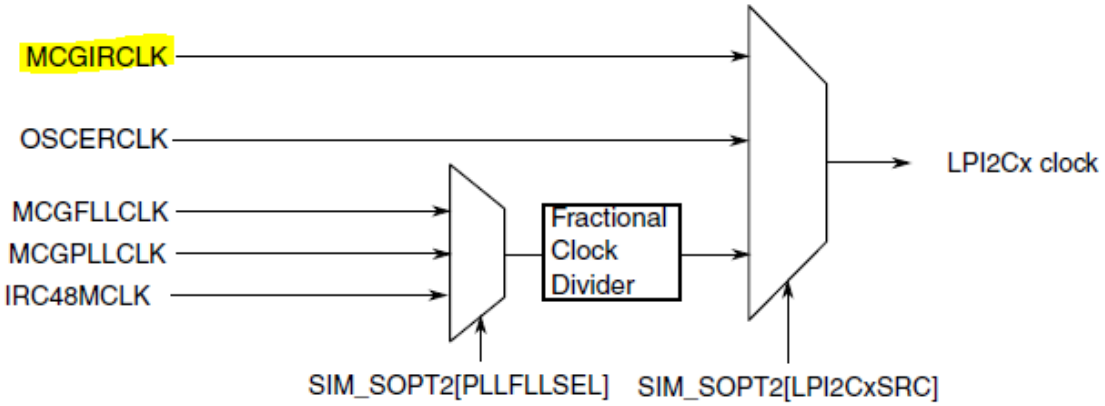

Figure 1. LPI2C clock sources

2.3. Data transfer consideration

To support I^2C data transfer without CPU interaction in low power mode, the DMA has to be involved to transfer data between memory (SRAM or Flash) and LPI2C Tx/Rx FIFO. The LPI2C supports raising requests to the DMA engine when data can be written to transmit FIFO or data is ready to read from receive FIFO according to the FIFO watermark. Also, in slave mode, when a valid address is ready in the address status register, it raises a request to the DMA for address fetching. Table 1 shows the LPI2C

status flag relationship with the DMA request. When the flag is set, the DMA request is sent to the DMA engine.

Table 1. LPI2C flag relationship with DMA request

LPI2C status flag	Master/Slave	Description	DMA Request
TDF	M/S	Data can be written to transmit data register	TX
RDF	M/S	Data can be read from the receive data register	RX
AVF	S	Address can be read from the address status register	RX

DMA works under asynchronous mode in STOP and VLPS. When a DMA request is detected in STOP or VLPS then the device initiates a normal exit from the low power mode. This includes restoring the on-chip regulator and internal power switches, enabling the clock generators in the MCG, enabling the system and bus clocks (but not the core clock) and negating the stop mode signal to the bus masters and bus slaves. The only difference is that the CPU will remain in the low power mode with the CPU clock disabled.

NOTE

When the DMA engine is working, part of the modules in SoC are in wakeup from STOP or VLPS mode, but the CPU is still kept in stop.

The use case described here makes LPI2C active in STOP or VLPS low power mode with DMA engine standby and CPU stop. LPI2C raises a DMA request when there are Tx/Rx requests on the I^2C bus. When the DMA receives those requests it either transfers the data or command from memory to LPI2C Tx FIFO data, or transfers the data from the LPI2C Rx FIFO data register to memory. When all the requested data is ready DMA wakes up the CPU to process the data.

Section 4, “LPI2C Master Work in STOP” and section 5, “LPI2C Slave Work in STOP” give further details on implementation and configuration for both Master and Slave use cases.

2.4. Baud rate consideration

Under the STOP or VLPS low power modes, the bus and flash clock cannot be high enough to support high speed I^2C baud rate. With 4 MHz bus and the LPI2C source clock, the I^2C baud rate can only be configured to maximum 500 kbps. Also, on the DMA side, the internal system bus is working in a very low speed, it may not be able to move data as fast as the high baud rate requires.

