Designing an Asynchronous FPGA Processor for Low-Power Sensor Networks

Yijun Liu, Guobo Xie, Pinghua Chen, Jingyu Chen and Zhenkun Li
Faculty of Computer
Guangdong University of Technology
Guangzhou, Guangdong, China, 510006
yjliu@gdut.edu.cn

***Abstract*—Battery-powered sensor nodes call for low power consumption. As a crucial component, a power-efficient sensor network processor greatly reduce the overall power consumption of a node. In the paper, we propose a low-power asynchronous event-driven sensor network processor mapped onto an off-the-shelf clocked FPGA. Since the processor employs a bundled-data asynchronous encoding scheme, we define a design flow that can use commercial synchronous design tools. No global clock is needed when the processor is in an idle state, thus the standby active power consumption is zero. The use of asynchronous design also results in a quick wakeup response with little design and power overheads. Moreover, an event-driven architecture is proposed to minimize the execution overheads caused by exceptions, interrupts and MMU handling of a conventional microprocessor.**

I. Introduction

With the development of micromechanics, microelectronics and integrated circuits, it is possible to integrate sensors, processing units (microcontrollers), memories, wireless communication modules and power supply systems (e.g. batteries) in a single node which is only several millimeters in dimension and costs less than one dollar. Hundreds and thousands of these nodes are organized together to form a sensor network using a wireless ad-hoc communication protocol [1]. The potential applications of sensor networks include chemical substance detecting in manufactories, environment monitoring in buildings, and habitat monitoring for wide species. In such applications, the sensor nodes are spread widely in a big area and it is very difficult (or sometimes impossible) to recharge their batteries. Therefore, battery life and power consumption are extremely important for single sensor nodes and whole networks. Minimizing the power consumption of a sensor network is a holistic problem and needs cares from the whole design hierocracy, including low-power design efforts in sensors, wireless transmission modules and communication protocols. In the paper, we mainly focus on the techniques to minimize the power consumption of processing units (sensor network processors).

The power consumption of a sensor network processor comes from two sides:

- The standby power consumed when the processor is in idle states;
- The active power consumed when the processor is executing codes — processing samples from sensors, executing communication protocols, etc.

For conventional processors, standby power may be negligible. However, for a sensor network processor spending most (may be 99%) of its time in idle states, standby power is very important. For example, a microprocessor with a 200 μA standby current will have a maximum lifetime of 1 year when powered by an AA-size battery even if it never leaves the standby state. In contrast, the lifetime of a microprocessor that burns only few μA of leakage current will be completely dominated by battery self-discharging and the active work to be done.

Many sensor nodes use low-power commodity off-the-shelf microprocessors (for example, Atmel Megal 128L in the motes [2]). These general-purpose microprocessors are not designed specially for sensor network applications. The execution overheads, such as precise exception handling, virtual memory support and OS thread contexts maintaining, use many power; while the percentage of overall active power used particularly by sensor network functions are small. To minimize execution overheads, application-specific sensor network processors (like the one in [3]) are designed. They use special architectures to handle different events from sensor nodes avoiding the requirement of an operating system like Berkeley Tiny OS [4], and therefore saving power. However, most of the existed sensor network processors are application-specific integrated circuits. In the paper, we implement an asynchronous microprocessor onto an off-the-shelf FPGA for low-power sensor networks.

The purposes of the design are:

1) Use asynchronous logic to minimize standby power, as well as active power;
2) Use special hardware architecture to minimize execution overheads and save active power;
3) Implement the processor using an off-the-shelf clocked FPGA.

The organization of the paper is as follows: Section 2 presents low-power asynchronous logic design for standby and active power saving; Section 3 describes the design flow of implementing the asynchronous processor onto a commercial clocked FPGA using a bundled-data encoding scheme; Section 4 presents the particular architecture of the processor, which minimizes the execution overheads in conventional general-purpose microprocessors; Section 5 gives the experimental results on performance and power consumption; and Section 6 concludes the paper.

