Smart re-use of hardware peripherals for better software UART

Jan Bělohoubek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
jan.belohoubek@fit.cvut.cz

***Abstract*—In this work, the efficient software implementation of UART is presented. The efficiency is achieved by using the microcontroller peripherals intended for the different purpose.**

I. Introduction

Today, microcontrollers are used in wide spectrum of applications. These are used to control various processes or in identification devices such as smart cards or key fobs. The amount of embedded control in different devices still rises. This caused, that the low-power design of microcontrollers became very important.

The hardware features and number of peripherals of the low-power microcontrollers are limited to reduce power consumption. The low-power microcontroller producers offer the products with the almost identical core and different sets of peripherals targeted for the specific applications. Such microcontrollers belong to *ASIC*s (Application Specific integrated Circuits).

The feature set of ASICs is limited. This applies especially to low-power ASICs because every additional part of the chip influences the power consumption. If a new product is introduced, it is advantageous, if the time on the market is as longest as possible. To achieve this, new applications of the product should be found. But new applications may require features originally not implemented in hardware.

The microcontrollers contain number of *GPIO* (General-purpose Input/Output) pins – these can be used to implement new chip features only by modifying the software. This reduces both the development and the manufacturing costs such as the time-to-market.

Unfortunately, the performance of any software implementation is reduced compared to hardware-supported implementation especially for low-power/low-performance microcontrollers. The overall performance of the microcontroller naturally degrades and the power consumption rises, because new (potentially sophisticated) tasks are performed.

II. Communication protocols in embedded systems

Today's systems often contain more collaborating microcontrollers. Today cars are a good example [1]. If there is an option, especially in the low-power area, the simplest solution should be used. This is why the old – traditional – protocols are very popular.

For the inter-chip communication, serial communication is advantageous. It reduces number of communication problems [2].

In *single-master, multi-slave* systems, the *SPI* (*Serial Peripheral Interface*) is often used [3]. In *single-master, multi-slave* or *multi-master, multi-slave* systems *I^2C* (*Inter-Integrated Circuit*) bus is widely used. For point-to point connections, years ago, *UART* (*Universal Asynchronous Receiver/Transmitter*) (*RS-232*) became de facto standard for communication between embedded systems and personal computers [2], [4].

Even if the RS-232 port has been replaced by the *USB* (*Universal Serial Bus*) port in today computers, UART is still prioritized in a wide spectrum of applications because of its simplicity. To connect a microcontroller with the UART interface into today's PCs equipped with USB ports, RS-232 to USB converters are available [5]. For devices using different voltage levels, simple level shifters may be used.

As denoted above, a low-power embedded system should not have a huge amount of different communication interfaces. If some unavailable communication protocols are required, they must be implemented (at least) partly in software. In this paper, an efficient software implementation of UART interface will be presented. The key idea is to effectively use available hardware peripherals to emulate UART behaviour.

A. SPI/UART point-to-point configuration comparison

UART (RS-232) uses at least two signals. One is used for data transmission (`Tx`), another (`Rx`) for reception. When no data transmitted, the signal remains in the *mark condition* (logic `1`). The transmission is initialized by the `START` bit with an inverted polarity compared to the mark condition. The start bit is used to synchronize the receiver. After the start bit, the defined number of data bits is transmitted and the transmission is finished by the `STOP` bit. The clock precision of transmitter and receiver must be sufficient to keep the clock synchronization in the interval between the start and stop bit. See Figure 1.

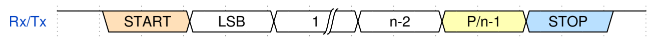

Fig. 1. UART transmission example. Mark condition value is equal to logic `1`. Transmission starts with `START` bit (`0`), after n bits, `STOP` bit (`1`) is transmitted. The last data bit (`n−1`) often serves as parity. After the transmission is finished, the signal remains in mark condition (value `1`).

SPI is a synchronous master-slave protocol. The transmission is initialized by the master. At least three signals are used – two data signals and a clock signal. The master generates the clock signal (SCL). Data from the master are transmitted to the slave using the signal denoted MOSI and data from the slave to the master using the signal MISO at the same time. The signal values are changed with a selected (rising or falling) edge of the clock signal and sensed with the other edge. See Figure 2.

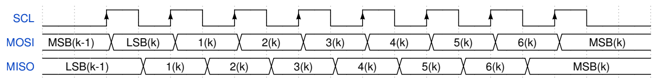

Fig. 2. SPI data transmission example – 8 bits, LSB first. Master reads incoming data on the rising edge and slave on the falling edge. Clock signal returns to zero after transmission is finished. Different transmission schemes are possible while the principle remains.

