Designing an RD53B Trigger Pattern Encoder for the YARR Readout System

by

Lucas Lopes Cendes

Supervised by Scott Hauck

A senior thesis submitted in partial fulfillment of

the requirements for the degree of

Bachelor of Science
With Departmental Honors

Paul G. Allen School of Computer Science & Engineering
University of Washington

June 2021

Presentation of work given on June 1st 2021

Thesis and presentation approved by 

Date _6/3/21_____

Abstract

The Large Hadron Collider (LHC) is currently undergoing the High-Luminosity (HL-LHC) upgrade to increase the number of collisions per unit of time in the collider. This update will require a new set of readout chips to be developed for the ATLAS detector used in the LHC. As a result of this, the RD53B readout chip architecture was created. These chips are designed to collect the data produced by the collisions that occur in the Hl-LHC. The data collection is initiated when the RD53B chips receive a trigger command word that specifies the specific set of data that should be sampled. These commands are produced by a readout system that converts a series of trigger pulses into the proper command encodings. One of these systems is the Yet Another Rapid Readout (YARR) data acquisition system. This system is composed of both a software and a hardware component. The YARR hardware is responsible for sending the properly encoded trigger commands to the readout chip and sending the collision data produced by the chip to the YARR software. This requires the YARR firmware to be loaded into an FPGA board that is connected to a host computer running the YARR software. The YARR firmware is currently being updated to support the command protocol used by the RD53B chip architecture. The project described in this thesis mainly focuses on designing the logic needed to produce the trigger commands used by the RD53B chips. This functionality is accomplished by adding a trigger code generator and a trigger extender module to the YARR firmware. The trigger code generator is mainly responsible for gathering 4-bit trigger patterns and converting them into the proper 16-bit trigger command words. The trigger extender is mainly responsible for extending the duration of a trigger pulse. To test these modules, a testbench that simulates a series of trigger pulses was designed. After that, the modified YARR firmware was tested on a real RD53B chip by running a YARR software scan.

Contents

Chapter 1

Background

1.1 Large Hadron Collider

The Large Hadron Collider (LHC) is currently the most powerful and largest particle accelerator in the world. It is maintained by the European Organization for Nuclear Research (CERN), and it is located on the border between Switzerland and France. The LHC mainly consists of a 27-kilometer ring composed of superconducting magnets. The particle collisions are produced by having two high-energy particle beams travel in opposite directions inside the accelerator [6]. These beams are composed of several intense bunches of protons and, when the two beams cross, the bunches on those beams collide. This event is known as a bunch crossing, and it occurs at a rate of 40 MHz. When a bunch crossing occurs, there is a chance that the individual protons contained in a given bunch will collide with the protons present at another bunch [13].

One of the main experiments at the LHC is ATLAS. It is a general-purpose particle physics experiment that intends to test the predictions of the Standard Model of particle physics [5]. In order to perform this experiment, the ATLAS detector was created. The device is a multi-layered instrument designed to detect the smallest and most energetic physical particles created by the collisions in the LHC. The innermost layer of the ATLAS detector is known as the Inner Detector, which is composed of three different components, with one of those being the Pixel Detector [4]. This device includes 1700 identical pixel modules, with each one of those modules containing an array of over 40 000 pixels. Each module is connected to the 16 front-end (FE) chips present in the Pixel Detector. These FE chips are responsible for processing the data produced by each one of the pixels [10].

The LHC is currently undergoing the High-Luminosity (HL-LHC) upgrade. This upgrade intends to increase the luminosity of the LHC by a factor of over ten times its current value, which will increase the number of particle collisions per unit of time. This will ultimately increase the amount of data gathered by the LHC experiments, thus enabling them to observe rare physical processes [3]. Since the HL-LHC will operate at a luminosity that is up to seven times higher than what the ATLAS detector was designed for, this upgrade will require new detectors to be developed and installed [1]. As a result of this, the RD53

Collaboration was established in 2013 with the intent of designing a new generation of FE chips for the pixel tracker. These efforts culminated in the creation of the RD53A prototype chip and its successor, the RD53B chip [2]. The latter chip will be discussed in more detail in the next section.

1.2 RD53B

The first readout chip produced by the RD53 Collaboration was the RD53A. This chip achieves all of the required functionality needed by the pixel upgrades that will be installed on the ATLAS detector. The RD53A, however, was never intended for production as it was simply designed to demonstrate the technology needed to produce readout chips for the ATLAS experiment. As a result of this, the RD53 Collaboration started working on the development of the RD53B chip architecture with the intent of developing a chip design that is suitable for production. The RD53B design builds upon the architecture of the RD53A with several technological advances that facilitate the integration of the RD53B into the Pixel Detector [2].

The availability of the physical RD53B is currently very limited, and making changes to the physical architecture of the chip is a complicated process. Because of that, the ACME laboratory at the University of Washington has developed an RD53B emulator that will facilitate the process of modifying and testing the architecture of the chip. Although the RD53B chip contains both physical and digital components, the emulator only simulates the digital components of the chip [11].

The two main components of the RD53B emulator are the TTC data processing block and the Command Processor. These blocks are mainly responsible for receiving and decoding the data sent through the serial input port of the emulator and operate in accordance with the RD53B command protocol, which is explained in detail in the next section [11].

1.2.1 RD53B Command Protocol

RD53B commands are sent as a continuous stream of serial data with a bitrate of 160 Mbps. Each command is composed of 16-bit frames that consist of two 8-bit symbols. Commands can span multiple frames, but this thesis will only focus on single-frame commands. Each 16-bit frame is DC balanced, which means that a single frame contains the same number of 0's and 1's, which ultimately results in every frame having a total of eight 0's and eight 1's. In addition, no command words start or end with more than two repeated bits. This ensures that, in most situations, the signal received from the data input port of the RD53B chip is set to the same value for at most four consecutive clock cycles. These properties guarantee that the input signal is constantly changing, which facilitates the process of recovering the clock signal used to send the data. This encoding also enables error detection since a single bit flip will result in an imbalance in the number of 0's and 1's [7].

There are three types of commands that are relevant for this project. One of them is the PLL_LOCK command, which ensures that the data input circuitry is locked to the correct 160 MHz clock frequency. This command is composed of an alternating series of 0's and

1's that mimics a clock signal. This command must be continuously sent to the TTC data processing block at the start of operation until a stable chip clock signal is produced. The PLL_LOCK command also functions as an idle pattern that is continuously sent to the TTC data processing block when no commands are being transmitted. Because of this, the PLL_Lock command will be referred to as the idle command from now on. The other relevant short command is the sync command. This command must be continuously sent at the start of operation of the TTC data processing unit for it to determine the proper frame boundaries. The input signal of the RD53B chip is held at the same value for exactly six clock cycles while sync commands are being sent, which means that this is the only situation in which the input signal is set to the same value for more than four consecutive clock cycles. This bit pattern cannot be produced through any combination of command words, which makes this command easily identifiable by the TTC data processing block [7]. The encoding of the sync and PLL_LOCK commands are shown in table 1.

Command	Binary Encoding	Hexadecimal Encoding
PLL_LOCK	1010_1010 1010_1010	0xAAAA
Sync	1000_0001 0111_1110	0x817E

Table 1: Binary and hexadecimal encoding of the PLL_LOCK and sync commands [7].

The third type of command that will be discussed in this section consists of trigger commands. These command words are composed of an 8-bit trigger encoding and an 8-bit tag encoding. The trigger encoding is used to determine if the chip should sample the data from a given bunch crossing. Since bunch crossings occur at a rate of 40 MHz and commands are transmitted at a bitrate of 160 Mbps, each command encoding represents a sequence of 4 consecutive bunch crossings, which means that a total of 16 clock cycles are needed to produce the command words. Since command words are 16-bit long and each bit is transmitted individually, the encoding used for these command words ensures that it also takes 16 clock cycles to transmit a single command word. This constraint guarantees that the transmission of command words is always aligned to the 40 MHz bunch crossing clock. The sequence of four bunch crossing that a command word refers to can be thought of as a 4-bit trigger pattern in which the presence of a trigger indicates that the chip should sample the data produced by the respective bunch crossing. Each one of those trigger patterns is mapped to an 8-bit trigger encoding [7]. The exact encoding for each pattern is shown in table 2.

Trigger Pattern	Binary encoding	Hexadecimal Encoding
000T	0010_1011	0x2B
00T0	0010_1101	0x2D
00TT	0010_1110	0x2E
0T00	0011_0011	0x33
0T0T	0011_0101	0x35
0TT0	0011_0110	0x36
0TTT	0011_1001	0x39
T000	0011_1010	0x3A
T00T	0011_1100	0x3C
T0T0	0100_1011	0x4B
T0TT	0100_1101	0x4D
TT00	0100_1110	0x4E
TT0T	0101_0011	0x52
TTT0	0101_0101	0x55
TTTT	0101_0110	0x56

Table 2: Encoding for each trigger pattern. For each trigger pattern, a T represents the presence of a trigger and a 0 represents the absence of a trigger [7].

The 8-bit tag encoding in each trigger command is used to identify the specific bunch crossings that correspond to a given trigger command. The data sampled from a bunch crossing is then associated with its respective tag. There are a total of 54 8-bit tag encoding that can be used in a trigger command. Each tag encoding is associated with a 6-bit tag base. This 6-bit value is used as an index into the rows of a master trigger table. Each trigger word received by the RD53B chip is placed at the row of the trigger master table that corresponds to the 6-bit tag base associated with the command [7]. The exact mappings for each one of those tags are shown in table A of appendix A.

