SPART: A Special Purpose Asynchronous Receiver/Transmitter

Introduction

In this miniproject you are tasked with implementing a Special Purpose Asynchronous Receiver/Transmitter (SPART). The SPART can be integrated into the processor of your final project to serve as the serial I/O interface between the processor and serial I/O port of the lab workstations. Using the Hypertermial Accessory program (such as Putty), this will permit you to input characters from the keyboard and to output characters to the screen on the lab workstations.

The objectives of this miniproject are to:

- Familiarize you with design in the ECE 554 board environment
- Practice the use of an HDL in design
- Generate a useful module for your final project
- Gain experience implementing a design as a team

SPART Design

SPART Functional Description

This section describes the specifications of the subsystem to be designed. First, some key terminology must be clarified. The terms output or write are used when the processor is sending information to the SPART interface. The term transmit is used when the SPART is transmitting data to the serial I/O port. Conversely, the terms input or read are used when the processor is retrieving information from the SPART interface. The term receive is used when the SPART is receiving data from the serial I/O port.

IOADDR	SPART Register
00	Transmit Buffer (IOR/W = 0); Receive Buffer (IOR/W = 1)
01	Status Register (IOR/W = 1)
10	DB(Low) Division Buffer
11	DB(High) Division Buffer

Table 1: Address Mappings

A top level diagram of the SPART and its environment is shown in Figure 1. The FPGA interfaces with a chip on the board which generates appropriate voltage levels for the I/O port. The TxD pin transmits serial data from the FPGA and RxD receives serial data.

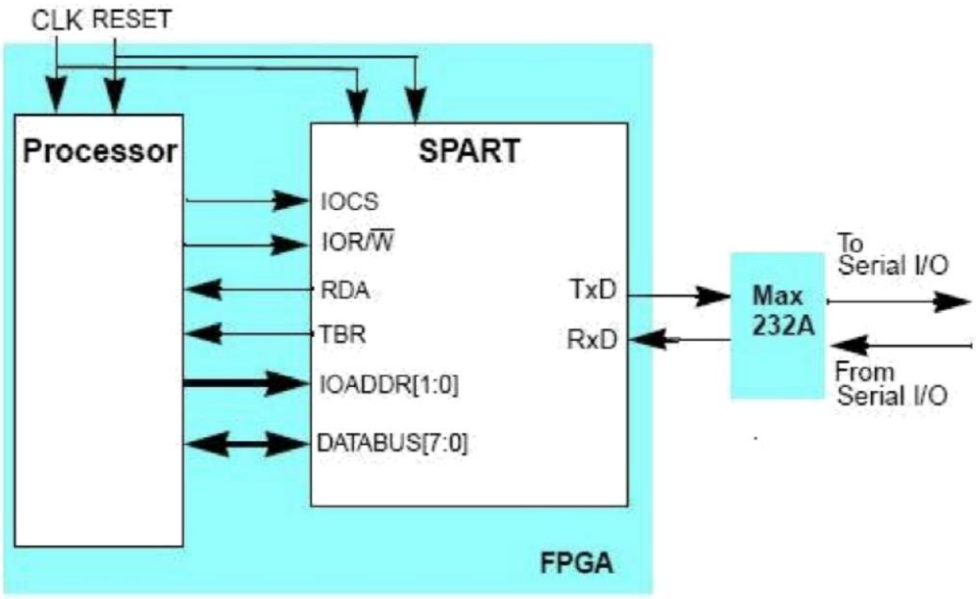


Figure 1: SPART Environment

The SPART and Processor driver share many interconnections that are used for controlling the reception and transmission of data. These signals are summarized in the following table:

DATABUS[7:0]	An 8-bit, 3-state bidirectional bus used to transfer data and control information between the Processor and the SPART.
IOADDR[1:0]	A 2-bit address bus used to select the particular register that interacts with the DATABUS during an I/O operation.
IOR/W	Determines the direction of data transfer between the Processor and SPART. For a Read (IOR/W=1), data is transferred from the SPART to the Processor and for a Write (IOR/W=0), data is transferred from the processor to the SPART.
IOCS	I/O Chip Select
RDA	Receive Data Available - Indicates that a byte of data has been received and is ready to be read from the SPART to the Processor.
TBR	Transmit Buffer Ready - Indicates that the transmit buffer in the SPART is ready to accept a byte for transmission.

