Design of the Embedded Serial and Parallel Communication Protocol Controller

Jie Liang, Ran Duan
Beijing Aerospace Automatic Control Institute
Beijing, China
E-mail: jessie7770@gmail.com, duandr@gmail.com

***Abstract*—Along with the developments of VLSI technologies, the entire industrial control computer system can be integrated into one chip, so called industrial control SoC. The serial and parallel interface plays an important role in the communications between industrial control system and the peripherals. The paper discusses the design of the serial and parallel protocol communication controller of industrial control SoC, and implements the serial and parallel communication with PC on a FPGA –based verification platform.**

Keywords-embedded; serial and parallel communication protocol controller

I. INTRODUCTION

Comparing with commercial PCs, industrial control computers have the advantages of execrable circumstances compatibility and high scalability, which bring about rapid development of industrial control systems. Typical industrial control systems connect with peripherals through communication interfaces, implementing accurate and rapid processing of run-time field information.

Along with the development of VLSI technique, the entire industrial control computer system now can be integrated into one chip. At present, miscellaneous I/O interfaces have been integrated in one peripheral controller and managed through configuration register. The integration can reduce the required I/O addresses, the cost and power consumption, and improve system reliability. Meanwhile the programming and management can be facilitated through centered configuration. Due to these advantages the integrated peripheral controller has gained popularity rapidly.

II. INDUSTRIAL CONTROL SOC

The typical industrial control SoC consists of three major parts: CPU, system controller and peripheral controller. The system controller includes SDRAM controller, bus controller for bus arbitration, ISA controller, programmable interrupt controller, timer, keyboard controller, RTC and DMA controller, etc. The peripheral controller is composed of hard disk controller, floppy disk and serial and parallel communication protocol controller. The functional block diagram is shown in Fig. 1. The serial (UART) and parallel port protocol controller are directly attached to ISA bus, implementing bidirectional information exchange with off-chip circuits and systems.

III. THE DESIGN OF SERIAL AND PARALLEL COMMUNICATION PROTOCOL CONTROLLER

The serial and parallel communication protocol controller includes two UART and an IEEE 1284 parallel protocol controller, as shown in Fig 2.

