A Low Power UART Design Based on Asynchronous Techniques

Dipanjan Bhadra, Vikas S. Vij, Kenneth S. Stevens
University of Utah

***Abstract*—Universal Asynchronous Receiver Transmitter (UART) implements serial communication between peripherals and remote embedded systems. The UART protocol is defined based on fixed frequencies with a sampling method to achieve robustness under reasonable frequency variations between systems. Such design specifications are natural for clocked domains. This work investigates whether this simple clocked hardware protocol can be advantageously implemented using asynchronous design techniques. A full duplex clocked and asynchronous UART are implemented and compared. The asynchronous design results in average power of about one-fourth that of the clocked design under standard operating modes.**

***Index Terms*—Universal Asynchronous Receiver Transmitter, UART, Asynchronous Circuits, Relative Timing**

I. Introduction

Universal Asynchronous Receiver Transmitter (UART) is a communication protocol which translates data from a parallel stream to a serial stream which is transmitted across the communication link. The UART communication protocol dictates that communication occurs at a predefined frequency. The two devices on the link generate their frequencies independently, and new events arrive asynchronously to the local clock. This requires that the data stream is sampled to ensure reliability even when the frequencies at the two ends of the link vary somewhat from the specified frequency. This is normally achieved through having a local internal clock with a frequency that is significantly higher than the channel sampling rate.

UART is an I/O communication protocol commonly used to communicate with slow peripherals. Wang et. al. [1] shows an FPGA implementation of a UART while Yu et. al. [2] implements a multichannel UART controller using asynchronous FIFOs (First In First Out) which allows communication with multiple blocks at different baud requirements. Idris et. al. [3] proposes the addition of BIST techniques to a UART design and Norhuzaimin et. al. [4] implements a UART for high speed operations. Many of these designs propose the addition of new hardware aiming at customizing the core for specific operations. However, all designs use an architecture similar to the synchronous UART designed by Litochevski [5], which was used for the clocked implementation described in this paper. It includes a full duplex asynchronous serial communication system used extensively to realize general serial bus protocols like RS232, RS422 and RS485. The synchronous design is derived from this clocked design by making control modifications to the clocked data path.

This work investigates whether the simple clocked UART hardware protocol can be advantageously implemented using asynchronous design techniques. Power reduction is the targeted advantage since performance is dictated by the protocol. Employing clocked design, the receiver must constantly sample the serial input line at the high frequency internal clock to determine the start of a new data transmission. The asynchronous design, on the other hand, does not need to sample the input line, rather it can reactively respond to the arrival of a new transmission. This is the basis for the expected power advantages of this study.

The contribution of this work are as follows. Firstly, an analysis on how to intrusively modify a synchronous peripheral into an asynchronous one is presented. Secondly, benefits of this approach at different activity and idle times is analyzed. Thirdly, a comparison with different ways of clock gating the clocked design are studied to measure the benefits in terms of power and energy.

II. Background

The clocked and asynchronous implementations are designed to be as comparable as possible. Therefore, an asynchronous bundled data design style is used. The data paths of the two designs are identical, and the same synthesis and physical design scripts are used for the two designs (minus clock optimization algorithms for the asynchronous design).

A. Relative Timing (RT)

Timing is the fundamental difference between clocked and asynchronous design flows. The effect of time on a system is to order and sequence events. In this work we have adopted the relative timing (RT) methodology to represent the sequencing that timing imposes on circuits [6]. A RT constraint consists of a common timing reference and a pair of events that are ordered in time for correct circuit operation. We call the common reference a point-of-divergence, or pod, and the ordered events the point-of-convergence, or poc. A constraint is represented as $\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1$ where poc_0 must occur in time before poc_1 with margin m. Hence the maximum path delay from pod to poc_0 must be less than the minimum path delay from pod to poc_1. This is represented by two related design constraint equations `set_max_delay` and `set_min_delay`, which perform timing driven synthesis that enforce the constraints on the logic paths.

B. Asynchronous CAD Tool Flow

The asynchronous CAD tool flow used is summarized in Fig. 1. Asynchronous controller design begins with a specification, which is synthesized either by hand or by using asynchronous synthesis tools like petrify, 3D, minimalist [7]–[9]. The result is a circuit definition which is then tech mapped to gates of the standard cell library. A reset signal is usually added to the element to ensure correct initialization. The element is then characterized. The timing graph of the circuit must be represented as a directed acyclic graph for the EDA tool flow to performing timing driven sizing, optimization and static timing analysis. Therefore, timing cuts are generated for the design as `set_disable_timing` constraints. The relative timing constraints are generated. RT constraints are represented as two separate delay paths mapped to `set_max_delay` and `set_min_delay` algorithms. The system is designed in Verilog using behavioral and/or structural representations of the characterized asynchronous design modules. The design is simulated for functional correctness. Relative timing constraints are mapped onto asynchronous module instances in the design which allow the commercial EDA tools to perform timing-driven synthesis and physical design and post-layout timing validation.