The total number of transmitted bits in UART is the number of data bits plus 2, compared to SPI, where only data bits are transmitted. The typical number of data bits for both interfaces is 8 (for UART ASCII terminals 7 bits are often).

Because SPI is a synchronous protocol, it requires the clock signal generated by the master and transmitted to the slave. Additionally, communication can be initialized only by the master (the slave can request data transfer by using a dedicated signal). On the other hand, UART receiver is synchronized by the start bit. Both sites can initialize the communication. Both protocols allow full-duplex communication – data are transmitted using dedicated signals in both directions. Another difference is that UART works in the return-to-zero mode – the mark condition value is held on data signals, when no transmission is in progress. In SPI, the value on data signals is not defined when the clock signal is inactive – usually the value of the last transmitted bit is present. The interconnect configuration distinctions are shown in Figure 3.

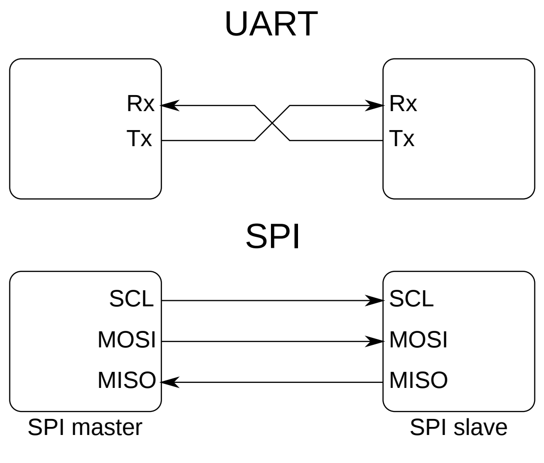

Fig. 3. SPI and UART configuration for point-to-point communication.

III. Traditional software UART implementations

In this section, the most common approaches used to implement UART in software will be described. The description is intentionally brief because UART implementations are common, e.g. [6], [7], etc. The excessive details are suppressed. Techniques combinations are possible.

The synchronization is performed at the beginning of the transaction and it must be kept until STOP bit is received. To realize successful communication, the UART clock deviation during transmission and reception must be minimized on both sides (receiver and transmitter). This usually requires to disable (subset of) interrupts depending on selected baud rate and the microcontroller performance. This may not be necessary, if the highest-priority interrupts are used.

It is assumed, that two GPIOs are used as Rx and Tx pin. Although periodical signal sampling may be used, using interrupts is usually advantageous and helps to increase the performance.

If the reception is implemented by using ISRs, the baud-rate is additionally limited by *ISR* (*Interrupt service routine*) invocation and execution time. The interrupt must be invoked and processed when the START bit is transmitted (at worst, it must be finished during the first data bit transmission).

Full-duplex communication is a complicated task for the software-implemented UART. Concurrent transmission/reception is simpler to realize, when data for the transmission is ready before START bit is received. The transmission is then performed during data reception initiated by the other communicating side.

A. *Instruction-counting-based techniques*

Instruction counting allows to achieve relatively high transmission speed. The transmission speed is only limited by the ability of microcontroller to switch values at the output (Tx) pin. If lower baud-rates are used, instruction-counting-based delay is used to achieve the defined period. This mostly means, that the microcontroller performs no beneficial actions (*NOP* instructions). Instruction-counting-based delay can be also introduced to achieve a defined delay between the sampling of data bits.

Assembly language implementation is generally required, especially when higher baud-rates are used (accurate number of instructions must be used during every period). Not required interrupts should be prohibited during both transmission and reception, especially when baud rate is high compared to microcontroller clock-speed.

Instruction counting is not very flexible. Changing the baud rate may require much afford. Still, instruction-counting-based techniques may be used to introduce shorter delays anywhere needed.

B. *Timer-based techniques*

The software-based UART clock is derived from a timer. The timer interrupts are generated periodically according to the selected baud rate. The data are received/transmitted bit-by-bit during consecutive *ISRs*.

Assembly language implementation is not necessarily required. The reachable baud rate is limited by *ISR* invocation and execution time. Other than timer interrupts should be disabled during transmission/reception.

A timer offers more flexibility than instruction counting when changing the baud rates.

IV. Proposed UART implementation

In this paper, efficient software-based, hardware-supported UART implementation, called *SPIUART*, is presented. The core idea is to use existing microcontroller peripherals to emulate UART. In the following lines it will be presented how to efficiently implement UART using an available SPI interface.