Since all data sent to the RD53B chip must be properly encoded, the serial data input port of the chip must be connected to a readout system that converts the raw trigger pattern into the proper encoding. One of these readout systems is YARR, which will be explained in detail in the next section.

1.3 YARR

The Yet Another Rapid Readout (YARR) system is a data acquisition system designed for the pixel readout chips used in the Pixel Detector. The overall architecture of YARR is discussed in detail at [8]. The system is composed of both a software and a hardware part that work together to convert a trigger pattern into the proper data encoding. YARR also analyzes the data that is outputted by the chip. The YARR system receives a set of trigger patterns and then uses them to send the properly encoded command words to the readout chip. The data that is produced by the readout chip is then sent back to the YARR system. Most of the data processing is done through software, which makes YARR less dependent on the hardware being used to host it, making it more portable across different hardware platforms [8] [14]. Figure 1 shows a diagram of the YARR system.

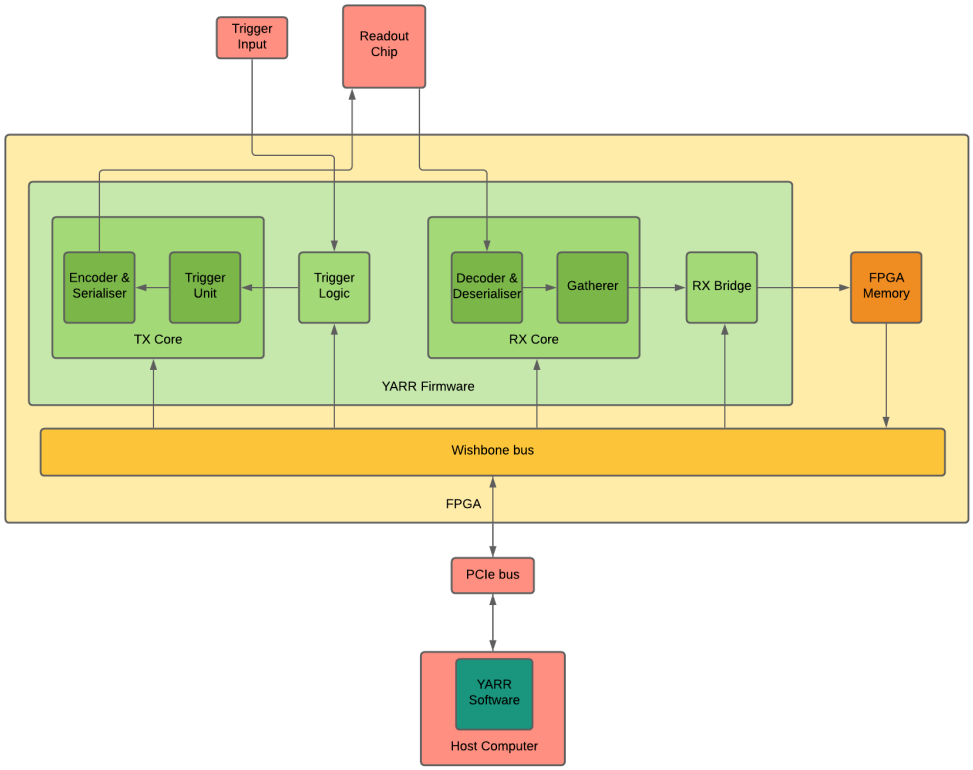

Figure 1: Diagram of the YARR system. Based on the diagram found at [8]

The YARR software is responsible for configuring the YARR hardware and performing most of the data analysis. Data is collected through the usage of scan procedures that accomplish a number of different actions. Firstly, the registers of the readout chip and the YARR hardware are configured by the YARR software. The YARR hardware then reads the data associated with each pixel in the readout chip. This data is then analyzed and processed by the YARR software [8] [14].

The YARR hardware layer mainly consists of a firmware loaded into an FPGA that is connected to a host computer through its PCIe port. The data sent through the PCIe port is then translated into the format used by the Wishbone bus used in the firmware. The Wishbone bus will be explained in more detail in section 1.3.1.1. The raw trigger pulse signal received by the firmware is sent to the TX Core, with the latter being responsible for encoding it and transmitting the encoded command words through a serial port. The data received from the readout chip is sent to the RX Core, which is responsible for sending the data to the host computer [8].

The YARR hardware firmware is currently being updated to support the RD53B chip architecture. The project work discussed in this thesis focuses on developing a suitable trigger code generator that can convert a series of trigger pulses into the 16-bit command words expected by the RD53B command protocol. This trigger code generator will be placed on the TX Core of the firmware and will depend on the existing functionality of that block. The overall architecture of the TX Core and all of its relevant components will be explained in the next section.

1.3.1 YARR Firmware TX Core

Currently, the TX core is mainly composed of a trigger unit and a TX channel. The TX core hosts a series of registers that can be used to configure its functionality. These registers can be configured through a Wishbone bus, with each register being associated with a specific Wishbone address. An external trigger pulse can be sent to the TX Core through one of its input ports [9]. Each relevant component of the TX core will be discussed in detail in the following sections.

1.3.1.1 Wishbone interface

The configuration registers of the TX core can be read and written to using the Wishbone interface. More information about how the Wishbone interface works can be found at [12]. In order to write to a register, a couple of Wishbone input ports must be configured. Firstly the Wishbone address port must be set to the address of the respective register. The Wishbone data port must then be set to the data that should be written to the register. Finally, the Wishbone write enable, strobe, and cycle signals must be asserted. When the write operation is completed, the Wishbone acknowledgment signal is asserted. The Wishbone address values used for the TX Core registers relevant to this project are shown in table 3 [9].

Address in hexadecimal	Register Name	Description
0x01	Command Enable	If set to 1, any command words received by the TX Channel will be outputted through its serial port. Otherwise, only sync and idle commands will be outputted.
0x03	Trigger Enable	If set to 1, the trigger unit will output trigger pulses. Otherwise, no trigger pulses are outputted.
0x05	Trigger Configuration	Used to configure the trigger unit. A value of 0 corresponds to internal mode, 1 corresponds to internal time mode and 2 corresponds to internal count mode. This is discussed in more detail in the trigger unit section.
0x18	TX Polarity	If set to 1, the bits transmitted by the output port of the TX channel are inverted.
0x20	Sync Interval	Sets the interval at which sync commands are sent. By default, this register is set to a value of 16, which means that a sync command will be sent after every 16 command frames.

Table 3: Wishbone address and description of each register used used in this project [9]

Additional Wishbone addresses were added to support the additional registers needed by

the new modules developed in this project. This is discussed in more detail in chapter 2.

1.3.1.2 Trigger Unit

The trigger unit is responsible for generating the trigger pulses that will be processed by the trigger code generator module developed in this project. If set to external mode, the external trigger signal inputted into the TX core will be used to generate trigger pulses. If that is the case, the trigger unit will wait for a rising edge in the external trigger pulse signal. When that happens, the trigger unit will wait a total of four clock cycles, and then it will output a trigger pulse through its output port. It will then set a dead time in which all external trigger pulses are ignored, and no trigger pulses are outputted. When this dead time elapses, the trigger unit will wait for another rising edge in its external trigger pulse input port [9].

If the trigger unit is set to any of the two other modes, the trigger unit will output trigger pulses at regular intervals. If that is the case, the module will extract a trigger frequency value from one of its input ports, and a counter will be initialized to 0. When the value of the counter is equal to the trigger frequency value inputted into the module, a trigger pulse will be generated, and the counter will be reset to 0 [9].

1.3.1.3 TX Channel

The TX channel is responsible for outputting the command words that are produced in the TX core. Any data sent to the TX channel must be 32-bit wide. When a command word is ready, it is sent to a priority encoder. When that happens, the command word with the highest priority is serialized and outputted through the serial port of the TX core. Sync command words have the second-lowest priority and are outputted at regular intervals. Idle command words have the lowest priority and are outputted whenever there are no other command words to transmit. If the TX channel is disabled, all command words other than sync and idle words are ignored [9]. A diagram of the priority encoder can be seen at figure 2.

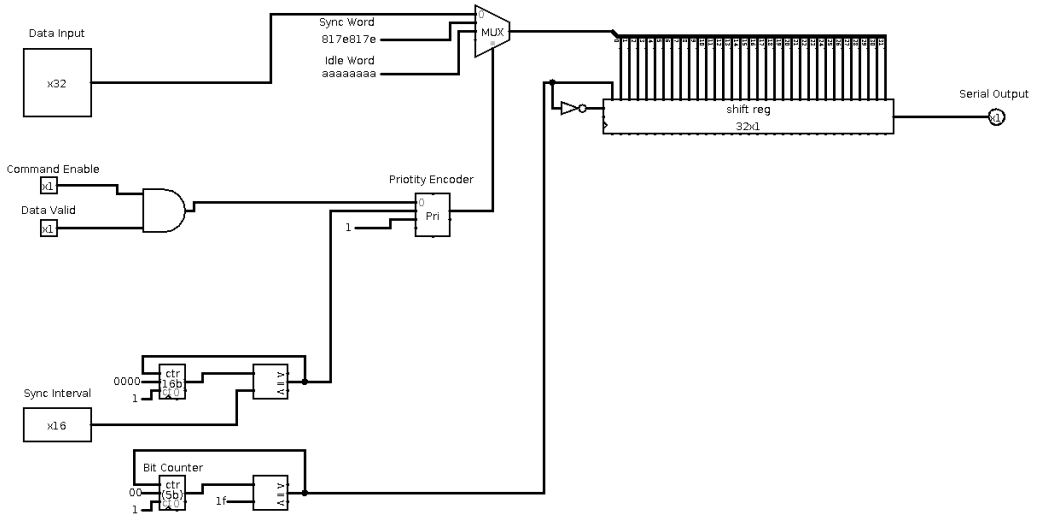

Figure 2: Circuit diagram of the priority encoder at the TX Channel [9].

