

Prac5: Trigger Surround Cache

Cameron F Clark[†] and Kian S Frassek[‡]

EEE4120F Class of 2024

University of Cape Town

South Africa

[†]CLRCAM007 [‡]KIAFRS001

I. INTRODUCTION

A Field Programmable Gate Array (FPGA) is an interconnected circuit that can be customized for specific applications. You can customize it using the coding language Verilog. In this lab, we will build a simple TSC Trigger Surround Cache using an ADC and a ring buffer memory device. ADC records an input analog signal and converts it to a digital signal. A ring buffer is a type of storage method where you have a fixed size storage, and you have two pointers - one pointer which is the head and one pointer which is the tail. The head points to the value you are going to read from, while the tail points to the value you are going to write to. The TSC must be able to communication with other devices using transfer protocols.

II. DESIGN AND IMPLEMENTATION

A. Hardware and Software

This was run on a MacBook Pro computer using Iverilog. Additionally, gtkwave was used to monitor the modules input and outputs.

B. TSC design overview

The TSC (Trigger Surround Cache) has a 3 bit state register, a 32-bit timer, a 32-bit TRIGGER_TM, and an internal ring buffer. It is connected to the ADC (Analog-to-Digital Converter) via a request (REQ), ready (RDY), and data (DAT) lines. Additionally, the TSC can communicate with other devices using triggered (TRD), Send Buffer (SBF), serial data (SD), and completed data (CD) registers and wires.

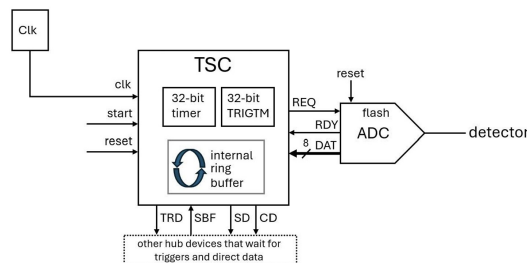


Fig. 1: Block diagram of the TSC

There is also an accompanying TSC_tb test bench which is used to initiate and test the TSC module.

C. CLock (clk)

A 250 MHz clock signal is set up on clk wire in the TS_tb test bench.

D. State register

The state register is a 3-bit register that has the following states:

- STOP (0b000) : State when the machine is powered on and has not been reset yet.
- READY (0b001) : State that is entered on the reset pin rising edge and it waits for the start pin rising edge.
- RUNNING (0b010): State that is entered from ready or Idle state once the start pin rising edge is pulled high. It incrementing the timer and write adc values to the ring buffer.
- TRIGGERED (0b011): State entered when the value read from the ADC is greater than the predetermined trigger value (TRIGVL). It capturing the next 16 values.
- IDLE (0b100); This state is entered when a trigger event has occurred and the TSC is waiting for the start pin or SPF pins rising edge.
- SENDING (0b101): This state is entered from the IDLE state when the SFB line is pulled high. It indicates that data is being sent on the SD line.

E. Timer

The timer is incremented on the rising edge of the clock. When a trigger event occurs the timer is saved in the TRIGTM register which is outputted to the test bench. The timer is reset if a transition into a running state occurs. To calculate the time stored in the TRIGTM register the timer is multiplied by the clock period (4 ps).

F. Ring Buffer

The ring buffer is used to store the values read by the ADC. It is made up of 32 8-bit registers stored in an array called ring_buffer. The tail pointer is named write_ptr and is initially set to 5'b11111. and the head pointer is named read_ptr and is initially set to 5'b00000. The 5 bit format for the head and tail index is used to induce rollover at value 32 (32 just becomes 0). To add a new value to the ring buffer the write_ptr is incremented and then the value is stored in the ring buffer at the write_ptr index then read_ptr is incremented. To read a value from the ring buffer the value at the read_ptr index is read and the read_ptr is incremented. this process is repeated

until the read_ptr = wright_ptr. Indicates that all the values have been read.

