CS 223 Term Project Assignment

Spring 2024

UART Hardware Communication Protocol Design and Implementation

Report Due: May 6, 2024 13:00

Introduction

You are going to implement Universal Asynchronous Receiver-Transmitter (UART) module in SystemVerilog. UART is a simple hardware for asynchronous serial communication, but it will allow you to demonstrate your knowledge in sequential logic, state machines, and read/write control logic. Your implementation will work in full duplex mode and be able to both transmit and receive data simultaneously between two Basys3 boards. You will also display the data you receive and transmit by controlling the on-board peripherals.

Stage 1: UART Device [40]

A simplified UART device is illustrated in Fig. 1. There are registers to store both the incoming and outgoing data. The communication happens at a pre-determined frequency called the baud rate. Baud rate expresses how many bits per second are sent over when the communication is initiated. Both sides must also agree on the amount of parity bits, stop bits, and the length of the data frame for UART to work properly. Additionally, the voltages must be the same, but this works out if you use the default voltage levels on your board. It is recommended to do some self-study on the details of the UART device before moving on to the implementation.

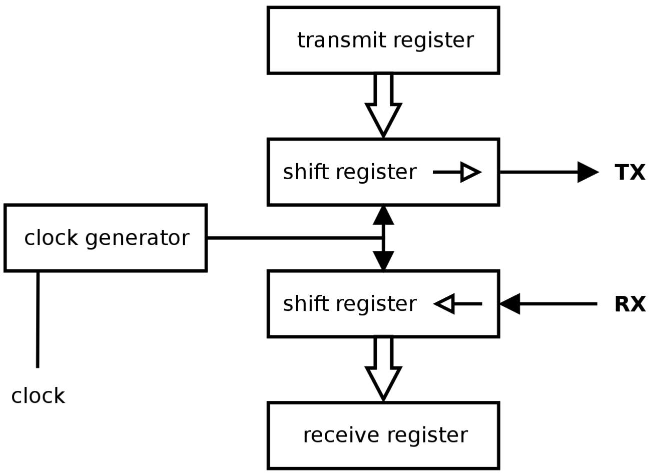

Figure 1: UART block diagram [2].

A UART frame typically consists of the following components:

- Idle - logic high
- Start bit - logic low
- Data bits - 5 to 9 bits
- Parity bit - 0 to 1 bit
- Stop bits - 1 to 2 bits

A UART operation can be briefly described as follows: The device maintains a constant high output when idle. The start of the transmission is signaled by pulling down the TX line. Upon detecting the pulled-down line, the receiver prepares for data reception on the next clock cycle on its RX line. The data is sent bit by bit starting from the least significant bit to most significant bit. The optional parity bit can be set to be even, odd, or none, serving as error-checking information. Stop bits mark the end of the data bits and is always logic high.

UART Implementation

In your UART design, adhere to the following specifications:

- 8 data bits per transmission (each transmission is a byte).
- 1 parity bit.
- For parity bit selection, use the following logic:
  - If the least significant digit of your Bilkent ID is odd, use odd parity.
  - Otherwise, use even parity.
  - Example: If your Bilkent ID is `21706543`, your least significant digit is `3`, therefore, use odd parity.
- 1 stop bit.

Note: Do not hardcode this configuration. Ensure customizability by using `parameter`s.

For the first stage of the project, your implementation should have a functioning transmitter and receiver module configured accordingly. Each module should have a 8-bit wide register as a buffer, storing the value to be sent or just received. Let us call them `TXBUF` and `RXBUF` for now. On the hardware, utilize the rightmost 8 switches (SW7-0) with pins `W13, W14, V15, W15, W17, W16, V16, V17` to be the input to the transmitter buffer where `W13` is the most significant bit and `V17` is the least significant bit. For example, inserting `hA5 = b1010_0101` would require the switches to be `W13=1, W14=0, V15=1, W15=0, W17=0, W16=1, V16=0 V17=1`. **DO NOT confuse the order of switches!** Most significant bit is assigned to the leftmost switch and the least significant bit is assigned to the rightmost switch.

Use down button, `BTND, pin U17`, to load the current byte from the switches into your transmitter registers according to the previously defined logic. For debugging purposes, wire the latest inserted value to the rightmost 8 LEDs, keeping the same order as switches. This enables verification of the value that is just inserted after `BTND` is pressed. Ensure that the `TXBUF` register is wired for this operation, and not the current configuration of switches, i.e, LEDs should not change state while switches are being changed to a new configuration.

