Prac5: Trigger Surround Cache

Cameron F Clark[†] and Kian S Frassek[‡]
EEE4120F Class of 2024
University of Cape Town
South Africa
[†]CLRCAM007 [‡]FRSKIA001

I. Introduction

A Field Programmable Gate Array (FPGA) is an interconnected circuit which can be coding using Verilog and can be customized for specific applications. This project details the building of a simple Trigger Surround Cache (TSC) using an ADC and a ring buffer memory device; an 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: a read pointer which is also called the head and a write pointer which is also called the tail. The TSC will also be able to communication with other devices using a clocked serial data transfer protocol.

II. Design and Implementation

A. Hardware and Software

This was run on a MacBook Pro computer with an intel processor using Icarus-Verilog. Additionally, gtkwave software 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 triggered time (TRIGTM), 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 HUB devices using triggered (TRD), Send Buffer (SBF), serial data (SD), and completed data (CD) registers and wires. Lastly, the TSC is connected to a external module that controls its clock (clk), start and reset lines.

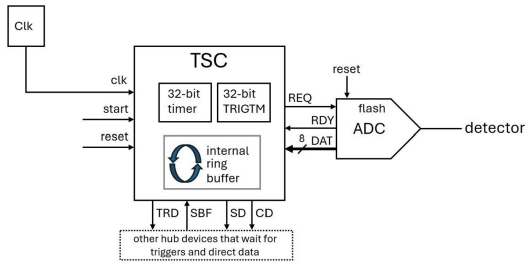

Fig. 1: Block diagram of the TSC

There is also an accompanying TSC_tb test bench which is used to initiate and test the TSC module. The test bench additionally simulates the HUB 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 TSC module is run as a state machine where the state is a 3-bit register that has the following states:

- `STOP` (0b000) : The state when the machine is powered on but has not been reset yet.
- `READY` (0b001) : The state where the module is awaiting a start command.
- `RUNNING` (0b010): The state entered after receiving a start command from `READY` or `IDLE`. In this state, the module increments the timer and writes adc values into the ring buffer.
- `TRIGGERED` (0b011): The state entered when a value read from the ADC is greater than the predetermined trigger value (TRIGVL). In this state, the module captures the next 16 adc values into the ring buffer
- `IDLE` (0b100); The state after the 16 adc values have been captured. This state signals to the HUB that the data is ready to be read, and the module awaits a start or SBF command.
- `SENDING` (0b101): This state is entered when it receives an SBF command from the `IDLE` state. 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 even occurs the timer is save 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 of the trigger relative to the start command, the TRIGTM value 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 byte registers stored in an array called ring_buffer. The write pointer or tail is a 5-bit register (reg [4:0] write_ptr). Similarly, the read pointer or head is a 5-bit register as well (reg [4:0] read_ptr) but is always one address ahead of the write pointer.

When adding new values to the ring buffer, the write pointer and read pointer are both incremented and then the value is written in the ring buffer at the address of the write pointer.

The write pointer is initially set to 5'b11111 such that the first write address is address 0.

When reading values from the ring buffer, the value at the read pointer is read and the read pointer is incremented; this is repeated until the read pointer is at the write pointer which indicates that all the values in the ring buffer have been read.

G. How the TSC interfaces with the ADC

The ADC is initialized in the TSC module, where the reset line is shared by both modules, the REQ line is controlled by the TSC module and the RDY and DAT lines are watched by TSC module.

When the TSC module requires a byte of data from the ADC module, it pulls the REQ line high, which causes the ADC module to output a byte of data on the DAT line and pull the RDY line high. The TSC module responds to the rising edge of the RDY line, stores the byte on the DAT line, and finally drops the REQ line low. The ADC module's data values come from a csv file of 256 randomly generated values.

Due to this being a simulation, all of the above would happen instantaneously, thus, artificial delay has been added to prevent the pulse from not showing on gktwave. This can be seen in the two code sections Lst. 1 and Lst. 2 below. It should be noted that both code sections have safety measures in place. Lst. 1 checks that data was actually requested and Lst. 2 check that the data has been recorded and acknowledged before setting the REQ line back high.