3. Environment Setup

3.1 Requirements

To implement and run this use case, there are some requirements for both software and hardware:

Software requirement

- Associated software demo package with this Application Note
- IAR Embedded Workbench 7.50.2

Hardware Requirement

- MAPS-KS22F256 Development Kit, including MCU board and Dock board. See Figure 2.
- Cables for connecting two boards(CN4) I^2C SDA/SCL/GND signals (in Slave case)

3.2 Hardware setup

Before building and running the demos, the KS22 hardware boards must be setup to ensure that the following jumpers are set:

- MAPS-KS22F256 MCU board
  - M1 (1-2), VDD power for KS22
  - JP10 (1-2), 3V3 for board power supply
- MAPS-Dock board
  - JP15, JP17 for on board debugger
  - JP7 for debug UART
  - JP4 for I^2C EEPROM

You can use a USB cable connecting to CN14 from PC for the total boards power supply. This CN14 acts as a power supply, OpenSDA JLINK debugger and USB serial debug port. For further details on the boards, refer to the [MAPS-KS22F256 Users Guide].

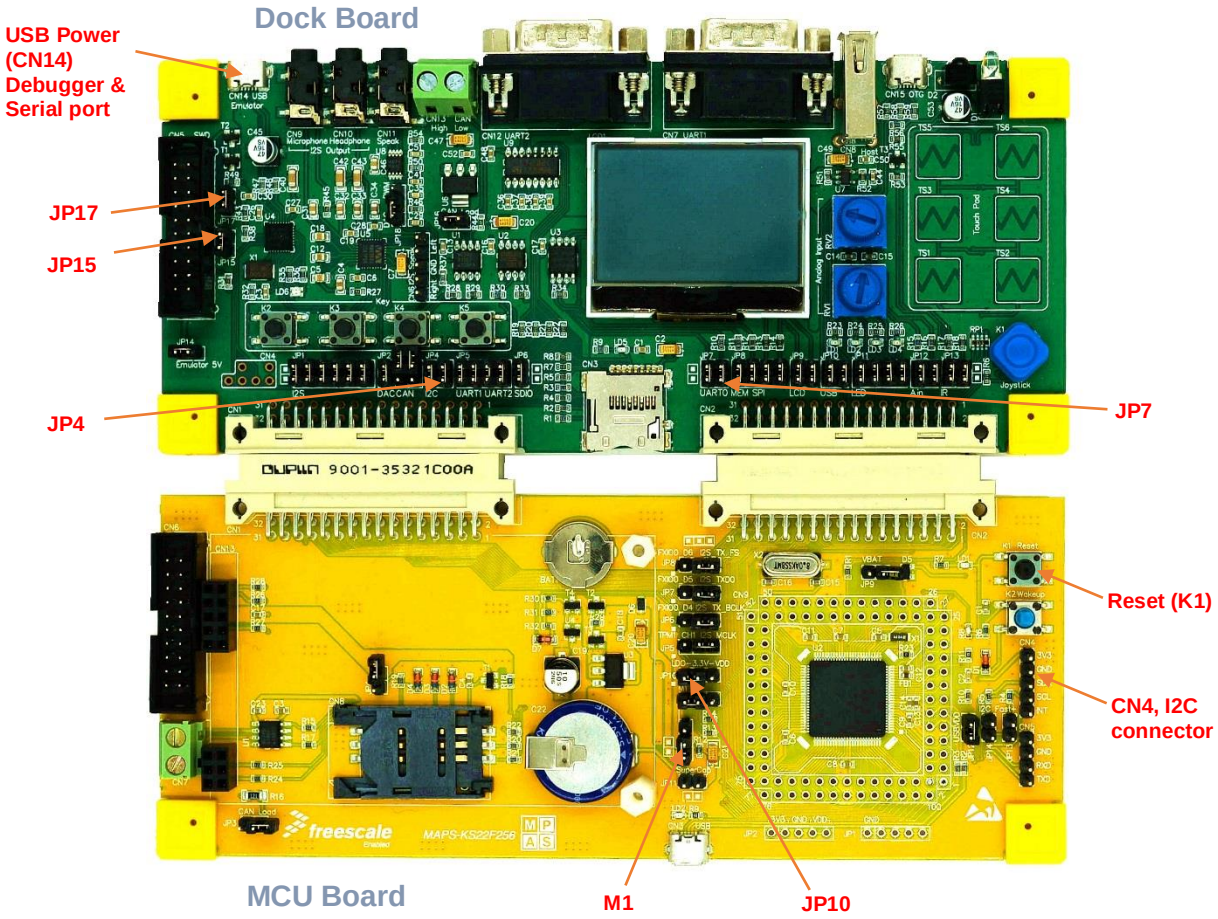

Figure 2. MAPS-KS22F256 Development Kit

4. LPI2C Master Work in STOP

This demo is to read the 64 bytes of data from I^2C EEPROM AT24C02 (on MAPS-Dock board) per second. The system enters VLPS mode by default, and a Low Power timer is configured to wake up the system and enable the DMA request for LPI2C Tx/Rx every 1 second. After timer timeout, the system reverts to VLPS mode with the DMA Tx started. The LPI2C sends START and the I^2C device address of AT24C02 to the bus with a READ request. It utilizes the AT24C02 sequential read feature, the AT24C02 sends data byte by byte in sequence before a STOP from the Master is detected. After a READ request is sent by the Master, the DMA Rx begins to receive data from the LPI2C receive data register to fill the 64 bytes buffer. The CPU is woken up by the Rx DMA channel completed interrupt and prints the 64 bytes on the debug console.

4.1. IP configurations

4.1.1. System clock

The system runs under the BLPI (Bypassed Low Power Internal) clock mode with 4 MHz IRC as clock source. The PLL and FLL are disabled, the core and bus clock are from the 4 MHz IRC with divider 1, so their frequencies are all 4 MHz. The MCGIRCLK is enabled, it also sources from the 4 MHz IRC without divider, and it is used by the LPI2C.

