The UARx (Universal Asynchronous Receiver-Serial to Parallel Converter)

by

Jahnabi Phukan

B.E., Visvesvaraya Technological University, 2010

A M Eng Project Submitted in Partial Fulfillment

Of the Requirements for the Degree of

Master of Engineering

in the Department of Electrical and Computer Engineering

The UARx (Universal Asynchronous Receiver-Serial to Parallel Converter)

By

Jahnabi Phukan

B.E., Visvesvaraya Technological University, 2010

Supervisory Committee

Dr. Fayez Gebali, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Kin Fun Li, Co-Supervisor

(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Fayez Gebali, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Kin Fun Li, Co-Supervisor

(Department of Electrical and Computer Engineering)

ABSTRACT

Universal Asynchronous Receiver Transmitter (UART) is a full duplex receiver/transmitter. It is a microchip with programming that controls a computer's interface to its attached serial devices and is widely used in data communication process especially for its advantages of high reliability, long distance and low cost. This project is specifically about the receiver (UARx) design that consists of two modules: bit-ASCII module and ASCII-word module. Each module specifies the function of their own individual sub-module. The bit-ASCII module, once detected the start bit, collects each bit serially and converts it to a valid ASCII character. After the conversion, each ASCII character is transferred to the ASCII-word module. The whole design provides non-clocked serial communications between two devices. The receiver logic implementation can be set up at different baud rates using a 50hz clock. All modules are designed and synthesized in VHDL and the reliability of VHDL implementation of the UARx is verified by simulated waveforms using Xilinx ISE 13.4 tool for simulation and synthesis.

Table of Contents

List of Tables

List of Figures

ACKNOWLEDGEMENT

Foremost, I would like to express my sincere and deepest gratitude to my supervisor Dr. Fayez Gebali and co-supervisor Dr. Kin Fun Li for the continuous support of my project work for their patience, motivation and immense knowledge. Dr. Gebali's guidance helped me in all time of research and writing of this project. It would not have been possible to complete my MEng project without his invaluable guidance.

Dr. Gebali has been amazingly supportive to my recovery when my steps faltered and overcoming many crisis and finishing this project. The amount of learning that I had from Dr. Gebali outweighs any course book that I have read. Co-supervisor Dr. Li provides valuable suggestions and assists me during the course of my degree.

I could not have imagined having a better supervisor and mentor for my MEng study without Dr. Gebali and Dr. Li.

I owe my gratitude to my family who have made this project possible by constant encouragement and because of whom my graduate experience has been one that I will cherish forever.

Chapter 1

Introduction

1.1 Serial Communication Interface (SCI)

A serial communication interface is a device that enables the serial (one bit at a time) exchange of data between a microprocessor and peripherals such as printers, external drivers and scanners. But in addition, the SCI enables serial communication with another processor or with an external network. A serial interface is the fastest and easiest way to get data into or out of a device. In some applications this interface is known as a UART (Universal Asynchronous Receiver/Transmitter). UART allows full-duplex communication in serial link thus has been widely used in the data communications and control systems. Basic UART needs only two signals to complete full-duplex data communication. Specifically, it provides the computer with the RS-232 data terminal equipment interface so that it can “talk” to and exchange data with modems and other serial devices. More advanced UART provides some amount of buffering of data so that the computer and serial data device streams remain coordinated. The most recent UART, the 16550 has a 16-byte buffer that can get filled before the computer’s processor needs to handle the data. A UART contains a parallel to serial converter that serves as a data transmitter and a serial to parallel converter that serves as a data receiver. There are different types of devices with enormous range of applications and perhaps the most common interfaces are USB, RS-232, RS-422 and HDMI. The primary benefit of serial communication is that it reduces the distortion of signal, therefore makes data transfer between two systems separated in great distance possible. The disadvantage of UART is that it is asynchronous, so the clock on both side needs to be pretty close to work, which often requires an external crystal (as opposed to internal RC oscillator). Since UART doesn’t have a clock line, there might be a potential damage in the data if there is a clock mismatch between the two ends or their settings are at different baud rates.

1.2 Hypothesis for Serial I/O

Digital data flows into or out of a circuit or device using two basic methods. The first way is via parallel data interfaces which transfer multiple bits over parallel data paths. This is the fastest way to transfer data, it’s expensive and hardware intensive because a driver and a receiver are needed for each path. Moreover, categorizing parallel interfaces include automotive interfaces providing data which include speed, medium and applications as shown in Table 1. The primary

limitation of a parallel interface is speed over distance. Such limitations not only restrict range but data rate as well. Table 1[1] shows the most common interface use today.

TABLE 1: AUTOMOTIVE INTERFACES

Interface	Speed	Medium	Application
Controller area network (CAN)	5 kb/s to 1 Mb/s	Twisted pair	Most automotive uses
Ethernet	100 Mb/s, 1 Gb/s	Shielded twisted pair	Most automotive uses
FlexRay	10 Mb/s max.	Twisted pair	Sensor, actuator to controller
Local Interconnect Network (LIN)	19.2 Kb/s max.	Single wire	Switches, doors, windows, seats, controls etc.
Media Oriented System Transport(MOST)	25, 50, 150 Mb/s	Plastic optical fibre (OPF) or twisted pair	Infotainment and Navigation Systems
On Board-Diagnostics II (OBD II)	10.4, 41.6 Kb/s	Multi-wire cable	Diagnostics and performance monitoring
Single Edge Nibble Transmission (SENT)	Clock tick interval 1.5 to 90µs (11 to 667 kb/s)	Any wire	Sensor to Controller only

And the second digital data flow method is serial data interfaces. In theory, data transfer in serial data interface is slower because of the bit–by-bit transfer. Serial connections require only a single path, either unbalanced or differential. Special line drivers and receiver with equalization deal with the various path distortions. The primary benefit of serial interfaces makes it possible to go faster over longer distances with less hardware at lower cost. The best way to categorize serial interfaces is by: wired baseband, wired broadband and wireless with the detailed data provided including speed, distance, media, application and standard designations in the tables below. Applications targeted by Wired-Baseband interfaces include consumer, automotive, industrial and PCB interconnects and backplanes [1].

TABLE 2: CONSUMER AND VIDEO INTERFACES

Interface	Speed	Distance	Medium	Application
DisplayPort	1.296,2.16,4.32 Gb/s	2m for max rate	20-wire cable, 4 diff. pairs	PC to video monitor, projector
High-Defination Multimedia	6 Gb/s max.(Version	5m for max rate	19-wire cable, 4 shielded twisted	TV set, DVR, cable box, other

Interface(HDMI)	2.0)		pairs	video
Lightning	480 Mb/s	Several feet	8-wire cable, 2 differential pairs	Apple iPhone, iPad data and charging
Thunderbolt	10 Gb/s, dual channel	3m(copper)50m (fiber)	20-wire cable, optional fiber optical cable	PC, Apple Mac peripherals
Universal Serial Bus(USB)	480 Mb/s(Ver.2) 5 or 10 Gb/s(Ver.3)	5m (ver.2.) 3m (ver.3.1)	4-wire cable, shielded twisted pair(ver.2.0). 10-wire cable with 3 twisted pairs(ver.3.1)	All purpose: PC peripherals, video, solid state drives, charging etc.

Consumer and video interfaces are used by everyone in everyday life. One of the best application is the USB interface, which is a widely used interface in the world (like RS-232 interface). It is also used for video connectivity or PC peripheral connections [1].

TABLE 3: INDUSTRIAL INTERFACES

Interface	**Speed**	**Distance**	**Medium**	**Application**
Ethernet	100 Mb/s, 1 and 10 Gb/s	10 m	Shielded or Unshielded twisted pair; fiber cable option	Connecting field buses to existing business networks or the Internet
Foundation Field Bus	H1.31.5 kb/s, HSE: 100 Mb/s, 1 Gb/s	1900 m max.	Shielded or Unshielded twisted pair	Connect sensors, actuators etc., in the process control
Highway-Addressable Remote Traansducer(HART)	1200 and 3600 b/s	< 10,000 ft	Shielded twisted pair	Analog and digital sensor and actuator connections in process control
Modbus	9.6 and 19.2 kb/s	< 1000 ft.	Shielded or Unshielded twisted pair	Monitor and Control with PLCs
Profibus	9.6 and 31.25 kb/s to 12 Mb/s	1200 m	Shielded or Unshielded twisted pair	Monitor and Control in process automation

RS-232	1.2 and 115.2 kb/s	< 50 ft.	Multiwire Cable	Connections to PC peripheral and industrial devices
RS-485	100 kb/s to 10 Mb/s	40 to 4000ft.	Shielded or Unshielded twisted pair	Industrial and commercial networks

Industrial interfaces are the one which connect computers to one another and to a huge variety of machines, robots, sensors, actuators and other devices such as widely deployed programmable logic controller (PLC). Some industrial interfaces define only basic physical layer, while other define specific networking configurations and protocol. Table 3 summarizes the most widely used industrial interfacers [1].

PCB/backplane interfaces are the interfaces to connect chip to chip or PCB to PCB. These interfaces are faster and are mostly used in routers, switches and other high-speed equipment. Table 4 summarizes the most widely used interfaces in PCB/backplane interface along with a number of special interfaces [1].

TABLE 4: PCB AND BACKPLANE INTERFACES

Interface	**Speed**	**Distance**	**Medium**	**Application**
100 Gigabit Ethernet attachment Unit Interface (CAUI)	Ten 10-Gb/s links for max. 100 Gb/s	50 cm	PCB and backplane traces	Chip and module connections on Ethernet equipment
HyperTransport	32 Gb/s per serial link	< 12 in.	PCB and backplane traces	Chip and module connections
Inter-Integrated Circuit (I^2C)	100,400 kb/s, 1, 3.4 Mb/s	< 12 in. (PCB), < several feet cable	PCB traces or 4-wire cable	Chip and module connections
PCI Express(PCIe)	2.5, 5, 8, 16 Gb/s link	< 12 in.	PCB and backplane traces	Connections between processor and peripherals, backplanes

RapidIO	1.25, 2.5, 3.125, 5, 6.25, 10 Gb/s per link	< 1 m	PCB and backplane traces	Connections between processor and peripherals, backplanes
Serial peripheral interface(SPI)	20 Mb/s up to 100 Mb/s	< 1m	PCB traces or short 4-wire cable	Connections between processor and peripherals, chip to chip
10/40 Gigabit Attachment Unit Interface(XAUI/XLAUI)	40 or 100 Mb/s	<50 cm	PCB and backplane traces	Chip and module connections on Ethernet equipment

Wired-Broadband Interfaces use modulation to transmit data on a cable. These interfaces are used primarily because data can be transmitted faster over longer distances. However, the downsides of wired-broadband interface are greater complexity and cost, since both ends on the cable connection require modulation and demodulation with the modem being the connecting device. Table 5 explains the detail of widely used broadband interfaces with their main characteristics [1].

TABLE: 5 MOST POPULAR BROADBAND INTERFACES

Interface	Standard	Speed	Medium	Modulation	Application
Data Over Cable Service Specifications(DOCSIS)	CableLabs, ISU	42.88 Mb/s max. per 6- Mhz channel	Optical fiber and coax cable	QPSK, m-QAM	Cable TV, Internet access, VoIP
Digital Subscriber line (DSL)	ANSI, ITU	Depends on version, from 8Mb/s to 1 Gb/s	Twisted-pair telephone cable	Discrete Multitone(DMT) with BPSK, QPSK, m-QAM	Internet access, VoIP
G3-PLC	IEEE, ITU	46kb/s, 300 kb/s	LV or MV ac power line	OFDM with DBPSK, DQPSK or D8PSK	Utility metering, smart grid, lighting, alternative energy monitoring

G.hn	ITU	Upto 1 Gb/s	Twisted pair, coax, or ac power line	OFDM with QAM	Home networking, video, Internet access
HomePlug	HomePlug Alliance, IEEE	Depends on version: 10 Mb/s, 200 Mb/s to 1 Gb/s	Home ac power line	OFDM with QAM	Home networking, video, Internet access
Multimedia over Cable Alliance (MoCA)	MoCA	700 Mb/s to 1 Gb/s	Home cable TV coax	OFDM with QAM	Home networking, video, Internet access
Powerline, Intelligent, Metering Evolution(PRIME)	PRIME Alliance, IEEE, ITU	5.4 kb/s, 128.6 kb/s, up to 1 Mb/s	LV or MV ac power line	OFDM with DBPSK, DQPSK or D8PSK	Utility metering, smart grid
X10	Authinx	120b/s	Home ac power line	ASK/OOK	Home lighting and appliance control

Wireless Interfaces are designed to cover short ranges, from a few inches up to about 100 meters and these are mostly used as a cable eliminator. Table 6 explains the details about this interface [1].