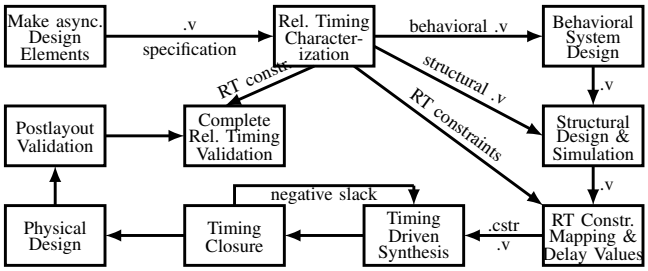

Fig. 1. Simplified Relative Timing Multi-Synch. Design Flow

C. Synchronous Implementation

The UART design consists of three main blocks as shown in Fig. 2 – transmitter, receiver, and baud generator, plus two status registers *tx_busy* and *rx_busy*. The baud generator deals with the generation of the baud frequency clock pulses from the input clock. The transmitter and the receiver block, which perform the data transfer, are independent of each other and hence this design can achieve full duplex communication.

1) Baud Generator: The baud generator creates a 16× baud clock based on the baud-rate (BR) specified for the communication which is provided to both transmitter and receiver blocks to generate their own baud clock pulse. The faster clock allows the receiver to align the sampling pulse as desired and provides a faster response time from the transmitter. The baud clock frequency and baud-limit are calculated as per Eqn. 1 and 2.

$$Baud\_Freq = \frac{16 \cdot BR}{gcd(Clock\_Freq, 16 \cdot BR)} \tag{1}$$

$$Baud\_Limit = \frac{Clock\_Freq}{gcd(Clock\_Freq, 16 \cdot BR)} - Baud\_Freq \tag{2}$$

2) Transmitter Block: A new data transfer in the UART is initiated by the *new_tx_data*, which indicates the availability of the *tx_data*. This data is stored in the *tx_hold_reg* in the next clock cycle. The parallel stream of data (*tx_data*) is converted into a serial stream using a shift register. The start of the shift operation sets the internal status register *tx_busy* to indicate a new data transfer. *tx_busy* can also be used as data validity signal for clock gating the transmitter block. The shift operation follows at the baud clock frequency to send the data out on the *ser_out* data line. A counter of 16 is used to generate the rising edge for the baud clock from the 16× the baud clock provided by the baud generator. The status of any data transfer is maintained by a status register to indicate a valid/invalid operation indicated by an interrupt signal. The resetting of the *tx_busy* status register is detected by another counter of 10 which counts the number of bits transmitted.

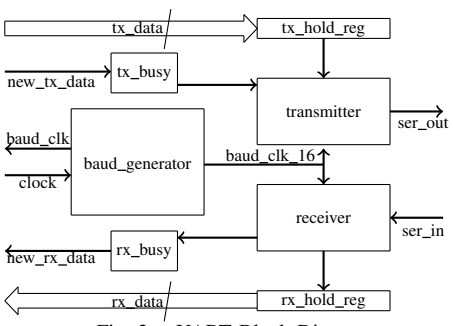

Fig. 2. UART Block Diagram

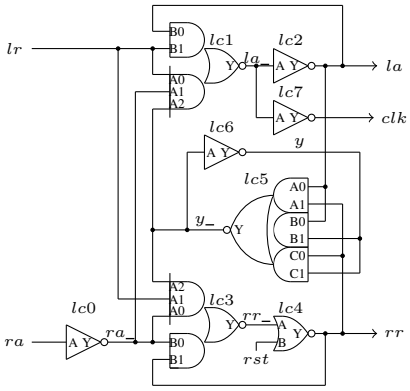

Fig. 3. LC circuit implementation

3) Receiver Block: The receiver block is the dual of the transmitter block. The UART communication format consists of a start bit followed by 8 data bits and one stop bit indicating the end of the communication. This block monitors the input line for new data which is indicated by the start bit. The *rx_busy* status register is set when the start bit is recognized. This block operates at two different edges of the baud clock pulse generated by a 16 bit shift counter from the 16× baud clock. The sampling and shifting at the shift register occurs at the middle of the incoming data pulse at the falling edge of the baud clock pulse which is generated at the count of 8 from the counter. After receiving the end bit of the *ser_in* data, the serial data is moved to the *rx_hold_reg* register at the rising edge of the baud clock frequency generated at the count of 16 on the counter. Thus a parallel stream of 8-bit data (*rx_data*) is generated which is indicated by the *new_rx_data* interrupt signal. There is also a 3-bit counter to track the number of input bits received.