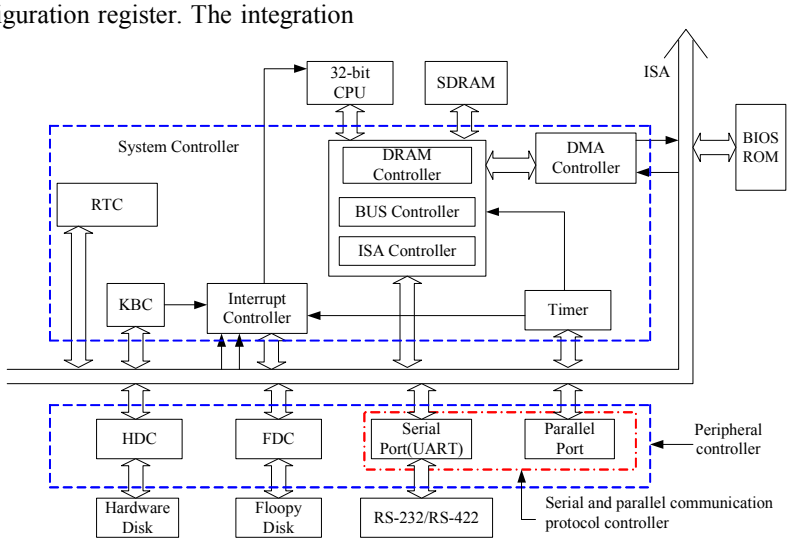

Figure 1. Industrial Control SoC

978-0-7695-4180-8/10 $26.00 © 2010 IEEE
DOI 10.1109/PCSPA.2010.111

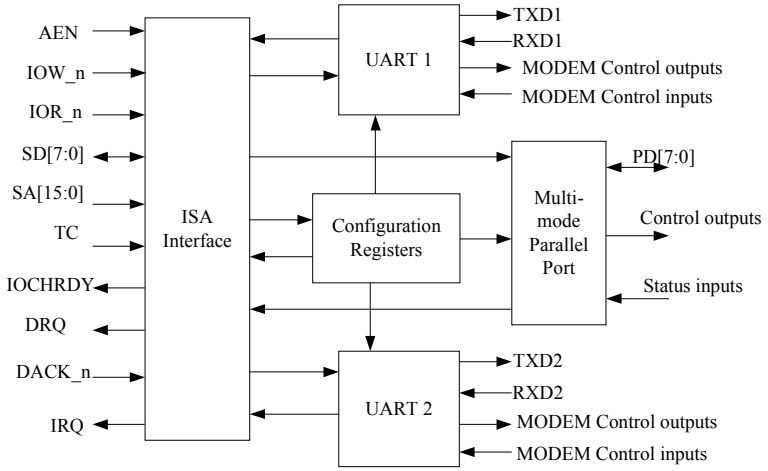

Figure 2. The Serial and Parallel Protocol Controller

The UART is full duplex and double buffered, compatible with GM16C550. The parallel protocol controller is compatible with IBM PC/XT and PC/AT architecture, and supports EPP and ECP protocol. In addition, the serial and parallel communication protocol controller is programmable, and can be enabled or disabled through configuration registers. The allocation of interrupt/DMA channel and communication protocol selection can also be designated through configuration registers.

A. The UART Design

The UART transmitter and receiver both have internal 16-byte FIFOs, which can be activated in FIFO mode. The UART receives the serial data from peripherals or MODEM and converts the serial data to parallel data. On the other hand, the UART converts parallel data from CPU to serial format. During UART operation, CPU can read complete UART status at any time, including communication types completed, and error information, etc [1].

1) Control of receiving

After initialization, the UART is prepared for receiving data and keep monitoring the serial input signal line. When a low level sample of input signal line is detected on the rising edge of the receiving clock, it is considered as the start bit and an internal counter is activated. From the moment on, after continuous eight low level samples are detected, the correct start bit is confirmed. Afterwards, every one bit time (16 pulses), the input signal line is sampled on the rising edge of receiving clock, just corresponding to the middle position of one bit time. The sampling continues until the whole character is received.

After receiving one frame, the synchronization control circuit removes start bit, parity bit and stop bit automatically according to the data format and baud rate set during initialization. Then the serial data are shifted in. After the Receiver Shift Register (RSR) receives one character, the data will be sent in parallel to the receiving data buffer or the Receiver FIFO (RCVR FIFO), and the 'receiving data ready' bit in the Line Status Register (LSR) will be set to '1', ready to be queried by CPU. If the interrupt is allowed, then the UART will send an interrupt to CPU, requesting read operation of received data. Meanwhile, the receiving synchronization circuit validates the data, such as parity bit, stop bit etc. If the format error is detected, the UART will set error flag in the LSR and send an interrupt to CPU requesting error handling.

The synchronous FSM is designed for controlling receiving process, as shown in Fig. 3.

2) Control of transmitting

When the Transmitter Holding Register (THR) is empty or the Transmitter FIFO (XMIT FIFO) is not full, the parallel data from CPU will be written into the THR or the XMIT FIFO. Then these data will be sent to the Transmitter Shift Register (TSR) when the TSR is empty. Under the synchronization of transmitting clock, the data in the TSR are converted into serial format and sent through the serial output pin. During the conversion, the start bit, parity bit and stop bit are added automatically according to the format settings. When there is no data left to transmit, the serial output is kept at high level.

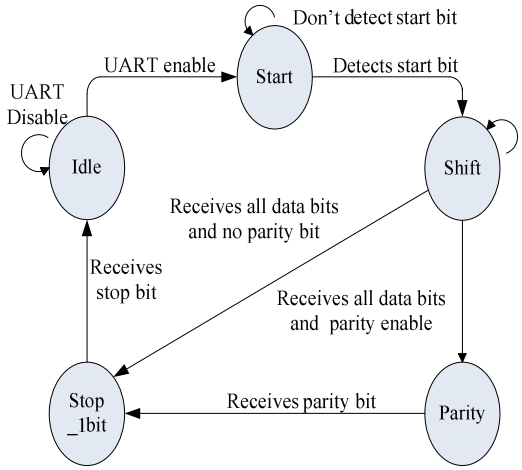

Figure 3. Receiving Synchronous FSM

Because the THR or the XMIT FIFO are written during the transmission of previous data frame, the next data frame will be transmitted immediately right after the completion of previous frame transmission, as long as the THR or the XMIT FIFO are not empty. The synchronous FSM is designed for controlling transmitting process, as shown in Fig. 4.