TABLE 6: MOST POPULAR SHORT-RANGE WIRELESS INTERFACES

Interface	**Standard**	**Frequency**	**Speed (max.)**	**Modulation**	**Range (max)**	**Application**
802.15.4	IEEE	868,902-928 Mhz, 2.4-2.835 Ghz	20,40,250 kb/s	DSSS with BPSK or O-QPSK	10-100m	Industrial,consumer,loT,utility
Bluetooth	Bluetooth SIG	2.4-2.4835 Ghz	1 Mb/s, 2.1 Mb/s, 3Mb/s	FHSS with GFSK,π/4-DQPSK and 8DPSK	1-100 m	Speakers,headsets,medical,fitness, sm-artphones and watches

Digital Enhanced Cordless Telecommunications	ETSI	1880-1930 Mhz	Upto 2Mb/s	GFSK,π/2-DBPSK, π/4-DQPSK, π/8 8DPSK	200m	Cordless phones, home automation
EnOcean	ISO/IEC	315,868,902-928 Mhz	125 kb/s	ASK	30 m	Building or home automation,Industrial
ISA100 - 11a	ISA, IEC, IEEE,WCI/ASCI	2.4-2.4835 Ghz	250 kb/s	DSSS with O-QPSK	10-100m	Process automation,Industrial,IoT
Near-field communications	ISO/IEC, ECMA, GSMA	13.56 Mhz	106-424 kb/s	ASK	< 20 cm	Payments, access, pairing
Ultra Wideband(UWB)	IEEE, WiMedia Alliance	3.1-10.6 Ghz	480 Mb/s, 1.3 Gb/s	OFDM, BPSK, pulse	< 10 m	Video, docking, military
Wi-Fi(802.11)	IEEE	2.4-2.4835 Ghz, 5.725-5.875 Ghz, 60 Ghz	11 Mb/s to 7 Gb/s	DSSS, mostly OFDM	100 m	LAN, Internet access, loT, Industrial
Wireless HART	HART comm. Foundation,IEEE	2.4-2.4835 Ghz	250 kb/s	DSSS with O-QPSK	10-100m	Process and building automation, sensor nets
ZigBee	IEEE, ZigBee Alliance	868,902-928 Mhz, 2.4-2.4835 Ghz	20,40,250 kb/s	DSSS with O-QPSK	10-100 m	Industrial, home automation, loT
Z-Wave	Z-Wave Alliance	908.42 Mhz	9.6 and 40 kb/s	GFSK	30 m	Home automation, loT

1.3 RS 232 – PC Serial Port

RS-232 is a standard interface used for connecting serial devices. In other words, RS-232 is a long established standard that describes the physical interfaces and protocol for relatively low-speed serial data communication between computers and related devices. RS-232 is the interface that a computer uses to talk and exchange data with a modem and other serial devices. The serial ports on most computers use a subset of the RS-232C standard. RS-232 can support data rates of up to 920kbps (normally 9600 and 115.2K are the maximum rates) and is commonly found in 9 or 25 pin configurations, however only three pins are required. Most applications drop many of the less commonly used pins, though some configurations such as data modem connect every pin for full handshaking capabilities. An RS-232 is a point to point connection made between a Data Terminal Equipment (DTE) device and a Data Communications Equipment (DCE) device. RS-232 has a maximum cable length of 50 ft. at 9600 baud. Baud rate specifies the number of bits that are being sent over the media and Bit rate describes the rate at which bits are transferred from one location to another, i.e., how much data are transmitted in a given amount of time. The only difference between baud rate and bit rate is that the bit rate measures the number of bits transmitted per second whereas the baud rate defines the number of symbols (symbol typically consists of a fixed number of bits depending on what the symbol is defined as) transmitted per second. RS-232 ports/cables can be found in factory automation and metering, medical equipment, consumer products and many other devices.

The primary benefits of RS-232 are its simple wiring and connectors, wide availability, low cost and most embedded processors include this interface. Some of the disadvantages of RS-232 are its incompatibilities in wiring and configuration between devices, short cable lengths and it is subject to noise interference and low data rates. Table 7 shows the pin description of RS-232 Interface Signals.

Table 7 Pin description of RS-232 Interface Signals.

Pin description	Pin description	Pin description
1. Protective Ground	10. Received for data set testing	19. Secondary request to send
2. Transmitted Data	11. Unassigned	20. Data terminal ready
3. Received Data	12. Sec. Rec'd line Sig. Detector	21. Signal quality detector
4. Request to stand	13. Sec. clear to stand	22. Ring Indicator
5. Clear to stand	14. Secondary transmitted data	23. Data Signal rate selector(DTE/DCE source)
6. Data set ready	15. Transmission signal element timing(DCE source)	24. Transmit signal element timing(DTE source)

7. Signal Ground(common return)	16. Secondary received data	25. Unassigned
8. Received line signal detector	17. Receiver signal element timing(DCE source)	
9. Received for data set testing	18. Unassigned	

1.4 RS-232 Specifications

RS-232 serial port is a UART standard. This means that the standard sets out to ensure compatibility between the host and the peripherals by specifying three parts: common voltage and signal levels, common pin wiring configurations and a minimal amount of control information between the host and the peripheral systems. Unlike many standards which simply specify the electrical characteristics of a given interface, RS-232 specifies electrical, functional and mechanical characteristics in order to meet the above three criteria.

Chapter 2

System Design

2.1: UARx (Universal Asynchronous Receiver)

The UART protocol is an asynchronous serial communication standard. Data is transferred one bit at a time. Essentially, the UART acts as an intermediary between parallel and serial interfaces. The UART serial communication module is divided into three sub modules: the baud rate generator, receiver module and transmitter module. Therefore, the implementation of a UART communication module is actually the realization of three sub-modules. Since asynchronous are self synchronizing, if there are no data to transmit, the transmission line can be idle. The UARx module is divided into two sub modules: bit_2_ASCII (where it converts serial bits into ASCII characters) and ASCII_2_word (where it converts ASCII characters into words). However, in this project, the design is mainly concerned with the receiver module of UART.

The UARx is used to receive serial signals and convert them into parallel data. There are two states in the signal line using logic 1 (high) and logic 0 (low) to distinguish respectively. In UARx, communication speed is defined by the baud rate. Since data are communicated serially it should have the same baud rate, else serial communication would not work. So if one sets different baud rates then the receiver might miss out the bits the transmitter is sending because it is configured to receive data and process it with a different speed. The baud rate specifies how fast data is sent over a serial line including start and stop bits. It is usually expressed in units of bits per second (bps). Common baud rates are 4800, 9600, 19200, 28800, and 38400, but other rates may also be used. The higher a baud rate goes, the faster data are sent/ received, but there are limits to how fast data can be transferred.

2.2: Baud Rate Calculation

Time required to send a bit is $\frac{1}{\text{Baud rate}}$ and is called as a time frame for a bit. A bit has to be sampled within this time frame. Therefore, when clock frequency requirement of the design is calculated, baud rate is taken into account. UART's clock is generated by counting master clock cycles using a counter. Assuming that the system clock rate is 50 Mhz and baud rate is 9600bps.

For the 9600 baud rate, the sampling rate has to be 153,600 (i.e., 9600*16) ticks per second.

Counter value = $\frac{1}{\text{Baud rate}}$* 50 Mhz

= $\frac{1}{9600}$* 50Mhz

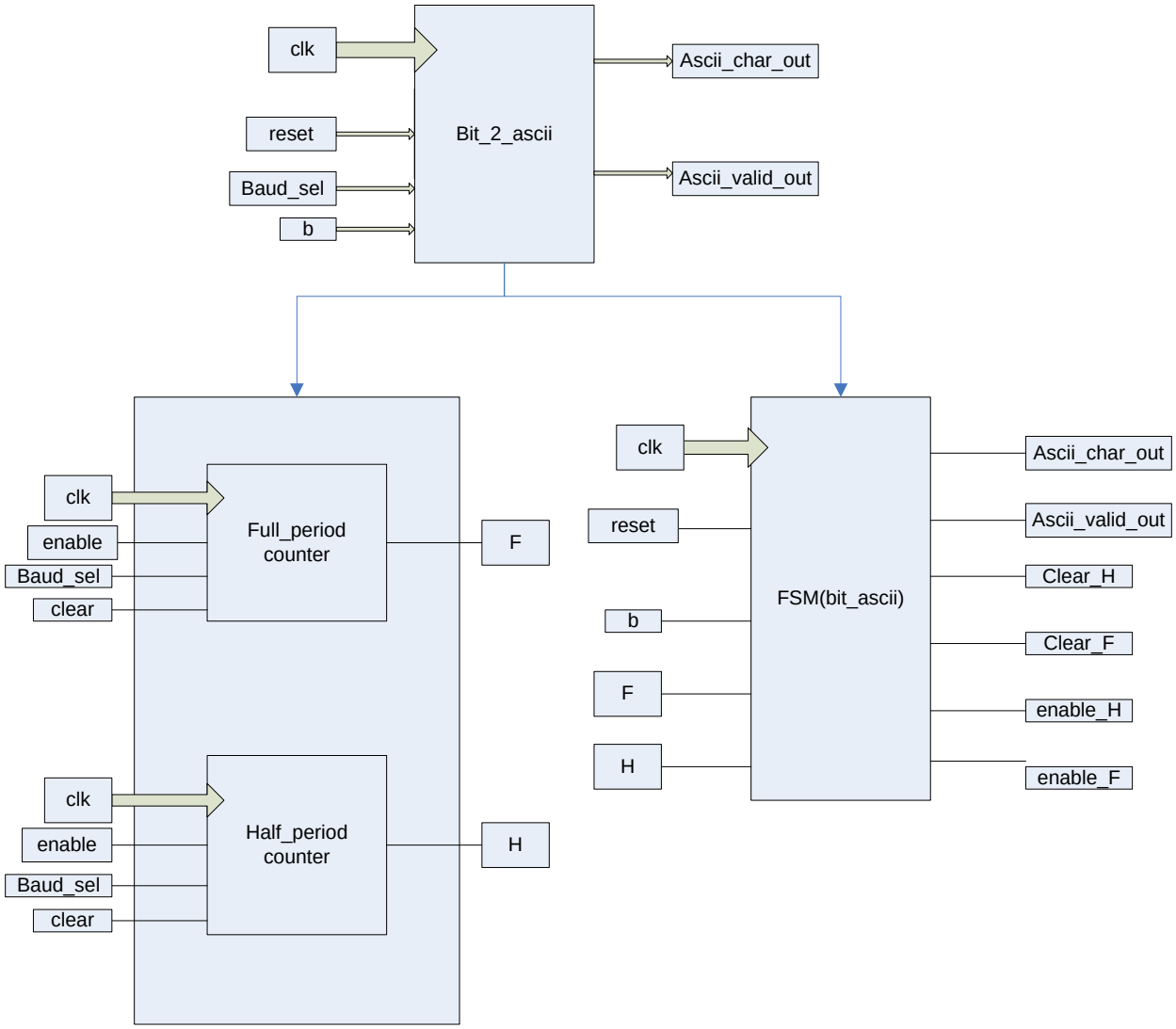
= 5208,33.

Then half of counter value = $\frac{\frac{1}{\text{Baud rate}} * 50\ \text{Mhz}}{2}$ = 2604,17

2.3 Bit_2_ASCII module

The function of the bit_2_ASCII module is to receive 8 serial bits in pin b and have a valid ASCII character at the output. The bit_2_ASCII module is divided into two sub modules: Counters (half period and full period) and Finite State Machine (FSM). In the counters, when the transmission line is logic “0” a LOW is detected on the data input (a start bit has been received). Bit detection feature is implemented in the design which requires the start bit to be low at least 50% of the baud rate clock cycle. The start bit needs to be low for at least eight cycles (which is half period in the half period counter) and is considered as a valid start bit. Once a valid start bit is received, the data bits will be sampled after every 16 cycles (which is full period in the full period counter) and is typically sampled at the middle (or at center) of each bit interval. The finite state machine controls the counters to shift out each bit of serial data into valid ASCII character. Figure 1, summarizes the bit_2_ASCII module.