The SPART is fully synchronous with the clock signal CLK; this implies that transfers between the Processor and SPART can be controlled by applying IOCS, IOR/W, IOADDR, and DATABUS (in the case of a write operation) for a single clock cycle and capturing the transferred data on the next positive clock edge. The received data on RxD, however, is asynchronous with respect to CLK. Also, the serial I/O port on the workstation which receives the transmitted data from TxD has no access to CLK. This interface thus constitutes the "A" for "Asynchronous" in SPART and may require an understanding of RS-232 signal timing and (re)synchronization.

SPART Structure

A block diagram of the SPART is given in Figure 2. More details about these subsystems can be found within this section.

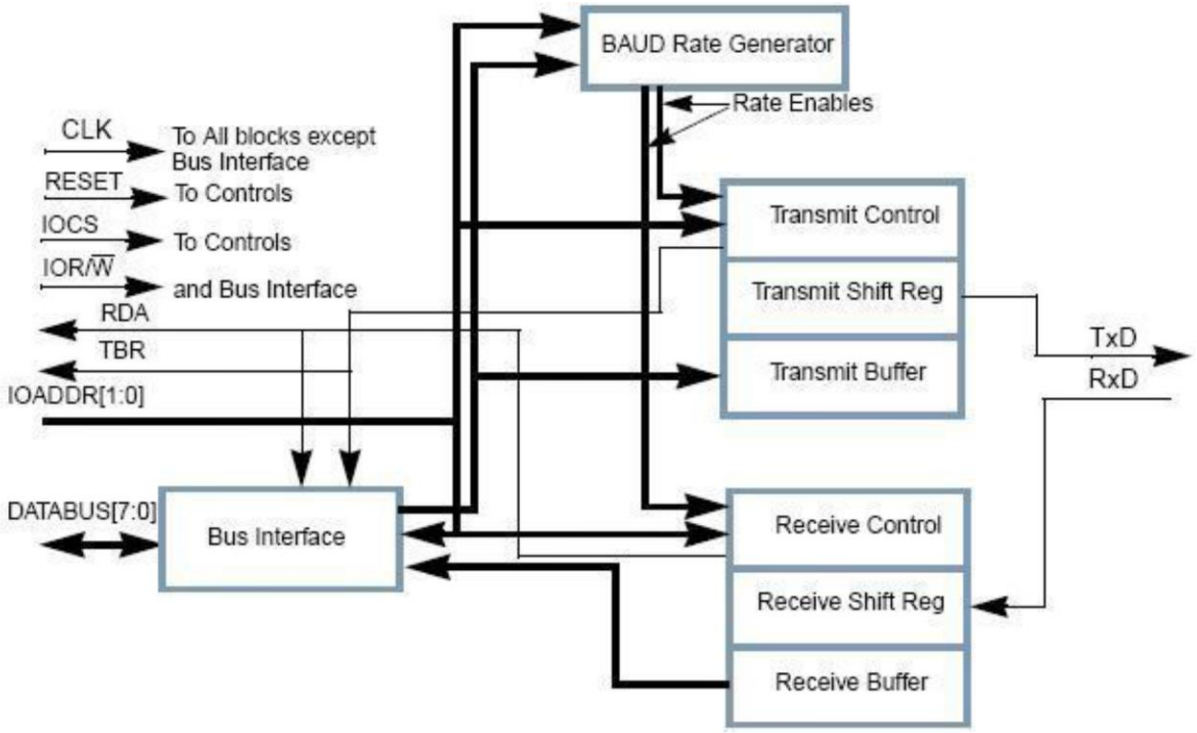


Figure 2: SPART Block Diagram

Bus Interface

The Bus Interface contains the 3-state drives which attach the SPART to the DATABUS. In addition, it contains the multiplexer which selects the Receive Buffer or the Status Register. The Status Register consists of RDA and TBR in positions 0 and 1, respectively. The Status Register is not actually a register, but just connections from RDA and TBR which are stored at their respective sources. The remaining six bits connected to the multiplexer for the Status Register are zeros. Note that RDA and TBR are provided both as direct signals to the Processor and as part of the Status Register content accessible by the Processor via the DATABUS. If interrupt-based I/O is used for the SPART, then the direct signals can be used as inputs to the interrupt system. If program-based I/O is used, then the Status Register content (RDA, TBR) can be accessed by the program using an I/O read operation on the Status Register to determine if an I/O data operation is needed. In either case, RDA and TBR can be used as part of a "handshake" between the processor and the SPART during I/O transactions.

In addition to the above datapath constructs, the Bus Interface also contains combinational control logic for the above. In particular, it uses IOCS and IOR/W to make sure that the 3-state drivers are never turned on in conflict with other drivers on DATABUS.

Baud Rate Generator