4.1.2. DMAMUX and EDMA Module

- DMAMUX
  - Enable DMA Channel0, connected with LPI2C0 Master/Slave TX request source.
  - Enable DMA Channel1, connected with LPI2C0 Master/Slave RX request source.
- EDMA
  - Channel0
    - Transmits bytes to LPI2C MTDR register with command and data
    - Disable interrupt after major loop done
    - Disable DMA request after major loop done
    - 4 bytes per minor loop
    - 3 minor loops per major loop
  - Channel1
    - Receive bytes from LPI2C MRDR register with data
    - Enable interrupt after major loop done
    - Disable DMA request after major loop done
    - 1 byte per minor loop
    - 64 minor loops per major loop (64 Bytes buffer created for receive data)
    - Work in asynchronous mode (must for STOP/VLPS mode)

The MTDR is a 32bit register (as shown in Figure 3) with DATA in [7-0] bits and CMD in [10-8] bits, a CMD/DATA buffer is constructed in a uint32_t array with 3 items:

1. START + Slave Address + R/W
2. Receive (DATA[7:0] + 1) bytes
3. STOP

It has 3 minor loops with a total of 12 Bytes for one major loop. In this way, DMA is utilized to send command and address to LPI2C, and complete the Master-Receive start address on the I^2C bus.
Enable interrupt after major loop done for receive channel1, to wakeup the CPU to print out the received 64 bytes buffer from EEPROM.

Address: Base address + 60h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																															
W	Reserved																					CMD			DATA							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LPI2Cx_MTDR field descriptions

Field	Description
31–11 Reserved	This field is reserved.
10–8 CMD	Command Data 000 Transmit DATA[7:0]. 001 Receive (DATA[7:0] + 1) bytes. 010 Generate STOP condition. 011 Receive and discard (DATA[7:0] + 1) bytes. 100 Generate (repeated) START and transmit address in DATA[7:0]. 101 Generate (repeated) START and transmit address in DATA[7:0]. This transfer expects a NACK to be returned. 110 Generate (repeated) START and transmit address in DATA[7:0] using high speed mode. 111 Generate (repeated) START and transmit address in DATA[7:0] using high speed mode. This transfer expects a NACK to be returned.
DATA	Transmit Data Performing an 8-bit write to DATA will zero extend the CMD field.