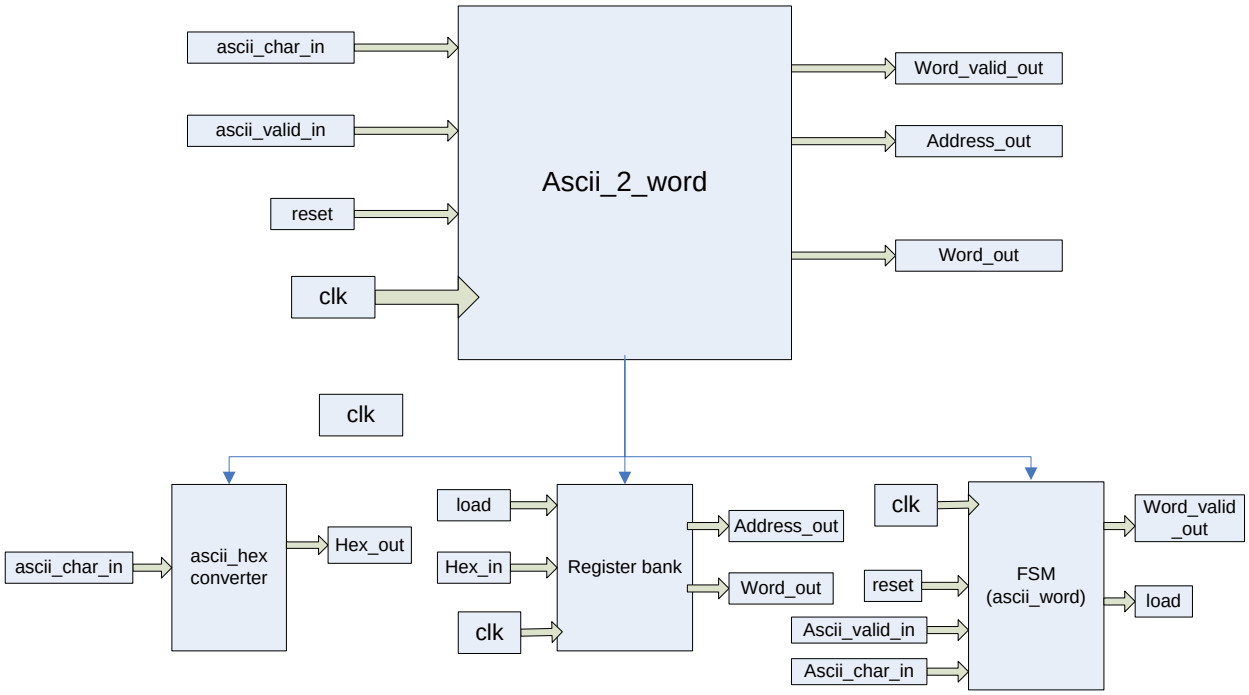
Figure 1. Bit_2_ASCII Module



2.4 ASCII_2_Word module

The function of the ASCII_2_word module is to convert ASCII character from bit_2_ASCII module into valid word out. However, the ASCII_2_word module is divided into three sub modules: ASCII to hex converter, register bank and a FSM (ASCII to word). The ASCII to hex converter module converts ASCII characters to hexadecimal. In the register bank module, the hexadecimals are stored from the converter in the register bank which holds 32 words and produces 4 bits each. Lastly, the finite state machine of ASCII to word helps in controlling the valid ASCII characters and shift out 9 load lines. Figure 2 shows the detail view of the bit_2_ASCII module.

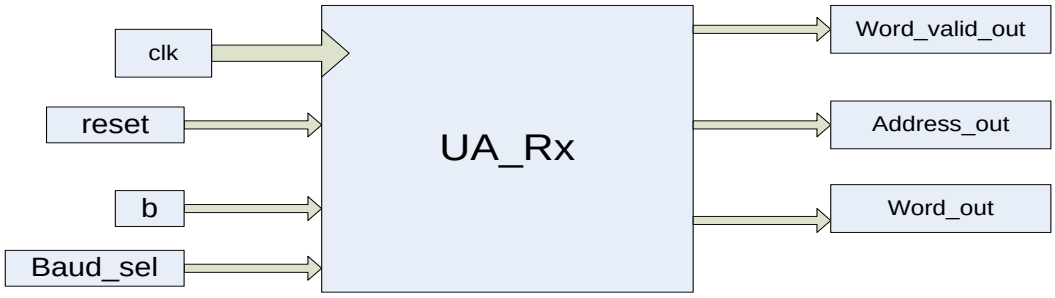
Figure 2. ASCII_2_Word Module



2.5 UARx Whole Design Module

The UARx receiver consists of two modules: bit_2_ASCII module and ASCII_2_word module. In bit_2_ASCII module, it receives 8 bits streaming serially and converts those bits into ASCII character and subsequently in the ASCII_2_word module the valid received ASCII characters from the bit_2_ASCII module is converted into word. Figure 3 shows the block diagram of the UARx receiver module.

Figure 3. UARx Receiver Module

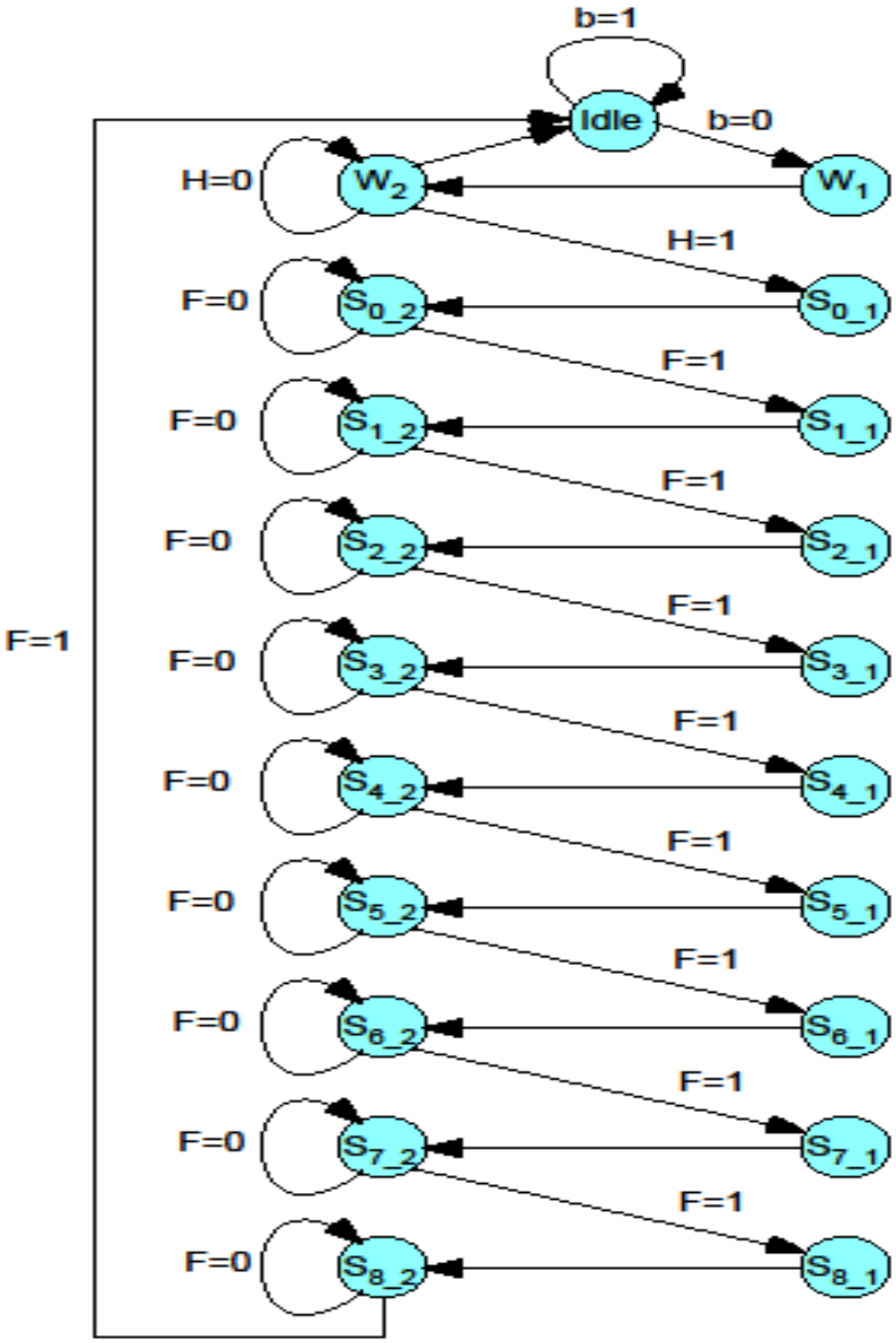


2.6 Finite State Machine of bit_2_ASCII

The FSM of bit_2_ASCII receives 8 bits of data serially. Since the serial frame is asynchronous, a high to low transition will be treated as the start bit of a frame. Based on the detection of the falling edge of the synchronized rxd (receiver) input, it waits half of a bit interval (8 clock cycles) and then after it considered receiving the start bit it re-samples for full period of each bit interval (16 clock cycles).

Once detecting the start bit, it advances 1 full bit period at a time (16 clock cycles) to end up in the middle of the 8 data bits, where it continues sampling the 8 data bits in the center of the serial frame until it reaches the stop bit. This behavior of bit_2_ascii is controlled by the FSM shown in the Figure 4.

Figure 4. Bit_2_ASCII State Machine



Idle: When the transition line (reset) is high in this state, it remains in the idle. Once a valid start bit (b) is detected, i.e., when b is low it moves to next state W_1 and whenever error is detected, i.e., if b is still high it goes back to idle to resample until b is low (b=0).

W_1 - W_2: In this state, when the transition line is still low, i.e., b=0 the FSM enables half period counter by measuring half period (H=0) which is half of a bit period (8 period clock cycles).Then moves to state W_2 and keeps enabling half period counter. Once half period is detected it clears the half period by making half period high (H=1) and then moves to next state S0_1.

S0_1- S0_2: After detecting the half period counter, it advances 1 full bit period at a time (16 clock cycle) i.e., it samples the start bit twice based on 8 clock cycle. So each of the subsequent bits will be sampled at the center of the bit itself (F=0). Once the full period counter is detected it clears the full period by making full period high (F=1) and then moves to next state S1_1 and the same process continues till S8_1.

S8_1- S8_2: After continuing the same process till state S8_1 it then samples for the stop bit. As long as logic 1 is sampled at the stop bit, it receives valid ascii character it then clears the full period counter by making full period high (F=1) and switches back to idle state after the stop bit sampling, or else if it samples as logic 0 at the stop bit it continue to be in the same state S8_2.

2.7 Finite State Machine of Ascii_2_Word

The finite state machine of ASCII to word controls to converts the ASCII character into word. This behavior of the ASCII_2_word finite state machine is shown Figure 5. Here are the explanations of each state in this finite state machine.

Idle: When the transition line (reset) is high in this state, it remains idle. Once a valid start ASCII character is detected, i.e., when ASCII_valid_in = “R,r”, it moves to next state Reg. However if error is detected, i.e., if ASCII_valid_in is low (ASCII_valid_in =0) it resamples until ASCII_valid_in =1 and remains in the same state or else it goes back to the idle state.

Reg: In the Reg state, when a valid ASCII character is received, i.e., when ASCII_valid_in = “0-9, a-z, A-Z” it switches to next state Address. If it doesn’t receive a valid ASCII character, i.e., ASCII_valid_in is low (ASCII_valid_in =0) it resamples until ASCII_valid_in =1 and remains in the same state or else it goes back to the idle state.

Address: In this state, it waits for two valid ASCII characters, once the two characters get detected, i.e., ASCII_valid_in = “Space” it moves to space 1 state and if ASCII_valid_in =“Equal” it moves to equal state; however if it doesn’t receive valid ASCII character, i.e., if ASCII_valid_in is low (ASCII_valid_in = 0) it resamples until ASCII_valid_in = 1 and remains in the same state or else it goes back to the idle state.