III. Asynchronous UART Design

The RT based asynchronous design approach adds asynchronous handshaking controls to enable and steer the local UART clock. Thus the datapath is nearly identical to that of the clocked design. This results in the asynchronous design being 4-8% larger than the clocked design. The power benefit of the asynchronous UART implementation derives from the presence of substantial idle times on the communication channel, and the reactive nature of the design to data transfer conditions rather than polling as occurs in clocked designs.

Operation of the asynchronous implementation is dictated by linear handshake controllers (LC), as shown in Fig. 3. The purpose of the controllers is firstly to generate the local timing reference to determine the frequency of operation and secondly to produce the high power clock signal that drives the latch and/or flip-flops in the data path thereby freeing the design of its need for a continuous clock. Two separate controller protocols are used to control and time the data flow in the Transmitter and Receiver blocks. Their operations are described in Fig. 7.

A. Baud Generator

The requirement of a specified baud frequency to transmit and receive data is the sole reason why the asynchronous UART consists of a baud generator and cannot be fully clockless. The design of this block is identical to that of the synchronous implementation. The baud clock needs to be active only during the data transmission, hence it can be paused during idle time. The pausing of this block is

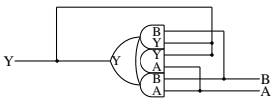

Fig. 4. A C-Element block

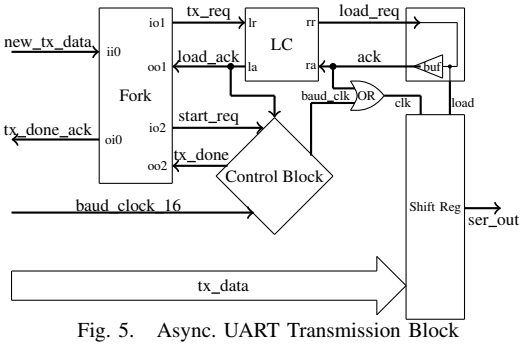

Fig. 5. Async. UART Transmission Block

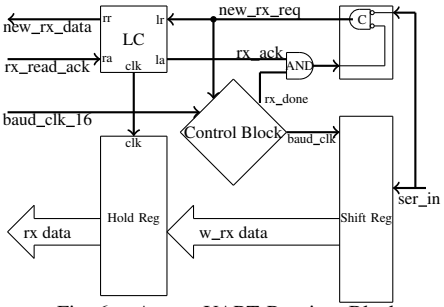

Fig. 6. Async. UART Receiver Block

Transmitter LC block function ::
$lr\uparrow \rightarrow rr\uparrow$ & $la\uparrow \rightarrow ra\uparrow \rightarrow rr\downarrow \rightarrow lr\downarrow \rightarrow la\downarrow$

Receiver LC block function ::
$lr\uparrow \rightarrow la\uparrow \rightarrow lr\downarrow \rightarrow la\downarrow$, $clk\uparrow$ & $rr\uparrow \rightarrow ra\uparrow \rightarrow rr\downarrow \rightarrow ra\downarrow$

Fig. 7. LC circuit operation for Transmitter and Receiver block

achieved by gating the incoming global reference clock signal at input based on the asynchronous handshake network. If both *new_tx_data* and *new_rx_data* are unasserted the clock to the baud generator is gated through a synchronizer.

B. Transmitter block

The transmitter block consists of the same shift register as the clocked design, but the clock control network is replaced by an asynchronous handshake network. An asynchronous handshake network consists of a generic fork element, a LC circuit, and a control block as shown in Fig. 5. The start of any new transmission is indicated by the *new_tx_data* signal. The fork broadcasts this request signal from the sender to the two outgoing channels (*tx_req* and *start_req*). The acknowledge associated with these two requests (*load_ack* and *tx_done*) is synchronized with a C-element before passing it as *tx_done_ack* [10]. The C-element output only changes when both the inputs are identical (Fig. 4). The upper channel goes into a LC block (Fig. 3) which generates a clock signal to load the parallel *tx_data* in the Shift Register, this is controlled by the *load_req* signal. The *load_req* signal also determines the mode of operation of the shift register. The *load_ack* signal indicates the latching of the data. The separation between loading the parallel data and shifting on the *ser_out* channel is achieved making *load_ack* the enabling signal along with *start_req* for the control block. This ensures the generation and supply of baud clock to the shift register is preceded by the loading operation. The OR gate allows the Shift Register to be clocked by either the baud clock or the asynchronous handshake request but it shifts data only when the load signal is set low.