Figure 3. LPI2C MTDR register description

4.1.3. LPTimer Module

- Work in time counter mode
- LPO 1 KHz as clock source to make sure it can work in STOP/VLPS mode
- Disable prescaler, set compare value to 1000. This means that the counter is equal to a compare value of 1000 per second.
- Enable interrupt

4.1.4. LPI2C Master Module

- Use LPI2C0 instance
- Enable Doze mode by setting LPI2C0_MCR[DOZEN], to make LPI2C active in STOP/VLPS mode
- Functional clock source from MCGIRCLK of 4 MHz by SIM_SOPT2[LPI2C0SRC]
- 100 kbps baud rate[1]
- Set Tx/Rx FIFO watermark to zero by LPI2C0_MFCR
- Enable DMA for both Tx/Rx by LPI2C0_MDER
- Enable Master mode by setting LPI2C0_MCR[MEN]

[1] Baud rate = Clock Source / Divide. The Divide is configured by the LPI2Cx_MCCR0 and LPI2Cx_MCFGR2 register.
Divide = ((CLKLO+CLKHI+2)*2^PRESCALER) + ROUNDDOWN ((2+FILTSCL)/2^PRESCALER)

4.2. Software work flow

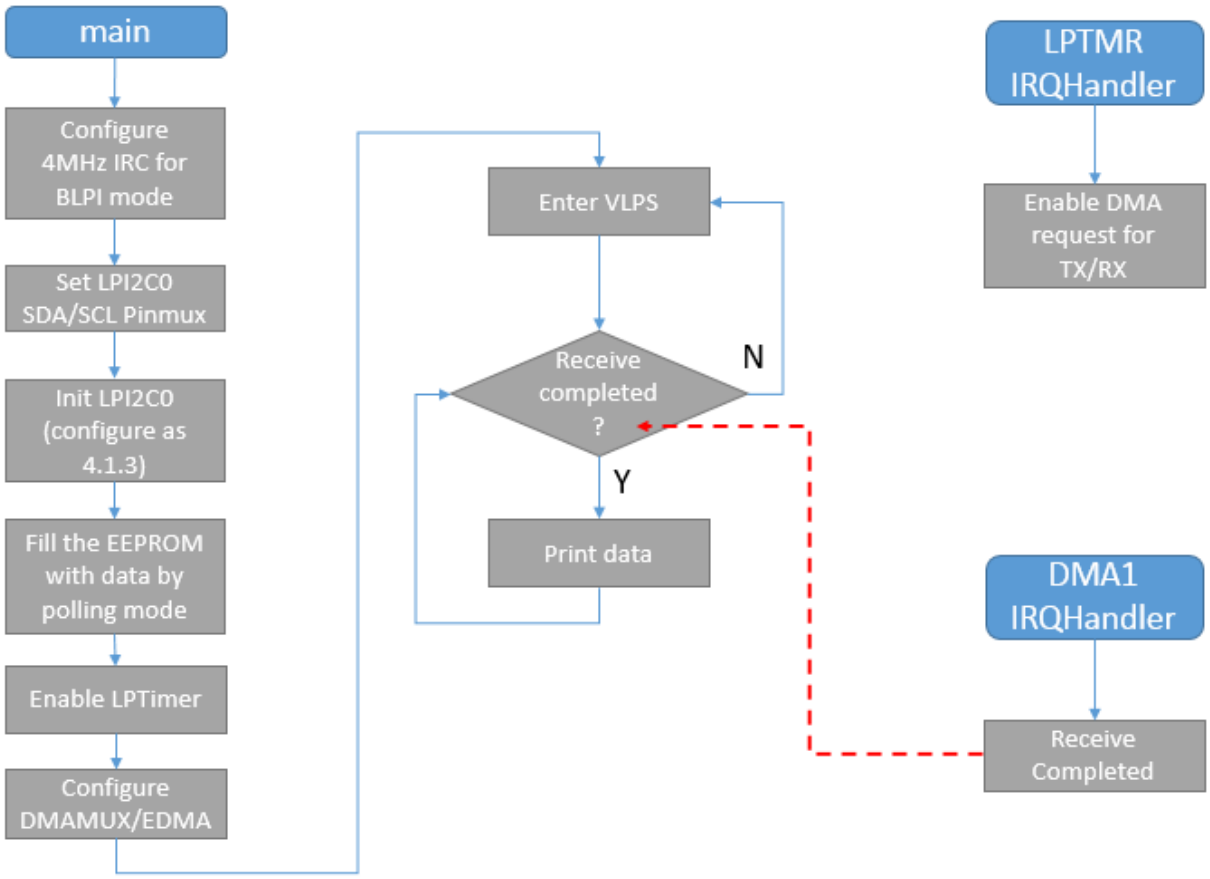

Figure 4. Master case software work flow

In this work flow, after the DMA configuration is complete, the CPU only handles IRQ and printing data buffer. It remains in VLPS low power mode most of the time. The LPTimer interrupt handler runs every second to enable the DMA request for I^2C master Tx/Rx, to trigger the DMA engine to start working. When channel0/1 have completed their work, DMA Channel1 major loop complete interrupt handler sets the receive complete flag for the main loop to print data.

4.3. Build and Run the demo

4.3.1. Build the demo

The project workspace files of the demo are located in:

examples/mapsks22f256/demo_apps/lpi2c_master_vlps/iar

The source files of the demo are located in:

examples/mapsks22f256/demo_apps/lpi2c_master_vlps/src

Open the IAR workspace file lpi2c_master_vlps.eww, click “Make” button to build the whole project.

4.3.2. Run the demo

Connect the USB cable with the MAPS-Dock board CN14 and PC Host to power up the board. User can find either CMSIS-DAP or JLINK debugger device be found on PC. The user can download a program image to the microcontroller through CMSIS-DAP or OpenSDA JLINK, depends on what on-board debugger MAPS-Dock installed.

Run a serial terminal tool like Putty or Terminal on PC host and open the USB serial port: MBED Serial Port or JLINK CDC port (COM port number can be found in the Windows Device Manager) with speed of 115200 bps and format of 8in1.