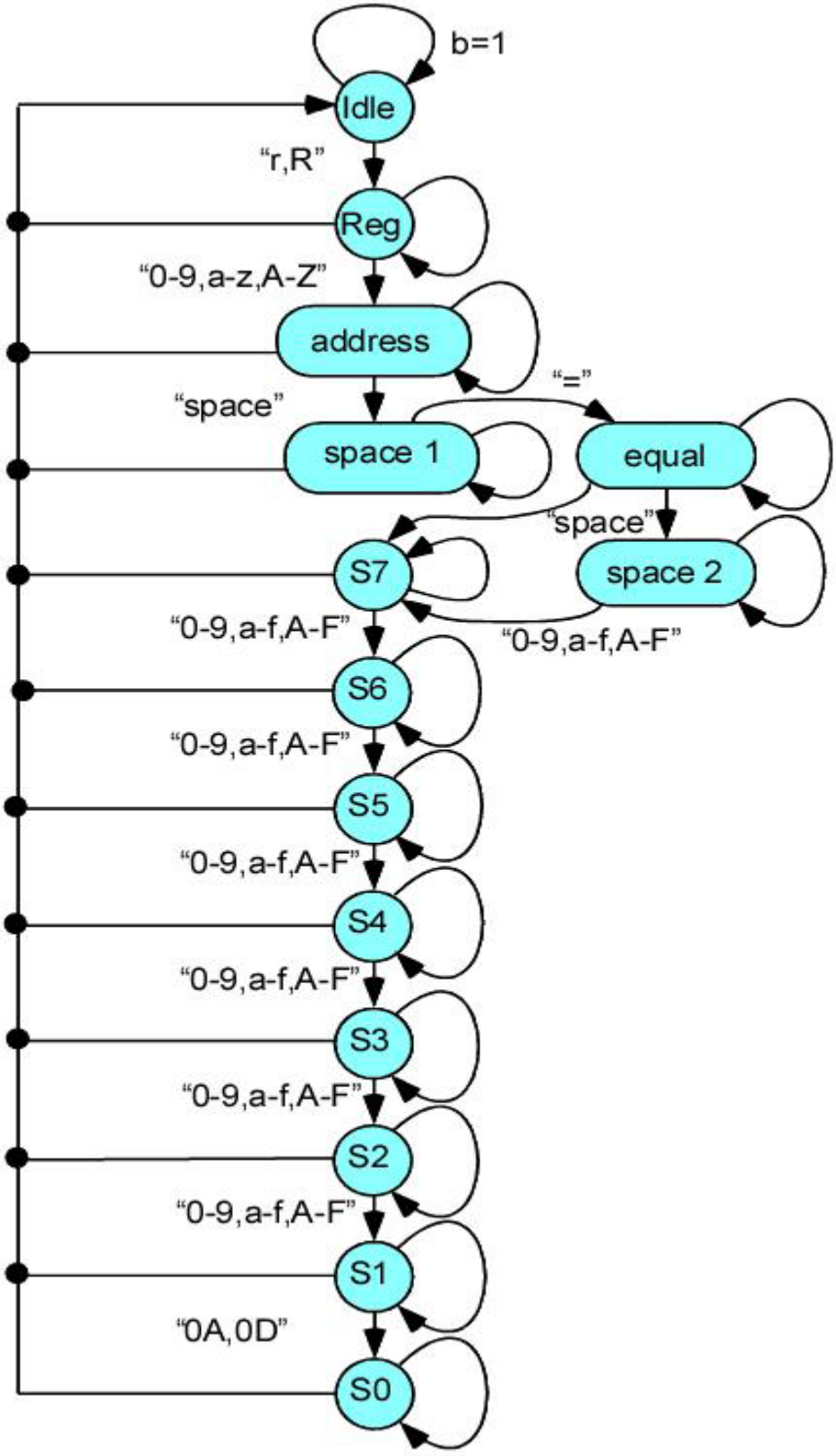
Equal: In the Equal state, it also waits for two valid ASCII characters, once the two characters get detected, i.e., ASCII_valid_in = “Space” it moves to Space 2 state and if ASCII_valid_in =“0-9, a-f, A-F” it moves to S7 state; however if it doesn’t receive valid ASCII character i.e., if ASCII_valid_in is low (ASCII_valid_in = 0) it resample until ASCII_valid_in = 1 and remains in the same state or else it goes back to the idle state.

Space 2: In the Space 2 state, it waits for a valid ASCII character, once the character get detected, i.e., ASCII_valid_in = “0-9, a-f, A-F” it moves to S7 state however if it doesn’t receive a valid ASCII character, i.e., if ascii_valid_in is low (ASCII_valid_in = 0) it resamples until ASCII_valid_in = 1 and remains in the same state or else it goes back to theidle state.

S7: In the S7 state, it waits for a valid ASCII character, once the character get detected, i.e., ASCII_valid_in = “0-9, a-f, A-F” it moves to S6 state and starts to load valid word; however if it doesn’t receive valid ASCII character, i.e., if ASCII_valid_in is low (ASCII_valid_in = 0) it resamples until ASCII_valid_in = 1 and remains in the same state or else it goes back to the idle state and the same process continues till state S1.

S0: In the S0 state, it waits for two valid ASCII characters, once the characters get detected, i.e., ASCII_valid_in = “0A” and ASCII_valid_in = “0D” it moves back to idle state and detects valid word; however if it doesn’t receive valid ASCII character, i.e., if ASCII_valid_in is low (ASCII_valid_in = 0) it resamples until ASCII_valid_in =1 and remains in the same state.

Figure 5. ASCII_2_Word State Machine



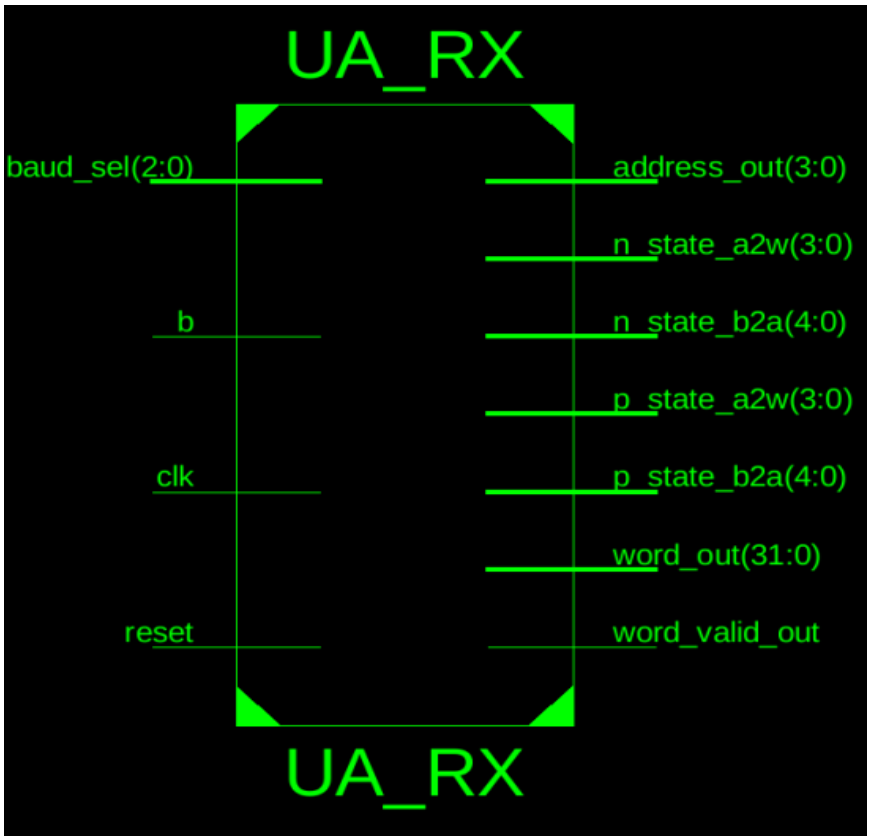
Chapter 3

Implementation

3.1 RTL Schematic of UARx Receiver

In this section it shows the Register Transfer Level (RTL) view of the top level block and the overall RTL schematic of the UARx (receiver of UART), and the associated hardware implementation. The top level block and the RTL schematic for UARx receiver are shown in Figure 6.

Figure 6. The Top Level Block and RTL Schematic of UARx Receiver



3.2 Performance Evaluation

The hardware device chosen for implementation is Xilinx Spartan-3E xc3s1200e-4fg320. This Spartan contains 8672 total number of slices out of which available logic utilization consists of 17344 flip flops, 17344 in 4 total number of input LUTs and 250 bonded IOBs. Apart from the usage of slices in logic, they are also used for routing signals within the device. The test study analyzes and compares their power consumption, device utilization and timing analysis using Xilinx ISE tool.

3.3 Device Utilization

Once the receiver (UARx) was successfully compiled, it was synthesized to access the device utilization and performance. The device utilization results are shown below in Table 8.

The device utilization summary can be obtained through the following:

- Go to ISE Navigator Design Pane > select Implementation (view) > select the design as "top module".
- In the process pane > select synthesize – XST (Xilinx Synthesis Technology) > view the Design Summary (synthesized window).

TABLE 8: Device Utilization Summary			
Design goal	**Used**	**Available**	**Utilization**
Number of Slices	132	8672	1%
Number of Slice Flip Flops	133	17344	0%
Number of 4 input LUTs	249	17344	1%
Number of bonded IOBs	61	250	24%
Number of Occupied slices	152	8672	1%

Table 8: Device Utilization Summary of UARx receiver

The on-chip logic utilization summary shows that UARx receiver uses a total number 132 of slices from the available 8672 which includes 61 IOBs (basic mapping and synthesis constraints) and 1 BUFGMUXs (multiplexed global clock buffer) for each of the three profiles.

3.4 Timing Analysis

Timing analysis shows the timing summary of the UARx receiver in which the clock signals are generated by combinatorial logic and having XST as the primary clock signals with speed grade of -4. The design uses the user defined constraints (UDC) file to define clock operating frequency. The clock duration was 14ns. Timing summary is described below:

- Maximum period: 7.481ns (maximum clock frequency of 133.669MHz)
- Maximum input arrival time before clock: 9.566ns
- Maximum output required time after clock: 12.914ns
- Maximum combinational path delay: 13.665ns

The clock frequency can be obtained by running the synthesize - XST option from the process panel in the ISE tool. Once the synthesis report is complete, the timing report can be viewed from right clicking the synthesize - XST. This report also reveals the source and the destination of the critical path (a signal).

The data path that cause the critical path delay for UARx are the following:

- For the Delay of 7.481ns

  Source: bit_2_ascii_instance/counter_full_period_instance/temp_0(FF)

  Destination: bit_2_ascii_instance/counter_half_period_instance/temp_31(FF)
- For the Offset of 9.566ns

  Source: baud_sel <2> (PAD)

  Destination: bit_2_ascii_instance/counter_half_period_instance/temp_31(FF)
- For the Offset of 12.914ns

  Source: bit_2_ascii_instance/fsm_bit_ascii_instance/ascii_char_out_3(LATCH)

  Destination: n_state_a2w<2> (PAD)
- For the Offset of 13.665ns

  Source: baud_sel <2> (PAD)

  Destination: n_state_a2 w<3> (PAD)

The overall operation completion time was obtained through running the ISIM (ISE simulator) simulation through the “simulation” view of the ISE tool and double clicking the “simulate

behavioral model", which will show the test bench output. The time difference can be calculated between vertical markers place on the rising edge when "start" signal becomes high till the rising edge instant when the "b" signal is set to low.

3.5 Power Consumption

The total on-chip power is given by the static power and the dynamic power. The static power results mainly from the leakage current within the device from the transistors and exists even when the transistor is logically "OFF". The dynamic power depends on the power utilized by the internal circuitary and the power consumed by the device's inputs and outputs.

The power consumption can be obtained through the following:

- Go to ISE Navigator Process Pane > select Implementation (view) > Place and Route Analyze Power Distribution (XPowerAnalyzer)
- Go to the XPA tool window and the clock frequency in three drop down list.
- Go to Tools in the menu bar > update power analysis.

The whole design uses 1% of the whole chip resources. The total power used in the design is 0.172W including static power of 0.159W and dynamic power of 0.014W. The dynamic power is 10 times more than static power depending on the clock operating frequency.