II. Low-power asynchronous design

In the past two decades, people designed many asynchronous devices, such as those in [5] [6] [7]. These circuits reveal that asynchronous logic design may potentially offer advantages over conventional synchronous logic design. One of the advantages is low power consumption. In asynchronous circuits, synchronization is indicated using a handshaking protocol. Therefore, a global clock is no longer necessary in these circuits. The absence of global clock makes the standby power consumption to be zero.

"The Amulet processors are especially good at doing nothing!" — this sentence is used to describe the zero standby power consumption in the Amulet2 [5] and the Amulet3 processors [7], both designed using an asynchronous bundled-data encoding scheme.

When running out of useful work, the Amulet2 processor is set to an idle state by a 'halt' instruction which stalls a signal in the asynchronous control network. The stall rapidly propagates throughout the control circuit and brings the whole processor to an inactive state. Since there is no global clock, the standby power consumption of the Amulet2 processor is zero except for leakage. When an interrupt occurs, it will activate the blocked control signal and reactivate the whole processor immediately.

A synchronous microprocessor can also enter a low-power idle state, but only with considerable effort. The global clock must be gated off to all parts of the system, except for the interrupt circuit. An interrupt must gate the global clock back on. Since the global clock still ticks in the idle state and the power overheads of clock gating are not avoided, the standby power consumption of a synchronous microprocessor can be quite significant compared to that of an asynchronous microprocessor.

It can be argued that a synchronous microprocessor can also switch off its oscillators and phase-locked loops (PLLs). However, stopped oscillators and PLLs take a considerable time to stabilize when they are turned back on, compromising response time when an interrupt occurs. A poor interrupt response is not acceptable in real-time systems. Some synchronous microprocessors require as much as 10 ms to enter and exit standby state and even fast synchronous microprocessors have wake-up times of circa 6 μs [8]. Asynchronous microprocessors need only a few tens of nanoseconds to wake up — several orders of magnitude faster than synchronous microprocessors.

Asynchronous design can thus achieve near zero standby dynamic power consumption quickly and efficiently with very little overhead.

The standby power consumption is extremely important in sensor nodes, because they may spend 99% of their time in an idle state [1]. A node is temporarily waked up by sensing events, and after that, it goes back to sleep again. In such an application, standby power is dominant and asynchronous design gains big advantage in standby power saving.

Asynchronous design can also minimize active power consumption because of its automatic fine-grained power management. In an asynchronous circuit, synchronization is represented by local hank-shake signals. No control signals are propagated to the blocks that are not required to perform operations, thus no activities and power consumption are imposed in these blocks. The power consumption used by an asynchronous circuit only depends on the useful operations need to do. In a synchronous circuit, on the other hand, blocks are connected with a global clock. Even with a clock-gating technique, the registers and clock-gating gates are switched with the global clock, introducing unnecessary activities.

For the reason mentioned above, we decide to design the sensor network processor using an asynchronous logic design scheme.

III. Implementing asynchronous circuits in clocked FPGA

Ekanayake et al proposed low-power ASIC asynchronous sensor network processor [3], which presents attractive performance in power efficiency. However, special-purpose and application-based ASIC circuits have difficulties in the time-to-market, development investment, flexibility, etc. The asynchronous sensor network processor in the paper is designed using a commercial clocked FPGA (a Xilinx Spartan3 FPGA [9]).

Commercial FPGAs, such as Xilinx Spartan series FPGAs and Altera Cyclone series FPGAs, are designed for synchronous (clocked) circuits and not for asynchronous ones. The limitations of the basic structure of these FPGAs introduce many difficulties when mapping an asynchronous circuit onto these FPGAs. The main difficulty comes from the fact that wire delays are no longer ignorable. Fig. 1 (a) illustrates the basic structure of a Xilinx Spartan3 FPGA. The FPGA is organized by a large number of *Slices* arranged in a matrix. Each slice contains a few look-up tables and registers, in which, logics are mapped. Routing is implemented using *local switch-boxes (LSBs)* and *global switch-boxes (GSBs)*. An output (except for the carry signals) of a slice cannot directly connect an input of a slice nearby. The connection must be routed via switch-boxes, resulting in a long delay as shown in Fig. 1 (b). This property makes it difficult to implement asynchronous elements, especially for those with feedback signals. Several papers [10] [11] proposed solutions to mapping asynchronous circuits onto clocked FPGAs. However, as the development of silicon technologies, commercial FPGAs become much more complex and larger than before, making those solutions difficult to practise.