Use center button, `BTNC, pin U18`, to initiate the transmission of the `TXBUF` register to the receiver. For debugging purposes, connect the `RXBUF` register to the leftmost 8 LEDs to verify the value that is just received. The data you send should directly appear on the `RXBUF` of the receiver side.

Implement your UART circuit based on the provided specifications. For testing purposes, set the baud rate to `115200`, but make it also configurable with other values. **Your design must be able to both receive and transmit data.** Before advancing in the project, prepare a testbench and verify your transmitter and receiver functionality. If you have difficulties getting your implementation to work with `115200` bauds on the FPGA, consider using `9600` bauds. Keep in mind that simulating with a lower baud rate is likely to result in lengthier simulations. However, this should not matter for your implementation on FPGA since everything runs on real time.

At the end of stage 1, your board design can transmit and receive 1-byte of data. Important points to note:

- You can load `TXBUF` according to the rightmost 8 switches by pressing `BTND` button.
- You display the 8-bit wide `TXBUF` register on the rightmost 8 LEDs.
- You signal the start of the tranmission by the press of `BTNC` button.
- Your receiver automatically detects the incoming transmission and stores it in the `RXBUF` register.
- You display the 8-bit wide `RXBUF` register on the leftmost 8 LEDs.

Tip: During project demo, there will be two different Basys3 boards. Each board will transmit data to and receive data from the other board. However, for development and debugging purposes, you can establish a loopback connection by connecting the TX (transmit) line to the RX (receive) line on the same board. This arrangement allows you to observe the transmitted data locally for verification. That being said, ensure that your design works between two different boards.

With that out of the way, how are you actually going to connect your TX and RX lines? You will use jumper wires and Pmod headers of Basys3. Refer to the Basys3 reference manual [1] for further details. Fig. 2 is the pinout for Pmod headers. There are 4 Pmod headers on the board, you can use the one you prefer. Just make sure to properly connect your signals with the physical port in the `.xdc` design constraints file.

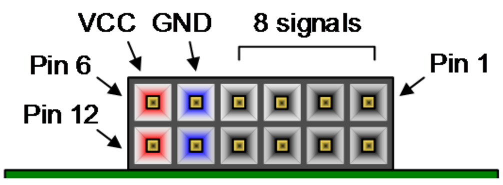

Figure 2: Basys3 pmod port pinout [1].

Important Note: Notice the `VCC` and `GND` pins in Fig. 2. Connect `GND` ports of two boards with a single wire to ensure signal integrity.

Stage 2: Register Files for Transmit and Receive Operation [15]

Now that you have a functioning UART transmitter and receiver, you will make a small modification to how you internally store the data. Allocate two 4-byte memory for storing transmitted and received data. Design the receiver register files to operate as a FIFO structure, discarding the earliest received data when the 5th byte arrives and there is no additional space. This FIFO structure is illustrated in Table 1 for a 4-byte register file as an example. Note that `hxx` notation represents the hexadecimal values.

Table 1: Example operation for the 4-byte memory.

Event	Register 0	Register 1	Register 2	Register 3	Discard
Initial Condition	`h00`	`h00`	`h00`	`h00`	-
`h01` received	`h01`	`h00`	`h00`	`h00`	`h00`
`h02` received	`h02`	`h01`	`h00`	`h00`	`h00`
`h03` received	`h03`	`h02`	`h01`	`h00`	`h00`
`h04` received	`h04`	`h03`	`h02`	`h01`	`h00`
`h05` received	`h05`	`h04`	`h03`	`h02`	`h01`

Basically, to expand the capacity of the previous `TXBUF` and `RXBUF` registers, increase their depth to accommodate more bytes. When `BTND` is pressed to load data to `TXBUF`, the rightmost 8 LEDs should display the most recent input data (Register 0 in Table 1). Initiating transmission is still signaled by pressing `BTNC`. However, the data that will be discarded in the next cycle of loading (Register 3 in Table 1) will be sent through your UART transmitter. Detection of a transmission should result in the leftmost 8 LEDs displaying the oldest received data (Register 3 in Table 1) as opposed to most recent in the previous version.