3.6 Hardware Simulation

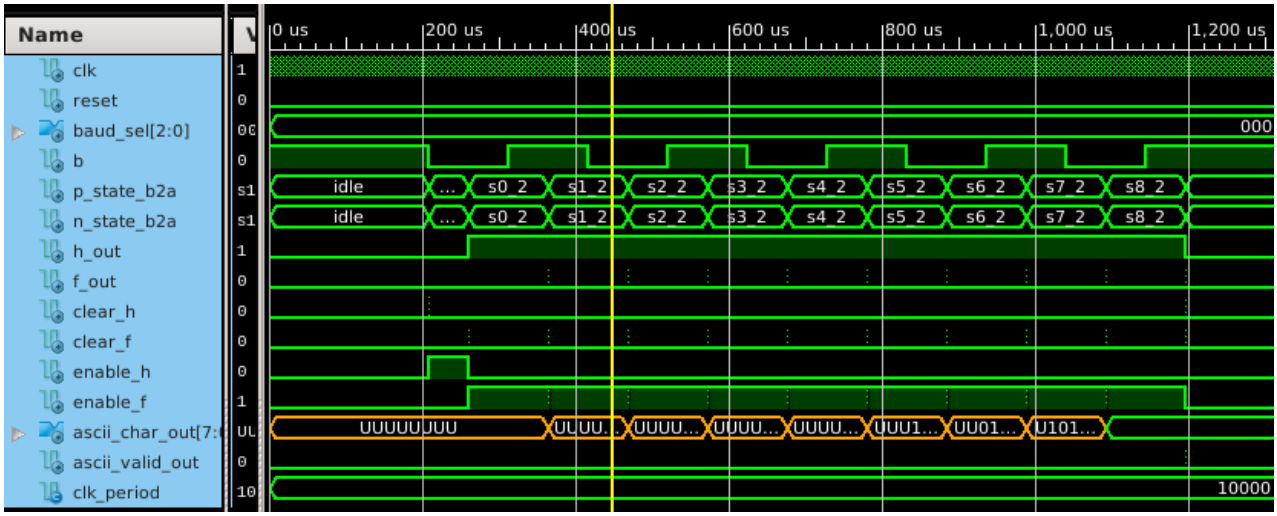
The UARx receiver was designed and synthesized and implemented in VHDL using Xilinx ISE Project Navigator 13.4. The implemented top level and overall RTL schematics were presented in previous section. The VHDL testbenchs were created and simulated to verify that the hardware performs correctly. There are screenshots of the testbench which are explained briefly in the following subsections.

3.6.1 VHDL Simulation of bit_2_ASCII module

The initial state is used as a system initialization mode which occurs upon reset. By observing external data input "b" as high it starts with the idle state. Once start-bit "b" is identified by detecting from high to low, it switches the state from idle to S0_0-S0_1 where the ideal time for sampling is at the half of a bit period for half period and then it samples 1 full bit period at a time which is the middle point of each serial data bit. And thereafter, it can be seen that a valid ASCII

character is reflected on ASCII_char_out. The screen shot of the bit_2_ASCII simulation is shown in Figure 6.

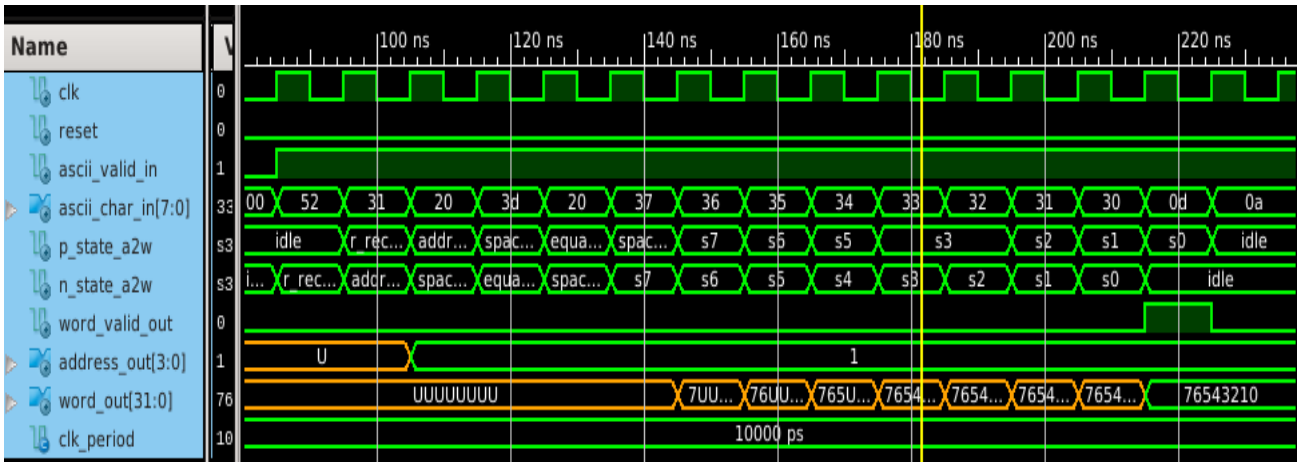
Figure 6. Screen shot of bit_2_ASCII testbench output



3.6.2 VHDL Simulation of ASCII_2_word Module

The initialization mode starts upon reset. By observing ASCII_valid_in as high it starts to sample valid ASCII characters. Once it reaches the idle state it detects a valid word which gets reflected in word_out. The screen shot of the bit_2_ASCII simulation is shown in Figure 7.

Figure 7. Screen shot of ASCII_2_word testbench output



3.6.3 VHDL Simulation of UARx

The initial state is used as a system initialization mode which starts with “high” reset. By observing external data input “b” as high, it starts with the idle state. Once a start-bit “b” is identified (detecting from high to low) it switches the states. Once it reaches state 7 it detects a valid word out for state 7. It can be seen valid word out for state 6 and state 5 by observing the screen shot of UARx start simulation in Figure 8. Thereafter, it can be observed that valid words out from state 4 until it receives all the valid 8 bits which is displayed in the screen shot of the UARx end simulation in Figure 9.

Figure 8. Screen shots of UARx start testbench output

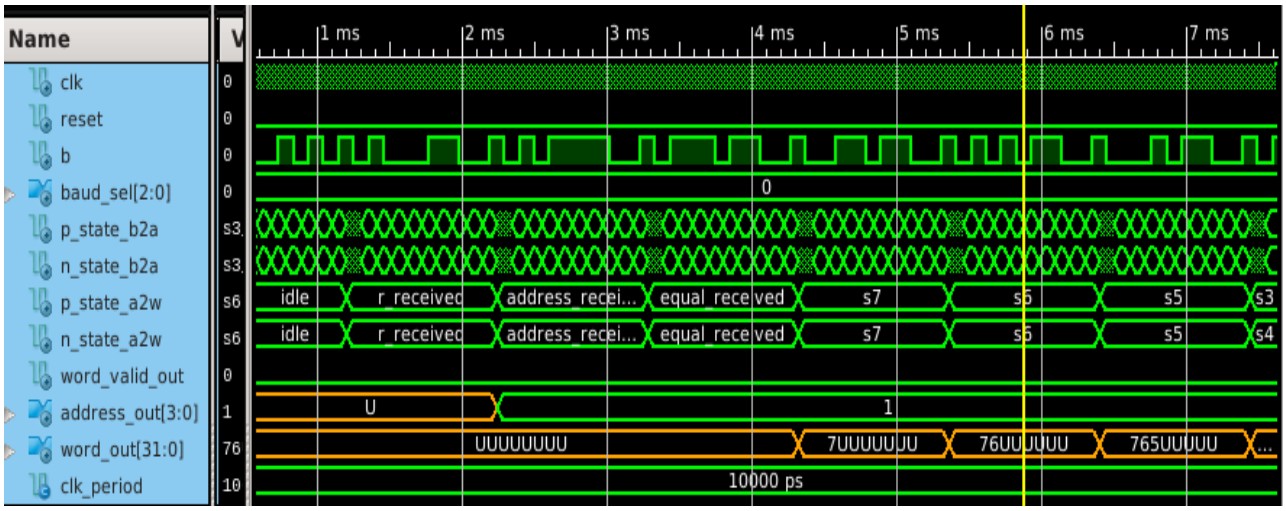
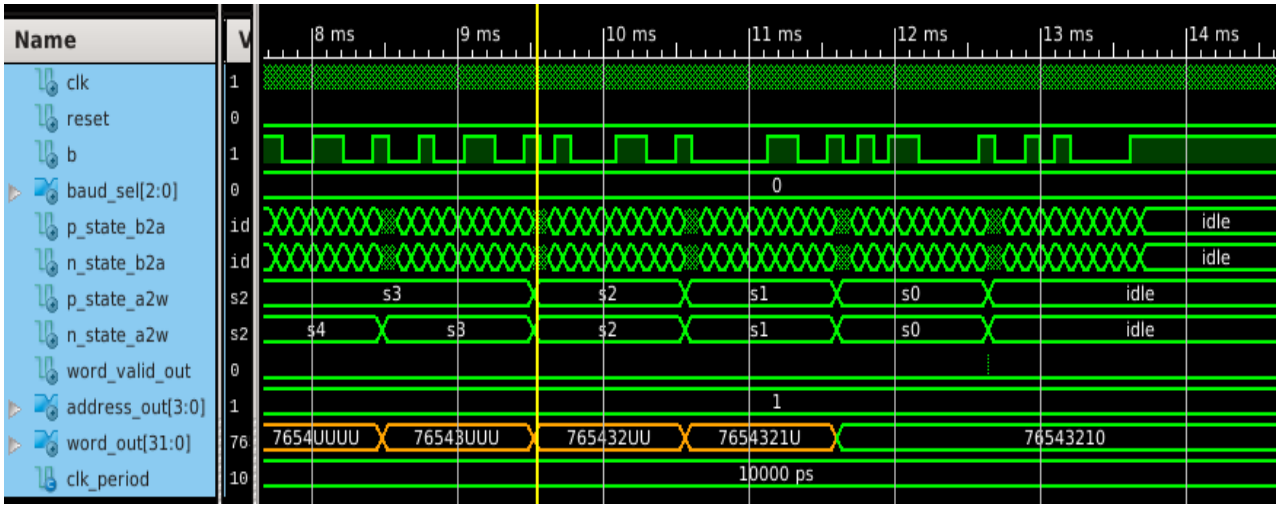


Figure 9. Screen shot of UARx end testbench



Chapter 4

Conclusion

This project uses VHDL as a design language to implement the modules of UARx receiver. UARx is a module in UART which is a microchip with programming that controls a computer's interface to its attached serial devices. The UARx is successfully implemented by using VHDL, compilation, elaboration, simulation and synthesis are performed by using Xilinx ISE 13.4 tool. The datas were transmitted serially at the maximum period of 7.41ns at clock frequency of 133.669MHz. The design has great flexibility and high integration. Especially in the field of electronic design, where SOC (System On Chip) technology has recently become increasingly mature, this design shows great significance.

Bibliography

[1] Frenzel, L. Serial I/O Interfaces Dominate Data Communications, Electronic Design, Sept 22, 2015.

[2] Norhauzaimin, J. Maimun, H.H. The Design of High Speed UART Applied Electromagnetics, Asia- Pacific Conference, pages 306-307, 2005.

[3] Agarwal, R.R. Mishra, V.R. The Design of High speed UART, Information & Communication Technologies (ICT) conference, pages 388- 390, 2013.

[4] Laddha, Neha, R. Thakere, A.P. Implementation of Serial Communication Using UART with Configurable Baud Rate, International Journal on Recent and Innovation Trends in Computing and Communication, vol: 1, issue: 4, ISSN 2321-8169, pages 263-268, Apr 2013.

[5] Bhadra, D; Vij, V.S; Stevens, K.S. A Low Power UART Design based on Asynchronous Techniques. IEEE 56th International Midwest Symposium Circuits and Systems, pages 21-24, 2013.

[6] Yongcheng Wang, Kefei Song. A New Approach to Realize UART, International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), vol: 5, Pages: 2749-2752, 2011.

[7] Khan, S. Implementation and Preference Analysis of Two Divison Algorithms. A thesis submitted in partial fulfillment of the requirement for the degree of Master of Applied Science in the Department of Electrical and Computer Engineering, University Of Victoria, Pages 1- 57, 2015.

[8] Wakhle, G.B. Aggarwal, I. Gaba, S. Synthesis and Implementation of UART using VHDL Codes Computer, Internationl symposium consumer and control(IS3C), pages 1-3, 2012.