Click the “Download and Debug” button to download the binary into KS22 flash, and debug the program from main(). Run the demo, it prints out 64 bytes of data read from EEPROM every second. It is a sequential read, when total read length over the EEPROM size, EEPROM would set its internal read pointer to the start of storage, then master would get the data from the beginning of the storage.

5. LPI2C Slave Work in STOP

This demo configures KS22 as an I^2C slave, provides an internal buffer (16 bytes) to store data for the Master side to read and write, just like an EEPROM. I^2C master can use Master-Receive to read the buffer by Single-byte or Multiple-byte read operations, the Multiple-byte read would make slave send the data in the buffer in loop until the master generates STOP. The master can also use Master-Transmit to write data into the slave internal buffer by Single-byte or Multiple-byte write operations.
This case requires two MAPS-KS22 boards with LPI2C0 SDA/SCL/GND pin connected (CN4 3, 4, 2), and the demo project is divided into two projects: master and slave. Each project run on one board. **Here only describes the LPI2C slave project, the master is designed to co-operated with slave demo without any special configurations for low power.**

5.1. IP configurations

5.1.1. System clock

System running under the BLPI (Bypassed Low Power Internal) clock mode with 4MHz IRC as clock source. The PLL and FLL is disabled, the core and bus clock are from 4 MHz IRC with divider 1, so their frequencies are all 4 MHz. The MCGIRCLK is enabled also source from 4 MHz IRC without divider, it is used by the LPI2C.

5.1.2. DMAMUX and EDMA Module

- DMAMUX
  - Enable DMA Channel0, connected with LPI2C0 Master/Slave TX request source.
  - Enable DMA Channel1, connected with LPI2C0 Master/Slave RX request source.
- EDMA
  - Channel0
    - Transmits bytes to LPI2C STDR register with data
    - Disable interrupt after major loop done
    - Keep DMA request enabled even after major loop done
    - 1byte per minor loop
    - 16 minor loops per major loop (16Bytes internal buffer)
  - Channel1
    - Receive bytes from LPI2C SRDR register with data
    - Disable interrupt after major loop done
    - Keep DMA request enabled even after major loop done
    - 1 byte per minor loop
    - 16 minor loops per major loop (16 bytes internal buffer)
  - Work in asynchronous mode (must for STOP/VLPS mode)

DMA is configured for transferring data between internal 16 Bytes buffer and the LPI2C slave transmit data register or receive data register. It is always working no matter if the major loop is completed or not, as the DMA request is always enabled.