Chapter 2

Design

In this project, two new modules were added to the TX core of the YARR firmware. One of them is the trigger code generator, which is responsible for converting 4-bit trigger patterns into 16-bit command words. The other module developed in this project is the trigger extender, which extends a trigger pulse for a given number of clock cycles. The following two sections will give a thorough description of the architecture of each one of those modules. The last section will explain the changes made to the existing YARR firmware code to ensure that these new modules are fully integrated into the TX core.

2.1 Trigger Code Generator

The trigger code generator module is responsible for converting a series of trigger pulses into a trigger command word. Each 16-bit command word is composed of an 8-bit trigger encoding and an 8-bit tag. The trigger encoding is produced by converting a sequence of 4 trigger bits into a DC balanced encoding. The trigger bits are received in a serial fashion, and the 4-bit binary value represented by those bits is mapped to a specific encoding. In order to produce the tags, the module keeps track of a 6-bit counter that is incremented every time a new trigger word is produced. The value of this counter is then converted into the appropriate DC balanced tag encoding.

A total of two 16-bit command words are outputted every 32 clock cycles. If no triggers are received during that interval, the command word output port is set to the idle command word, and the command word ready port is set to 0 until the next 32 clock cycle interval. If only one command word is produced during that interval, one of the 16-bit words in the command word output port is set to the idle pattern. The trigger code generator module can be enabled or disabled by the YARR software at any time.

A diagram of the trigger code generator can be seen in figure 3. The following sections will go over each of the essential components of the trigger code generator module in detail.

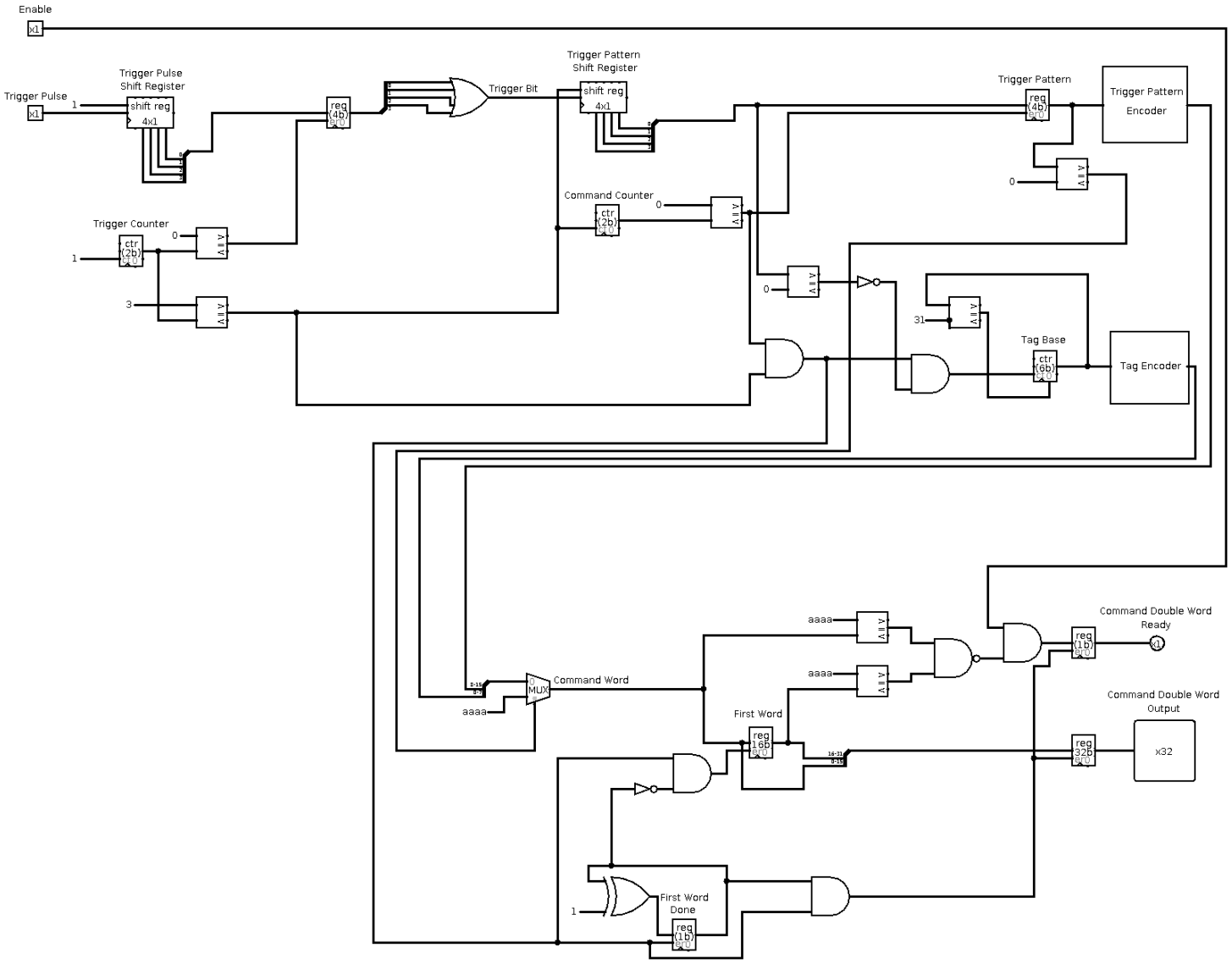

Figure 3: Circuit diagram of the trigger code generator

2.1.1 Trigger Pulse Processing

Since the bunch crossings occur at a 40 MHz frequency and the trigger code generated module uses a 160 Mhz clock, each trigger pulse will take up a total of 4 clock cycles. A trigger is detected if the trigger input port is set to 1 at any of the four clock cycles within that interval. Otherwise, if the value of the port is set to 0 throughout the entire interval, no triggers are detected.

In order to accomplish this functionality, the module keeps track of a 2-bit trigger counter and a 4-bit shift register. At each clock cycle, the current value of the trigger input port is shifted into the shift register and the counter is incremented. When the value of the trigger counter rolls back to 0, the module will set the value of the next trigger bit to the result of ORing all bits in the shift register.

2.1.2 Trigger Command Generation

In order to keep track of each 4-bit trigger pattern received, the module keeps track of a 2-bit command counter, a 4-bit trigger shift register, and a 4-bit trigger pattern register. Whenever the trigger counter described in the previous section is set to 3, a new trigger bit is shifted into the trigger shift register, and the command counter is incremented. When the command counter rolls back to 0, the trigger pattern register is set to the current value of the trigger shift register. The 4-bit trigger pattern stored in the trigger pattern register is then converted into an 8-bit DC balanced trigger encoding. The encodings for each respective trigger pattern are shown in table 2

2.1.3 Tag Generation

Tags are generated by converting a 6-bit tag base into an 8-bit DC balanced tag encoding according to table A of appending A. The tag base is represented by a 6-bit counter that is incremented whenever a new command word is produced. The maximum possible base tag value is 49, and the counter rolls back to 0 whenever it is incremented past that value.

2.1.4 Command Word Transmission

New code words are produced in 16 clock cycle intervals. A code production interval ends whenever both the trigger and command counters are set to 0. Whenever that happens, the module will update the value of the register that keeps track of the last command word. In order to do that, the module will first check if any trigger pulses were received during the last code production interval. If no triggers were detected, the register is set to the idle command word. Otherwise, the higher-order bits of the register are set to the last 8-bit trigger encoding produced, and the lower-order bits are set to the respective 8-bit tag encoding.

In order to conform with the 32-bit width of the serial port in the TX channel module, the trigger code generator must output two 16-bit command words at the same time. Because of that, the output of the trigger code generator is only updated after two new command words have been produced. This, therefore, means that new command word pairs are outputted in 32 clock cycle intervals, which will be defined as a code output interval.

To ensure that the code words are transmitted properly, a number of steps need to be taken. Whenever a new 16-bit trigger command word is produced, we must determine if the given command word is the first one produced in the current code output interval. In order to do that, the module keeps track of a first word done flag and 16-bit first word register. If the flag is set to 0, this means that the current command word is the first one produced during the current code output interval. If that is the case, the first word register is set to the current command word, the first word done flag is set to 1, and the output ports of the module remain unchanged. When the next command word is produced, the module will check the first word done flag again. Since the flag is set to 1, the module will start taking the necessary steps needed to update the output ports of the module. Firstly, the higher-order bits of the 32-bit code output port are set to the value of the first word

register and the lower-order bits of the port are set to the last command word produced. The module will then check the value of the two command words produced during the current code output interval. If both command words are set to the idle command word, this means that no command words should be transmitted during the current code output interval. This, therefore, means that the module will set the command word ready output port to 0. Otherwise, if at least one of the two command words is not an idle word, the module will check if it is enabled. If the trigger code generator module is enabled, the command word ready output port is set to 1, and the newly produced command word pair is transmitted to the TX channel. Otherwise, the command word ready signal is set to 0 and the command words are not transmitted.