[9] Mahat, N.F. Design of a 9-bit UART Module based on Verilog HDL. IEEE Conference Semiconducctor Electronics, pages 570-573, 2012.

Appendix

A. Bit_2_ascii module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;


entity bit_2_ascii is
    Port ( --inputs
                  clk : in  STD_LOGIC;
                reset : in  STD_LOGIC;
                baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
                b: in  STD_LOGIC;
           -- debug outputs
             p_state_b2a, n_state_b2a : out fsm_bit_ascii_state_type;
               H_out, F_out : out std_logic;
               enable_H, enable_F : out std_logic;
               clear_H, clear_F : out std_logic;
             -- outputs
           ascii_char_out : out  STD_LOGIC_VECTOR (7 downto 0);
           ascii_valid_out : out  STD_LOGIC);
       end bit_2_ascii;

architecture Behavioral of bit_2_ascii is
           signal slow_clk_internal : std_logic;
           signal H_internal : std_logic;
           signal F_internal : std_logic;
           signal clear_H_internal : std_logic;
           signal clear_F_internal : std_logic;
           signal enable_H_internal : std_logic;
           signal enable_F_internal : std_logic;
begin

--         slow_clk_instance: slow_clk port map (
--         clk => clk,
--         baud_sel => baud_sel,
--         -- debug outputs
--         count_out => OPEN,
--         -- output
--         slow_clk => slow_clk_internal
--         );

           counter_half_period_instance: counter_half_period
```

```
Port map (   -- inputs
        clk  => clk,
        baud_sel  => baud_sel,
        clear  => clear_H_internal,
        enable  => enable_H_internal,
          -- debug outputs
        count  => OPEN,
          -- output
        H  => H_internal
                        );

  counter_full_period_instance: counter_full_period
              Port map (  -- inputs
         clk  => clk,
        baud_sel  => baud_sel,
        clear  => clear_F_internal,
        enable  => enable_F_internal,
          -- debug outputs
        count  => OPEN,
          -- output
        F => F_internal
                        );

   fsm_bit_ascii_instance: fsm_bit_ascii
        Port map( --inputs
        clk => clk,
        reset => reset,
        b => b,
        H => H_internal,
        F => F_internal,
          --debug outputs
         p_state_out => p_state_b2a,
         n_state_out => n_state_b2a,
          --outputs
        ascii_valid_out => ascii_valid_out,
        ascii_char_out => ascii_char_out,
        clear_H => clear_H_internal,
         clear_F => clear_F_internal,
        enable_H => enable_H_internal,
        enable_F=> enable_F_internal
                        );

                        H_out <= H_internal;
                        F_out <= F_internal;
                        clear_H <=clear_H_internal;
                        clear_F <= clear_F_internal;
```

```
                    enable_H <= enable_H_internal;
                    enable_F <= enable_F_internal;

end Behavioral;
```

B. Half_period_counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter_half_period is
    Port (   -- input
                clk : in  STD_LOGIC;
           baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
           enable : in  STD_LOGIC;
           clear : in  STD_LOGIC;
                           -- debug outputs
           count : out  integer;
      -- output
           H : out  STD_LOGIC);
end counter_half_period;

architecture Behavioral of counter_half_period is
signal half_period_value: integer;
begin
           define_half_period: process (baud_sel,enable) is
           begin
               if enable= '1' then
                       half_period_value <= 0;
                           end if;
                   C1:case baud_sel is
                           when "000" => half_period_value <=baud_000H;
                           when "001" => half_period_value <= baud_001H;
```

```
                    when "010" => half_period_value <= baud_010H;
                    when "011" => half_period_value <= baud_011H;
                    when others => half_period_value <= baud_othersH;
                end case C1;
        end process define_half_period;

P1:     process (clk, enable,clear) is
        variable temp :integer;
        begin
     IF rising_edge(clk) THEN
       IF clear = '1' THEN
         temp := 0;
       ELSE
         IF enable = '1' THEN
           temp := temp + 1;
         END IF;
       END IF;
    END IF;
                count <= temp;
                if temp = half_period_value then
                        H <= '1';
                else
                        H <= '0';
                end if;
        end process p1;

end Behavioral;
```

C. Full_period_counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity counter_full_period is
   Port ( -- inputs
                        clk : in  STD_LOGIC;
                              baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
                              enable : in  STD_LOGIC;
                              clear: in std_logic;
                        -- debug outputs
                        count : out integer;
                        -- output
         F : out  STD_LOGIC);
end counter_full_period;

architecture Behavioral of counter_full_period is
        signal full_period_value: integer;
begin
        define_full_period: process (baud_sel) is
        begin
                C1:case baud_sel is
                        when "000" => full_period_value <= baud_000F;
                        when "001" => full_period_value <= baud_001F;
                        when "010" => full_period_value <= baud_010F;
                        when "011" => full_period_value <= baud_011F;
                  when others => full_period_value <= baud_othersF;
                end case C1;
        end process define_full_period;

        P1:     process (clk, enable,clear) is
        variable temp :integer;
        begin
     IF rising_edge(clk) THEN
       IF clear = '1' THEN
         temp := 0;
       ELSE
         IF enable = '1' THEN
           temp := temp + 1;
         END IF;
       END IF;
     END IF;
                count <= temp;
                if temp = full_period_value then
                        F <= '1';
                else
                        F <= '0';
                end if;
```

```
        end process p1;

end Behavioral;
```

D. Fsm_bit_2_ascii (Finite state machine)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fsm_bit_ascii is
    Port ( --inputs
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           b : in  STD_LOGIC;
           H : in  STD_LOGIC;
           F : in  STD_LOGIC;
                           --debug outputs
                           p_state_out, n_state_out : out fsm_bit_ascii_state_type;
                           --outputs
           ascii_valid_out : out  STD_LOGIC;
           ascii_char_out : out  STD_LOGIC_VECTOR (7 downto 0);
                           clear_H : out  STD_LOGIC;
           clear_F : out  STD_LOGIC;
                           enable_H : out  STD_LOGIC;
           enable_F : out  STD_LOGIC
                           );

end fsm_bit_ascii;

architecture Behavioral of fsm_bit_ascii is
              SIGNAL p_state: fsm_bit_ascii_state_type;
              SIGNAL n_state: fsm_bit_ascii_state_type;
begin

   P1: PROCESS (clk,reset)
          BEGIN
```

```
 IF reset ='1' THEN
  p_state<= idle;
 ELSIF rising_edge(clk)THEN
  p_state<= n_state;
 END IF;
END PROCESS P1;

p2: PROCESS(p_state,b,H,F)
BEGIN
  CASE p_state is
                when idle =>
                        p_state_out <= idle;
                        ascii_valid_out <= '0';
                        IF b='0' THEN
                                n_state<= W_1;
                                n_state_out <= W_1;
                                clear_H<= '1';
                                clear_F<= '0';
  enable_H <='0';
  enable_F <='0';
                        ELSE
                                n_state<= idle;
                                n_state_out <= idle;
                                clear_H<= '0';
                                clear_F<= '0';
  enable_H <='0';
  enable_F <='0';
                        END IF;

                when W_1 =>
                        p_state_out <= W_1;
                        ascii_valid_out <= '0';
                        n_state<= W_2; n_state_out <= W_2;
                        clear_H<= '0';
                        clear_F<= '0';
enable_H <='1';
enable_F <='0';

          WHEN W_2=>      -- when p state is "W_2"
                        p_state_out <= W_2;
                        ascii_valid_out <= '0';
                        IF H= '1' THEN
                                n_state<= S0_1; n_state_out <= S0_1;
                                clear_H<= '0';
                                clear_F<= '1';
                                enable_H <='0';
```

```
                        enable_F <='0';
                    ELSE
                        n_state<= W_2 ;         n_state_out <= W_2;
                        clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='1';
                        enable_F <='0';
                    END IF;

        WHEN S0_1=>
                    p_state_out <= S0_1;
                    ascii_valid_out <= '0';
                    n_state<= S0_2; n_state_out <= S0_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S0_2=>
                    p_state_out <= S0_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                        ascii_char_out(0) <= b;
                        n_state<= S1_1; n_state_out <= S1_1;
                        clear_H<= '0';
                        clear_F<= '1';
                        enable_H <='0';
                        enable_F <='0';
                    ELSE
                        n_state<= S0_2;         n_state_out <= S0_2;
                clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='0';
                        enable_F <='1';
                    END IF;

        WHEN S1_1=>
                    p_state_out <= S1_1;
                    ascii_valid_out <= '0';
                    n_state<= S1_2; n_state_out <= S1_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S1_2=>
```

```
                    p_state_out <= S1_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                        n_state<= S2_1; n_state_out <= S2_1;
                        ascii_char_out(1) <= b;
                        clear_H<= '0';
                        clear_F<= '1';
                        enable_H <='0';
                        enable_F <='0';
                    ELSE
                        n_state<= S1_2;        n_state_out <= S1_2;
                        clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='0';
                        enable_F <='1';
                    END IF;

        WHEN S2_1=>
                    p_state_out <= S2_1;
                    ascii_valid_out <= '0';
                    n_state<= S2_2; n_state_out <= S2_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S2_2=>
                    p_state_out <= S2_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                        n_state<= S3_1; n_state_out <= S3_1;
                        ascii_char_out(2) <= b;
                        clear_H<= '0';
                        clear_F<= '1';
                        enable_H <='0';
                        enable_F <='0';
                    ELSE
                        n_state<= S2_2;        n_state_out <= S2_2;
                        clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='0';
                        enable_F <='1';
                    END IF;

        WHEN S3_1=>
                    p_state_out <= S3_1;
```

```
                    ascii_valid_out <= '0';
                    n_state<= S3_2; n_state_out <= S3_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S3_2=>
                    p_state_out <= S3_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                            n_state<= S4_1; n_state_out <= S4_1;
                            ascii_char_out(3) <= b;
                            clear_H<= '0';
                            clear_F<= '1';
                            enable_H <='0';
                            enable_F <='0';
                    ELSE
                            n_state<= S3_2;         n_state_out <= S3_2;
                            clear_H<= '0';
                            clear_F<= '0';
                            enable_H <='0';
                            enable_F <='1';
                    END IF;


        WHEN S4_1=>
                    p_state_out <= S4_1;
                    ascii_valid_out <= '0';
                    n_state<= S4_2; n_state_out <= S4_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S4_2=>
                    p_state_out <= S4_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                            n_state<= S5_1; n_state_out <= S5_1;
                            ascii_char_out(4) <= b;
                            clear_H<= '0';
                            clear_F<= '1';
                            enable_H <='0';
                            enable_F <='0';
                    ELSE
```

```
                    n_state<= S4_2;         n_state_out <= S4_2;
                    clear_H<= '0';
                    clear_F<= '0';
                    enable_H <='0';
                    enable_F <='1';
                END IF;


        WHEN S5_1=>
                    p_state_out <= S5_1;
                    ascii_valid_out <= '0';
                    n_state<= S5_2; n_state_out <= S5_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S5_2=>
                    p_state_out <= S5_2;
                    ascii_valid_out <= '0';
                    IF F= '1' THEN
                        n_state<= S6_1; n_state_out <= S6_1;
                        ascii_char_out(5) <= b;
                        clear_H<= '0';
                        clear_F<= '1';
                        enable_H <='0';
                        enable_F <='0';
                    ELSE
                        n_state<= S5_2;         n_state_out <= S5_2;
                        clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='0';
                        enable_F <='1';
                    END IF;

        WHEN S6_1=>
                    p_state_out <= S6_1;
                    ascii_valid_out <= '0';
                    n_state<= S6_2; n_state_out <= S6_2;
                    clear_H<= '0';
                    clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S6_2=>
                    p_state_out <= S6_2;
```

```
                ascii_valid_out <= '0';
                IF F= '1' THEN
                        n_state<= S7_1; n_state_out <= S7_1;
                        ascii_char_out(6) <= b;
                        clear_H<= '0';
                        clear_F<= '1';
                        enable_H <='0';
                        enable_F <='0';
                ELSE
                        n_state<= S6_2;         n_state_out <= S6_2;
                        clear_H<= '0';
                        clear_F<= '0';
                        enable_H <='0';
                        enable_F <='1';
                END IF;

        WHEN S7_1=>
                        p_state_out <= S7_1;
                        ascii_valid_out <= '0';
                        n_state<= S7_2; n_state_out <= S7_2;
                        clear_H<= '0';
                        clear_F<= '0';
enable_H <='0';
enable_F <='1';

        WHEN S7_2=>
                        p_state_out <= S7_2;
                        ascii_valid_out <= '0';
                        IF F= '1' THEN
                                n_state<= S8_1; n_state_out <= S8_1;
                                ascii_char_out(7) <= b;
                                clear_H<= '0';
                                clear_F<= '1';
                                enable_H <='0';
                                enable_F <='0';
                        ELSE
                                n_state<= S7_2;         n_state_out <= S7_2;
                                clear_H<= '0';
                                clear_F<= '0';
                                enable_H <='0';
                                enable_F <='1';
                        END IF;

         WHEN S8_1=>
                        p_state_out <= S8_1;
                        ascii_valid_out <= '0';
```

```
                        n_state<= S8_2; n_state_out <= S8_2;
                        clear_H<= '0';
                        clear_F<= '0';
        enable_H <='0';
        enable_F <='1';

                WHEN S8_2=>
                        p_state_out <= S8_2;
                        IF F= '1' THEN
                                n_state<= idle; n_state_out <= idle;
                                clear_H<= '1';
                                clear_F<= '1';
                                enable_H <='0';
                                enable_F <='0';
                                if b = '1' then
                                        ascii_valid_out <= '1';
                                else
                                        ascii_valid_out <='0';
                                end if;
                        ELSE
                                n_state<= s8_2;         n_state_out <= S8_2;
                                ascii_valid_out <= '0';
                                clear_H<= '0';
                                clear_F<= '0';
                                enable_H <='0';
                                enable_F <='1';
                        END IF;
                END CASE;
        END PROCESS P2;
end Behavioral ;
```