G. How the TSC interfaces with the ADC

The ADC is initialized in the TSC module. This connect the adc_request to REQ, adc_request to RST, adc_ready to RDY, and adc_data to DAT. The adc_request line is connected to the main reset line this mean that the ADC module is reset when the TSC_td module is pull the reset line hight. once the ADC is reset the ADC will pull the adc_ready line hight to indicate that the ADC is ready to send data over the 8 bit DAT line. The bytes of of data being sent over the DAT line are extracted from a CSV file in the ADC module. When the TSC module detects the adc_ready line is hight and the TSC is in the running state. The TSC will pull the adc_request line hight on the posed of the clk line for 1 ps to request data from the ADC on the adc_data line. This can be seen in the two code section named listing 1 and listing 2 below. The subsequent data is then stored on the ring buffer.

Listing 1: Code for storing data and moving pointers in the posedge adc_ready

```
always @(posedge adc_ready) begin
    if (adc_request) begin
        #1 //delay so the pulse doesn't disappear on the echo. TO BE REMOVED

        //manage trigger_value
        //... the trigger code is here

        //store data and move pointers around
        ring_buffer[write_ptr] = adc_data;
        read_ptr++;

        adc_request = 0; //pull request down
    end
end
```

Listing 2: Code for requesting data from the ADC on posedge of the clock when in RUNNING state

```
'RUNNING: begin
    timer++;
    if (~ adc_request)
        adc_request = 1; //request new adc value (handled with posedge adc_ready)
end
```

H. Triggering

The TSC module has a trigger value (TRIGVL) that is set in TSB module. However, TRIGVL can also be set in the TSC_tb. When the TSC is in the RUNNING state and the ADC is outputing to adc_data if the value adc_data is greater than the TRIGVL the following event will occurs:

- The TSC module transitions to the TRIGGERED state.
- The TRIGTM register is set to the current value of the timer.
- The reg remaining_values is set to equal 5'h10 (16 in decimal). This is done to help record 16 more values.

This Implementation is demonstrated in the code below.

Listing 3: Code for triggering event in the TSC module

```
//if it hasn't been triggered already, check for a valid trigger
if (state != 'TRIGGERED) begin
    if (adc_data > TRIGVL) begin
        state = 'TRIGGERED;
        TRIGTM = timer; //capture time of trigger
        remaining_values = 5'h10; //set remaining adc values to 16 (handled by posedge clk)
    end
end
```

When the TSC module is in the TRIGGERED state the TSC dose the following on the positive edge fo the clock. It incremented the timer value because as per project brief the timer must only stop being incremented once the IDLE stat has been entered. Then the TSC will check if remaining_values =0 to see if 16 byte have been recorded. If all 16 bytes have been recoded it will transition to the IDLE state. else it will pull the adc_request line hight to request more data from the ADC and decrease the remaining_values by 1 to indicate that it has recorded one more value. This this can be seen in the code below.

Listing 4: Code for triggering event in the TSC module

```
'TRIGGERED: begin
    timer++;
    //check that there are remaining values left to capture and decrement.
    if(remaining_values-->0) begin
        if (~ adc_request)
            adc_request = 1; //request new adc value (handled with posedge adc_ready)
        end else begin //all 16 values have been captured, wait for start or SBF command
            state = 'IDLE;
            TRD = 1'b1;
        end
    end
end
```

I. Sending data to to HUB devices

To sens da date the following state variables where made:

J. Sending data to HUB devices

To send data, the TSC module uses the following bit definitions for the current serial_bit when transmitting data via the SD line:

- WAIT_BIT (4'b1101): used to wait for at least 1 positive edge of the clock for the first start bit.
- START_BIT (4'b1110): Use to pull the SD line low when the first byte is being sent.
- END_BIT (4'b0111): Used to indicates a byte has been sent.
- FIRST_BIT (4'b1111): The 'first' bit of the byte. The serial bit is always incremented BEFORE the bit is sent.