Listing 1: Verilog code for storing data and moving pointers on the posedge of RDY

```
always @(posedge RDY) begin
    if (REQ) begin
        #1 //delay so the pulse doesn't disappear on gktwave.

        //manage trigger_value ....

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

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

Listing 2: Verilog code for requesting data from the ADC on the posedge of the clock while in RUNNING state

```
always @(posedge clk) begin
    case(state)
        `\texttt{RUNNING}: begin
            timer++;
            if (~ adc_request)
                adc_request = 1; //request new adc value (handled with posedge adc_ready)
            end
        end

    //other cases

    endcase
end
```

H. Triggering

The TSC module has a constant trigger value (TRIGVL) which is set in the TSC module but can also be overwritten in the test bench. A trigger occurs when the TSC module is in the RUNNING state and the requested ADC DAT value is greater than the TRIGVL. This causes the following to happen:

- The TSC module transitions to the TRIGGERED state.
- The TRIGTM register is set to the current value of the timer.
- The register remaining_values is set to 5'h10 (equivalent of decimal 16). This value will count down as the following 16 adc values are recorded into the ring buffer.

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 != `TRIGERED) begin
          if (adc_data > TRIGVL) begin
              state = `TRIGERED;
              TRIGTM = timer; //capture time of trigger
              //set remaining adc values to 16 (handled by posedge clk)
              remaining_values = 5'h10;
          end
    end
```

When the TSC module is in the TRIGGERED state, the TSC does the following on the positive edge of the clock.

- It increments the timer value because as per project brief the timer must only stop being incremented once the IDLE stat has been entered.
- It checks if remaining_values is greater than zero, if so, it records the next byte from the adc into the ring buffer.
- If remaining_values is zero, then all 16 bytes have been recoded and it transitions to the IDLE state, and pulls the TRD line to the HUB module high.

Listing 4: Verilog code for recording 16 values after a trigger event

```
always @(posedge clk) begin
    case(state)
        `TRIGERED: begin
            timer++;
            //check that there are remaining values left to capture and decrement.
            if(remaining_values--)  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 = `\texttt{IDLE};
                    TRD = 1'b1;
                end
            end
        end

    //other cases

    endcase
end
```

I. 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): Wait for at least 1 positive edge of the clock for first start bit.
- TRANSMISSION_START_BIT (4'b1110): Use to pull the SD line low when the first byte is being sent.
- END_BIT (4'b0111): Data byte only has bits 0–7, so reaching bit 7 means the byte has finished.
- START_BIT (4'b1111): Start bit of the byte transmission 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 module in sending state and the serial_bit = WAIT_BIT, on the positive clock edge the serial_bit will be set to equal TRANSMISSION_START_BIT. This ensures that the TSC module will wait at least one full clock cycle before starting the fist byte of data. Then on the next negative edge of the clock the start bit will be sent where the SD line will be set low and serial_bit will be set to equal START_BIT.

The TSC will then send the 8 bit data on the SD line on each negative edge of the clock using little endian format and incrementing serial_bit each time. After the 8 bits have been sent, the serial_bit is set back to the START_BIT, and the read pointer is incremented. If the read pointer is at the write pointer, it means that the full ring buffer has been sent and the following occurs:

- The read pointer is set back to in front of the write pointer.
- The state transitions into READY.
- The SD line is pulled low.
- The CD line is 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. Reseting and Starting

