Reducing Energy Consumption in Asynchronous and Synchronous Data Transfers

Lubomir Valeriev Bogdanov and Racho Marinov Ivanov

Abstract – **Deeply embedded systems use general purpose microcontrollers that have a fixed inner architecture. Each chip houses many peripheral modules and usually not all of them are used. In this paper we try to better utilize the unused resources of a chip in the favour of reducing the energy consumption. The asynchronous and the synchronous interfaces are the focus of our studies.**

Keywords – **embedded systems, energy optimization, asynchronous/synchronous interfaces.**

I. INTRODUCTION

The following paper shows an attempt to reduce the energy consumption of an embedded system by better utilizing the unused UART and SSI modules. The tested boards are Texas Instruments' EK-LM4F232 using an ARM Cortex-M4 based microcontroller. The mentioned modules are in excess, having 8 x UART and 4 x SSI.

The main idea behind the experiment is to reduce the time of the transmission frame which in turn would lead to energy savings. The problem that needs researching is whether the dynamic power of the additional modules would add up and exceed the time-energy savings or not. That's why we must test this case and use the two possible types of data transfers:

- Asynchronous
- Synchronous

Intuition makes us think that the fixed baud rate of the asynchronous data transfers would exclude the possibilities for energy reduction using time reductions. On the other hand the synchronous data transfers will achieve such reductions because the time length of the bit can be varied.

II. THE ASYNCHRONOUS CASE

Even though the UART interface may seem outdated, the purpose of the experiment is independent of the specific interface. All that matters is that the bit time length is standardized and is not related to the system's main clock. This means that as the system frequency increases, the energy consumption will also increase. This is due to the fact that dividers are used inside the module to scale down the frequency.

The energy optimization is not possible unless we make some additional changes to the system. One of the approaches is to increase the baud rate. But in our case we consider this parameter as a constant –it's a decision of the firmware developer. Next, the only possible solution is to look for hardware enhancements that would lead to energy reduction.

In Fig. 1 a basic approach for UART data transfer is shown between two devices. Each data token is at least ten bits wide (1 start, 8 data and 1 stop bit) [1]. Let's assume we have a fixed number of data tokens that need to be transferred from device A to device B. In this case we use a single UART.

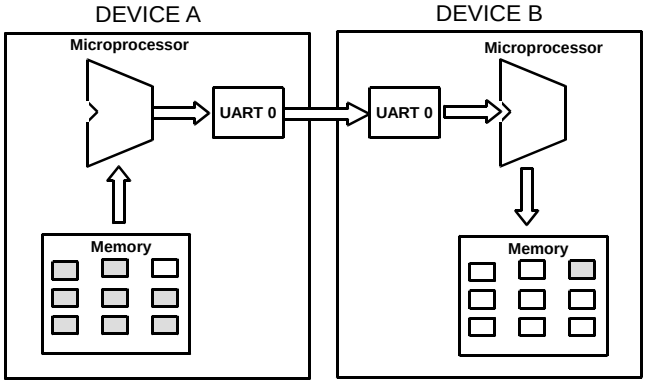

Fig. 1. Single channel transfer of data tokens over an asynchronous interface.

At any instant of time we can transmit only a single data token. The reading and writing is buffered (exactly the case of the UARTs in the LM4F232 [2]). The microprocessor will have to wait in an endless loop before the last data token is send from the UART buffer. During this time no other code can be executed.

The first obvious energy optimization can be achieved exactly in this loop. The ARM Cortex-M microprocessors can execute the WFI (Wait For Interrupt) instruction [3] which will gate off the clocking signal from the core without stopping the generator and the rest of the peripheral modules. The only energy leak in this case would be the one induced by the static power of the core and the additional stack push/pop upon entering the sleep state.

Then we can further develop this case by trying to utilize the hardware better. As we previously stated the rest of the UART modules remain unused. If we make a multi-channel communication link, the time for transmitting one array of data tokens would be reduced. This means that the energy for the transmission may be reduced and this statement must be checked with an experiment. For a successful optimization the energy reduction due to the speed up must compensate the data processing overhead of the message aggregation. The multi-channel interface would require that the message be decomposed into small

L. Bogdanov is with the Department of Electronics, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: lbogdanov@tu-sofia.bg

R. Ivanov is with the Department of Electronics, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: r.ivanov@tu-sofia.bg

chunks of data tokens at the transmitter side and be brought back together (aggregated) at the receiver side. We cannot avoid this process by any means. The multi-channel version of the example is shown in Fig. 2. Notice the number of the transferred data tokens per single communication time quant.