E. Ascii_2_word module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;


entity ascii_2_word is
    Port ( --inputs
                clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           ascii_valid_in : in  STD_LOGIC;
```

```
        ascii_char_in : in  STD_LOGIC_VECTOR (7 downto 0);
                        --debug outputs
                        p_state_a2w, n_state_a2w : out fsm_ascii_word_state_type;
                        --outputs
        word_valid_out : out  STD_LOGIC;
        address_out : out  STD_LOGIC_VECTOR (3 downto 0);
        word_out : out  STD_LOGIC_VECTOR (31 downto 0));
end ascii_2_word;

architecture Behavioral of ascii_2_word is
         -- Declare all internal signals
         signal hex_internal: STD_LOGIC_VECTOR (3 downto 0);
         signal load_internal: STD_LOGIC_VECTOR (8 downto 0);

begin

  converter_ascii_hex_instance: converter_ascii_hex
      Port map ( -- input
         ascii_in => ascii_char_in,
          --output
        hex_out => hex_internal
                        );

  register_bank_instance : register_bank
      Port map( -- inputs
        clk => clk,
        load => load_internal,
        hex_in => hex_internal,
                        --outputs
                        word_out=> word_out,
                        address_out=> address_out
        );

 fsm_ascii_word_instance: fsm_ascii_word
     Port map( --inputs
               clk => clk,
             reset => reset,
        ascii_valid_in => ascii_valid_in,
          ascii_char_in => ascii_char_in,
                       --debug outputs
                        p_state_out => p_state_a2w,
                        n_state_out => n_state_a2w,
                        --outputs
                         load_out => load_internal,
                        word_valid_out => word_valid_out
                        );
```

```
end Behavioral;
```

F. Converter_ascii_hex

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity converter_ascii_hex is
   Port (    -- input
             ascii_in : in  STD_LOGIC_VECTOR (7 downto 0);
            --output
              hex_out : out  STD_LOGIC_VECTOR (3 downto 0)
                        );
end converter_ascii_hex;

architecture Behavioral of converter_ascii_hex is

begin
    ascii_hex: process (ascii_in) is
           begin
               if     (ascii_in = x"30") then -- ascii for '0'
                          hex_out <= x"0";
                       elsif (ascii_in = x"31") then -- ascii for '1'
                         hex_out <= x"1";
                       elsif (ascii_in = x"32") then -- ascii for '2'
                         hex_out <= x"2";
                       elsif (ascii_in = x"33") then -- ascii for '3'
                         hex_out <= x"3";
                       elsif (ascii_in = x"34") then -- ascii for '4'
                         hex_out <= x"4";
                       elsif (ascii_in = x"35") then -- ascii for '5'
                         hex_out <= x"5";
                       elsif (ascii_in = x"36") then -- ascii for '6'
                         hex_out <= x"6";
                       elsif (ascii_in = x"37") then -- ascii for '7'
                         hex_out <= x"7";
```

```
                        elsif (ascii_in = x"38") then -- ascii for '8'
                          hex_out <= x"8";
                        elsif (ascii_in = x"39") then -- ascii for '9'
                          hex_out <= x"9";
                        elsif (ascii_in = x"41" or ascii_in = x"61") then
                          hex_out <= x"A";
                        elsif (ascii_in = x"42" or ascii_in = x"62") then
                          hex_out <= x"B";
                        elsif (ascii_in = x"43" or ascii_in = x"63") then
                          hex_out <= x"C";
                        elsif (ascii_in = x"44" or ascii_in = x"64") then
                          hex_out <= x"D";
                       elsif (ascii_in = x"45" or ascii_in = x"65") then
                          hex_out <= x"E";
                       elsif (ascii_in = x"46" or ascii_in = x"66") then
                          hex_out <= x"F";
            else
                    hex_out  <= x"0";
                  end if;
                end process;


end Behavioral;
```

G. Register bank

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity register_bank is
    Port ( -- inputs
                            clk : in  STD_LOGIC;
          load : in  STD_LOGIC_VECTOR (8 downto 0);
          hex_in : in  STD_LOGIC_VECTOR (3 downto 0);
                            --outputs
                            word_out: out STD_LOGIC_VECTOR (31 downto 0);
                            address_out: out STD_LOGIC_VECTOR (3 downto 0));
```

```
end register_bank;

architecture Behavioral of register_bank is
        COMPONENT hex_register is
   Port ( -- inputs
                        clk : in  STD_LOGIC;
        load : in  STD_LOGIC;
        hex_in : in  STD_LOGIC_VECTOR (3 downto 0);
                        --outputs
        hex_out : out  STD_LOGIC_VECTOR (3 downto 0));

        END COMPONENT hex_register;
begin
 Component_0: hex_register  Port MAP(
        load => load(0),
        hex_in=> hex_in ,
        hex_out => word_out(3 downto 0),
                        clk => clk);

Component_1: hex_register Port MAP(
        load => load(1),
        hex_in=> hex_in,
        hex_out=> word_out(7 downto 4),
                        clk=> clk);

Component_2: hex_register Port MAP(
        load => load(2),
        hex_in=> hex_in,
        hex_out=> word_out(11 downto 8),
                        clk=> clk);

Component_3: hex_register Port MAP(
        load => load(3),
        hex_in=> hex_in,
        hex_out=> word_out(15 downto 12),
                        clk=> clk);

Component_4: hex_register Port MAP(
        load => load(4),
        hex_in=> hex_in,
        hex_out=> word_out(19 downto 16),
                        clk=> clk);

Component_5: hex_register Port MAP(
        load => load(5),
        hex_in=> hex_in,
```

```
        hex_out=> word_out(23 downto 20),
                                clk=> clk);

Component_6: hex_register Port MAP(
        load => load(6),
        hex_in=> hex_in,
        hex_out=> word_out(27 downto 24),
                                clk=> clk);

Component_7: hex_register Port MAP(
        load => load(7),
        hex_in=> hex_in,
        hex_out=> word_out(31 downto 28),
                                clk=> clk);

