K-Line Communication

List of Authors (Adam Jesse, Trey Plichta, Jacob Alam, Ruger Stellberger)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: ajesse@oakland.edu, treyplichta@oakland.edu, jalam2@oakland.edu, rstellberger@oakland.edu

***Abstract*—The purpose of this document is to describe the process of achieving K-line communication between the Nexys A7 FPGA and a 9141 OBD ECU simulator board designed by Ozen Elektronik. The design, methodology, setup, results and conclusions will be drafted along with supporting visuals of components and software diagrams.**
keywords - FSM, ECU, LED

I. INTRODUCTION

This project uses the Nexys A7 FPGA board and the ISO 9141-2 OBD ECU simulator board. The purpose is to initialize a communication session between the two boards and then request and display select data from the ECU simulator on Nexys Board. The ECU simulator provides signals that will be requested and received using the Nexys A7 board. A simple high level block diagram is provided below that covers the 4 circuits of this project and how they are intertwined.

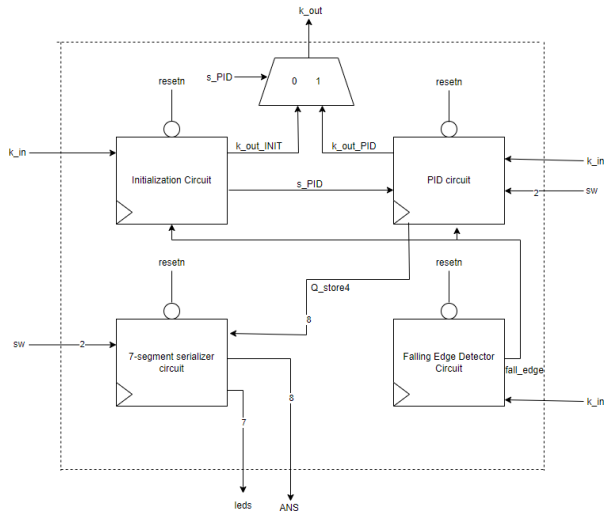


Figure 1. Digital Circuit Overview

II. METHODOLOGY

A. Initialization Process

The first step to allow the Nexys A7 FPGA to communicate with the ECU simulator board is to initialize a communication session. This is done through serial communication at a baud rate of 5. A single wire communication protocol, K-line communication, is used. This process requires a series of steps that include sending specific packets at different bit speeds. Once a session is initialized, a green LED will turn on indicating a successful session has been initialized.

According to the document [1], to begin the initialization process, the simulation board requires an address of 33 at precisely 5 bits per second. The total time needed to transmit this message will last roughly two seconds. The simulator will then respond with a synchronization byte of 55 to inform the user of the new baud rate of 10,400 bits per second. The simulator will then wait between 5ms to 20ms for the user to set the new baud rate with the Nexys A7 and follow up with two key bytes. After the key bytes have been sent, the user will then send an inverted byte number 2 after waiting for 25 to 50ms. After this is completed, the simulation board will then invert the address value 33 and send it back to the user as a ready to communicate signal.

The requirements to initialize the emulator board via the Nexys require a 3.3V to 12V amplifier circuit, a 12V to 3.3V voltage divider circuit, a finite state machine, and several software components implemented via the FPGA board. First, the 12V to 3.3V down converter was used to set the voltage levels of the K-line to acceptable GPIO voltage levels for the Nexys A7. The Nexys A7 will accept a maximum of 3.3V through GPIO with any levels exceeding this value to cause damage to the FPGA board.

The next requirements for the emulator initialization process are the components programmed in vhdl and implemented on the Nexys A7. The components used in the initialization process include: shift registers, registers, a falling edge detector, counters, comparator, and a finite state machine. The purpose of these components are as follows: the shift registers were used to output data from the Nexys, registers were used in this design to hold and store data when necessary, the falling edge detector was used to detect the start bits of incoming messages, counters were used to determine the waiting period to send data packets and set varying baud rates, a comparator to verify the transmitted bits are correct,and the state machine was used as a control circuit to complete the necessary requirements in correct sequence.