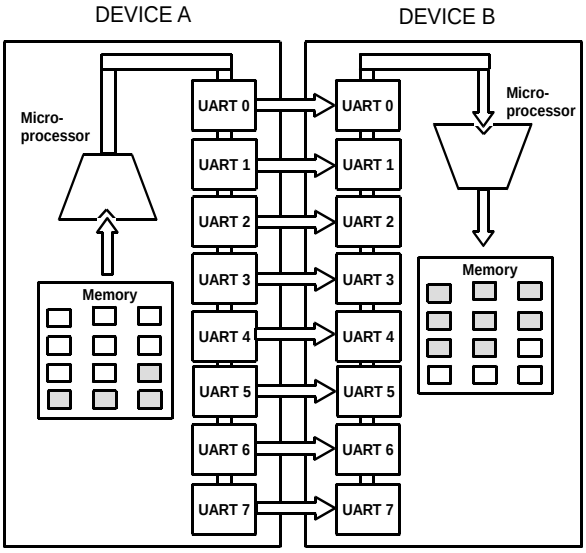

Fig. 2. Multi-channel transfer of data tokens over an asynchronous interface.

Again we can apply the sleep mode here – the microprocessor operates at much higher frequency than the UART modules, meaning that the time for filling their buffers is negligible compared to the time for sending the data tokens over the interface.

III. The Synchronous Case

The same experiment can be conducted with a synchronous interface, such as the SPI. With the SPI, the time length of one bit is not constant and is reduced with the increase in system frequency. Again the data transfer is buffered and that's why we can also use the ARM Cortex-M sleep mode. There will be a slight difference compared to the UART experiment - the number of the SPI modules is 4. Furthermore, the interface being a synchronous one would require a clocking signal. The latter means that if we increase the number of the communication channels, then we increase the number of the clocking signals. The clocking signal is a square wave with a 50 % duty cycle that produces many 0-to-1 and 1-to-0 transitions during the transmit time frame. This yields high dynamic power and the energy optimization due to the time decrease might be overwhelmed by the consumption of the additional clocks.

Texas Instruments have named their SPI module SSI because it can operate in various modes, including the legacy SPI. In our experiments we use SPI mode, CPOL = 1, CPHA = 1 and 8-level buffered data transfers.

IV. Experiment Setup

The setup of our experiment is shown in Fig. 3. We use two EK-LM4F232 boards whose UART interfaces are connected together. We spare UART number 0 for controlling the experiment through a PC. Actually we needed two PC hosts due to technical difficulties concerning closed source software. We have also used one GPIO of each LM4F232 microcontroller for a SYNC signal. The SYNC signal is used to measure the time duration of the transmission/reception. The $I_{DD}(t)$ signal is produced by a differential amplifier built into the demo board. It represents the instantaneous current consumption of the microcontroller. Those two signals (SYNC and $I_{DD}(t)$) are fed into a Tektronix oscilloscope.

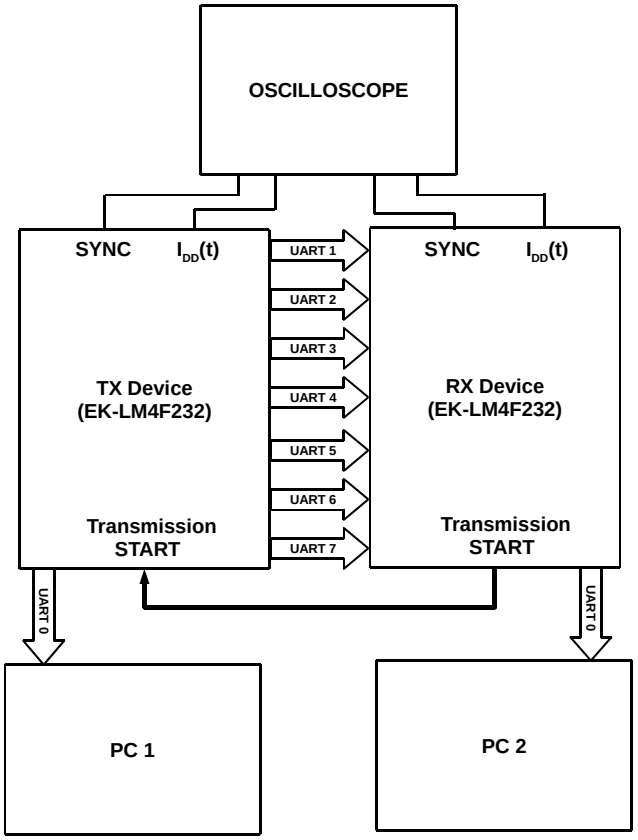

Fig. 3. Experimental setup for measuring energy consumption in one-way UART data transfer.