2.2 Trigger Extender

The trigger extender module allows the YARR software to extend the duration of a trigger pulse for a given number of clock cycles. If the extension interval input port is set to 0, the module will work as a pass-through that outputs whatever trigger pulse it receives. Otherwise, if the extension interval is set to a non-zero value, the module will check if its pulse input port is set to 1. If that is the case, it will initialize a cycle counter to the value sent through the extension interval port. The counter is then decremented at each clock cycle, and the module will set its pulse output port to 1 until the counter reaches 0. When that happens, the counter will stop being decremented and it will go back to its pass-through behavior until it receives a new pulse. Changes to the extension interval input port will only have an effect after a new pulse is received and cycle counter has reached 0. The output pulse signal of the trigger extender is connected to the trigger code generator. A diagram of the trigger extender can be seen in figure 4.

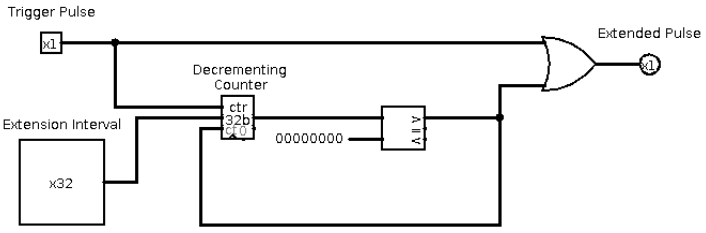

Figure 4: Circuit diagram of the trigger extender

2.3 Modifications to Existing Code

2.3.1 TX Core

Two additional entries were added to the address map of the Wishbone bus at the TX core. These entries are shown in table 4.

Address in hexadecimal	Register Name	Description
0x20	Trigger Extension Interval	The duration of trigger pulses are entended according to the value of this register. For example, if this register is set to 2, any triggers issued by the trigger unit will have their duration extended by 2 clock cycles.
0x21	Trigger Code Generator Enable	If set to 1, command words produced by the trigger code generator are transmitted through the serial port. Otherwise, no trigger commands are transmitted.

Table 4: Wishbone address and description of the registers added to the TX core.

2.3.2 TX Channel

The outputs of the trigger code generator were connected to the TX channel and were mapped to the highest priority entry in the priority encoder. The serial port is set to transmit the 32-bit code double word outputted by the trigger code generator whenever the trigger code ready signal is asserted.

Chapter 3

Testing

Two separate tests were done to verify the operation of the two modules designed in this project. The first one of these tests was a simulation of both the TX core of the YARR firmware and the RD53B emulator. This was accomplished through the development of a VHDL testbench. The other tests involved running a YARR scan with the modified YARR firmware loaded in an FPGA. A real RD53B chip was connected to the FPGA board used in this test. These tests are described in more detail in the following sections.

3.1 Simulation Testbench Design

A VHDL testbench was used to simulate the functionality of the trigger code generator and the trigger extender. The testbench instantiates both the TX core module and the top-level module of the RD53B emulator. The testbench simulates a series of trigger pulses and trigger extender interval configurations. In order to test that code words produced by the trigger code generator are being correctly outputted by the serial port of the TX core, the data output port of the TX core is connected to the data input port of the RD53B emulator. The following sections describe the functionality of the testbench in more detail.

3.1.1 Trigger Pattern Generation

A 5-bit pulse synchronization counter is used to synchronize the trigger pulses generated by the testbench with the code output intervals of the trigger code generator module. The counter is initialized to a value of 11 and it is incremented at every clock cycle. This ensures that the counter will have a value of 0 at the beginning of a code output interval.

The trigger pattern generator procedure takes an arbitrarily sized binary string and generates the appropriate series of trigger pulses with the correct timing. When the procedure is called, it will first wait until the pulse synchronization counter has a value of 0. When that happens, the procedure will start iterating through the binary string. For each bit of the string, the procedure will firstly set the trigger pulse signal to the value of the current bit for one clock cycle. Then, in the next clock, the value of the trigger pulse signal is set to 0.

The procedure will then wait for 3 clock cycles before moving on to the next bit.

3.1.2 Wishbone Interface Configuration

A few TX core registers have to be configured before the pulses are generated. Firstly, the trigger configuration register has to be set to the external mode to allow the testbench to generate the trigger pulses and send them to the TX core. After that, both the trigger unit and the trigger code generator are enabled by setting the respective registers to a value of 1. The TX polarity is then set to 0 to ensure that the TX core outputs the data correctly.

After those preliminary steps are done, the command protocol between the YARR firmware and the RD53B emulator must be initialized. To do that, the YARR firmware must keep sending a series of sync frames until the RD53B emulator locks to the correct channel. To accomplish this, the testbench will firstly set the sync interval register of the YARR firmware to 0. The testbench will then wait for a total of 300 clock cycles. After this time has elapsed, it will set the sync interval register back to its default value of 16. This results in the TX core constantly sending sync signals throughout the 300 clock cycles in which the sync interval register is set 0. The TX core then returns to its regular operation after the sync interval is set to its default value. After the synchronization is done, the command enable register is set to 1, allowing the serial port to output any command words sent to it. Once this is done, the testbench will start simulating the trigger pulses.

3.1.3 Test Cases

The first part of the test involves sending a series of individual trigger pulses without configuring the trigger extender. Firstly, the patterns 1000, 0001, 0000, and 1001 are sent consecutively. After that, the testbench waits for a few clock cycles, and then it sends the patterns 0010 and 0100. When this step is done, the testbench will start testing the trigger extender. To do that, the testbench will first set the trigger extender register to 7 and then issue a 1000 trigger pattern. After that, the testbench will set the trigger extender interval to 11, and it will resend the 1000 trigger pattern. Finally, the trigger extender interval is set to 15 and the 1000 trigger pattern is sent one more time.

3.2 Simulation Testbench Result Analysis

The waveforms produced by the simulation of the testbench are shown in appendix B. In each of the figures, waveforms shown in green are the clock signals used by the testbench. The blue waveforms are the signals used to configure the wishbone interface. The waveforms shown in cyan represent the signals used by the trigger extender module. The waveforms represented in yellow show the signals that are present in the trigger code generator module. Finally, the waveforms shown in magenta represent the signals present in the RD53B emulator.

Figures A to D show the TX core configuration process. As seen in figure A, the first part of this process sets the trigger configuration to external mode. The wishbone address, rep-

resented by the wb_addr_i signal, is set to a value of 5. In addition, the wishbone input data port, represented by the wb_dai_i signal, is set to a value of 0. Finally, the wishbone write-enable flag, represented by the wb_we_i is asserted. After that, the same process is repeated for the trigger enable and trigger code generator enable registers. Figure B shows the setup of the TX polarity and the TX configuration registers. After the sync interval is set to 0, the TX core will repeatedly send sync frames to the RD53B. Figure C shows that these frames being received by the RD53B emulator through its data input port, which is represented by the magenta datain signal. After 300 clock cycles have elapsed, the sync interval register is set to its default value of 16 and the TX channel enable register is set to 1. These last two configuration steps can be seen in figure D.

After the configuration is done, the testbench starts simulating the pulse signal. Figure E shows the process of generating the 1000, 0001, 0000, and 1001 trigger patterns. The trigger extender interval input port, which is represented by the cyan ext_interval_i signal, is set to its default value of 0 throughout this process. As a result, the trigger extender module is simply outputting the pulses that it receives from the trigger unit. This is demonstrated by the fact that pulses being sent through the pulse output port of the trigger unit, represented by the cyan trig_pulse_o signal, are identical to the pulses being received by the trigger code generator module, which is represented by the yellow pulse_i signal. In addition, these pulses are being interpreted correctly. This is demonstrated by the fact that the value of the command_word register used by the trigger code generator module is being set to the value of the trigger patterns produced by the testbench. In addition, the trigger code generator is incrementing the base tags correctly, which is demonstrated by the fact that the yellow base_tag register is only being incremented when a new trigger pattern containing at least one trigger is received. Furthermore, the value of the yellow code_o output port of the trigger code generator module demonstrates that trigger patterns and base tags are being encoded correctly. It can also be seen that the second code double-word produced by the module has its higher order bits set to the idle pattern. This demonstrates that the code generator module correctly outputs idle patterns when no triggers are received during a code production interval. Finally, the yellow code_ready_o flag is being set to 1 when a new command double-word is ready.

The results of transmitting the command words produced in the previous part can be seen in both figures E and F. The magenta datain signal present in both of those figures shows the serial data that was transmitted to the RD53B emulator. Figure F demonstrates that the command words have been recognized and decoded by the RD53B emulator. This is demonstrated by the fact that the magenta trig_data signal used by the RD53B emulator is equal to the decimal representation of the binary trigger patterns produced by the testbench. In addition, the fact that the magenta trig_detect signal is being set to 1 when a new code word is decoded further demonstrates that the RD53B is detecting the trigger command words. The results of sending the 0010 and 0100 trigger patterns can be seen in figures F to H. These figures demonstrate that these trigger patterns are also being correctly encoded and outputted by the YARR firmware. A simplified waveform diagram of the process of generating all of the previously mentioned trigger patterns can be seen in figure 5.