5.1.3. LPI2C Slave Module

- Use LPI2C0 instance
- Enable Doze mode by set LPI2C0_MCR[DOZEN], to make LPI2C active in STOP/VLPS mode
- Functional clock source from MCGIRCLK of 4 MHz by SIM_SOPT2[LPI2C0SRC]
- Set the slave address into LPI2C0_SAMR
- Configure the TDF only be set in the Slave-Transmit condition[1] by LPI2C0_SCFGR1[TXCFG]
- Enable the TX Data SCL Stall and RX SCL Stall for clock stretching on SCL[2]
- Enable DMA by set LPI2C0_SDER
- Enable Slave mode by set LPI2C0_SCR[SEN]
- Enable interrupt for potential bus or FIFO error

[1] If TDF is not configured to be set in the Slave-Transmit condition, the TDF would always be set when slave Tx FIFO is empty, this will mis-trigger the DMA engine to transmit data, and mess data would occur on the I2C bus.

[2] The clock stretching on SCL for slave is must for working in STOP/VLPS mode, because the DMA engine is not working as faster as normal RUN mode, the data would not be present on the SDA bus as master expected on time. Slave must tell master it's preparing the data after receiving the request by clock stretching.

5.2. Software work flow

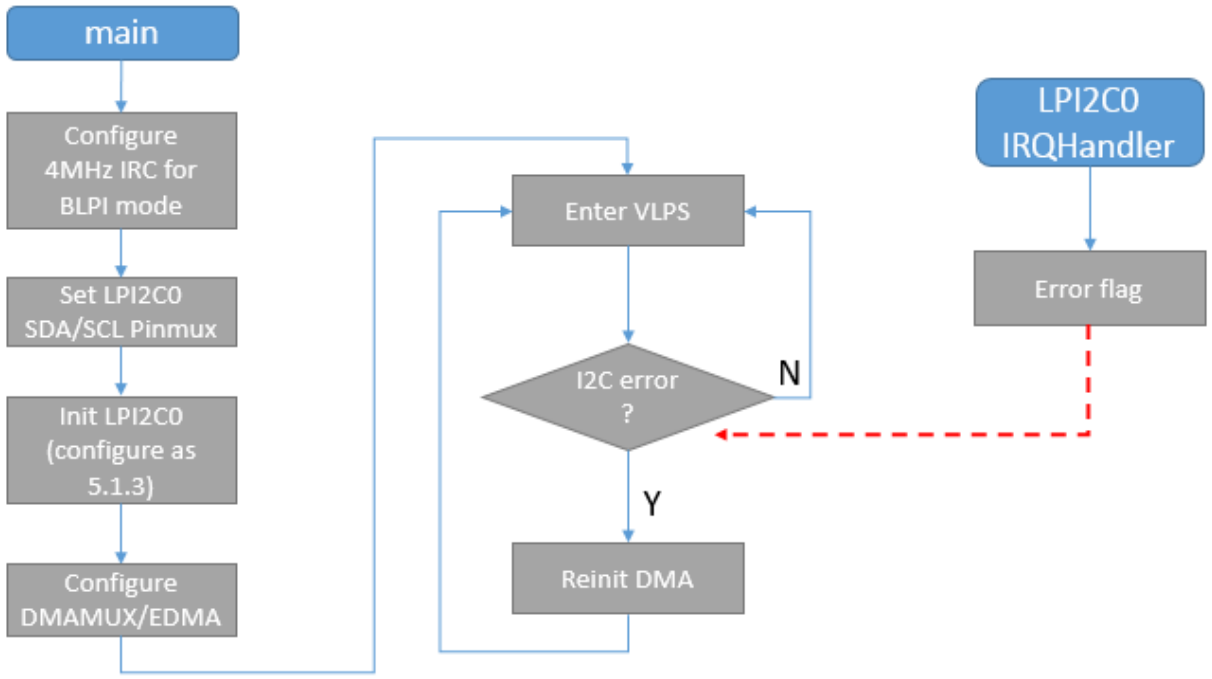

Figure 5. Slave case software work flow

In the work flow above, the CPU does nothing after DMA configuration is completed, except for handling IRQ if there is error in the I^2C bus. It mostly stays in the VLPS low power mode. The DMA engine is responsible for transferring data between the internal buffer and the LPI2C slave TDR and RDR registers when there is a request from the Master on the I^2C bus. The Address Match, Slave-Transmit, and Slave-Receive progress does not require action from the CPU.