Thus, if the current consumption does not change during the time frame (outlined by the SYNC signal), we can measure the energy:

$$E_{TOT} = E_{TX} + E_{RX} = \\ = V_{DD}.\left(I_{DDTX}.\Delta t_{TX} + I_{DDRX}.\Delta t_{RX}\right) \quad (1)$$

where E_{TOT} is the total energy spent for one complete data transmission, I_{DDTX}/I_{DDRX} – the current consumption of the transmitter/receiver for the time period $\Delta t_{TX}/\Delta t_{RX}$ and V_{DD} = 3,3 V. The sum:

$$t_{TOT} = \Delta t_{TX} + \Delta t_{RX} \quad (2)$$

represents the total amount of time t_{TOT} that is consumed by the transmitter and the receiver (not elapsed time) to make one data transmission. Exactly this time is of interest to us. The current consumption is technology- and architecture-

dependant and won't be discussed in this paper. The energy reduction in the receiver-transmitter system can be achieved through reduction in t_{TOT}.

The firmware of the devices is separated in two projects – one for the UART, and one for the SSI case. Each project is further split into a one channel and multi-channel versions. Then it is further split into two – with and without sleep( ) API calls. The latter variations are implemented with #if-else directives. The test code remains exactly the same throughout the projects except for the communication modules initializations and low-level send/receive functions (the first case is for UART, the second – for SSI). We use GCC cross compilers for ARM [4] and both codes, the receiver's and the transmitter's, have been compiled with the same version and command line parameters of the tools. The StellarisWare libraries [5] are also identical. We send 910 chars from device A to device B in both cases.

In our test we use bare-metal firmware whose structure is shown with the pseudo-code below.

```
//Transmitter side
while(1){
    waitForStartSignal( )
    SYNC = 1
    sendMessageOnUART( )
    SYNC = 0
}

//Receiver side
while(1){
    SYNC = 1
    sendStartSignal( )
    waitForEntireMessage( )
    verifyMessage( )
    SYNC = 0
}
```

The sleep API is added in the waitForEntireMessage( ) function that actually does nothing useful - it just loops endlessly while the message is being received in UART interrupt handlers.

The multi-channel version slightly differs from the single channel one. The init( ) (not shown) function is altered. Then the decomposeMessage( ) and the aggregateMessage( ) functions are added.

```
 //Transmitter side
while(1){
    waitForStartSignal( )
    SYNC = 1
    decomposeMessage( )
    sendMessageOnUARTs( )
    SYNC = 0
}

//Receiver side
while(1){
    SYNC = 1
    sendStartSignal( )
    waitForEntireMessage( )
    aggregateMessage( )
    verifyMessage( )
    SYNC = 0
}
```

Also the number of the interrupt handlers is increased. The multi-channel version with sleep( ) uses interrupt handlers in the transmitter as well (for wake-up).

The synchronization method is the following:

- For a single-channel data transfer the size of the packet with tokens is 8. They are sent sequentially and when the last member of the packet is sent the transmitter immediately sends a new packet. The receiver is able to process the incoming packets without any delay because the work frequency of the microprocessor is higher than the frequency of the UART module.
- For a multi-channel data transfer the size of the packet is 7x8 tokens (and 4x8 for the SSI case). First, the microprocessor fills the module 1 buffers. Then it immediately fills the module 2 buffers and so on. When it reaches the last module, the Cortex-M4 loops endlessly until the last token from it has been sent. When this happens, the process starts over immediately. The receiver device gets the portions of the packet on each UART module. When the last packets is received, the message is aggregated from the receive buffers. The first buffer contains the first 8 data tokens, the second – the second 8 data tokens and so on.

As we can see from the above descriptions, the complexity of the hardware, as well as the complexity of the software, is increased. This complication is done to achieve energy reduction. The changes have to be considered in the early stages of the ESL design. If the hardware of the embedded system had already been produced, then this optimization would not be applicable. Furthermore – if the device resources are limited (less UART/SSI modules, less memory, less µP speed), the optimization will fail again.

V. RESULTS

The energy measurements are conducted for 5 different system frequencies – 10, 20, 40, 50 and 80 MHz. The results for the UART and SSI versions are shown in Fig. 4 and Fig. 5 respectively.

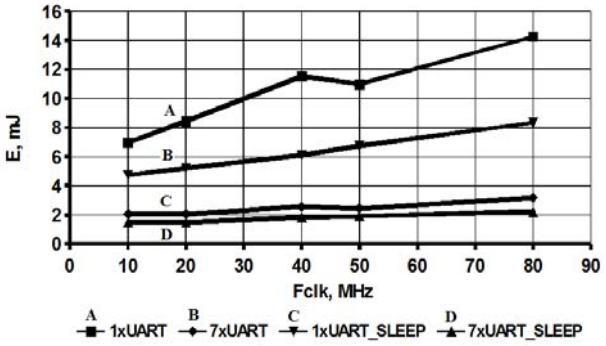

Fig. 4. Microcontroller energy consumption using UART data transfers between two EK-LM4F232 boards.