The BAUD Rate Generator (BRG) produces an enabling signal or signals for controlling the transmitter and the receiver. In traditional UART designs, transmitter and receiver clocks, which typically are the same frequency, are used to perform the necessary timing for controlling the BAUD rate of the transmitted serial information and for controlling the sampling of received information. Since we have no separate clock source, we cannot use this approach, but must instead depend upon the BRG to produce enable signals for these purposes instead of separate clocks. The reason for producing an enable signal instead of a clock is to avoid the problem of having multiple clock domains. The enable signal is produced by a down counter and decoder circuit to perform divisions of the frequency of CLK. Note that in Verilog, the enable signal syntax is a conditional within an always block:

```
always@(posedge clk)
  if (enable)
    ...
```

The enable signal is asserted for a duration of one CLK period, typically with a frequency of 2^n × baud rate, where n ranges from 2 to 4. We will assume that 4 is used. In the design specification of the BRG, we have a special problem in that we will vary the frequency of CLK driving the BRG. Thus, the BRG must be programmable to maintain a fixed baud rate in the face of a changing clock frequency. Programming is achieve by the processor loading two bytes, DB(High) and DB(Low) into the Divisor Buffer in Fig. 3. This buffer drives the data inputs to the down counter as shown in Figure 3. The divisor is the nearest integer to (clock frequency/(2^n × baud rate) − 1). When the counter contains 0, it is reloaded with the divisor. So the count goes from the divisor to zero at the CLK rate. If the decoder consists of a zero detect on the entire counter, then an enable is produced for a single clock period at a rate of one every (CLK/divisor + 1) = CLK/(2^n × baud rate) With the divisor ranging from 1 to 65,535, division by 2 through 65,536 can be accomplished. By using appropriate divisor and designing appropriate decoder, pulses can be produced at 2^n times the baud rate. If an additional enable is required, for example, at the baud rate itself, it can be generated by a 4-bit down counter with a zero decoder and with the (2^n × baud rate) enable as an input. The following example illustrates the Basic Baud Rate concept.