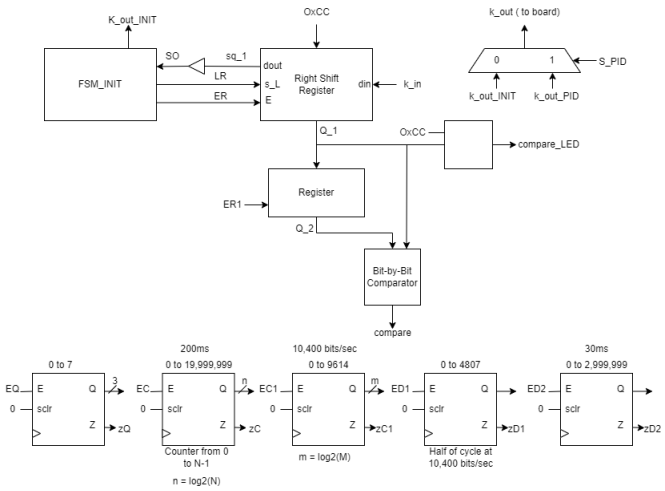


Figure 2. Initialization Circuit Diagram

The finite state machine is by far the most complex component used by the Nexys A7 during the initialization process. This component contained 19 states to send data packets at their required time interval and determine what values to transmit to the emulator. Since the FSM acted as a control circuit, it determined when to activate and utilize the other components detailed in the initialization process. The FSM was in direct control of the counters and the right shift register. The counters were used to time the output of data, allowing the FSM to control the output of data packets. The FSM also utilized the shift register to shift data bits out of the Nexys and to the Emulator board. Due to the size of the FSM, an ASM diagram is provided in Appendix I.

The 19 states created in the finite state machine allow for the initialization steps to occur in proper sequence. State one waits for the button on the Nexys to be enabled to start the upcoming sequence of events. State two waits for the user to hit a button to start the initialization process. It also loads 0xCC into the shift register before moving to state three. Note that this 0xCC will eventually be inverted before being shifted out. State three signals the start bit of the data packet and signals the counter component to count to 200ms. After this time has passed, the FSM reaches state four where it will shift the message in the timing diagram with each bit lasting 200 ms.

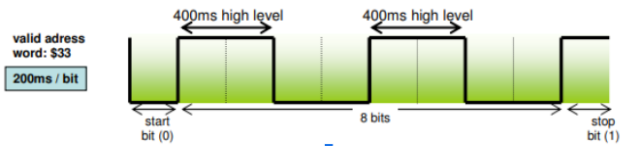


Figure 3. Message Timing Diagram

FSM state 5 sends out a high stop bit for 200 ms. Next, the FSM waits for a response from the ECU simulator board. The response is detected with the falling edge detector in state 6. When reading data from the ECU simulator board, it is desired to read a steady value from the middle of the transmission time. Therefore, state 7 uses a counter to delay to the middle of the start bit. State 8 is where the data byte is shifted into the right-shift register. Every time zC1 is high, which means the respective counter has reached 9614, a data bit is shifted into the register. This happens until zQ hits a maximum value of 7, so this process sees a byte shifted into the right shift register. State 9 sends a stop bit out onto the k-line. According to the ISO 9141-2 5 Baud Initialization Process, the ECU simulator will need to send three bytes to the Nexsy board; 0x55, 0x08, 0x08. The last two bytes are the same. Therefore, instead of using additional states, a comparator was used to detect the last two bytes being the same before the process could proceed. State 10 sees this comparator in action with the FSM returning to state 6 to wait for another falling edge and shift in another data byte when the data in the shift register and storage register are unequal. Once 0x08 is successfully compared to 0x08, the FSM proceeds to state 11 which is a delay of 30 ms achieved through a 0 to 2,999,999 counter. State 12 sees the Nexys board sending a start bit out onto the k-line. According to the ISO 9141-2 5 Baud Initialization Process, we must now send the 0x08 back to the board, but inverted. Therefore, the 0x08 that is still stored in the right shift register is inverted and sent back out onto the k-line at 10.4 kbps using a 0 to 9614 counter (duration of each bit) and a 0 to 7 counter (number of bits) in state 13. State 14 sends a stop bit out onto the k-line. Finally, the ECU simulator sends one final message to the Nexys board. This occurs in states 15 to 18 which are a direct mirror of states 6 to 9. At this point, the Nexys has reached state 19 in the FSM Init component and is ready to continue on the next process of reading data using PIDs. It informs the system that the initialization process is complete by setting signal s_PID to 1. The FSM rests in this state until resetn is activated. Additionally, the emulator is expected to activate the LED signifying the activation confirmation.

A few things to note in the initialization process. The first message sent from Nexys to ECU simulator in states 3 to 5 is the only message sent at a 5 baud rate rate. Every other message whether sent from Nexys to ECU simulator or vice versa is sent at 10.4 kbps. Additionally, the k-line idles high. Therefore, all delay and rest states see the k-line set to 1.