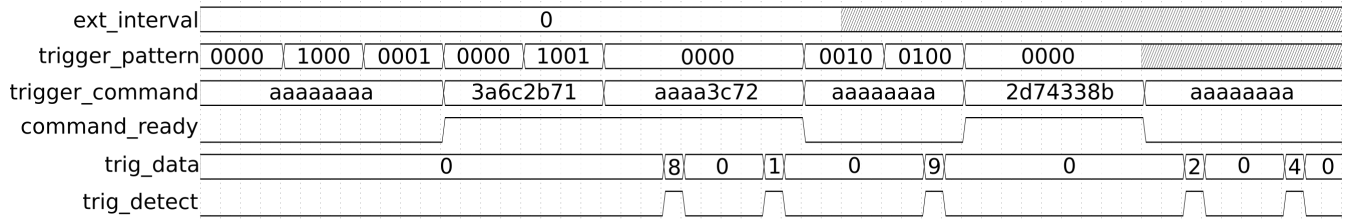

Figure 5: Simplified waveform diagram of the process of generating trigger patterns

After the first trigger patterns are sent, the testbench starts testing the trigger extender module. Figure F shows the wishbone bus configuration used to set the trigger extender module to a value of 7. Furthermore, the waveform for the cyan `ext_interval` shows that the trigger extender is receiving the new trigger extender interval value. The results of sending the 1000 pattern can then be seen in figure G. The aqua waveform for the `trig_pulse_o` demonstrates that the trigger extender is receiving the trigger pulse. Also, the yellow `pulse_i` signal shows that the trigger extender is extending the trigger for seven additional clock cycles, and the trigger code generator is receiving the extended signal. In addition, the value of the yellow waveform of the `code_word` register used by the trigger code generator demonstrates that the extended pulse is correctly being interpreted as the 1100 trigger pattern. The yellow waveform for the `code_o` signal in Figure I demonstrates that the 1100 trigger pattern is being encoded correctly, and the trigger code generator is outputting the respective command word. Finally, the magenta waveform for the `trig_data` in the same figure demonstrates that the code word is being detected and correctly decoded by the RD53B emulator. The results of setting the trigger extender interval to 11 and sending the 1000 trigger pattern are being shown in figures G to K. The results of setting the trigger extender interval and sending the same trigger pattern as before are shown in figures I to M. These figures demonstrate that, in both of these cases, the trigger pulses are being extended correctly, and the resulting trigger pattern is being encoded correctly. A simplified waveform diagram of the process of testing the trigger extender module can be seen in figure 6

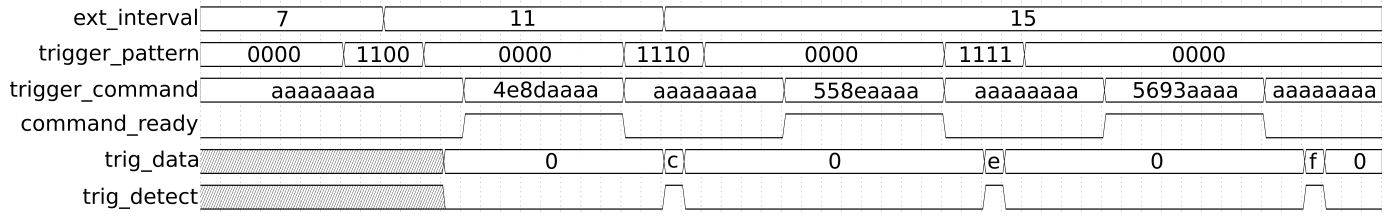

Figure 6: Simplified waveform diagram of the process of testing the trigger extender

3.3 YARR Scans on Real Hardware

Due to the difficulty of simulating trigger pulses on real hardware, the RD53B scans done for this test only check if the additions done to the YARR firmware in this project do not interfere with the functionality already present in the firmware. These scans simply inject 100 digital pulses into each pixel of a real RD53B chip and then produce an occupancy map showing the number of hits in each pixel. If the YARR firmware is functioning correctly, the resulting occupancy map should report that every pixel of the RD53B chip has received 100 hits.

To run these scans, a Trenz Electronic TEF1001 board was connected to one of the PCIe ports of a computer. The YARR firmware was then flashed into the Xilinx Kintex-7 FPGA contained in the TEF1001 board. An Ohio adapter card was then connected to the TEF1001 board, and port A of the Ohio adapter card was connected to the CMD/DATA port of the RD53B chip board using a display port cable. Finally, a custom power cable was then connected to the RD53B chip and a voltage of 1.6 V was supplied to the board. The scans were then run from the computer to which the TEF1001 board is connected. The occupancy map obtained is shown in figure 7

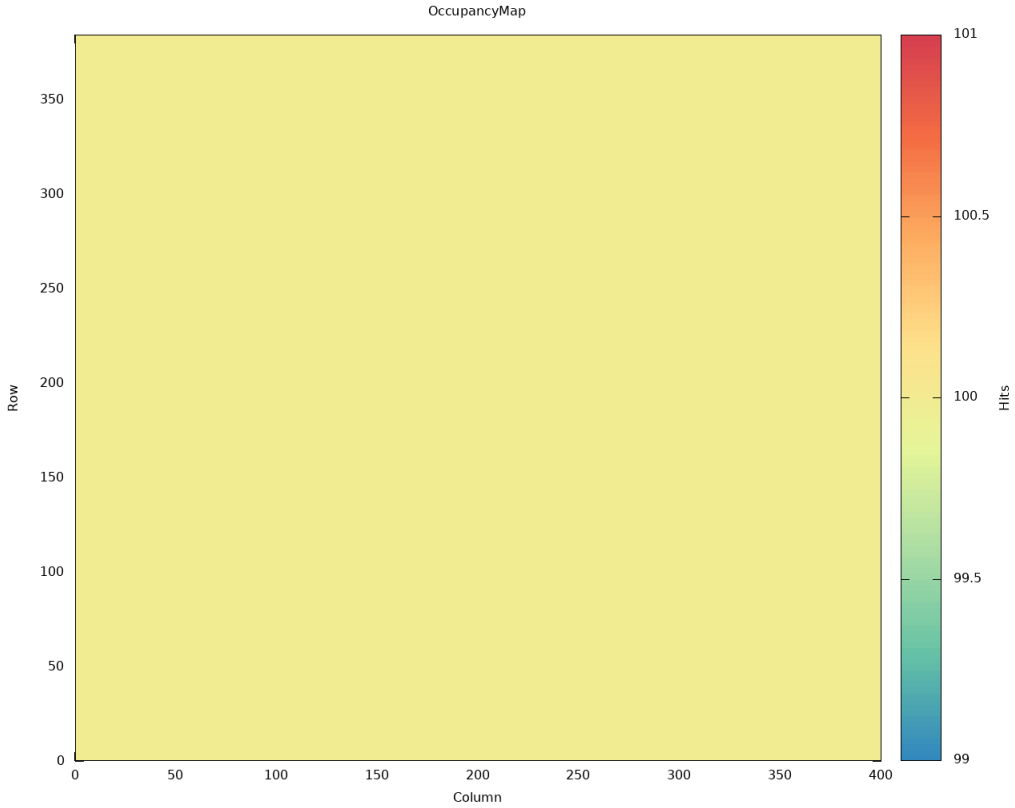

Figure 7: Occupancy map produced by the RD53B scans

Figure 7 shows that every pixel in the RD53B chip used for testing received a total of 100 hits. This indicates that changes made to the YARR firmware in this project did not interfere with the existing functionality of the firmware.

Acknowledgments

Firstly, I would like to thank my advisor Scott Hauck for giving me the opportunity to work at the ACME laboratory. His guidance was essential for the completion of this thesis. The weekly lab meetings organized by him significantly helped me to take steps that needed to be taken to complete this project successfully.

I would also like to thank Timon Heim for guiding me through developing and testing the project described in this thesis. His guidance was essential for me to understand the YARR firmware and the RD53B command encoding. I would not have been able to complete this project without Timon's help in debugging the issues encountered while I was developing this project.

Next, I would like to thank Geoff Jones for helping me to code in VHDL. The code review sections I had with him were essential for me to catch bugs present in the code I wrote. I am also very grateful for his help in writing the testbench used to test the modules designed in this project.

Finally, I would like to thank Lauren Choquer and Donavan Erickson for giving me the necessary background for the project discussed in this thesis. Lauren was extremely helpful in getting me started with the project and getting me up to speed with everything that I needed to know. Donavan's help in explaining the basics of the command processor of the RD53B emulator was essential for me to be able to integrate it into the testbench used in the project.

References

[1] K. Anthony. "Preparing ATLAS for the future". In: *CERN* (Dec. 20, 2018). URL: https://atlas.cern/updates/news/preparing-ATLAS-for-future.

[2] F. Arteche Gonzalez et al. *Extension of RD53*. Tech. rep. Geneva: CERN, Sept. 2018. URL: https://cds.cern.ch/record/2637453.

[3] CERN. *High-Luminosity LHC*. URL: https://home.cern/science/accelerators/high-luminosity-lhc (visited on 05/14/2021).

[4] CERN. *The ATLAS Detector*. URL: https://atlas.cern/discover/detector (visited on 05/14/2021).

[5] CERN. *The ATLAS Experiment*. URL: https://atlas.cern/about (visited on 05/14/2021).

[6] CERN. *The Large Hadron Collider*. URL: https://home.cern/science/accelerators/large-hadron-collider (visited on 05/14/2021).

[7] CERN. *The RD53B Pixel Readout Chip Manual*. May 27, 2020.

[8] T. Heim. "YARR - A PCIe based Readout Concept for Current and Future ATLAS Pixel Modules". In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 032053. DOI: 10.1088/1742-6596/898/3/032053. URL: https://doi.org/10.1088/1742-6596/898/3/032053.

[9] T. Heim, A. Sautaux, and V. Baratham. *Yarr Firmware*. Aug. 31, 2020. URL: https://github.com/Yarr (visited on 05/16/2021).

[10] F. Hugging. "The ATLAS pixel detector". In: *IEEE Transactions on Nuclear Science* 53.3 (2006), pp. 1732–1736. DOI: 10.1109/TNS.2006.871506.

