I^2C-Like Communication for the Power Meter IC

M. Cvetkovic, M. Jevtic and M. Dimitrijevic

Abstract – In this paper, we will present the communication controller for the power meter IC. The realized controller is based on the I^2C communication standard. The I^2C standard was modified in order to provide addressing (for reading and writing) 24-bit memory words in DSP part of the PMIC, instead of addressing individual PMICs. The controller contains the mechanism for the synchronization with DSP, which doesn't interfere DSP's performance. The design is synthesized using AMS 0.35 μm standard cells, and can be utilized in final product realization. Simulation results are also presented in the paper.

I. Introduction

Power meter integrated circuit (PMIC) is the integrated circuit that provides power consumption measurement. It consists of analog and digital part (DSP). The realized communication controller is a part of the digital design and has to provide tuning, remote control and monitoring of the PMIC.

There are many common solutions for realizing a communication interface for digital control/signal processing ICs. Among these solutions, Inter-Integrated Circuit (I^2C) bus seemed the most appropriate. The main benefit of I^2C is its simplicity, which provides the designer to implement intelligent application-oriented signal processing circuits without encountering numerous interfacing problems. This interface is structured for economical, efficient and versatile serial communication.

The I^2C bus is a two-wire serial interface developed by the Philips® Corporation [1]. It consists of serial clock line (SCL) and serial data line (SDA) providing bi-directional communication, facilitated through the use of "wire-and" (i.e., active-low, passive-high). Each device connected to the I^2C-bus can operate as either a transmitter or receiver, and, in addition, devices can also be considered as masters or slaves. I^2C specification supports 7-bit and 10-bit address format, and data transmission is 8-bit oriented, with no limit in the number of bytes transferred. Concerning transfer rate, this specification supports data transmission up to 100Kbps in standard mode, and up to 400Kbps / 3.4Mbps in fast/high-speed mode.

At this point of the project development, the slave communication controller for the prototype PMIC is realized. The main purpose of this interface is to connect two-wire serial bus and 24-bit parallel bus in the DSP part of the PMIC in order to provide additional test function for the prototype. In this paper we will present the controller design, as well as some modifications of the I^2C standard that had to be done in order to satisfy all the requirements needed for testing.

M. Cvetkovic, M. Jevtic and M. Dimitrijevic are with the Department of Electronics, Faculty of Electronic Engineering, University of Nis, Beogradska 14, 18000 Nis, Serbia & Montenegro, E-mail: mcvetkovic@elfak.ni.ac.yu

II. Communication controller design

The testing is about to be performed by reading from and writing to the internal memory words of the PMIC. At that point we had to modify the standard I^2C interface, because it was primarily intended for master–slave communication, where master communicates to a number of slaves by addressing them individually. Our test concept considers one master reading from and writing to internal memory words of one slave device – the PMIC prototype. Thus, master is not addressing the slave units, but individual words within the PMIC. For the present, only the measurement part of the IC is developed with 64 internal 24-bit words and one 24-bit command/status register. Thereby, 7-bit address format was adequate to address all the words in the chip. The address formats supported by I^2C standard are detailed in [1].

In the following subsections we will present the main features of the controller, its operational flow, with the emphasis on synchronization mechanisms, implemented to insure accurate performance of the controller.

A. Main Features and Structure

Features of the realized I^2C-like controller are:

- Performance in slave mode;
- Data transfer of 400 kbit/s (f_{SCL} = 400 kHz);
- 7-bit addressing;
- Filtering of incoming data from SDA and SCL lines;
- Detection of START, STOP and REPEATED START conditions [1];
- Latching of serial data bits from the SDA line with every falling edge of the SCL clock;
- 4-byte communication protocol;
- Synchronization with DSP part of the chip;
- Synchronization with master (inserting Wait-states).

In I^2C-bus communication master initiates data transfer and generates serial clock signal on the SCL line [1]. Since there's no need for the PMIC to initiate the transfer, only the slave mode is supported. We decided to use 400 kHz SCL frequency, because it gives us the ability to read all sampled data from the measurement part of the chip every second.

A block diagram of the slave communication controller, showing the basic structure of the design, is given in Fig. 1. There are three main parts of the block diagram: I^2C interface, Address decode/internal bus interface and the on-chip memory (64 24-bit words) with the command/status register. First two blocks together compose the slave communication controller, and the third part represents the memory words and registers that are being read from and written to in the test process. On the other hand, I^2C interface consists of I^2C control logic block, 24-bit data shift register and 8-bit address shift register. Data shift register holds data to be sent to the master over the serial bus or data received from the master. Address shift register is holding the 7-bit address of the memory word that is being read from or written to, as well as the R/W bit (LSB in the register), which determines the direction of data transfer. I^2C control logic is connected to the SDA and SCL lines and is responsible for START, STOP and REPEATED START detection, incoming data filtering, and controlling the whole data transfer process. The synchronization with master is also performed in this block. This synchronization is firmly connected to the synchronization with DSP part of the chip, which is performed in Address decode/internal bus interface block. Both synchronization mechanisms are very important for proper functioning of the communication controller and they will be detailed in the latter subsections. In Fig. 1 is also shown that both SCL and SDA lines are pulled up with external resistors, which is needed in order to perform "wired-and" function.