For the reason mentioned above, traditional Quasi-Delay-Insensitive (QDI) circuits are nearly impossible to be mapped onto commercial FPGAs. The proposed processor is designed using a bundled-data encoding scheme. The correctness of a circuit using a bundled-data scheme depends on the assumption that the delay of each block (including logic and

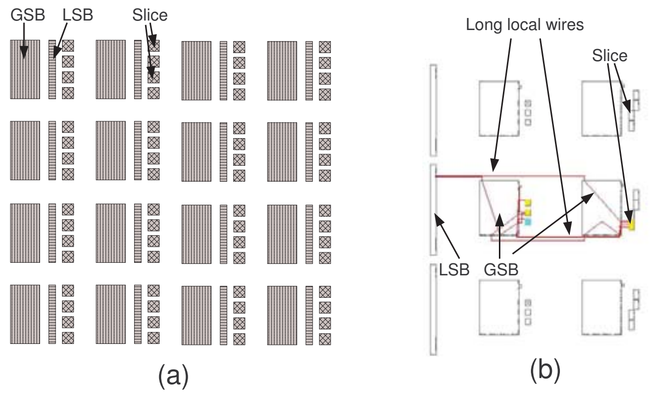

Fig. 1. The basic structure of a Xilinx Spartan FPGA and logic mapping

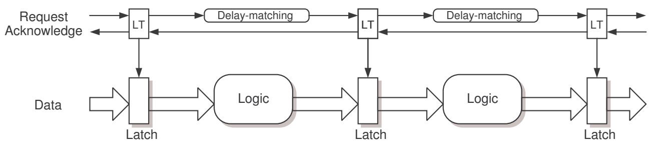

Fig. 2. An asynchronous bundled-data pipeline architecture

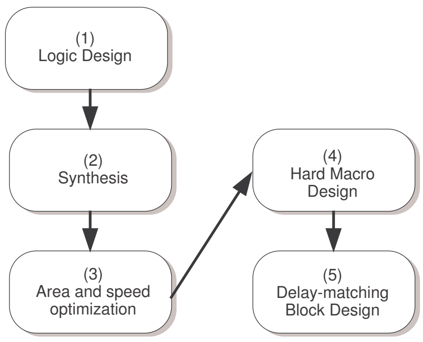

Fig. 3. The design flow of mapping asynchronous circuits onto FPGAs

routing) is predictable and designers can use a delay-matching block to satisfy timing constraint. In this case, bundled-data asynchronous design is somewhat like synchronous design, in which, commercial timing analysis tools, like Xilinx timing analyzer, are used to find the longest delay path(s) in a circuit. Based on timing analysis data, a global clock cycle is defined.

To maximize performance, asynchronous circuits are usually designed using an asynchronous pipelining technique. Fig. 2 shows an asynchronous bundled-data pipeline. The delay-matching blocks are inserted in control channel to make sure that the operations of the function blocks finish before the hand-shake control signals arrive.

The workflow of designing a function block in the proposed asynchronous FPGA processor contains five steps using Xilinx commercial series tools as shown in Fig. 3:

1) Use a schematic or a Hardware Description Language (HDL) to design the logic block;
2) Use the Xilinx ISE tool to synthesis the logic block;
3) Use Timing Analyzer to find and optimize the critical path(s); Use Floorplanner and FPGA Editor to optimize the area and routing length of the block;
4) Use FPGA Editor to map the function block into a *hard macro*, so its operation delay is no longer altered; the shape of the hard macro is made to be rectangular;
5) Make a delay-matching *hard macro*, which has a comparative delay and dimension (width or height) with the function block.