B. Reading Data

To retrieve data from the ECU simulator board, a request message must be transmitted, which must follow the specifications outlined in the ISO 9141-2 standard. This request message consists of header bytes that contain the format byte, the target byte, and the source byte, followed by a data length code byte that specifies the number of data bytes to be sent. The subsequent data bytes can be up to a maximum of 7, and they contain the PID of the signal. Lastly, a checksum byte is sent to ensure data integrity.

Once the ECU simulator receives a correct request message, it will send a response that contains specific information about the requested signal. For instance, if a PID indicating ECT is sent, the response data byte will contain the hexadecimal value of the engine coolant temperature. In this project, the ECU simulator is constantly requested for values of ECT, VSS, O2, and Time since

power up, which are then displayed on the 7 segment display of the Nexys A7 board. The user can select which reading to view using switches 1 and 2 on the board - 00 for ECT, 01 for VSS, 10 for O2, and 11 for time since startup.

To implement this process, the circuit diagram in Figure 3 depicts the components involved. A multiplexer is used to load the appropriate shift register with preloaded request messages. The selection of this multiplexer is controlled by a counter that increments each cycle of the FSM. The request message is shifted out one bit at a time at a baud rate of 10,400. Once the request is sent and a leading 0 is detected, the FSM reads the response from the simulator by shifting each bit into a register. The data byte of the response is obtained from this register, and its hexadecimal value corresponds to the actual value of the signal. This value is then passed to the 7 segment serializer and displayed.

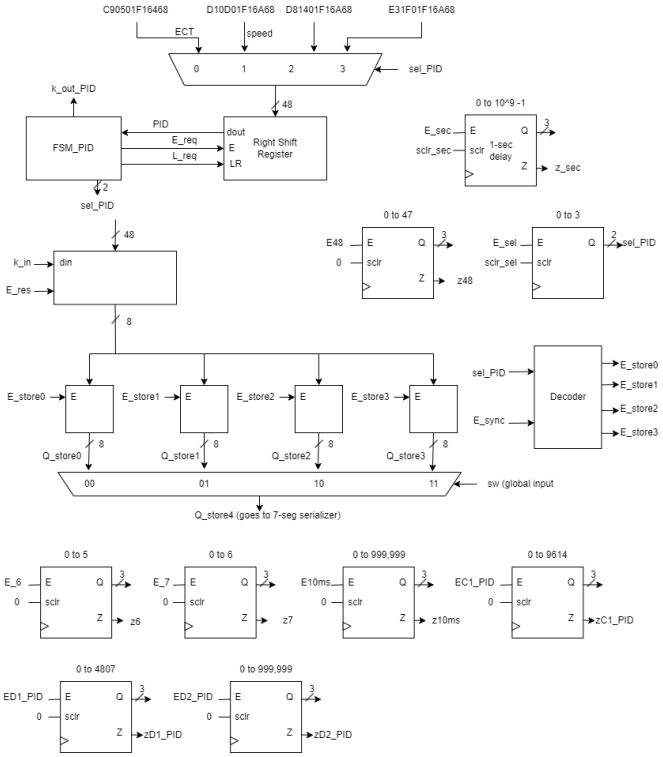


Figure 3. Reading Data Circuit Diagram

The FSM_PID, shown in Figure 4, controls the entire circuit. This is how the request is sent at the correct baud rate and how the response is received. This FSM had 10 states that repeat about every half a second. The first state would wait for a signal that came from the initialization FSM that would only become high after a successful initialization sequence was completed. States 2 through 5 were responsible for sending the start and stop bit along with each individual byte. Once all 6 bytes were sent, states 6 and 7 waited for the falling edge indicating that the ECU simulator had sent its start bit. Then it is delayed to the middle of this start bit. State 8 is where the reading process begins and the response is shifted into the register. Depending on the sel_PID signal, the data byte is passed to 1 of the 4 different registers and the rest is ignored because it is not needed. Once here, the data will be passed to the 7 segment serializer to be displayed.