B. The Design of Parallel Protocol Controller

The IEEE 1284 standard parallel protocol controller supports IBM XT/AT compatible parallel port, PS/2 bidirectional parallel port, EPP (Enhanced Parallel Port) and ECP (Extended Capability Port). The communication mode includes Compatibility Mode, Nibble Mode, Byte Mode, Enhanced Parallel Port Mode and Extended Capability Port Mode [2]. The mode and base address (378h、278h、3BCh) can be set through configuration registers [5]. For asynchronous operations described in IEEE 1284 standard, the input and output operations are implemented through handshakes with peripherals. That is, the data are read or written on the rising or falling edge of handshake signal. The synchronous design of parallel protocol controller is presented in the paper, in which the data can be input or output on the rising edge of the clock while the handshake signal is active.

In IEEE 1284 standard, any transfer among communication modes needs negotiation. The negotiation is initiated by the host, and the mode transfer can only happen just after successful negotiation, otherwise the Parallel Protocol Controller has to work in Compatibility Mode. In order to make mode transfers, the host FSM has to enter termination state first and then goes back to the idle state, which means the Parallel Protocol Controller returns to default Compatibility Mode. Afterwards, the host FSM begins negotiation again to enter the target mode. The above process is obeyed unless the target mode is Compatibility Mode. The two level FSMs are designed to implement the parallel protocol controller, as shown in Fig. 5. The CPU originates the negotiation by writing '1' to 'expanding request bit' of data register.

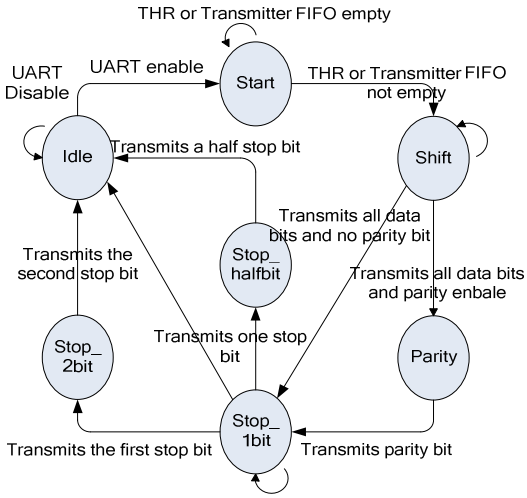

Figure 4. Transmitting Synchronous FSM

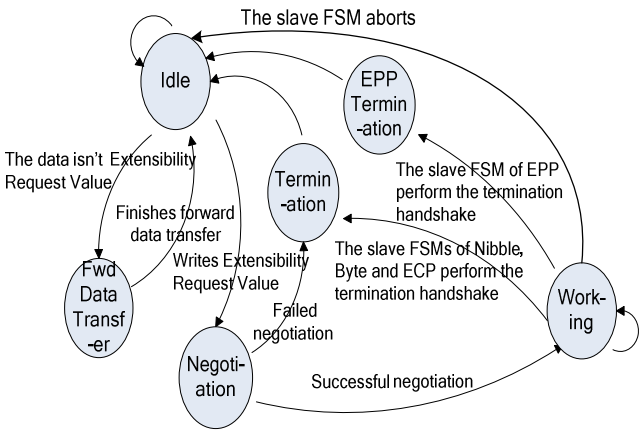

Figure 5. Host FSM

Once the negotiation successes, the host FSM starts the slave FSM, and the corresponding FSM begins to run. The host FSM is in working state when the slave FSM is running. The host FSM will start terminating process and then return to idle state when the slave FSM sends the terminating signal.

Limited to length, the paper only discusses the ECP mode in detail below.

The ECP Mode is a high speed, bidirectional parallel communication mode, and is compatible with standard PC parallel port. The ECP and EPP modes have the characteristic of bidirectional transmission in common. Besides, the ECP Mode allows transmitting dada through DMA, transmitting/receiving data through FIFO buffer, and compressing data at run-time using RLE (Run Length Encoding) [3]. Through the configuration of highest three bits in the Expand Control Register, the ECP Mode is compatible with all the communication modes in IEEE 1284 Standard.

The whole data transmission process is controlled by hardware, and the synchronous FSM is designed to shake hands with peripherals. The state transfer diagram is shown in Fig. 6.