The function blocks in the datapath of the processor using a bundled-data encoding scheme are not necessary to be hazard-free since they are sampled when hand-shake signals arrive. However, any spurious transitions on the control circuit as shown in Fig. 2 will cause unpredictable operations in the datapath, thus failing logic correctness. Therefore, the control circuit must be hazard-free. The main components of the asynchronous control circuit are *latch controllers* (LT in Fig. 2), which interpret hand-shake signals to control the states of pipeline registers. Many asynchronous bundled-data asynchronous circuits use 4-phase hand-shake latch controllers [**?**] [7]. In these circuits, asymmetrical delay-matching blocks are used to minimizing the return-to-zero period. An asymmetrical delay-matching block propagates a ‘1’ slow but propagates a ‘0’ fast. As we find it is difficult to implement asymmetrical delay-matching blocks in FPGAs, we propose a 2-phase hand-shake latch controller. The Signal Transition Graph (STG) description of the latch controller is shown in Fig. 4 and its gate-level schematic is shown in Fig. 5. As can be seen from the STG, one event/transition on Rin (which means generating a data token) and one event/transition on $Aout$ (which means consuming a data token) open the pipeline latch for one time. Since there are no return-to-zero operations in hand shakes, the datapath using the 2-phase latch controller is faster than the one using a 4-phase latch controller. The latch controller is also designed as a rectangular hard macro.

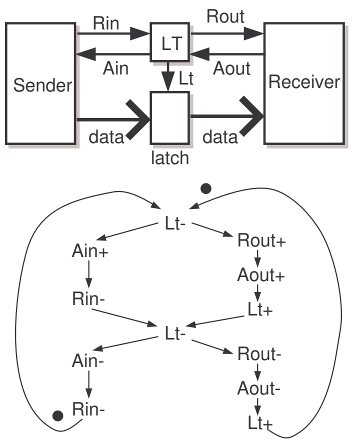

Fig. 4. STG description of the proposed 2-phase latch controller

Once all the hard macros for the latch controller, the logic

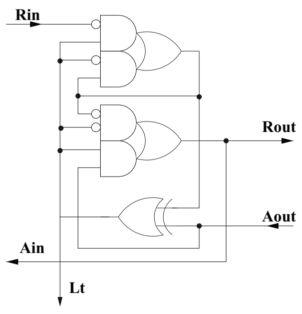

Fig. 5. The latch controller implementation

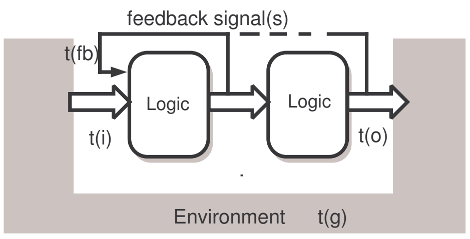

Fig. 6. The timing constraint of a feedback signal

blocks and the corresponding delay-matching blocks are designed, they are assembled together to form the sensor network processor pipeline like the one in Fig. 2. As can be seen from the mention above, the bundled-data asynchronous sensor network processor is designed using many commercial EDA tools provided for synchronous design. The asynchronous logic design work flow shown in Fig. 3 is somewhat like a synchronous logic design work flow, but with more careful timing analysis and restriction, especially for datapath with feedback signals.

Fig. 6 shows an asynchronous datapath with some feedback signals. The timing constraint should be defined very carefully. The assumption for the logic correctness of the circuit is that the maximal feedback delay $t(fb)$ has to be not larger than the sum of the minimal delays in the output port $t(0)$, the environment for generating a new data token $t(g)$ and the input port $t(i)$. The timing constraint for the circuit in Fig. 6 is:

$$0 < t(fb) \leq t(o) + t(g) + t(i) \tag{1}$$

IV. The dataflow processor architecture

Many commercial sensor nodes use commodity off-the-shelf microprocessors. For example, the Berkeley serious sensor nodes use Atmel Mega 128L [2]. Such microprocessors are general-purpose and can be programmed to meet the performance requirement of sensor network nodes. However, those microprocessors are not specially designed for sensor network applications and their general-purpose functionality makes them cost more power consumption than a specific sensor network processor that is designed based on the characters of sensor network applications. We believe that a good sensor network microprocessor should efficiently and quickly react to the events from sensors, timers and wireless transmission modules. The desirable properties for such a microprocessor are as follows:

1) Low standby power;
2) Low active power;
3) An efficient states-switching mechanism that can wake it up and let it sleep quickly with low overhead.