Similarities and distinctions between SPI and UART were briefly described in II-A. In the following lines, 8-bit SPI interface is assumed. Both communicating sites can be communication initiators when UART is used. This implies that the SPI interface emulating UART must be set to *master mode*. In SPIUART, the SPI pin directions are preserved. The `MOSI` pin represents the UART `Tx` pin and the `MISO` pin represents the UART `Rx` pin – see Figure 4.

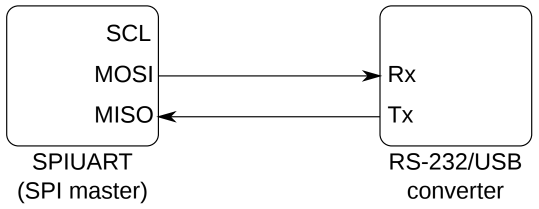

Fig. 4. Proposed configuration for point-to-point communication.

If SPI is implemented using 8-bit shift register, at most 8-bits can be transmitted. The comparison noted above implies that two bits (the first and the last) in the UART transmission have a special meaning – the `START` and `STOP` bit. For the transmission, the SPI shift register must be configured in such a way to transmit at first the `START` bit, then 6 data bits and finally the `STOP` bit. This implies that SPIUART can be used to transmit 6 data bits. If the receiver is configured to receive more than 6 data bits, `STOP` bit values are received.

The data reception has to be implemented as follows: The interrupt must be configured to the edge of the `START` bit. When `START` bit is detected, ISR is executed. In the ISR, the `MOSI` register is filled in by 8 `STOP` bits and SPI transaction is initialized. During the reception of 8 bits (`Rx`), the parallel transmission of 8 `STOP` bits is performed. It preserves the *mark condition* value at the `Tx` pin (constant *mark condition* value means *no transmission*). `STOP` bits are shifted to the `MOSI` line while the bits from the `MISO` line are shifted into the input SPI register.

Up to 8 bits can be received using SPI interface, if the first received bit is the bit following the `START` bit and the `STOP` bit is not considered.

The full-duplex communication is possible in a limited way. During the reception, transmission can be realized if `START`, `STOP` and 6 data bits are loaded into the `MOSI` register instead of the 8 `STOP` bits. The duplex communication must be initialized by the full-featured UART interface.

The resulting implementation is the (limited) full-duplex UART. The data bit length is up to 8 (up to 6 data bits can be transmitted and up to 8 data bits can be received). No flow control signals are considered.

Note that it is possible to transmit 8 data bits using SPI interface if new data can be loaded into the `MOSI` shift register after actual content is transmitted. However this brings additional timing requirements and it is also not considered advantageous.

V. Experimental results

Proprietary microcontroller of *EM Microelectronic-Marin SA* based on *CoolRISC* core has been used for experiments.

A. Used microcontroller characteristics

The detailed information about the microcontroller used for experiments are confidential. Yet very similar products offered by *EM Microelectronic-Marin SA* for general purpose are *EM6812* [8] or *EM6819* [9].

The microcontroller is equipped with 10 freely programmable I/Os. It has an adjustable RC oscillator with frequency range up to 10MHz. Its 8-bit RISC architecture is designed for very low power consumption. With 2 clocks per instruction, the used microcontroller executes up to 5 MIPS at 10 MHz. The instruction word length is 22 bits and the available program memory allows to store up to 16k instructions (44 kB Flash memory). 512 bytes of on-chip static RAM is available. For more details about CoolRISC, see EM6812 datasheet [8].

B. SPIUART implementation

The solution proposed in IV was successfully implemented for the mentioned microcontroller. The implementation was completely realised in C programming language without any assembly construction. The standard tools and the development kit supplied by the manufacturer including *GCC 4* based C compiler were used.

As described in IV, SPI interface is in the master mode. Transmission is performed as a regular SPI transaction with fixed 2 (`START` and `STOP`) of 8 transmitted bits. The reception is started from ISR after the start bit is detected (falling edge on `MISO`). The end of both transactions is detected by *end of SPI transmission* interrupt.

The SPI clock is generated using integrated *PWM* (*Pulse Width Modulation*). Usage of PWM allows simple modification of the baud rate up to 9600. This is the limit baud rate for the used microcontroller. The time constraint is composed from the time required for the `START` bit detection and from the SPI start-up procedure.

As noted in section IV, up to 6 valid data bits can be transmitted and up to 8 data bits can be received. One can ask how to configure the other side interconnected with SPIUART. Complete symmetry is achieved, if 6 data bits are used for both transmission and reception. To maximize the data rate in direction to SPIUART, 8 bit length should be used.