Concerning the communication protocol, we decided that every transfer would include the transmission of exactly four bytes: the first byte for address and R/W bit, and the remaining three for the 24-bit data. This is needed for the proper generation of some signals used for the synchronization with DSP. Apart this change of the I^2C standard, we also decided to replace standard I/O stages with the bi-directional I/O pad buffers with internal active pull-up resistance and Schmidt-trigger circuit for input signal filtering.

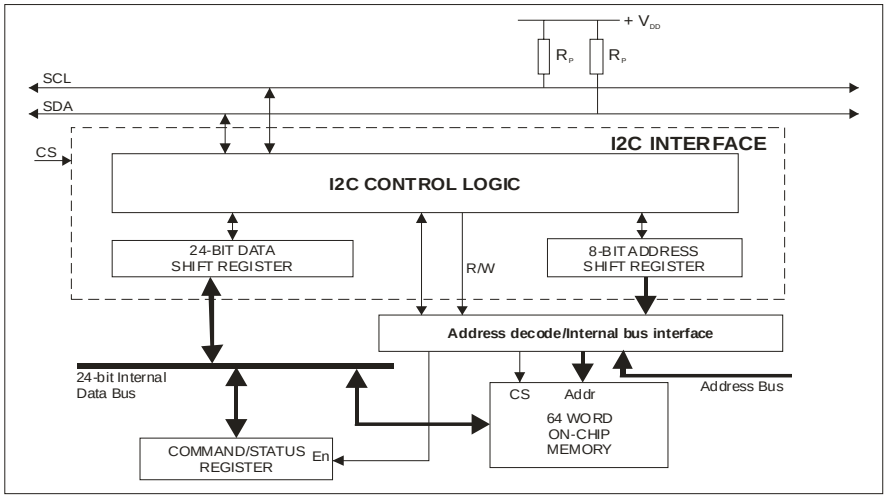

Fig. 1. Block diagram of the realized communication controller

B. Operational flow

We've already mentioned that it's always the master who initiates the transfer over the I^2C bus. Master, first generates START condition, than sends the 7-bit address to the slave, and after that, the data transmission is performed [1]. On the other side of the bus line, slave controller's I^2C control logic circuit monitors SCL and SDA lines in order to detect START condition. When this condition is detected the slave controller receives the address, together with the R/W bit. After that, data transfer is performed, and at the end, master can generate either STOP condition, meaning the end of the transfer, or REPEATED START condition, meaning initiation of another transfer session.

The operational flowchart for the I^2C-like communication controller is given in Fig 2. After the detection of START condition, the controller receives the address sent by master. In the realized design the address represents the address of the memory word in the DSP part or the address of the command/status register. These locations can either be read from or written to, which is determined by the R/W bit. Memory words contain the measured or calculated values of current, voltage and power. After the R/W bit is checked, the controller transitions to the transmitting or the receiving state. When receiving, it receives data from master byte by byte. I^2C control logic block contains two counters: bit-counter, which counts the received bits so the controller would know when the whole byte is received, and byte-counter, so the receiver would know when all three bytes are received. Every data bit is shifted in the data shift register with the falling edge of SCL signal. When byte-counter counts to 3, meaning all three bytes are received, signal data_rdy is set to "1". This signal is needed to indicate that data is received and ready to be transferred from the data shift register to the addressed memory word in DSP, i.e.

represents the request for data transfer from the controller to the DSP. After this the controller waits for the dtc (data transfer complete) signal, which indicates that the transfer is complete, so the I^2C control logic can reset data_rdy signal. At that point, the controller is ready for another master-slave transfer.

When transmitting, the controller first has to fetch data from the DSP and load it to the data shift register. Therefore, it first sets the addr_rdy signal to "1", which, similarly to the data_rdy signal, represents the request for data transfer from the DSP to the communication controller. After the dtc signal is set to "1" indicating data is loaded to the data shift register, controller starts the transmitting process, sending three bytes of data, byte by byte.

Signals addr_rdy, data_rdy and dtc are used in synchronization mechanisms that are to be discussed in the next subsections.