The first thing that should be noted, before jumping to conclusions, is that the flash memory of LM4F232 can work with up to 40 MHz of system frequency. For frequencies higher than this value read/write buffers are

turned on and the flash clock is reduced. That's the reason we see an energy drop at 50 MHz.

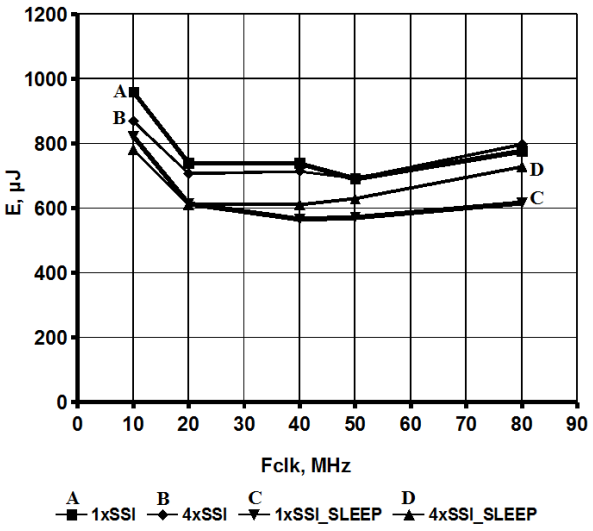

Fig. 5. Microcontroller energy consumption using SSI (SPI) data transfers between two EK-LM4F232 boards.

VI. Conclusion

Observing Fig. 4 we can conclude that the optimization method is correct. Trace D is the best case where both multi-channel connection and sleep APIs are combined. This drastic reduction is achieved mainly through time reduction – 166,0 ms for 80MHz/1xUART and 32,8 ms for 80MHz/7xUART.

The synchronous case yielded completely different results, as shown in Fig. 5. The shape of the graph is different due to the inversely proportional relation of time and frequency. This effect was expected [6]. What is interesting is that there is almost no optimization between the normal and the sleep versions. Furthermore – trace D rises high above trace C at high speeds. Even though the microcontroller is a black box to us, we suppose that this increase comes from the fact that at high frequencies the energy reduction due to the time reduction is canceled out by the dynamic power of the additional SSI modules. On the other hand the UART cases introduce no such effect due to the vast time reduction (166,0 ms for 1xUART and 32,8 ms for 7xUART) which is not that well seen in the SSI (9,37 ms for 1xSSI and 7,49 ms for 4xSSI).

Fig. 6 shows a graph of the relative energy reduction achieved with the multiple channel data transfers in the cases with and without sleep modes. We've calculated these values by simply dividing the energy value of the single over the multi-channel case (e.g. Reduction = E(1xSSI) / E(4xSSI)). The biggest relative reduction appeared in the UART case – approximately 4,5 times.

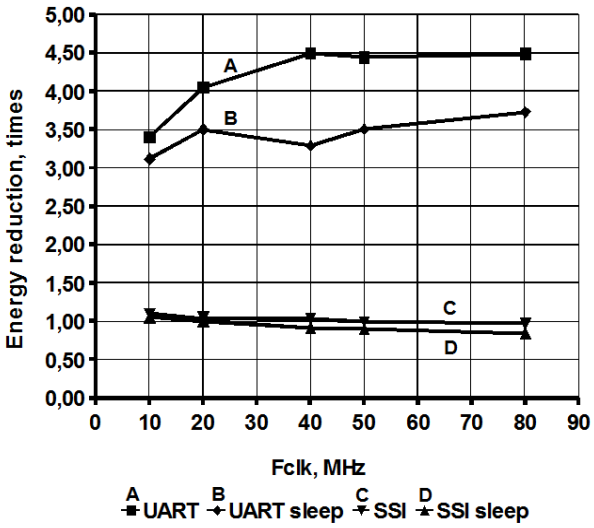

Fig. 6. Energy reduction in single/multiple channel data transfers with UART and SSI interfaces using no sleep and sleep modes.

References

[1] J. Axelson. *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*, Second Edition, Lakeview Research LLC, 2007.

[3] *Cortex-M4 Devices*, Generic User Guide, ARM, 2010.

[2] *Stellaris LM4F232H5QD Microcontroller*, Data Sheet, Texas Instruments, 2012.

[4] R. Stallman. *Using the GNU Compiler Collection for GCC version 4.5.2*, GNU Press, 2008.

[5] *Stellaris Peripheral Driver Library*, User's Guide, Texas Instruments, SW-DRL-UG-9107, 2012.

[6] R. Ivanov, L. Bogdanov. *Dynamic Frequency Scaling in Embedded Systems*, Annual Journal Of Electronics, p.95 – 98, ISSN 1314-0078, TU-Sofia, 2013.