At the end of this stage:

- You have 4-byte arrays for `TXBUF` and `RXBUF` operating as FIFO.
- You load a byte from switches into `TXBUF` the same way as before.
- The order you load the data from is up to you to decide. Nevertheless, be consistent with which one you display on LEDs and which one you send over TX line:
  - If you input from right and shift left, you load data to `TXBUF[3]` and discard `TXBUF[0]`. You display `TXBUF[3]` and `RXBUF[0]` on LEDs.
  - If you input from left and shift right, you load data to `TXBUF[0]` and discard `TXBUF[3]`. You display `TXBUF[0]` and `RXBUF[3]` on LEDs.

    - In other words, you display the most recent byte on `TXBUF`, but send and display the oldest byte on `TXBUF` and `RXBUF` respectively.
- You send the oldest byte you entered into `TXBUF` by pressing `BTNC`.
- You display the most recent byte you loaded on the **rightmost** 8 LEDs.
- You display the oldest byte you received on the **leftmost** 8 LEDs.

Stage 3: Display Contents of Register Files on 7-Segment Display [25]

You utilized your UART device for data transmission and reception, and now you are able to store some previous transactions. However, the stored data is currently displayed using a temporary method implemented in stage 2 to quickly verify the functionality of your memory array. It is preferable to visualize the contents of both the transmit and receive registers on the onboard 4-digit 7-segment display of the Basys3 board. Each byte consists of 8 bits, or two hexadecimal digits. You can uniquely represent all hex digits on a single 7-segment digit: `F, E, d, c, b, A, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0`. Additionally, you can uniquely display all characters in the English alphabet. Convince yourself that this is the case. In your implementation, there are 8 hex digits, or 4 bytes to represent for one memory array. Each byte has a position ranging from 0 to 3, and a named array block: `TXBUF` or `RXBUF`. You can concurrently display 4 digits at most, meaning you would need to "scroll" across "pages" to see all values stored in registers.

Implement a display function to access bytes from your 4-byte registers and display them on the 7-segment display with four pages. Each page should fully utilize the full 4 digits of the display. Organize the information shown on your display as follows:

- First digit: A letter "t" or "r" depending on which register group is active for `TXBUF` and `RXBUF` respectively.
- Second digit: Position index ranging from 0 to 3 depending on which byte is being displayed.
- Third and fourth digit: The actual number stored in the current group and current index in hexadecimal format.
- Example: `t186` denotes that `TXBUF` has hexadecimal number `h86` stored in Register `1`.

For this purpose, set left button, `BTNL, pin W19`, to shift the display index towards left by 1. Set right button, `BTNR, pin T17`, to shift the display index towards right by 1. For example, assume that hex numbers `[h45], [h67], [h89], [hAB]` are loaded into your `TXBUF` memory. Initially, you display the first position of `TXBUF` on the 7-segment display:

- `t` `0` `4` `5` is on display.
- Press `BTNR`.
- `t` `1` `6` `7` is on display.
- Press `BTNR`.
- `t` `2` `8` `9` is on display.
- Press `BTNR`.
- `t` `3` `A` `B` is on display.
- Press `BTNR`.
- `t` `0` `4` `5` is on display (loop back).
- Press `BTNL`.
- `t` `3` `A` `B` is on display (loop back).
- Press `BTNL`.
- `t` `2` `8` `9` is on display.

By default, display the contents of the transmitter memory, `TXBUF`. Use up button, `BTNU, pin T18`, to switch between which memory is being displayed. The same operation procedure applies except this time you denote that you are displaying `RXBUF` by displaying `r` on the first digit. Each button press should consistently flip the display once. Erroneous, multiple flips are not acceptable. Functionality of the rightmost 8 LEDs and the leftmost 8 LEDs should remain unchanged.

Note: The exact forms of letters "t" and "r" are not important. If you want to follow the convention, see [3]. You can light up only the top segment for "t" and the bottom segment for "r". As long as they are unique and easily differentiable, you are set. That being said, feel free to customize to improve readability as you see fit and ensure you display the index and the byte with proper hexadecimal numbers.

At the end of this stage:

- You load, send, and receive data the same way as before.
- You can display the contents of the memory arrays on the 7-segment display.
- You can switch pages (each has a byte) using `BTNL` and `BTNR`.
- You show the current page on the second digit with one of `0` `1` `2` `3`.
- You can switch which memory arrays using `BTNU`.
- You show the current memory array on the first digit with one of `t` `r`.
- You show the hex number stored on third and fourth digits.