When the TSC module is in the IDLE state and the SBF line is pulled high, the TSC module will transition to the SENDING state. During this transition, it will set SD high, CD low and serial_bit = WAIT_BIT. If the TSC in sending state and the serial_bit = WAIT_BIT, on the positive clock edge the serial_bit will be set to equal START_BIT. Then on the next negative edge of the clock SD will be set low and serial_bit will be set to equal FIRST_BIT. The TSC will then send the 8 bit data on the SD line on each negative edge of the clock. After the 8 bits have been sent the serial_bit will be set to equal END_BIT. This will results in the SD line being pulled low and serial_bit is set to FIRST_BIT unless read_ptr pus 1 equal write_ptr in which case the TSC will transition to the READY state and the CD line will be pulled high.

III. TESTING AND VALIDATION

This section details the tests that were conducted on the TSC module while it is connected to the ADC and HUB module. The tests conducted were only aimed to prove the functionality of the module and weren't aimed to test the module protection

of misuse, even-though the TSC module was written to handle such cases.

The test bench runs a simple test of sending a pulse on the reset line followed by a pulse on the start line to move the module into running mode. The module will next get triggered by one of the 256 ADC values going above the TRIGVL. The TRIGVL is set at 0xC8 and there are only two values in the ADC csv file which are above 0xC0 which are the 150th and 200th values. The test bench is programmed to send a start pulse on the first trigger and an SBF pulse on any subsequent triggers.

The test was run as one continuous test and several snippets of gktwave were taken and explained in chronological order.

A. Resetting and Starting

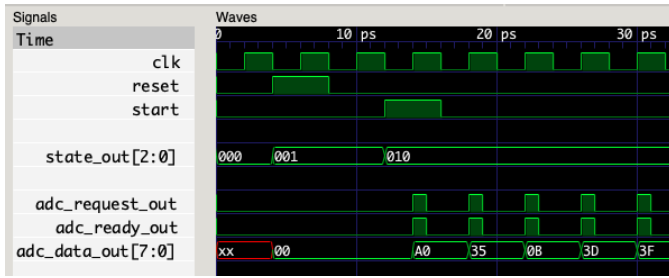


Fig. 2: Resetting and Starting gktwave Output

Firstly, a reset then start pulse are sent and the TSC module response by changing the state from STOP (0b000) to READY (0b001) and then to RUNNING (0b010). Once the module has entered running mode, it correctly requests data from the ADC module every rising clock edge, and the adc replies by pulling the ready line high and outputs a new byte on the data bus. In a real world implementation of this there would be a slight delay between the request being pulled high and the ready being pulled high. Next, the TSC module acknowledges the adc ready and resets the request line. Again, in a real world implementation there would be a slight delay between these edges.

B. Ring Buffer Writing

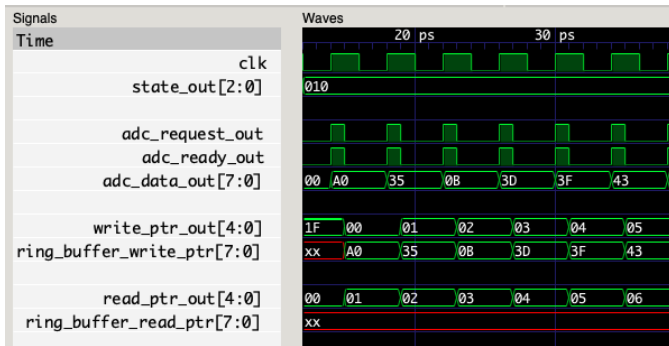


Fig. 3: Ring Buffer Writing gktwave Output

In RUNNING state, the adc value is written into the ring buffer. As it is shown in Fig. 3, the first byte the ADC module is 0xA0 which is correctly written into address 0 of the ring buffer, and the second byte 0x35 is written into address 1 of the ring buffer, etc. The byte at the read buffer pointer is unknown at it has not been written yet, and will only output a known value once a full loop of the ring buffer has been written.