[11] N. Mittal. "Development of an FPGA emulator for the RD53B chip". MSc Thesis. University of Washington, Aug. 14, 2020. URL: http://hdl.handle.net/1773/45785 (visited on 05/14/2021).

[12] A. Sautaux. *Wishbone Express Core*. Aug. 30, 2017. URL: https://github.com/Yarr/Yarr-fw/blob/master/rtl/kintex7/wbexp-core/README.md (visited on 05/14/2021).

[13] B. Schmidt. "The High-Luminosity upgrade of the LHC: Physics and Technology Challenges for the Accelerator and the Experiments". In: *Journal of Physics: Conference Series* 706 (Apr. 2016), p. 022002. DOI: 10.1088/1742-6596/706/2/022002. URL: https://doi.org/10.1088/1742-6596/706/2/022002.

[14] N. L. Whallon et al. "Upgrade of the YARR DAQ system for the ATLAS Phase-II pixel detector readout chip". In: *PoS* TWEPP-17 (2018), 076. 5 p. DOI: 10.22323/1.313.0076. URL: https://cds.cern.ch/record/2312402.

Appendices

A Tag Encodings

Base Tag	Binary Encoding	Hex Encoding	Base Tag	Binary Encoding	Hex Encoding
0	0110_1010	0x6A	27	1100_1010	0xCA
1	0110_1100	0x6C	28	1100_1100	0xCC
2	0111_0001	0x71	29	1101_0001	0xD1
3	0111_0010	0x72	30	1101_0010	0xD2
4	0111_0100	0x74	31	1101_0100	0xD4
5	1000_1011	0x8B	32	0110_0011	0x63
6	1000_1101	0x8D	33	0101_1010	0x5A
7	1000_1110	0x8E	34	0101_1100	0x5C
8	1001_0011	0x93	35	1010_1010	0xAA
9	1001_0101	0x95	36	0110_0101	0x65
10	1001_0110	0x96	37	0110_1001	0x69
11	1001_1001	0x99	38	0010_1011	0x2B
12	1001_1010	0x9A	39	0010_1101	0x2D
13	1001_1100	0x9C	40	0010_1110	0x2E
14	1010_0011	0xA3	41	0011_0011	0x33
15	1010_0101	0xA5	42	0011_0101	0x35
16	1010_0110	0xA6	43	0011_0110	0x36
17	1010_1001	0xA9	44	0011_1001	0x39
18	0101_1001	0x59	45	0011_1010	0x3A
19	1010_1100	0xAC	46	0011_1100	0x3C
20	1011_0001	0xB1	47	0100_1011	0x4B
21	1011_0010	0xB2	48	0100_1101	0x4D
22	1011_0100	0xB4	49	0100_1110	0x4E
23	1100_0011	0xC3	50	0101_0011	0x53
24	1100_0101	0xC5	51	0101_0101	0x55
25	1100_0110	0xC6	52	0101_0110	0x56
26	1100_1001	0xC9	53	0110_0110	0x66

Table A: Tag encodings used in trigger command words [7]

B Simulation Results

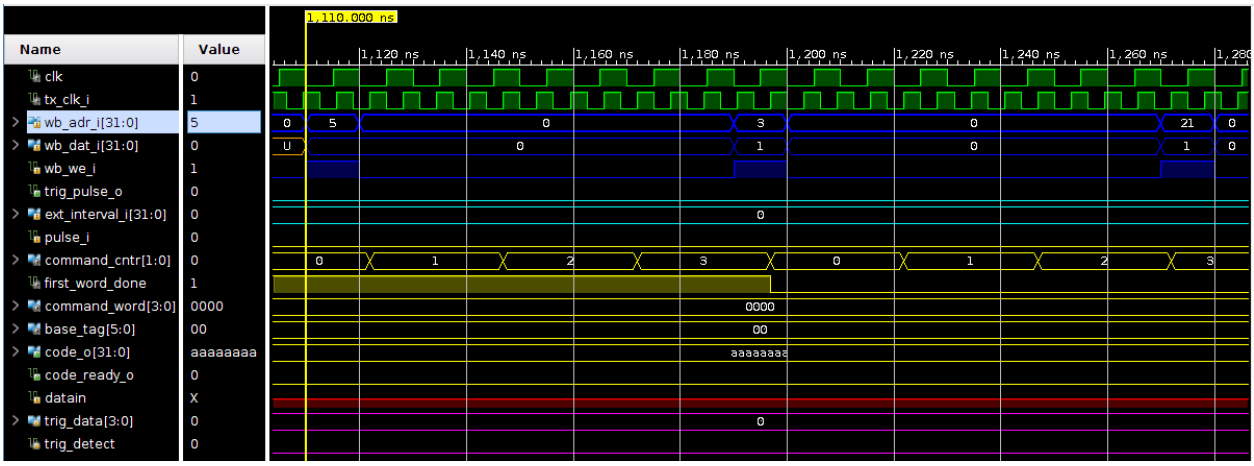

Figure A: Process of setting the trigger configuration, trigger enable and trigger code generator registers to the appropriate values

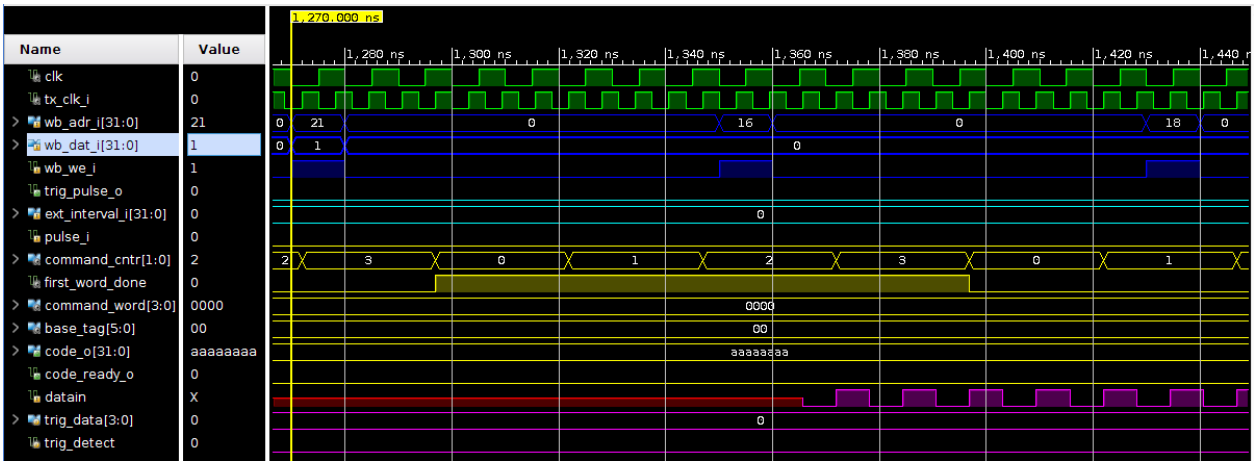

Figure B: Process of setting the trigger code generator, TX polarity and sync interval registers to the appropriate values

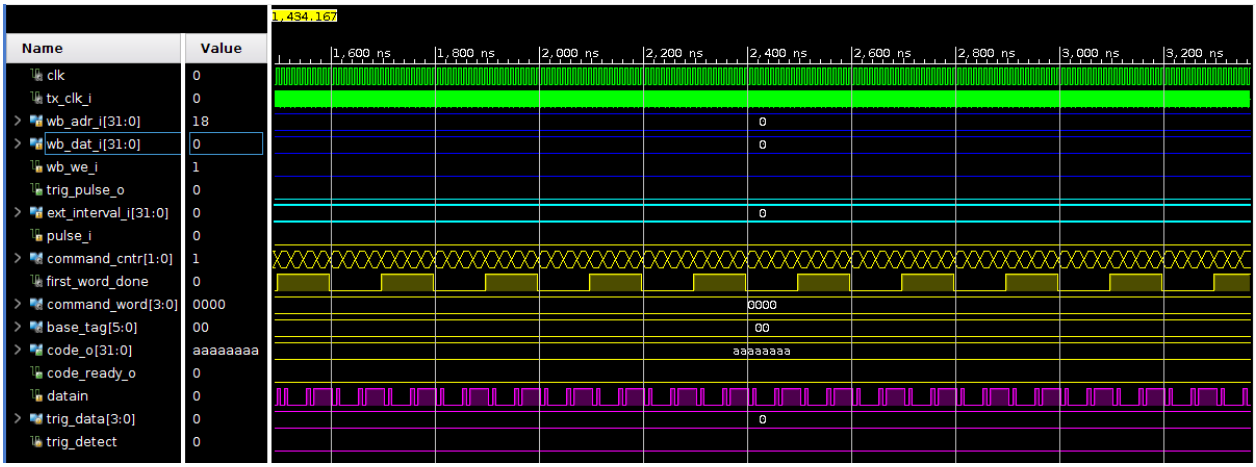

Figure C: Initialization of the command protocol between the YARR firmware and the