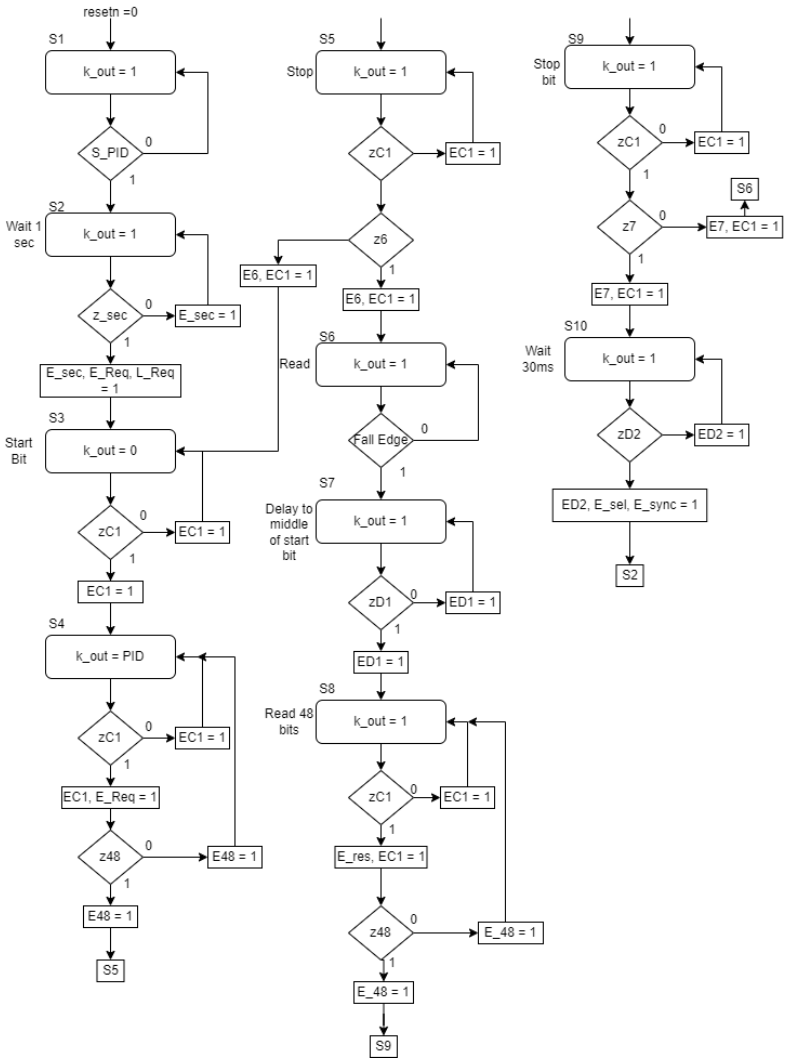


Figure 4. Reading Data Control Circuit

C. 7-Segment Serializer

In figure 5, the FSM and attached components are used to show all of the signals on separate 7-segment displays to the human eye simultaneously by cycling through every 1ms(0.5ms may be necessary). The state diagram for the aforementioned FSM is shown in figure 6. The FSM increases its state every time E is 1, which as the diagram shows, will occur every 0.001s (1ms). Hence, ‘s’ will increase every 1ms, thus switching what data value to fetch from the multiplexor each millisecond. The ‘s’ selector also selects which 7-segment display to turn on at a given moment since only one display can be utilized at a time because of hardware limitations. Hence why the 1ms delay is present to let the board switch displays constantly so it is only utilizing one display at a time while simultaneously seeming as though the image is still to the human eye thanks to the low delay. Note: These components were not a focal point of the project. Therefore, VHDL code and diagrams were taken directly from notes and online resource website [2].

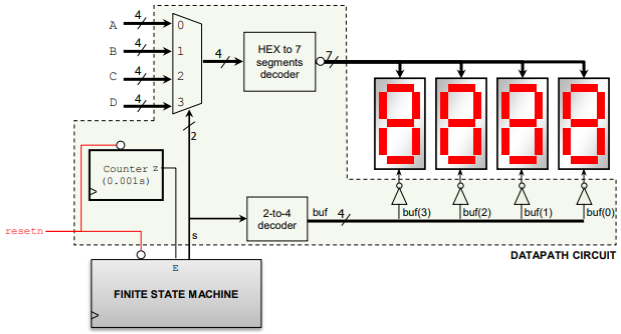


Figure 5. 7-Segment Serializer Circuit [2]