Stage 4: Automatic Transfer [10]

So far, you implemented an operational UART along with memory and means to display their contents. However, you are still manually sending only 1-byte data with each `BTNC` press, which is inconvenient and limited by the speed at you operate. In this stage, you will implement another state, from which you can send the whole transmitter array `TXBUF` automatically. If you properly implemented the FIFO architecture for your memory arrays, you should be able to observe the contents of your transmitter array `TXBUF` appear at the receiver array `RXBUF` on the other end with the press of a button, without making any modifications to your receiver design.

Use leftmost switch, `SW15, pin R2`, to implement this mode. You must preserve the same functionality at the end of stage 3 when the switch is low (`SW15=0`). When the switch is high (`SW15=1`), pressing `BTNC` sends the contents of your 4-deep, 8-bit wide `TXBUF` array byte by byte over UART to the receiver device, which is then saved to the receiver's `RXBUF` array. The 7-segment display module functions exactly the same as before.

- `SW15=0` : Keep stage 3 level functionality, transmit 1 byte with each `BTNC` press.
- `SW15=1` : Transmit all 4 bytes of `TXBUF` array with a single press of `BTNC`.
- Your receiver functions the same as before, capturing and saving all 4 bytes automatically into `RXBUF`, that is if you implemented stage 2 correctly.
- You navigate the 7-segment display exactly the same way as before.

Remarks

At the end, you successfully implemented and verified a UART communication protocol, seamlessly incorporating fundamental digital design concepts into your work. Congratulations! Here are a few remarks that you should consider:

- You can directly implement the whole project in one go, skipping the intermediate steps. However, stages allow you to incrementally implement the project, ensuring you can easily fix issues as you encounter them. You can pinpoint and fix the problem faster if you have less components to worry about.
- You may receive partial points if your design has partial functionality; for instance, if it functions correctly only up to stage 3. In that case, you might want to include proper simulations, and testbenches for each stage in your report. This is another reason you might want to follow the project stages.

- Please remember that you will be demonstrating your designs with another student in the same lab session. You can arrange this with a friend, or TAs will randomly choose a partner for you. It is preferable if you test everything beforehand. Lab hours should be reserved solely for the demo, so it is advised to complete your work before coming to the lab.
- You can and should try to receive the data you transmit while working on your implementation. However, keep in mind that the demo will involve two different boards, each receiving and transmitting data.
- **Do not forget to connect the grounds of the two boards while testing UART!**
- **While you may choose not to, it is strongly advised that you treat Vivado warnings as errors. More often than not, they indicate potential issues. Vivado only raises errors when it encounters a fatal error, i.e., when your design is not synthesizable or when it cannot generate a bitstream. Possible logical errors might still be implementable; thus, Vivado does not consider them as errors, only as warnings.**
- **[10]** Be ready for any questions your TA may ask during and after your demo.

Project Report

In the project report, you need to submit the following:

- Cover page with your name, surname, ID, and section number
- Detailed explanation of your design and implementation along with UART functionality
- RTL schematics for UART TX, RX, memory arrays, 7-segment display, continuous transfer
- State diagram for UART TX, RX, 7-segment display, continuous transfer
- Block diagram of each module you implement
- Testbenches (optional, mainly for partial points if everything does not work properly)
- References (if you have any)
- **Appendix that contains all of your code as text, *NOT* image. This will be checked and points will be deducted if you submit *ANY* portion of your code in image format.**

References

[1] Sam Bobrowicz. Basys 3 Reference - Digilent Reference — digilent.com. `https://digilent.com/reference/basys3/refmanual`. [Accessed 13-11-2023].

[2] Wikipedia contributors. Universal asynchronous receiver-transmitter — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter`, 2023. [Online; accessed 13-November-2023].

[3] Dave Madison. Segmented led display - ascii library — GitHub. `https://github.com/dmadison/LED-Segment-ASCII?tab=readme-ov-file#7-segment`, 2017. [Online; accessed 27-March-2023].

Last updated: Barış Bilgin Şenol, March 2024.

Prepared by: Barış Bilgin Şenol, November 2023.