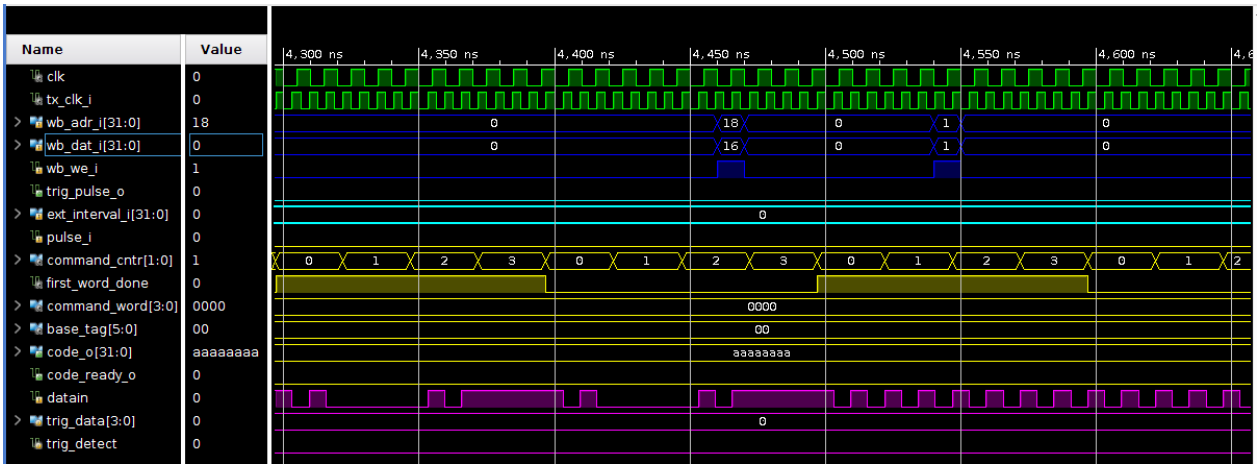

Figure D: Process of setting the sync interval register and the TX channel registers to the appropriate values

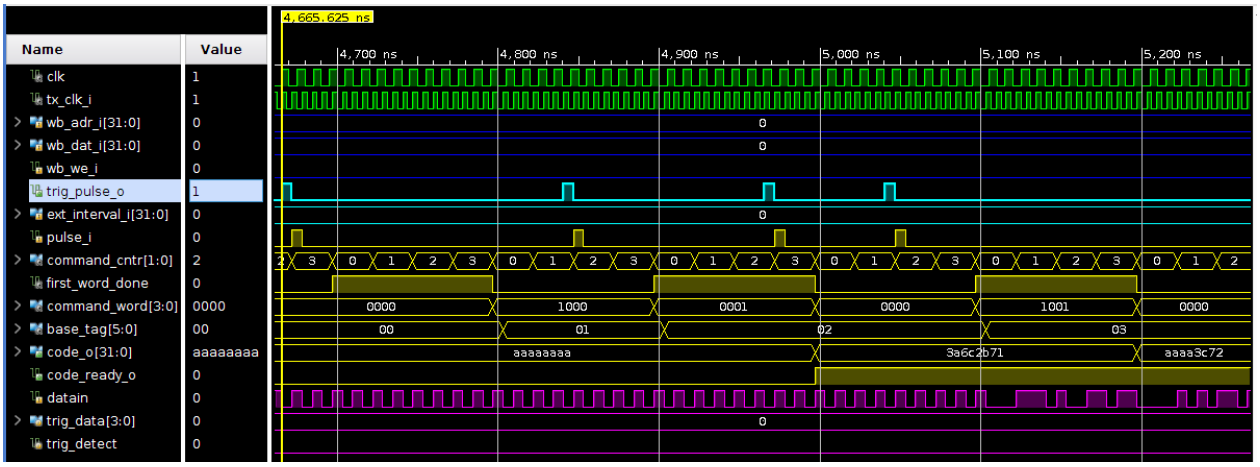

Figure E: Results of sending the 1000, 0001, 0000 and 1001 trigger patterns

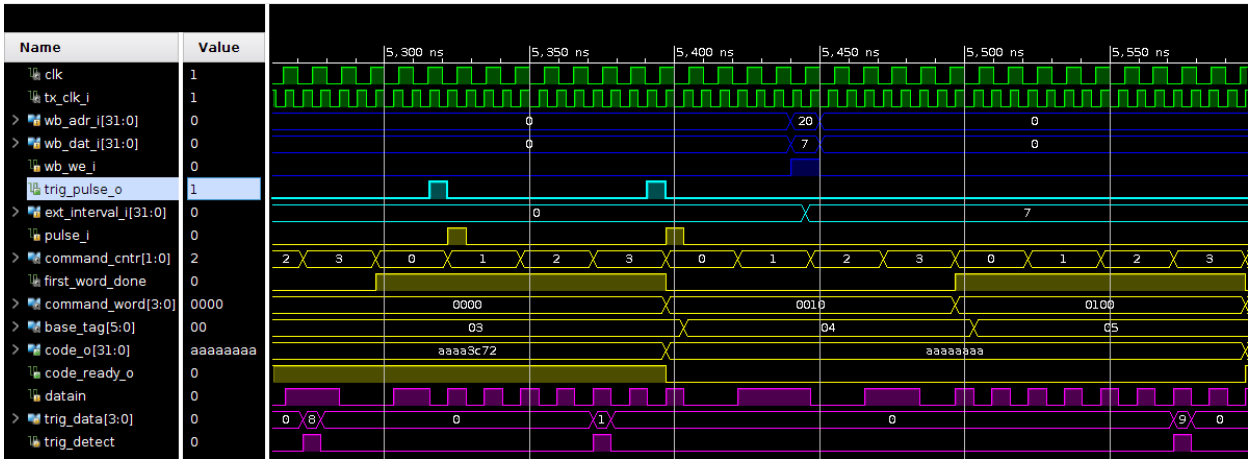

Figure F: Results of sending the 0010 and 0100 trigger patterns

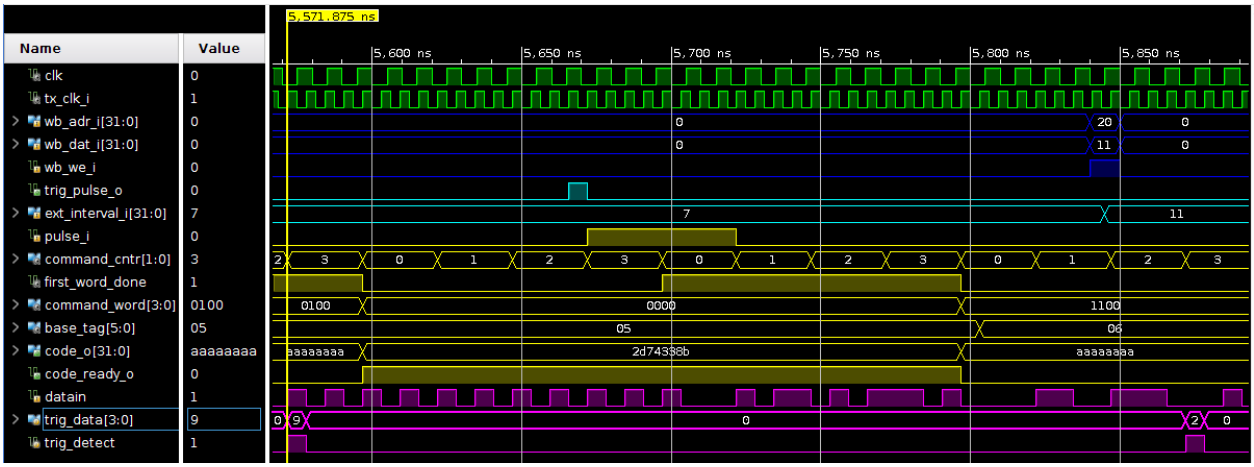

Figure G: Results of setting the trigger extender interval to 7 and sending the 1000 pattern