Component_8: hex_register Port MAP(
        load => load(8),
        hex_in=> hex_in,
        hex_out=> address_out,
                                clk=> clk);
end Behavioral;
```

H. Fsm_Ascii_word (Finite state Machine)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fsm_ascii_word is
   Port ( --inputs
               clk : in  STD_LOGIC;
                           reset : in  STD_LOGIC;
         ascii_valid_in: in  STD_LOGIC;
                           ascii_char_in : in  STD_LOGIC_VECTOR (7 downto 0);
                           --debug outputs
                           p_state_out, n_state_out : out fsm_ascii_word_state_type;
```

```
                        --outputs
        load_out : out  STD_LOGIC_VECTOR (8 downto 0);
                        word_valid_out : out  STD_LOGIC
                        );

end fsm_ascii_word;

architecture Behavioral of fsm_ascii_word is
      SIGNAL p_state : fsm_ascii_word_state_type;
                SIGNAL n_state : fsm_ascii_word_state_type;
begin

    P1: PROCESS (clk,reset)
          BEGIN
           IF reset ='1' THEN
            p_state<= idle;
           ELSIF rising_edge(clk)THEN
            p_state<= n_state;
           END IF;
           END PROCESS P1;

    p2: process(p_state,ascii_valid_in,ascii_char_in) is
                variable var1,v_0_9,v_lc_a_z, v_uc_A_Z,v_uc_A_F,v_lc_a_f : boolean;
    BEGIN
                 var1 := ascii_valid_in = '1';
                      v_0_9 := (ascii_char_in >= X"30") and (ascii_char_in <= X"39"); -- check
0 <= char <= 9
                      v_uc_A_Z := (ascii_char_in >= X"41") and (ascii_char_in <= X"5A"); --
check A <= char <= Z
                      v_lc_a_z := (ascii_char_in >= X"61") and (ascii_char_in <= X"7A"); --
check a <= char <= z
                      v_uc_A_F := (ascii_char_in >= X"41") and (ascii_char_in <= X"46"); --
check A <= char <= F
                      v_lc_a_f := (ascii_char_in >= X"61") and (ascii_char_in <= X"66"); --
check a <= char <= f

                CASE p_state is
                 when idle => -- looking for "r" or "R"
                        p_state_out <= idle;
                            word_valid_out <= '0';
                            load_out <= O"000";
                           IF (var1 and ((ascii_char_in = X"72") or (ascii_char_in = X"52")))
THEN -- "r" or "R"
                                n_state <= R_received;
                                n_state_out <= R_received;
                           ELSE
```

```
                        n_state<= idle;
                        n_state_out<= idle;
                    END IF;

when R_received =>  -- state that "R" or "r" has been received. Looking for Reg address.
                    p_state_out <= R_received;
                    word_valid_out <= '0';
                    if var1 then
                        IF (v_0_9 or v_lc_a_z or v_uc_A_Z) THEN -- valid address
                            n_state <= address_received;
                            n_state_out <= address_received;
                            load_out <= O"400"; -- load_8 = '1': 100 000 000
                        else
                            n_state<= idle;
                            n_state_out<= idle;
                            load_out <= O"000";
                        end if;
                    ELSE
                        n_state<= R_received;
                        n_state_out<= R_received;
                        load_out <= O"000";
                END IF;

        WHEN address_received=> -- state that address has been received
                p_state_out<= address_received;
                    word_valid_out <= '0';
                    load_out <= O"000";
                    if var1 then
                            IF (ascii_char_in = X"20") THEN -- "20" is ASCII space
                                n_state <= space1_received;
                                n_state_out <= space1_received;
                            elsif (ascii_char_in = X"3D") then  -- "3D" is ASCII =
                                n_state <= equal_received;
                                n_state_out <= equal_received;
                            else
                                n_state<= idle;
                                n_state_out<= idle;
                                load_out <= O"000";
                            end if;
                ELSE
                  n_state<= address_received;
                        n_state_out<= address_received;
                END IF;

               WHEN space1_received =>
                  p_state_out<= space1_received;
```

```
word_valid_out <= '0';
            load_out <= O"000";
            if var1 then IF (ascii_char_in= X"3D") THEN -- "3D" is ASCII =
       n_state <= equal_received;
                  n_state_out <= equal_received;
                                 else
                        n_state<= idle;
                        n_state_out<= idle;
                                 END IF;
            ELSE
          n_state<= space1_received;
                n_state_out<= space1_received;
       END IF;


    WHEN equal_received =>
        p_state_out<= equal_received;
 word_valid_out <= '0';
             if var1 then
               IF (ascii_char_in = X"20") THEN  -- "20" is ASCII space
       n_state <= space2_received;
                  n_state_out <= space2_received;
                        load_out <= O"000";
            elsif (v_0_9 or v_lc_a_f or v_uc_A_F) THEN  -- correct bit 7 received
                                 n_state <= s7;
                                 n_state_out <= s7;
                  load_out <= O"200"; -- load_7 = '1'
                           ELSE
             n_state<= idle;
                  n_state_out<= idle;
                  load_out <= O"000";
          END IF;
               ELSE
          n_state<= equal_received;
                n_state_out<= equal_received;
                load_out <= O"000";
        END IF;

    WHEN space2_received =>
       p_state_out<= space2_received;
word_valid_out <= '0';
             if var1 then
               IF (v_0_9 or v_lc_a_f or v_uc_A_F) THEN
      n_state <= s7;
                  n_state_out <=s7;
                        load_out <= O"200"; -- load_7 = '1'
```

```
                    else
                        n_state<= idle;
                        n_state_out<= idle;
                    load_out <= O"000";
                END IF;
                    ELSE
                n_state<= space2_received;
                    n_state_out<= space2_received;
                    load_out <= O"000";
            END IF;


WHEN S7=>           -- when p state is "S7"
   p_state_out <= S7;
   word_valid_out <= '0';
                    if var1 then
                      IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
            n_state <= s6;
                      n_state_out <= s6;
                      load_out <= O"100"; -- load_6 = '1'
                ELSE
                 n_state<= idle;
                      n_state_out<= idle;
                      load_out <= O"000";
                END IF;
                        ELSE
                n_state<= S7;
                    n_state_out<= S7;
                    load_out <= O"000";
            END IF;



WHEN S6=>           -- when p state is "S6"
   p_state_out <= S6;
   word_valid_out <= '0';
                                if var1 then
                                    IF (v_0_9 or v_lc_a_f or v_uc_A_F) THEN
            n_state <= s5;
                      n_state_out <= s5;
                      load_out <= O"040"; -- load_5 = '1'
                                    ELSE
                n_state<= idle;
                      n_state_out<= idle;
                    load_out <= O"000";
                                    END IF;
```

```
                    ELSE
                n_state<= S6;
                    n_state_out<= S6;
                    load_out <= O"000";
              END IF;

WHEN S5=>           -- when p state is "S5"
   p_state_out <= S5;
   word_valid_out <= '0';
                                if var1 then
                                    IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
            n_state <= s4;
                    n_state_out <= s4;
                    load_out <= O"020";
                ELSE
                   n_state<= idle;
                        n_state_out<= idle;
                        load_out <= O"000";
                END IF;
                                ELSE
                n_state<= S5;
                    n_state_out<= S5;
                    load_out <= O"000";
              END IF;


WHEN S4=>           -- when p state is "S4"
   p_state_out <= S3;
   word_valid_out <= '0';
                        if var1 then
                                IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
            n_state <= s3;
                    n_state_out <= s3;
                    load_out <= O"010";
                ELSE
                  n_state<= idle;
                        n_state_out<= idle;
                        load_out <= O"000";
                END IF;
                        ELSE
                n_state<= S4;
                    n_state_out<= S4;
                    load_out <= O"000";
              END IF;

WHEN S3=>           -- when p state is "S3"
```

```
   p_state_out <= S3;
   word_valid_out <= '0';
                        if var1 then
                          IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
               n_state <= s2;
                           n_state_out <= s2;
                           load_out <= O"004";
                    ELSE
                     n_state<= idle;
                           n_state_out<= idle;
                           load_out <= O"000";
                  END IF;
  ELSE
                     n_state<= S3;
                           n_state_out<= S3;
                           load_out <= O"000";
                  END IF;

WHEN S2=>             -- when p state is "S2"
   p_state_out <= S2;
   word_valid_out <= '0';
                         if var1 then
                          IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
                n_state <= s1;
                            n_state_out <= s1;
                            load_out <= O"002";
                     ELSE
                        n_state<= idle;
                              n_state_out<= idle;
                             load_out <= O"000";
                     END IF;
                                 ELSE
                     n_state<= S2;
                           n_state_out<= S2;
                           load_out <= O"000";
                  END IF;


               WHEN S1=>              -- when p state is "S1"
   p_state_out <= S1;
   word_valid_out <= '0';
                         if var1 then
                          IF(v_0_9 or v_lc_a_f or v_uc_A_F) THEN
                 n_state <= s0;
                            n_state_out <= s0;
                            load_out <=O"001";
```

```
                    ELSE
                       n_state<= idle;
                             n_state_out<= idle;
                            load_out <= O"000";
                    END IF;
        ELSE
                    n_state<= S1;
                          n_state_out<= S1;
                          load_out <= O"000";
                  END IF;

               WHEN S0=>            -- when p state is "S0"
        p_state_out <= S0;
                          if var1 then
                                                IF ((ascii_char_in = X"0A") or (ascii_char_in =
X"0D")) THEN
                                                      n_state <=idle;
                                                      n_state_out <= idle;
              word_valid_out <= '1';
                          load_out <= O"000";
                                                ELSE
                                                      n_state <=S0;
                                                      n_state_out <= S0;
                                                      word_valid_out <= '0';
                                                      load_out <= O"000";
                                                END IF;
                                          ELSE
                                                n_state<= S0;
                                                n_state_out<= S0;
                                                word_valid_out <= '0';
                             load_out <= O"000";
                    END IF;
               END CASE;
    end process p2;
end Behavioral;
```

I. UA_RX receiver

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.uart_package.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
```

```
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UA_RX is
    Port ( -- inputs
              clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           b : in  STD_LOGIC;
           baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
                               -- debug outputs
          p_state_b2a, n_state_b2a : out fsm_bit_ascii_state_type;
          p_state_a2w, n_state_a2w : out fsm_ascii_word_state_type;
                    --H_out, F_out : out std_logic;
           ----enable_H, enable_F : out std_logic;
           --clear_H, clear_F : out std_logic;
                               -- outputs
           word_valid_out : out  STD_LOGIC;
           address_out : out  STD_LOGIC_VECTOR (3 downto 0);
           word_out : out  STD_LOGIC_VECTOR (31 downto 0));
end UA_RX;

architecture Behavioral of UA_RX is


           signal ascii_char_internal:   STD_LOGIC_VECTOR (7 downto 0);
           signal ascii_valid_internal:   STD_LOGIC;

begin

  bit_2_ascii_instance: bit_2_ascii PORT MAP (
          clk => clk,
          reset => reset,
          baud_sel => baud_sel,
          b => b,
                                  p_state_b2a => p_state_b2a,
                                  n_state_b2a => n_state_b2a,
                                  H_out => open, --H_out,
                                  F_out => open, --F_out,
                                  clear_H => open, --clear_H,
                                  clear_F => open, --clear_F,
                                  enable_H => open, --enable_H,
                                  enable_F => open, --enable_F,
```

```
        ascii_char_out => ascii_char_internal,
        ascii_valid_out => ascii_valid_internal
      );


  ascii_2_word_instance: ascii_2_word PORT MAP (
        clk => clk,
        reset => reset,
        ascii_valid_in => ascii_valid_internal,
        ascii_char_in => ascii_char_internal,
        p_state_a2w =>  p_state_a2w,
        n_state_a2w => n_state_a2w,
        word_valid_out => word_valid_out,
        address_out => address_out,
        word_out => word_out
      );

end Behavioral;
```

J. UA_RX Package

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package uart_package is

        constant baud_000F          : integer := 650*16; -- 10,400 full period count,
corresponds to 4,800 baud rate
        constant baud_001F          : integer := 325*16; --  5,200, full period count corresponds
to 9,600 baud rate
        constant baud_010F          : integer := 162*16; -- 2,600, full period count corresponds
to 19,200 baud rate
        constant baud_011F          : integer := 80*16; -- 1,300, full period count corresponds
to 38,400 baud rate
        constant baud_othersF       : integer := 20;

        constant baud_000H          : integer := 650*8;
        constant baud_001H          : integer := 325*8;
        constant baud_010H          : integer := 162*8;
        constant baud_011H          : integer := 80*8;
        constant baud_othersH       : integer := 10;

        TYPE fsm_ascii_word_state_type IS (
        idle, R_received, address_received, space1_received,
        equal_received,space2_received,s7,s6,s5,s4,s3,s2,s1,s0
```

```
            );


TYPE fsm_bit_ascii_state_type IS (
            idle,W_1,W_2,S0_1,S0_2,S1_1,S1_2,S2_1,S2_2,S3_1,
            S3_2,S4_1,S4_2,S5_1,S5_2,S6_1,S6_2,S7_1,S7_2,S8_1,S8_2
            );
      -- Declare our components



            Component counter_half_period is
                  Port (   -- input
        clk : in  STD_LOGIC;
    baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
    enable : in  STD_LOGIC;
    clear : in  STD_LOGIC;
                     -- debug outputs
    count : out  integer;
                      -- output
    H : out  STD_LOGIC);
            end component counter_half_period;

            component counter_full_period is
                  Port ( -- inputs
                    clk : in  STD_LOGIC;
                          baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
                          enable : in  STD_LOGIC;
                          clear: in std_logic;
                     -- debug outputs
                     count : out integer;
                     -- output
    F : out  STD_LOGIC);
            end component counter_full_period;

            component fsm_bit_ascii is
                  Port (  --inputs
        clk : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    b : in  STD_LOGIC;
    H : in  STD_LOGIC;
    F : in  STD_LOGIC;
                     --debug outputs
                     p_state_out, n_state_out : out fsm_bit_ascii_state_type;
                     --outputs
    ascii_valid_out : out  STD_LOGIC;
```

```
      ascii_char_out : out  STD_LOGIC_VECTOR (7 downto 0);
                        clear_H : out  STD_LOGIC;
      clear_F : out  STD_LOGIC;
                        enable_H : out  STD_LOGIC;
      enable_F : out  STD_LOGIC
                        );
               end component fsm_bit_ascii;

               component converter_ascii_hex is
      Port ( -- input
                              ascii_in : in  STD_LOGIC_VECTOR (7 downto 0);
                         --debug outputs
                         --output
      hex_out : out  STD_LOGIC_VECTOR (3 downto 0)
                        );
   end component converter_ascii_hex;

               component register_bank is
      Port ( -- inputs
                        clk : in  STD_LOGIC;
      load : in  STD_LOGIC_VECTOR (8 downto 0);
      hex_in : in  STD_LOGIC_VECTOR (3 downto 0);
                        --outputs
                        word_out: out STD_LOGIC_VECTOR (31 downto 0);
                        address_out: out STD_LOGIC_VECTOR (3 downto 0));
     end component register_bank;

               component fsm_ascii_word is
      Port ( --inputs
            clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
      ascii_valid_in: in  STD_LOGIC;
                        ascii_char_in : in  STD_LOGIC_VECTOR (7 downto 0);
                        --debug outputs
                        p_state_out, n_state_out : out fsm_ascii_word_state_type;
                        --outputs
      load_out : out  STD_LOGIC_VECTOR (8 downto 0);
                        word_valid_out : out  STD_LOGIC
                        );

     end component fsm_ascii_word;

COMPONENT bit_2_ascii
 PORT( --inputs
               clk : in  STD_LOGIC;
      reset : in  STD_LOGIC;
```

```
      baud_sel : in  STD_LOGIC_VECTOR (2 downto 0);
                          b: in  STD_LOGIC;
                                -- debug outputs
                          p_state_b2a, n_state_b2a : out fsm_bit_ascii_state_type;
                                H_out, F_out : out std_logic;
                                enable_H, enable_F : out std_logic;
                                clear_H, clear_F : out std_logic;
                          -- outputs
      ascii_char_out : out  STD_LOGIC_VECTOR (7 downto 0);
      ascii_valid_out : out  STD_LOGIC
    );
END COMPONENT;

COMPONENT ascii_2_word
PORT(
    clk : IN  std_logic;
    reset : IN  std_logic;
    ascii_valid_in : IN  std_logic;
    ascii_char_in : IN  std_logic_vector(7 downto 0);
    p_state_a2w : OUT  fsm_ascii_word_state_type;
    n_state_a2w : OUT  fsm_ascii_word_state_type;
    word_valid_out : OUT  std_logic;
    address_out : OUT  std_logic_vector(3 downto 0);
    word_out : OUT  std_logic_vector(31 downto 0)
   );
END COMPONENT;
end uart_package;
```