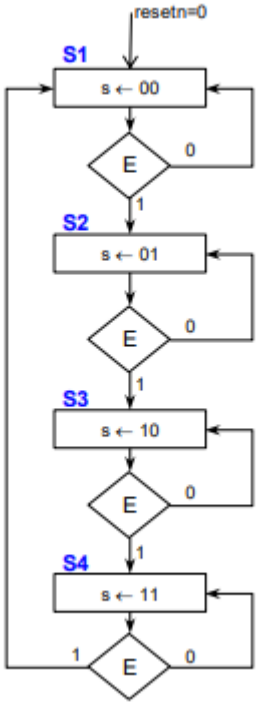


Figure 6. 7-Segment Serializer FSM [2]

D. Falling Edge Detector

Both the Initialization FSM and the FSM responsible for reading data utilized a common falling edge detector circuit in order to begin reading data from the emulator board. After receiving an alert from the Falling Edge Detector circuits, these control FSMs could proceed to the next state. The FSM responsible for detecting the falling edge of input from the k-line is shown in figure 7.

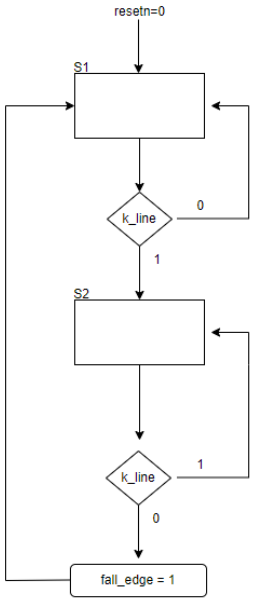


Figure 7. Falling Edge Detector FSM

III. Experimental Setup

The setup necessary to set the connection between the Nexys and the ISO 9141 OBD ECU simulator board required a series of several components. The project required the utilization of Pmod headers on the Nexys A7 FPGA, the ODB connector terminal of the emulator board, a 12V to 3.3V down converter, and an OBD connector. The Nexys Pmod headers JA 1 and 2 were utilized as K line connectors in and out respectively. The OBD terminal pins used in this system were the K line pin 7, 12V pin 12 and GND pin 5. The final component needed in the system setup is a 12V to 3.3V down/step converter. The Nexys requires an input of 3.3V for communication while the emulator board needs 12V. To allow the boards to communicate, a regulator is needed to control the power level input into each board. The Nexys was then powered using the micro usb terminal utilizing a 5V input and the emulator required a 12V power supply. Once the setup is complete, the system is ready for use.

IV. Results

After much debugging, a functioning project was observed. Links to videos of that project are given below.

Video DEMO #1

Video DEMO #2

One problem with the outcome of the project is the

tediousness of the setup. Our FSMs were designed to only handle expected outcomes. Therefore, if any data was corrupted, the FSMs will find themselves stuck in a state and the ECU simulator will time out with 5 seconds of inactivity on the k-line. We found that corrupted data did often exist due to the use of physical circuitry in our amplifier/voltage divider circuit. Any movement to these components could result in the failure of communication. Therefore, our circuit was observed to have communication sessions as short as 20 seconds and as long as 3 minutes when, in theory, they should have lasted forever.

Another challenge of the entire project was the debugging process. A testbench was used to ensure the expected behavior of the initialization process. However, since the first 10 bits were sent at a 5 baud rate, this process took over 2 seconds and the behavioral simulation took over 10 minutes to run. A waveform is not included in this report as it would be unreadable without a zoom function.

This project often turned from a design, testing, and validation process to a research project. Finding data on ISO 9141-2 5 Baud Init proved to be a challenging task. Some examples of roadblocks encountered include start and stop bits being sent between each data byte when sending/receiving requests with no delay besides these two bits; setting the k-line high during any idle state of an FSM; and configuring the correct way to send data out. These were all tricky tasks as information on the web on ISO 9141-2 5 baud INIT is limited and the group had limited automotive communication knowledge before the project. Overall, the group found the standard to be very "UARTish", a topic covered in large detail during the course.