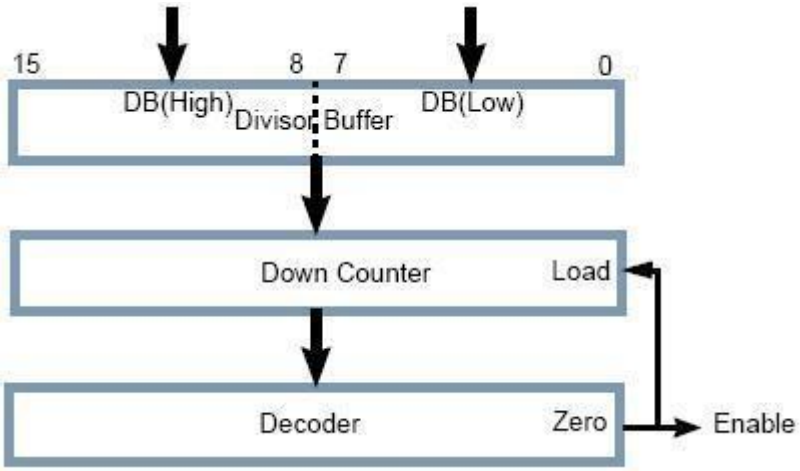


Figure 3: Baud Rate Generator

Example: *Suppose that CLK has frequency 25 MHz and that the desired enable frequency is (2^n. baud rate) where n = 4 and the baud rate is 9600 (bits/second). The divisor required is 25,000,000/(16 . 9,600) − 1 = 161.76 which is rounded to 162. This becomes A2 in hexadecimal. At count time 0, the counter will be loaded with 162 and will be decremented every 40 ns. The counter will be 0 at count time 163, so the interval of time for the counter to be loaded and count down is 163 . 40 ns = 6.52 µs. Inverting, the frequency is 153,374.23 Hz. Dividing by 16, this corresponds to a baud rate of 9586 bits/second. Based on calculations, we have estimated that an error of + or - 3 percent is tolerable, so this design is well within tolerance.*

In your final project, to facilitate communication with the "console" (Hyperterminal) immediately after a reset, a divisor should be loaded into the Divisor Buffer upon processor reset. This divisor should be in dedicated locations in memory. The memory should also contain a "boot program" that executes automatically on reset to load the divisor in memory into the Divisor Buffer. The clock frequency has been set before the reset is applied. Further, in order to provide a means of setting a divisor before your system can execute code, the reset should initialize the Divisor Buffer to the divisor corresponding to a clock of 50 MHz and 9600 bits/second. You can modify this value to something else if necessary for your design.

More Information

For information on the remaining SPART components, you may consult the RS232 UART implementation in the Quartus IP catalog. Note that there are many differences between the UART and SPART design specifications and that the UART IP is not to be used for purposes other than as an example in this assignment.

Hardware Testbench

In order to test your SPART in the lab, you will need circuitry to mimic the behavior of the Processor. We will refer to this as a hardware testbench. This hardware testbench should be able to:

- Demonstrate the ability to transmit and receive characters by, for example, entering characters on a dumb terminal keyboard and echoing them back to the dumb terminal display
- Loading the Baud Rate Generator with an arbitrary value.

The testbench needs to provide four hardwired divisor values. The testbench must load one of these values into the Divisor Buffer after reset has been applied and removed. The value loaded will be determined by the values provided by two DIP switches that are set before reset is applied.

DIP Setting	Baud Rate
00	4800
01	9600
10	19200
11	38400

Hardware Harness

Due to the complexity of the board and the possibility of causing damage by improper design or pin assignment, you are required to use a harness that surrounds your design. Later designs done by you will use your own pin assignments and configurations, but for now a harness will be provided. There are one provided file for the miniproject which is described below.

- lab1-spart.v – This is the top level of the project which connects to external I/O pins as well as instantiates the SPART and your mock "Processor" which is referred to as the "driver" in the file.
- You need to implement SPART and driver as part of your assignment.

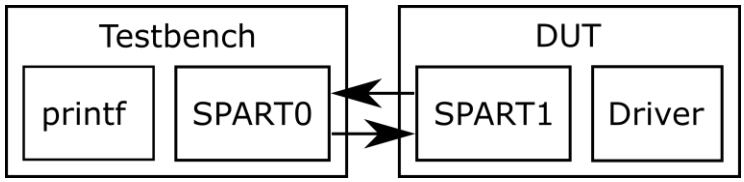
Lab Demo

In lab you are to demonstrate the operation of your SPART as follows:

- Show that characters can be transferred between the dumb terminal and your hardware testbench via the SPART.
- Demonstrate transmission at multiple baud rates

Simulation Testbench

In order to test your SPART and driver, you should implement a printf-like simulation testbench configured as in the diagram below. The DUT is the same SPART and driver used for the hardware testbench. The same SPART module can be instantiated within the simulation testbench to simplify interaction with the serial interface. The testbench should be implemented as Verilog or System Verilog. It need not be synthesizable.



Report

Your report should consist of the following:

- Verilog code for your entire design with clear, useful commenting
- An accompanying narrative describing the function of the SPART and each of the blocks including the testbench
- A record of the experiment conducted including the characters transmitted for a basic test
- A discussion of problems encountered in the design and solutions employed.