C. Ring Buffer Reading

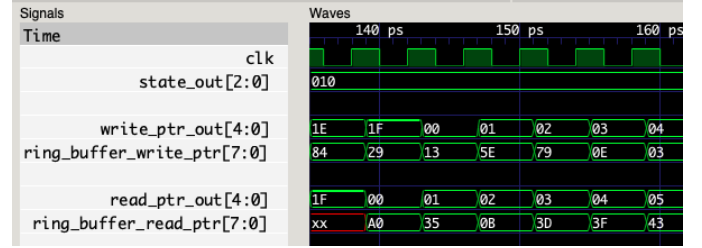


Fig. 4: Ring Buffer Reading gktwave Output

As mentioned in Subsec. III-B, a full loop has been written into the ring buffer and the read pointer is correctly return 0xA0 for address 0, and 0x35 for address 1, etc.

D. Triggering

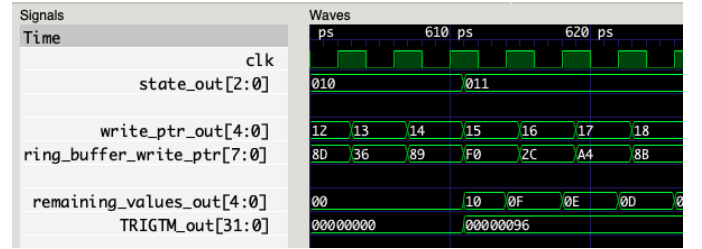


Fig. 5: Triggering gktwave Output

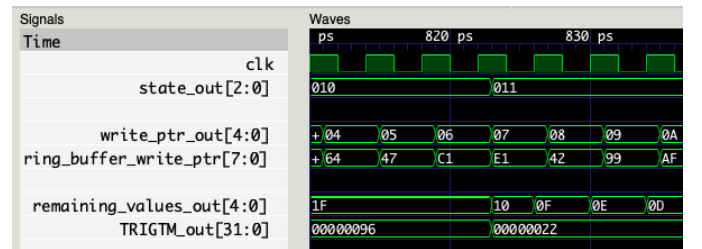


Fig. 6: Re-triggering gktwave Output

Fig. 5 and Fig. 6 show an initial triggering and re-triggering of the TSC module respectively. At the point of triggering, the state correctly changes to TRIGGERED (0b011) and the remaining ADC values is set to 16 to indicate that there are 16 more adc values to be written into the ring buffer. The remaining values then starts counting down with every adc value saved to the ring buffer. Additionally, the TRGRTM for

the initial trigger is correctly updated to the current value of the timer which is:

$$\frac{\text{trigger time} - \text{start time}}{\text{clock period}} = \frac{612\text{ps} - 12\text{ps}}{4\text{ps}} = 150 = 0x96$$

E. Raising TRD Line

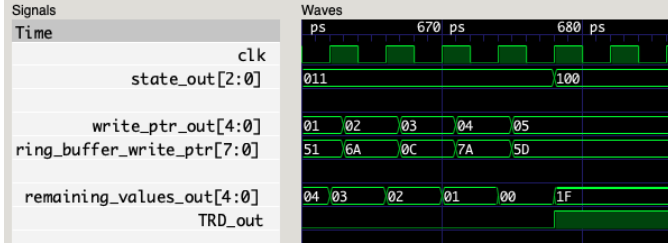


Fig. 7: Raising TRD Line gktwave Output

Once there are no remaining values (the remaining value register has reached 0), the TSC module correctly changes state to IDLE (0b100), stops recording adc values, and pulls the TRD line to the HUB module high.