Conclusions

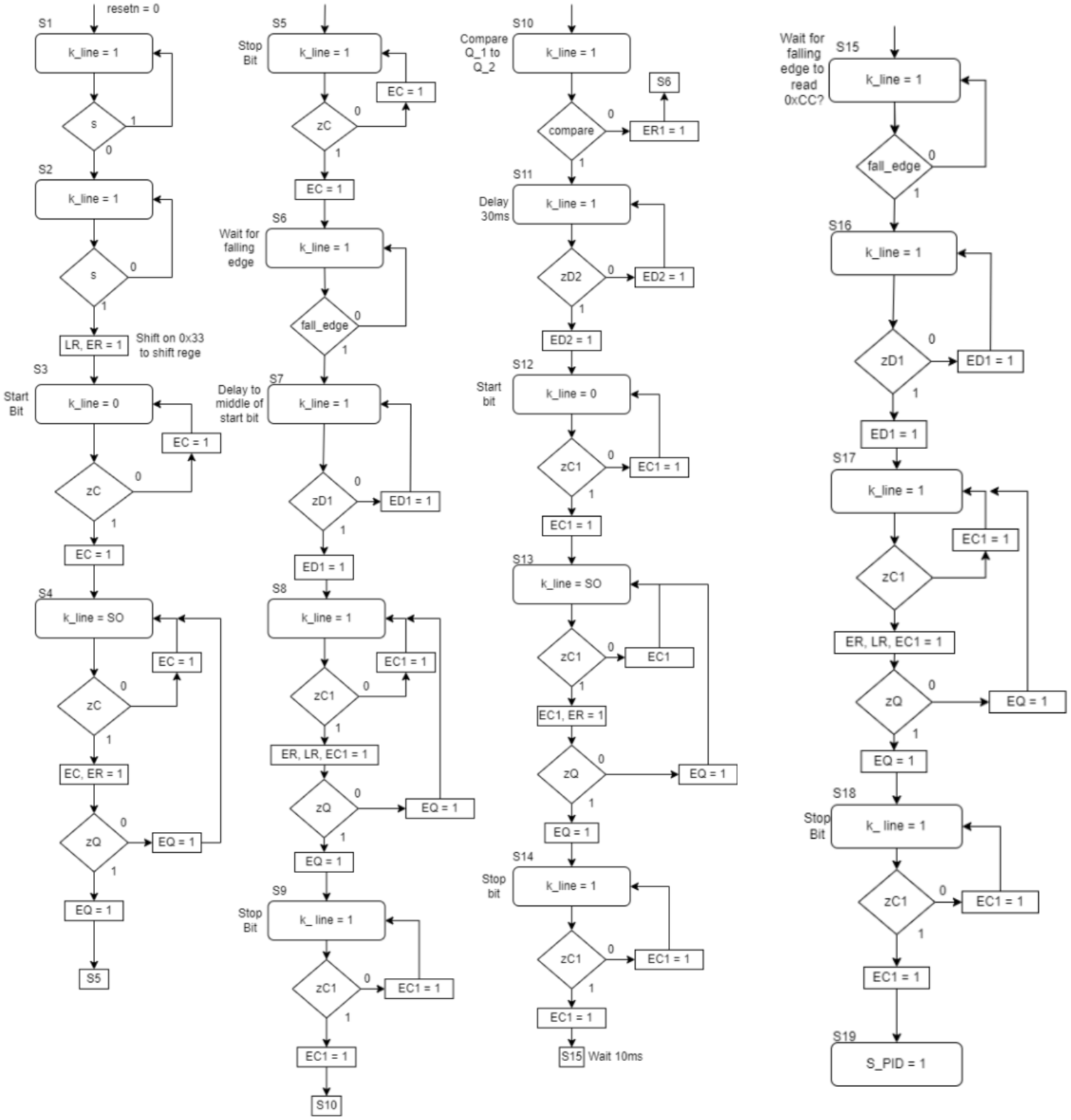
In conclusion, this project aimed to initialize a communication session between the Nexys A7 FPGA board and the ISO 9141 OBD ECU simulator board, and to request and display select data from the ECU simulator on the Nexys board. This was achieved through serial communication at a baud rate of 5 using a single wire communication protocol, K-line communication. The project required a 3.3V to 12V amplifier and 12V to 3.3V divider circuit, a finite state machine, and several software components implemented via the FPGA board. The finite state machine was the most complex component and contained 19 states to send data packets at their required time interval and determine what values to transmit to the emulator. The project successfully implemented the 5 Baud Initialization Process as described in the ISO 9141-2 standard, and allowed for data to be requested and displayed on the Nexys board. Overall, the project achieved its objectives and successfully demonstrated the use of the Nexys A7 and ISO 9141 OBD ECU simulator boards for automotive applications.

References

[1] B. Gruszczynski, "K-line Communication Description," Nov. 2009. Accessed: Apr. 12, 2023. [Online]. Available: https://www.obdclearinghouse.com/Files/viewFile?fileID=1380

[2] Llamocca, Daniel. VHDL Coding for Fpgas, Oakland University. https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.

APPENDIX I (INIT FSM)



APPENDIX II (Reading FSM)