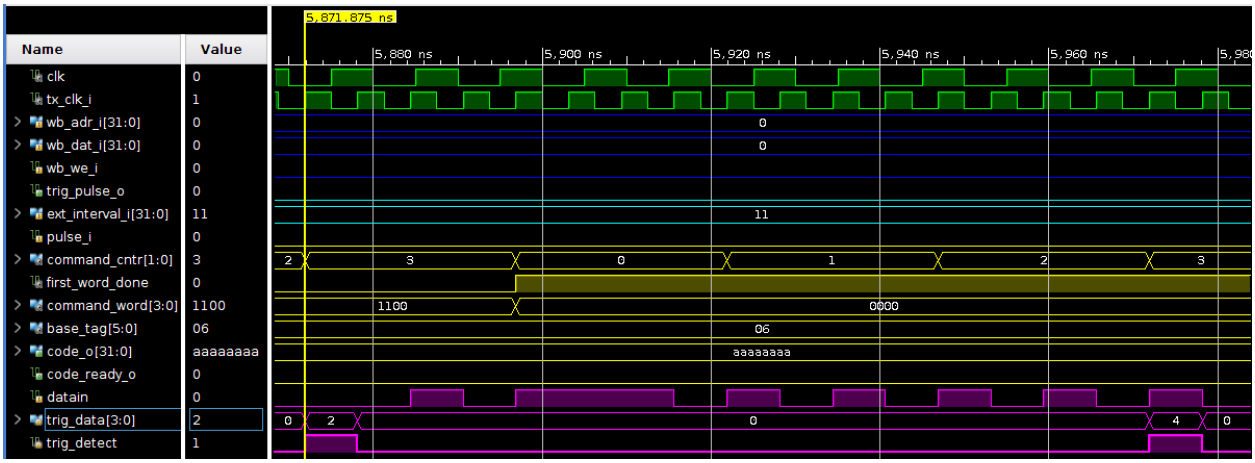

Figure H: Results of setting the trigger extender interval to 11

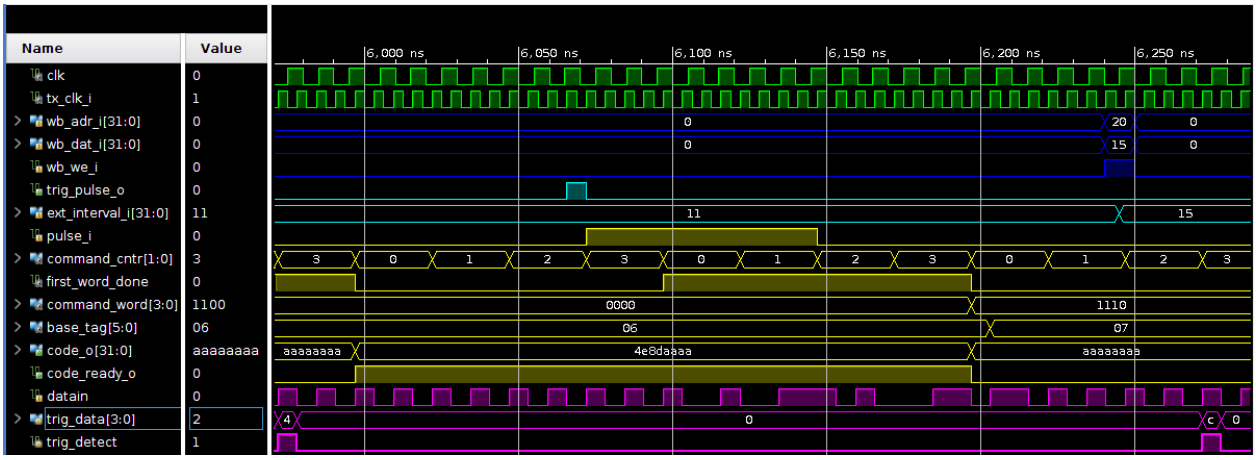

Figure I: Results of sending the 1000 pattern after setting the trigger extender interval to 11

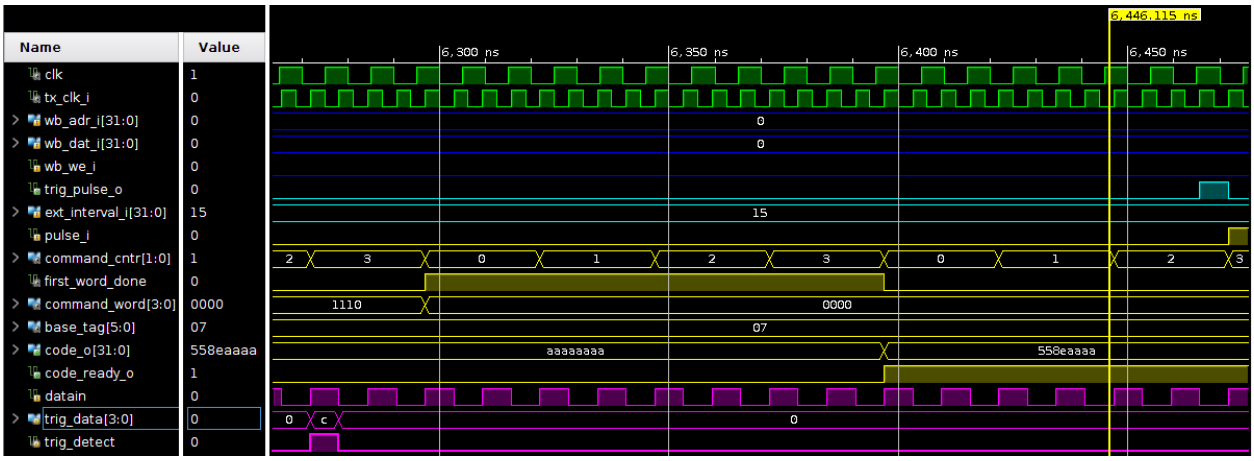

Figure J: Results of setting the trigger extender interval to 15

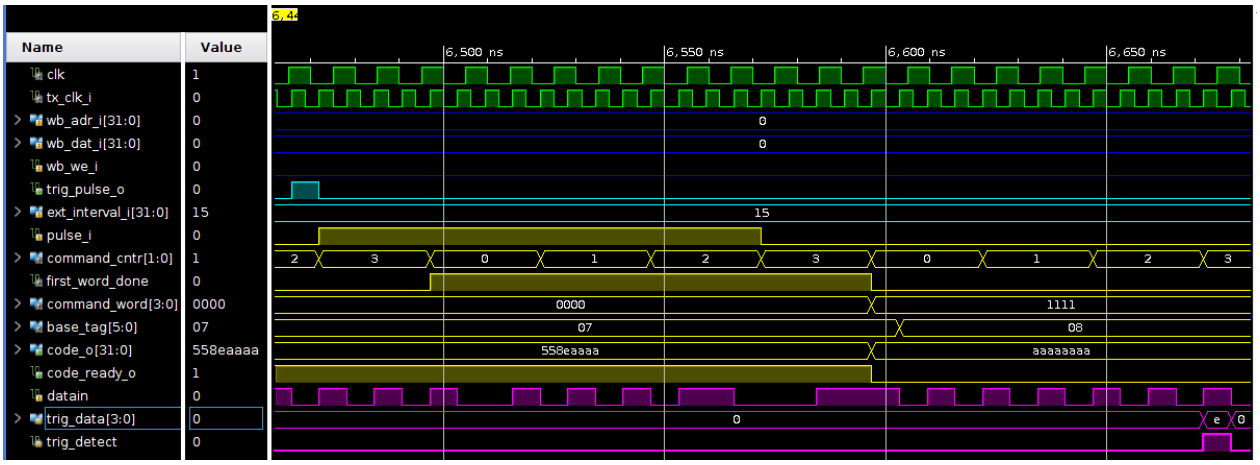

Figure K: Results of sending the 1000 pattern after setting the trigger extender interval to 15

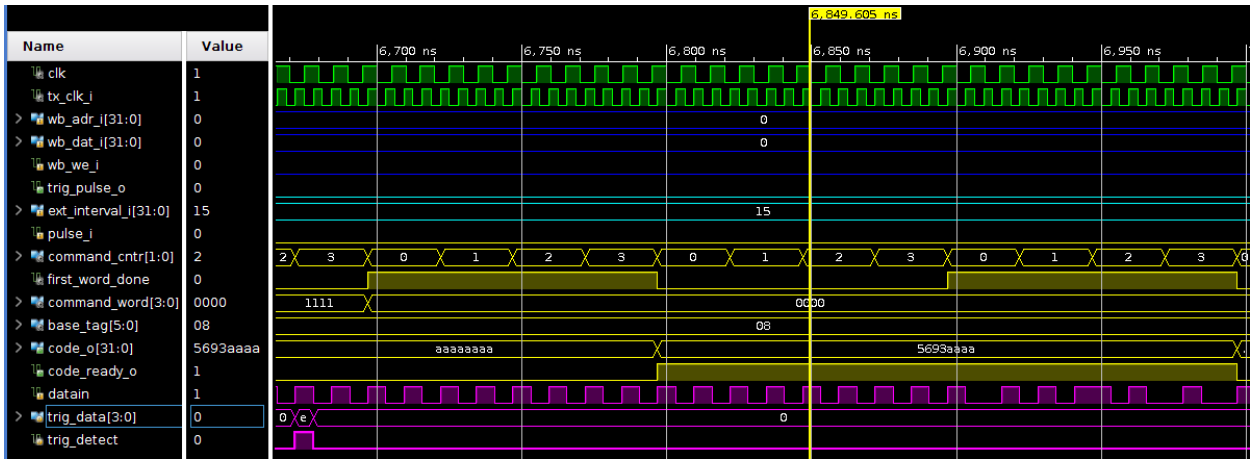

Figure L: Results of waiting for the RD53B emulator to receive the command generated from the 1110 trigger pattern

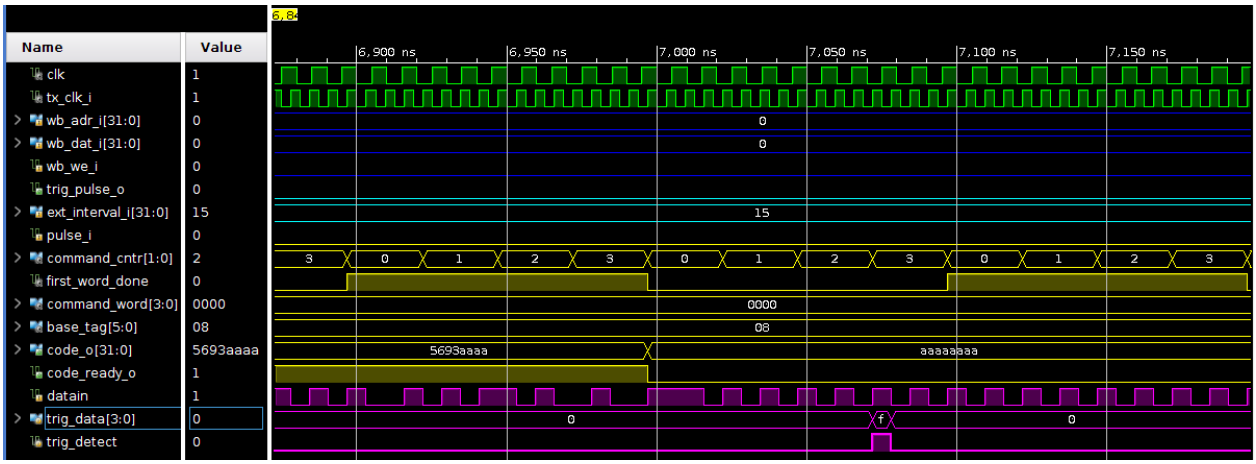

Figure M: Results of waiting for the RD53B emulator to receive the command generated from the 1111 trigger pattern