We use asynchronous logic to realize the first and the third properties. When sleeping, the proposed microprocessor is not necessary to maintain a global clock, thus the standby power is zero expect for leakage. When an event occurs, a request signal is immediately propagated to every required function logic blocks of the processor. The microprocessor responses the event in a few tens of nanoseconds, several orders of magnitude faster than a synchronous microprocessor having a quick wake up time of circa 6 microseconds [2]. When the desirable logic functions are finished, hand-shaking asynchronous control is done, sending the processor go back to sleep immediately. Near no performance sacrifice and power overhead are needed to realize such an efficient wake up and sleeping mechanism.

As for the second property, we design the microprocessor using an event-driven architecture as shown in Fig. 7. In sensor network applications, performance is not the first design concern and 10 MIPS is more than enough in most circumstances. Moreover, the programs in sensor nodes are usually short and parallelization is not necessary. For these considerations, we design the sensor network processor ignoring many properties of a general-purpose microprocessor, such as exceptions, interrupts and MMU, to save power and FPGA area.

When an event occurs, an event vector is sent into and propagated through the event queue (FIFO), and then to the fetching block, leading the processor to execute the corresponding code segments in IMEM. Each event has a code segment to define the responding operations. Only when finishing the response of one event does the processor cope with another event, thus concurrency will not happen. At the end of a code segment, a special ‘HALT’ instruction stops instruction fetching and sends the processor to sleep.

The instruction architecture of the sensor network processor is memory-memory. The word length of the processor is 16-bit long. The instruction lengths are not fixed as many RISC microprocessors; the instructions can be 16-bit or 32-bit long. One data processing instruction fetches two operands directly

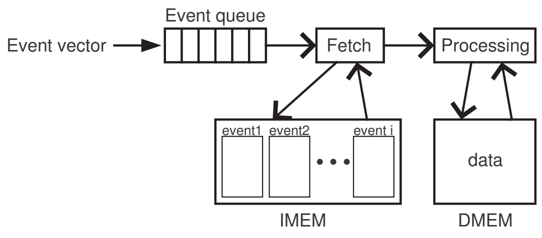

Fig. 7. The proposed sensor network processor architecture

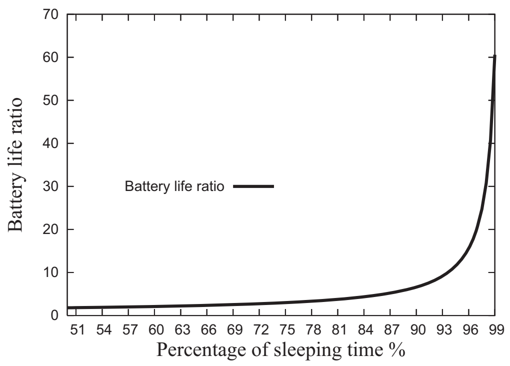

Fig. 8. The battery life time comparison under different sleeping percentages

from the DMEM and sends the result back to the DMEM. Therefore, no register file is existed in the processor. The power overhead caused by data transmission and duplication between a register file and memory is saved. The instruction set includes traditional data processing and branch instructions, such as arithmetic, logic and jump.