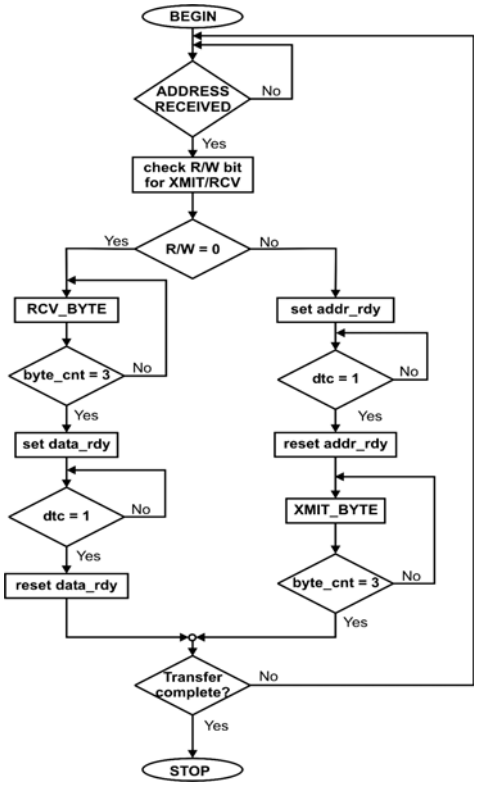

Fig. 2. I^2C-like slave communication controller operational flowchart

C. Synchronization with DSP

During I^2C master-slave communication there is a possibility that slave receives certain request from master, but also to be unable to respond because it's busy performing some other operation. In this case a problem may occur if master resumes the transfer, without waiting for the slave to finish current job and properly respond to the request. Therefore, some kind of synchronization mechanism is needed, in order to force the master into a wait state until the slave is ready for another transfer. I^2C specification defines a clock synchronization mechanism that can be used to enable devices to cope with the fast data transfers, on either byte level or bit level. This mechanism is a type of handshake procedure, and it's performed by slave holding the SCL line low after reception of a byte, on byte level, or stretching the low period of SCL signal, on bit level.

In our design it is certain that the controller is able to receive bytes at 400 kbit/s rate, but it may need more time to store received data or prepare data to be transmitted. In both cases, the core of the problem is that the communication controller and DSP are using the same 24-bit internal bus for data transfer. DSP is using it during data processing, and it is obvious that the bus may be busy when the controller generates the request for data transfer. Therefore the controller needs to pull SCL line low until data transfer between the communication controller and DSP is complete. In that manner, the wait states are inserted. Since all data in DSP is 24-bit long, it is not needed to check if it is necessary to insert a wait state after each transferred byte. If needed, wait states are inserted after the address is received (slave transmitter), and after all three data bytes are received (slave receiver). On the other side, we designed a synchronization circuit, as a part of Address decode/internal bus interface, which synchronizes data transfer between the controller and the DSP, and signals to the I^2C control logic when the transfer is complete. When I^2C logic receives this signal, it can end the wait state, and the master can resume the transmission.

I^2C standard defines that transmission of every byte is followed by an acknowledge bit (ACK), generated by the receiver. Wait states, if needed, are inserted after the ACK bit. The duration of the wait state is determined by the mechanism for synchronization with DSP. In the previous section, we have mentioned addr_rdy, data_rdy and dtc signals. Signals addr_rdy and data_rdy represent the request to read from and write to the addressed memory word, respectively. When one of these signals is set to "1", the synchronization circuit checks if internal bus is busy. If the bus is not busy - data can be transferred immediately and the dtc signal is set to "1". If it is busy, this circuit waits for the bus to be released, and then performs the data transfer, setting dtc signal active just afterwards. The dtc signal set to “1” determines the end of the inserted wait state, i.e. that the slave should release the SCL line.

This kind of synchronization mechanism implies that data shift register should have asynchronous load. Data is loaded in this register only when the controller has to transmit data to the master. On the other hand, this register is clocked by the external SCL signal, generated by master. If the wait state is inserted, there is no clock on the SCL line, so data couldn’t be loaded in the register when ready, if the load was synchronous. Therefore, we designed the synchronization circuit to generate the asynchronous load signal for the data shift register. The dtc signal is set to “1” just after the data is loaded to the register.