When *load_ack* is asserted, a new data bit will be shifted out every 16 *baud_clock_16* ticks. The need to maintain a count for the number of bits transmitted is met by a 4-bit counter clocked by the baud clock. The number of bits to be transmitted include 8 data bits, and one start and one stop bit. The *tx_done* signal is generated at the count of 10, thus indicating the end of the data transmission. Availability of *tx_done* results in *tx_done_ack* being generated and hence completing active part of the handshake. The reset phase of the handshake follows which resets the counters in the control block.

C. Receiver block

Any new data reception in a UART is indicated by a start bit. At reset the *rx_done* signal is low. When the *ser_in* goes low, indicating the start of a new communication, the C-element output enables the control block. The control block generates the baud clock with a clocked counter to store incoming data bits at the shift register. There is another counter to count the number of incoming bits. When all the 10-bits (the 8-bit data and start and stop bits) are received the *rx_done* signal is enabled indicating completion data reception. This resets the C-element output thus completing the four-cycle handshake. The clock signal for the hold register is generated at the end of the handshake cycle, converting the serial stream into parallel 8-bit data. A new data reception is then indicated to the core by a request on the *new_rx_data*.

IV. Results

The synchronous and asynchronous UART circuits are implemented using the Artisan academic library in IBM's 65nm 10SF process. Each circuit is written in Verilog, synthesized using Design Compiler (DC) and place and routed using SoC Encounter. The functional validation of these designs is performed using Modelsim simulator with their parasitics Standard Delay Format (SDF) file back-annotated. Both these designs are made to transmit and receive the same set of data. A Value Change Dump (VCD) file is generated for each simulation using the parasitics SDF from place and route. The VCD file along with the Standard Parasitic Exchange Format (SPEF) are used to generate the power numbers for the designs using Primetime PX.

Three different designs for the synchronous UART are implemented and compared against the asynchronous UART for power and area benefits. The first design (*Clocked – Not gated*) is a synchronous version without any clock gating. Since the UART is a very slow peripheral with large idle times, a better comparison to asynchronous designs would be against a clock gating design which is implemented in the second and the third design. Clock gating was introduced in the second design (*Clocked – Auto gated*) using Design Compiler, while for the third design (*Clocked – Manually gated*) clock gating was introduced manually. The third design used a behavioral definition for clock gating and prevented DC from modifying it. Since the receiver block always polls for the incoming data, only the transmitter block was clock gated. All registers in the

TABLE I
Area and Power Benefits for Asynchronous Design.

	Area	Power		
	Core Area	Active(100%)	Idle(100%)	Idle(90%)
Asynchronous Design	1.00×	1.00×	1.00×	1.00×
Clocked – Not gated	0.96×	1.72×	5.00×	4.67×
Clocked – Auto gated	0.92×	1.01×	4.32×	3.99×
Clocked – Manually gated	0.96×	1.66×	4.14×	3.89×

TABLE II
COMPARISON OF AREA AND POWER NUMBERS FOR DIFFERENT DESIGNS.

	Area	Power (μW)								Simulation-Time
	Core Area	Active (0% Idle Time)				Idle (100% Idle Time)				
	μm sq	switching	internal	leakage	total	switching	internal	leakage	total	μs
Asynchronous Design	2268.14	4.43	15.38	6.68	26.49	0.00	00.71	6.62	07.33	4000
Clocked – Not gated	2186.94	2.90	37.29	5.42	45.61	2.79	28.18	5.27	36.25	4000
Clocked – Auto gated	2088.94	3.00	18.81	5.04	26.85	3.81	22.88	4.97	31.66	4000
Clocked – Manually gated	2186.75	3.81	34.95	5.32	44.07	2.76	22.68	5.20	30.63	4000