If 7 data bits are considered, the MSB is always `1` during transmission from SPIUART. If ASCII terminal is used, almost all received data are displayed as readable characters (excluding `0x7F`). SPIUART is able to receive up to 8 data bits – accepting only 7 valid data bits is not any problem at all (bit masking is a simple operation in software). This behaviour can

	EM68xx softUART	SPIUART FS1	SPIUART FS2
Transmission bit length	8	6	6
Reception bit length	8	8	7
Baud rate [bit/s]	fixed 2400	up to 9600	up to 9600
Total number of ISRs executed during transmission	9	1	1
Number of ISRs executed during transmission per data bit	1,125	0,167	0,167
Total number of ISRs executed during reception	10	2	2
Number of ISRs executed during reception per data bit	1,250	0,250	0,286
C code including preprocessor directives	up to 140 lines	up to 250 lines	up to 300 lines
Assembly language equivalent	240 lines	127 lines	625 lines
Number of instructions	972	504	2496
Used programme memory	6%	3%	16%
Number of *NOP* instructions	87	0	0
Instructions in ISRs	656	268	268
The share of instructions in ISRs	67%	53%	11%
Used I/O PINs	2	3	3

TABLE I. SOFTWARE UART IMPLEMENTATIONS COMPARISON.

be advantageous for debugging. This is why 7 is considered to be a good choice.

Two feature sets were implemented. The first feature set (denoted *FS1*) allows only transmission of 6 data bits and the reception of up to 8 data bits. In the second feature set (denoted *FS2*), additional functionality is included. This feature set includes functions for sending byte arrays in 6-bit words and receiving byte arrays in 7-bit words. Transmissions are secured using one parity bit for every word or for the transmitted array[1].

The number of I/O pins occupied by the proposed implementation is 3. Used pins are `Rx` (`MISO`), `Tx` (`MOSI`) and `SCL`, which is not necessary for UART but it remains occupied while the SPI peripheral is in master mode.

Interrupts should be inhibited only when higher baud rate is used and `START` bit is expected. After the SPI transaction is started, interrupts can be permitted immediately.

Both the feature sets were compared to the implementation of software UART supplied by EM Microelectronic-Marin SA for EM68xx device family [10] (it is implemented similarly to III-B). This implementation is functionally comparable to FS1. The results of this comparison are in Table I.

VI. CONCLUSIONS

The proposed software implementation of UART exploits the available microcontroller features. It is presented, that reusing hardware intended for the different purpose can be more efficient than full software implementation even if some nice properties of emulated hardware were sacrificed (data bit length symmetry for transmission and reception). In the resultant implementation, the number of ISR calls is reduced. The baud rate is relatively high while interrupts are permitted during transactions.

[1]For the full-featured communication with *SPIUART FS2* from Windows-based PC station, basic C library was implemented.

ACKNOWLEDGMENT

The author would like to thank to ASICentrum s r.o. for the provided hardware and the helpful cooperation. This work has been in part supported by CTU grant SGS15/119/OHK3/1T/18.

REFERENCES

[1] EM Microelectronic-Marin SA, "Automotive," http://www.emmicroelectronic.com/applications/automotive-0, 2014.

[2] A. S. Tanenbaum, *Structured Computer Organization.*, 5th ed. Pearson Education, 2007.

[3] Martin Schwerdtfeger, "SPI – Serial Peripheral Interface," http://www.mct.net/faq/spi.html, 2006.

[4] Christopher E. Strangio, CAMI Research Inc., Acton, Massachusetts, "The RS232 STANDARD," http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html, 2015.

[5] *FT230X (USB to BASIC UART IC)*, Future Technology Devices International Ltd., 2 2015, version 1.3.

[6] "Implementation of a software uart on tms320c54x using general-purpose i/o pins," Texas Instruments, 7 1999, application Report, SPRA555.

[7] Prashant Mehta, "Software_UART – UART stack implemented in C for uNiBoard v1.1 (ATmega128)," https://code.google.com/p/uniboard/wiki/Software_UART, 2010.

[8] *EM6812*, EM Microelectronic-Marin SA, 5 2005, rev. E.

[9] *EM6819Fx-A00x, EM6819Fx-A10x, EM6819Fx-B00x, EM6819Fx-B10x*, EM Microelectronic-Marin SA, 10 2014, version 9.2.

[10] EM Microelectronic-Marin SA, "EM6812 Downloads," http://www.emmicroelectronic.com/products/microcontrollers/multi-io/em6812#node_prod_full_group_downloads, 2004.