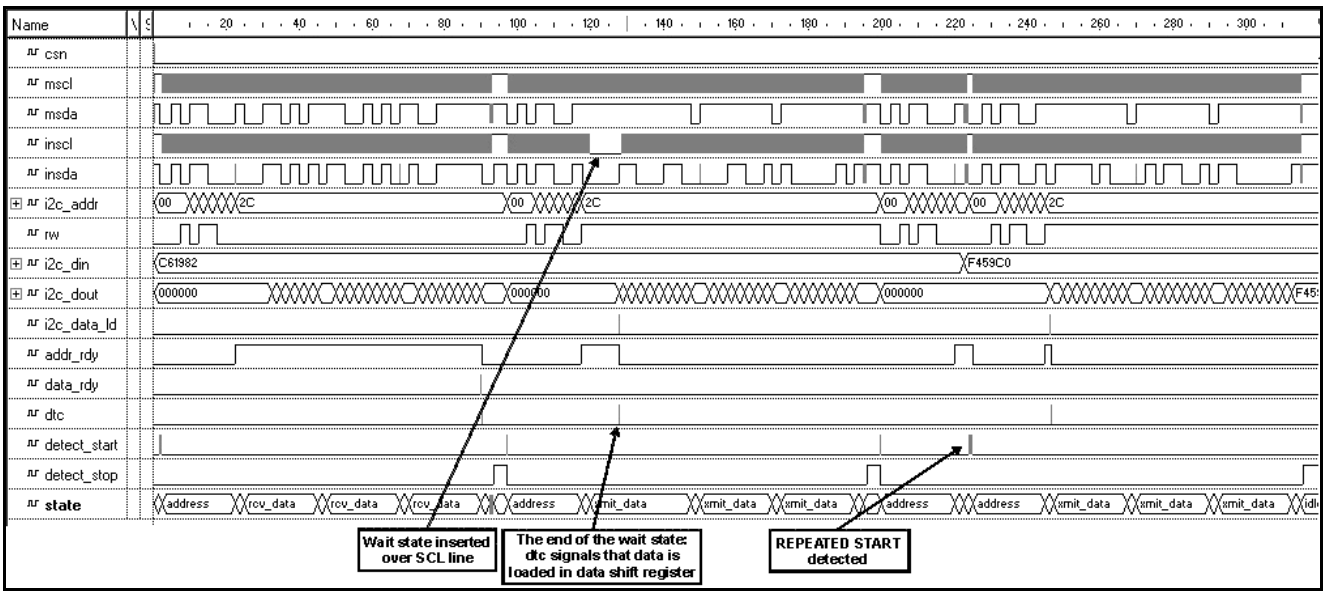

Fig. 3. Simulation waveforms

III. Simulation results

After the VHDL description [2], the design was verified on the functional level. The design was tested for the two modes of operation: transmitter and receiver mode. Besides, we needed to specifically test the detection of REPEATED START condition and, also, to check if the synchronization circuit is performing properly. The waveforms obtained during the simulation are presented in Fig. 3.

As it can be seen from the figure, the controller was first tested in the receive-mode, in the transmit-mode, and, at the end, detection of the REPEATED START condition was tested. In all three cases, the controller responded properly. Fig. 3 also shows the transitions of addr_rdy, data_rdy and dtc signals during the wait-state (marked part of the figure). The simulation was performed for the worst-case wait-state period, which lasts for four periods of SCL. During that time, SCL is held low by the receiver. Signal detect_start from the waveform also shows proper detection of REPEATED START condition.

IV Further Work

For the utilization of the presented controller in the final product, further modification has to be performed. Since the controller will have to provide remote tuning, control and monitoring of a number of the PMICs, the protocol must be modified. First 3 bytes in message (24-bit sequence) would represent unique ID of the PMIC. They would be followed by another 24 address bits, with upper 16 bits being insignificant at this point, and lower 8 bits intended for additional identification/command coding. These 8 bits will have the least significant bit (R/W) defining whether data is being written-to or read-from the PMIC, and upper 7 bits providing reading from and writing to all 128 24-bit memory words and registers in the final PMIC [3]. At the end of each message, a 24-bit information for message integrity checking would be appended.

V Conclusion

The realized communication controller fulfills the imposed requirements. After the VHDL-description based simulations, design synthesis was performed. The controller design is prepared for AMS 0.35 μm technology, using CADANCE Design Tools [4]. Post-synthesis simulation results match with the results obtained in VHDL-description based simulations. Verification of the whole digital part of the chip was also performed, and all the simulation results showed that the mechanism for synchronization with DSP doesn't interfere its performance.

Apart of the appliance in PMIC, this design can be implemented as a part of other integrated circuits with the communication function based on I^2C standard or some of its specific modifications.

References

[1] Philips Semiconductors, *"The I^2C-bus Specification - v2.1"*, www.semiconductors.philips.com

[2] K. C. Chang, *"Digital System Design with VHDL and Synthesis: An Integrated Approach"*, IEEE Computer Society, Los Alamitos, California, 1999.

[3] Milun Jevtic, Marko Cvetkovic, " *I^2C-bus communication"*, Faculty of Electronic Engineering - LEDA Annual Report, Nis, December 2002

[4] www.cadence.com