F. Starting or Transmitting From Idle

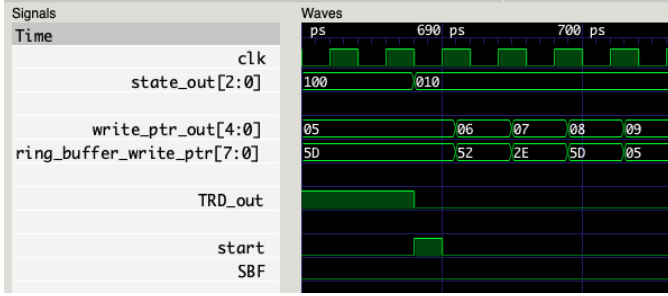


Fig. 8: Starting From Idle gktwave Output

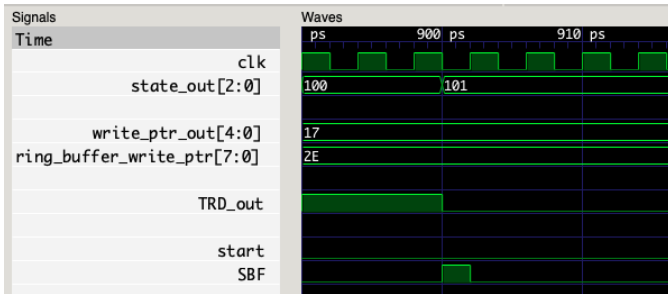


Fig. 9: Transmitting Data From Idle gktwave Output

From the IDLE state, the TSC module can either transition back to RUNNING or SENDING depending on the next command sent. In Fig. 8, the TSC module receives a start pulse, correctly transitions back to RUNNING (0b010) state, and correctly continues writing the ADC values into the ring buffer. In Fig. 9, the TSC module receives a SBF pulse, correctly transition into SENDING (0b101) state, and starts transmitting data with is shown in Subsec. III-G.

G. Starting Data Transmission

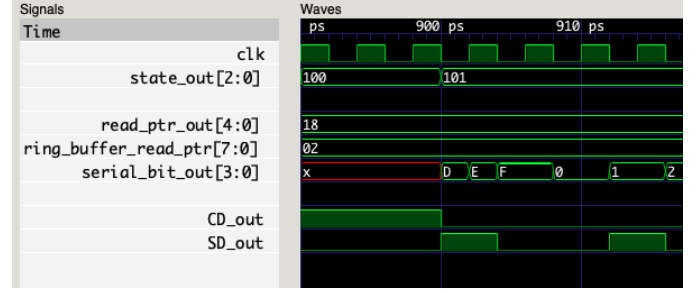


Fig. 10: Starting Data Transmission gktwave Output

H. Byte Transmitting

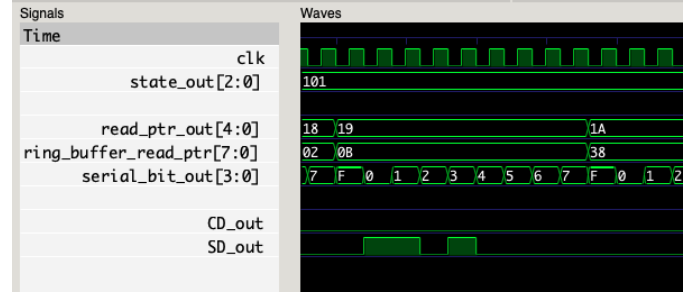


Fig. 11: Byte Transmitting gktwave Output

I. Ending Data Transmission

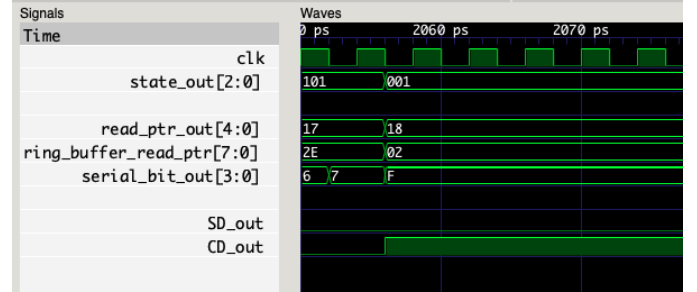


Fig. 12: Ending Data Transmission gktwave Output

IV. CONCLUSION

The Trigger Surround Cache (TSC) module was successfully implemented and tested. The system was robust and worked as expected in all of our test. The next step is to test against miss use. Then the tests should be repeated with a better ADC module. Final Physical testing on an FPGA should be preformed to see if it works in real-world application. The TSC module could be improved buy simplifying the code and/or make it more efficient. More specifically the ring buffer size could be changed to be more efficient and a user manual could be written for the TSC module to help users.