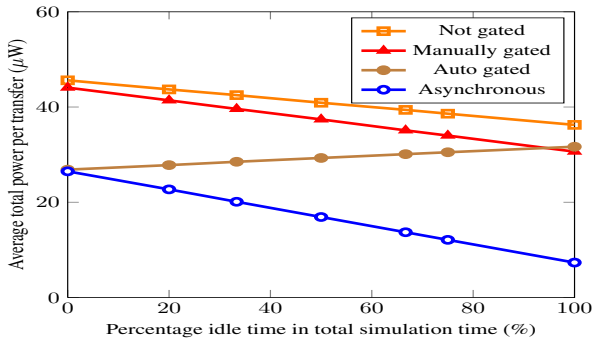

Fig. 8. Average Power with respect to percentage idle time

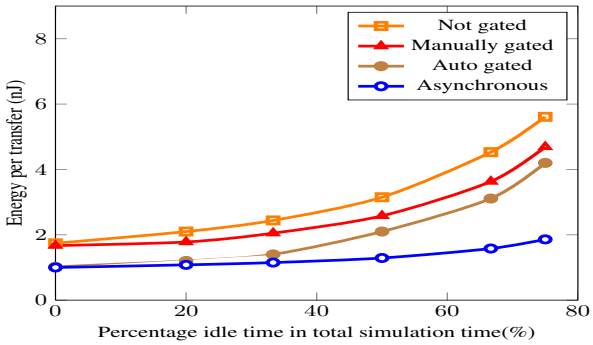

Fig. 9. Energy per transmission/reception

asynchronous design are clocked by handshake controls only when valid data is present.

Tab. I and II report the power and area numbers for the designs. Since UART can have a lot of Idle time, power consumption comparisons have two modes of operation – idle and active. In the idle mode the test bench enforces no transfer of data either from the transmission block or from the receive block. For the active mode, both the transmitter and the receiver is forced to transfer at the highest frequency permitted by the protocol. Comparison of the total active power shows a 72% and 68% benefit as compared to the designs with no clock gating and manual clock gating respectively, while minor benefit are seen with automatic clock gated design. Major benefits are seen for the 90% idle time power comparison with the synchronous UART designs consuming about 4$\times$ the power of the asynchronous design. No switching power occurs in idle mode for the asynchronous circuit since the clock is enabled to the baud generator only in the active mode. There is the cost of a small increase in area due to the asynchronous handshake network.

The designs were also analysed for variations in the activity factor by varying the idle time for the devices. Fig. 8 and 9 show a comparison of performance of the designs in terms of power and energy consumption per transfer for the same data transmitted but over varying idle time percentage. The curve is generated by fitting it with respect to the points marked on the graph. Fig. 8 along with Tab. II provide a good comparison for all the designs. The numbers for clock gating done automatically and manually shows a difference between fine-grained and coarse grained clock gating respectively. Fine grained clock gating results in big switching power savings during active mode but the extra logic for this gains comes at the cost of increased switching activity in the idle mode. This overhead shows a gradual increase in the average total power with increase in idle time in the graph, whereas the other designs show a decrease in the average total power. The comparison for energy per transfer also shows the impact of the clock and the clock gating overhead for the synchronous designs. The increase in the energy per transfer of the synchronous UARTs as idle time increases is substantial. For the asynchronous design this is mostly only attributed to leakage.

V. CONCLUSION

This work presents a comparison of four designs for UART consisting of three synchronous ones with and without clock gating and an asynchronous implementation. The asynchronous design uses the same tools and flows as the synchronous design. The asynchronous design shows a substantial benefit in terms of total power. The drawbacks of clock and clock gating overhead are reflected by these comparisons with about 4$\times$ higher power consumption by synchronous designs as compared to the asynchronous version. The asynchronous techniques allows a reactive generation of clock pulses at the time of transfer as opposed to having a continuous clock for the synchronous design. This results in an overall improvement in terms of power and energy for slow peripherals like UART, thus making them ideal for such peripheral which remain idle most of the time.

REFERENCES

[1] Y. Wang and K. Song, "A new approach to realize uart," in *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, vol. 5, aug. 2011, pp. 2749 –2752.

[2] S. Yu, L. Yi, W. Chen, and Z. Wen, "Implementation of a multi-channel uart controller based on fifo technique and fpga," in *Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on*, may 2007, pp. 2633 –2638.

[3] M. Idris and M. Yaacob, "A vhdl implementation of bist technique in uart design," in *TENCON 2003. Conference on Convergent Technologies for the Asia-Pacific Region*, vol. 4, Oct., pp. 1450–1454 Vol.4.

[4] J. Norhuzaimin and H. H. Maimun, "The design of high speed uart," in *Applied Electromagnetics, 2005. APACE 2005. Asia-Pacific Conference on*, Dec., pp. 5 pp.–.

[5] M. Litochevski, "Uart to bus core specification," Opencores, Tech. Rep., 2012.

[6] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative Timing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 11, pp. 129–140, Feb. 2003.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.

[8] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementation)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 101–117, Feb 1999.

[9] R. M. Fuhrer and S. M. Nowick, *Sequential Optimization of Asynchronous and Synchronous Fininte State Machines: Algorithms and Tools*. Kluwer Academic, 2001.

[10] R. Miller, *Switching theory*. Wiley, 1965, vol. 1.