5.3. Important note

For LPI2Cx_SCFGR1[TXCFG]: Transmit Flag Configuration, the transmit data flag will always assert before a NACK and STOP is detected at the end of a slave-transmit transfer. This can cause an extra word to be written to the transmit data FIFO, but not present on the I^2C bus. This would not cause an issue on the I^2C bus, but it means when using DMA to transmit data to slave data FIFO, the DMA channel current source address would be plus 1 offset. If DMA is not re-configured for the next transmit, one word would be missed, which is already transmitted in the previous DMA job. Keep this in mind when you are using the DMA automatically for slave transmit.

5.4. Build and Run the demo

5.4.1. Build the demo

The project workspace files of the demo are located in:

examples/mapsks22f256/demo_apps/lpi2c_slave_vlps/iar

The source files of the demo are located in:

examples/mapsks22f256/demo_apps/lpi2c_slave_vlps/src

Open the IAR workspace file lpi2c_slave_vlps.eww. There are two projects in this workspace:

1. lpi2c_master: it acts as I^2C master, firstly it reads the data from the slave internal buffer; it then overwrites those data using Master-Transmit, then reads the data back again to verify. Build it by selecting it, and click "Make" button.
2. lpi2c_slave_vlps: it act as I^2C slave. Build it by selecting it, and clicking the "Make" button.

5.4.2. Run the demo

Two MAPS boards are needed to run this demo, one for master program mark as #1, another for slave program mark as #2. They are connected by cable for 3 signals on MAPS-KS22 MCU board CN4: SDA/SCL/GND.

1. Download the slave program. Connect the USB cable with the #2 MAPS-Dock board CN14 and PC Host to power up the board. The user can find either CMSIS-DAP or JLINK debugger device on their PC. The user can download a program image to the microcontroller through CMSIS-DAP or OpenSDA JLINK, depending on what on-board debugger MAPS-Dock installed. Click the "Download and Debug" button to download the built out master binary into KS22 flash. Exit the IAR debug window, re-plug the USB cable to power on the slave again.
2. Download the master program. Connect the USB cable with the #1 MAPS-Dock board CN14 and PC Host to power up the board. Click the "Download and Debug" button to download the built out slave binary into KS22 flash, then run the demo in IAR. Run a serial terminal tool like Putty or Terminal on PC host and open the USB serial port: MBED Serial Port or JLINK CDC port (COM port number can be found in the Windows Device Manager) with speed of 115200 bps and format of 8in1. Use this terminal to monitor the master program print out and status.

When the master starts to run, the user can see the original data from the slave, then the overwritten data from the master which is read back from the slave.

6. Conclusion

The LPI2C IP module is well designed for low power use cases, as it can work in the STOP low power mode without CPU interaction. This feature can significantly reduce the power consumption for product (like Sensor Hub) that need I^2C communication in idle or other non-critical path, where CPU can STOP. Besides the low power feature, LPI2C module also supports different speed mode: Standard, Fast, Fast Plus, Ultra-Fast and HS-mode in slave, which means the user can utilize the LPI2C to achieve a very high speed I^2C communication, up to 1 Mb/s in Fast Plus mode and 5 Mb/s in Ultra-Fast mode.

Due to the new LPI2C module the user can design the system in a very flexible, low power and high performance way.

7. References

1. Kinetis KS22 SoC Reference Manual and Data Sheet
2. MAPS-KS22F256 and MAPS-Dock: Freescale MAPS Platform for Kinetis MCUs
3. MAPS-KS22F256 Users Guide
4. KINETIS_SDK: Software Development Kit for Kinetis MCUs

8. Revision history

Table 2. Revision history

Revision number	Date	Substantive changes
0	01/2016	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5245
Rev. 0
01/2016