V. Experimental results

Comparing the FPGA processor with an ASIC microprocessor does not mean anything because they use different technologies. To evaluate the asynchronous FPGA processor, a comparable synchronous FPGA processor is designed, which uses the same architecture and supports the same instruction set. The two processors are mapped onto a Xilinx Spartan-3A FPGA [9]. Powered by a 1.8 volt supply voltage, their performance is about 65 MIPS. The power efficiency of the asynchronous processor when active is 320 MIPS/watt and the power efficiency of the synchornous processor is 180 MIPS/watt. The global clock in the synchronous processor uses 34% of the overall power. If the global clock is not switch off and a sensor node spend $n\%$ of the time sleeping, the battery life ratio between the asynchronous processor and the synchronous one is plotted in Fig. 8.

Based on the experimental data, an AA-size battery can hold the asynchronous processor for 60 days, but only hold the synchronous processor for one day if the sleeping percentage is 99%.

VI. Conclusion

In the paper, a low-power asynchronous event-driven sensor network processor is designed onto a commercial clocked FPGA. The processor employs a bundled-data asynchronous encoding scheme, which makes it possible to use the clocked FPGA design tools provided by Xilinx Company. A 2-phase asynchronous latch controller is proposed. The latch controller is faster than other 4-phase latch controllers because it eliminates return-to-zero operations. The properties of the processor are listed as follows:

- Zero standby power consumption

With an asynchronous control, the processor does not need to maintain a global clock when sleeping;

- Quick wakeup response

When an event occurs, a request signal is sent to the processor. The handshake signals are propagated to every required function blocks and wake them up. The wakeup time is several orders of magnitude shorter than a good synchronous microprocessor and the wakeup mechanism has little design effort and power overheads;

- Low active power consumption

This property comes from two sides: I) nature asynchronous clock-gating and II) particular architecture designed for sensor networks. From the first side, the asynchronous handshake control signals only switch the required function blocks, while the other blocks are not switched and consume no power. The power consumption of the processor depends on the jobs need to do and no extra power is wasted. From the second side, the processor uses an event-driven and memory-memory architecture. It saves the power caused by the execution overheads of a general-purpose microprocessor, including exceptions, interrupts and MMU handling and data duplication between memory and register.

Based on the experimental results, the FPGA sensor network processor runs at the speed of 65 MIPS, enough to satisfy most sensor network applications. If we assume a sensor node spends 99% of its time sleeping, the battery life of the proposed processor is 60 times longer than the one use a comparative synchronous processor.

Acknowledgment

The project was funded by a grant from Ministry of Personnel of the People's Republic of China (No. 2006-164), a grant from Guangdong Science and Technology Department (No. 2006B11801010) and a Natural Science Foundation of Guangdong (No. 07300421). The authors would like to acknowledge with gratitude these grants.

References

[1] M. Tubaishat and S. Madria., Sensor Networks: an Overview., IEEE Potentials, 22, 2, 20-23, April 2003
[2] Intel Mote Research Project. http://www.intel.com/research/exploratory/motes.htm
[3] V. Ekanayake, C. Kelly, et al., An Ultra Low-Power Processor for Sensor Networks., ASPLOS'04, October 2004
[4] Tiny OS website, www.tinyos.net
[5] S. B. Furber, J. D. Garside, S. Temple and J. Liu., AMULET2e: An Asynchronous Embedded Controller. Asyn'97, 1997
[6] Alain J. Martin, Mika Nystrom and Catherine G. Wong, Three Generations of Asynchronous Microprocessors. IEEE Design and Test of Computers 20(6): 9-17 (2003)
[7] S.B. Furber, D.A. Edwards and J.D. Garside, AMULET3: a 100 MIPS Asynchronous Embedded Processor, Proceeding on ICCD2000, 2000
[8] J.L. Hill, System Architecture for Wireless Sensor Networks, PhD thesis, University of California, Berkeley, 2003
[9] Xilinx Spartan 3A datasheet, http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan3a_fpgas/index.htm, 2007
[10] K. Maheswaran and V. Akella, Hazard-Free Implementation of the Self-Timed Cell Set in a Xilinx FPGA, Technical report, U.C.Davis, 1994
[11] S.W. Moore and P. Robinson, Rapid Prototyping of Self-timed Circuits, ICCD'98, 1998

This Page Was Intentionally Left Blank