If the negotiation is successful and the mode field in expand control register is set as ECP operation, the host FSM signals the ECP mode slave FSM to start. Then the slave FSM enters setup state, otherwise stays in idle state. While in setup state, the slave FSM transfers to 'forward idle' state if the peripherals are revealed in the ECP Mode. In 'forward idle' state, the ISA writing signal is kept detected, and the forward transfer will begin as soon as the host accesses the ECP address FIFO or data FIFO. If the host starts terminating process, the slave FSM returns to ECP idle mode. During the forward transfer, the host will perform 'Host Transfer Recovery' operation to recover the transfer if the peripherals do not response in 35ms. If the host request reversing the data channel, the slave FSM transfers to 'forward to reverse' state and starts reverse transfer. In the same way, the host can request reversing the data channel again to return to 'forward transfer' state.

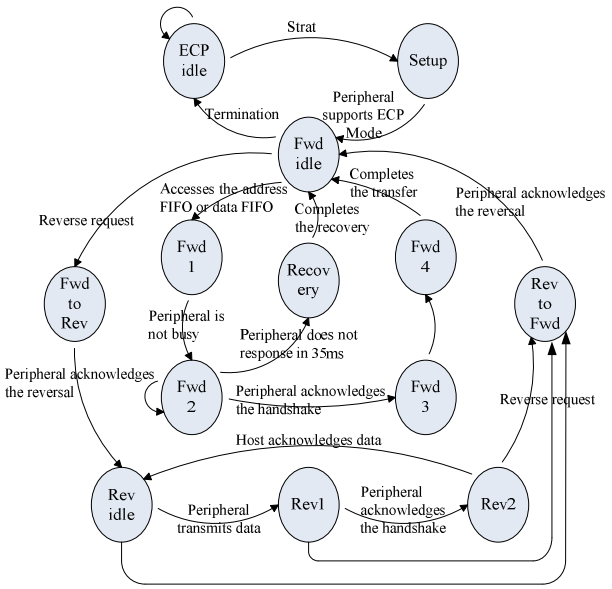

Figure 6. ECP Mode slave FSM

IV. SYSTEM VERIFICATION

The platform is built to verify the design of protocol controller, as shown in Fig. 7. The verification platform includes a FPGA, a main board with ISA slot and serial/parallel interface, and a circuit board with SDRAM. The kernel of the verification platform is the XC2V8000 FPGA from Xilinx for loading the design [4]. After the design is loaded, the logic will be initialized by the program ROM. Then the program in the ROM will be executed to carry out the communication with PC. During the operation, the diagnostic card on the ISA slot can output the run-time status information.

There are two methods to debug FPGA. We can use soft logic analyzer in Xilinx FPGA to capture the system state under the specified condition or when the specified signal value appears. We can also observe the output information from diagnostic card to track the execution of application program, making sure the system is running correctly. Besides, the Debugging software tools for serial and parallel port should be installed on PC to carry out the data exchange with FPGA. A lot of test programs have been executed on the platform, and the results have verified that the serial and parallel communication protocol controller communicates with PC correctly and smoothly.

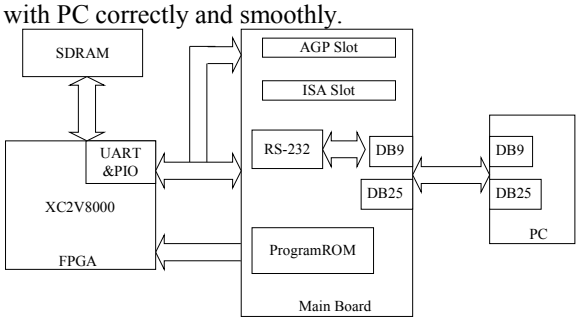

Figure 7. FPGA verification platform

V. CONCLUSION

The serial and parallel communication protocol controller is designed following top-down methodology. The serial receiving/transmitting and different parallel communication mode are implemented in synchronous control pattern. The communication status flag will be set based on the monitoring of the receiving state and the correctness of the receiving data. The design has been verified on the specified platform and implements the serial and parallel communications with PC successfully. The achievements of the paper may be contributing to the similar designs or products.

REFERENCES

[1] GM16C550 Asynchronous Communications Element with FIFOs, LG Semlcon Co., Ltd.

[2] IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers, IEEE –SA Standards Board, September 2000.

[3] Extended Capabilities Port: Specifications, Microsoft Development Library, July 14 1993.

[4] Virtex-II Platform FPGA User Guider, August 2004.

[5] Frank van Gilluwe, THE UNDOCUMENTED PC: A Programmer's Guide to I/O, CPUs and Fixed Memory Areas, Addison-Wesley Professional, 2 edition, December 23, 1996.