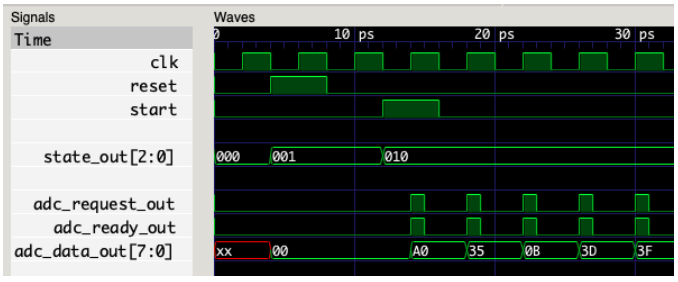

Fig. 2: Resetting and Starting gktwave Output

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

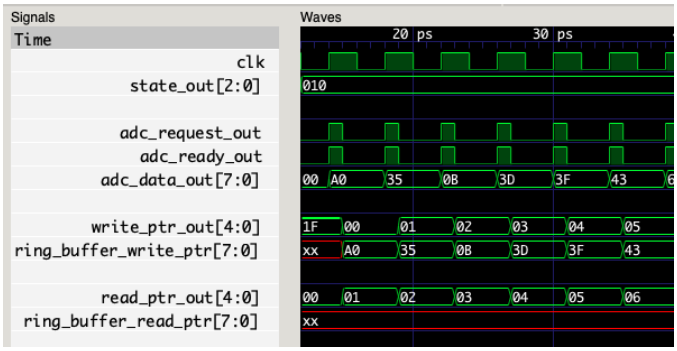

Fig. 3: Ring Buffer Writing gktwave Output

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

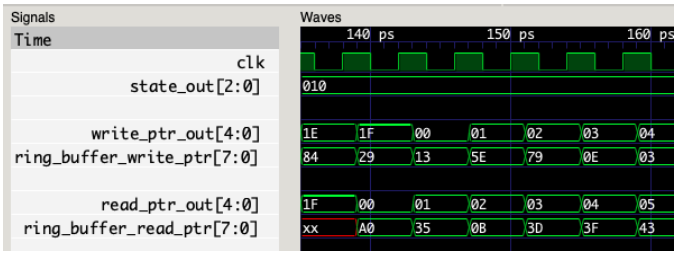

Fig. 4: Ring Buffer Reading gktwave Output

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

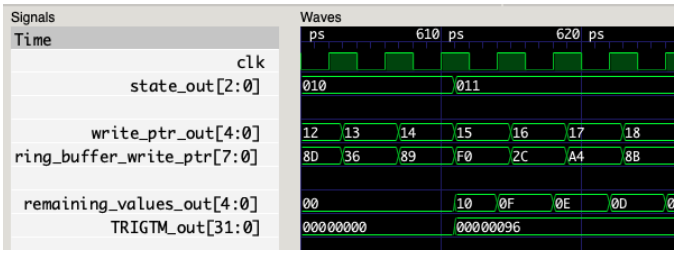

Fig. 5: Triggering gktwave Output

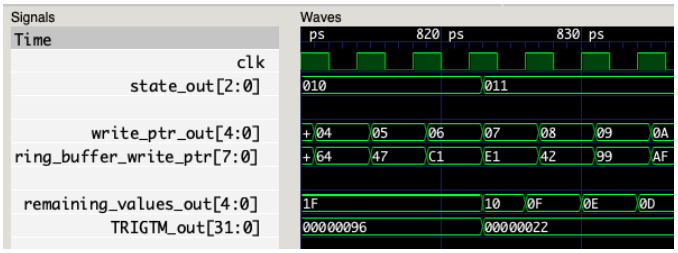

Fig. 6: Re-triggering gktwave Output

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:

$$\frac{trigger\ time - start\ time}{clock\ period} = \frac{612ps - 12ps}{4ps} = 150 = 0\text{x}96$$

E. Raising TRD Line

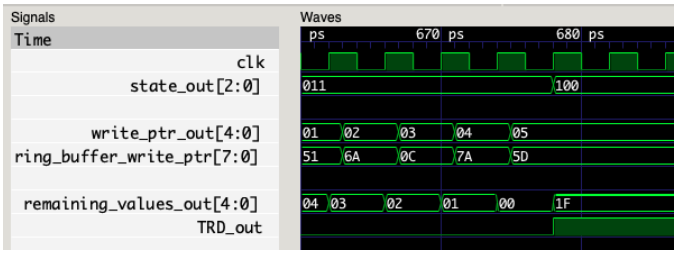

Fig. 7: Raising TRD Line gktwave Output

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

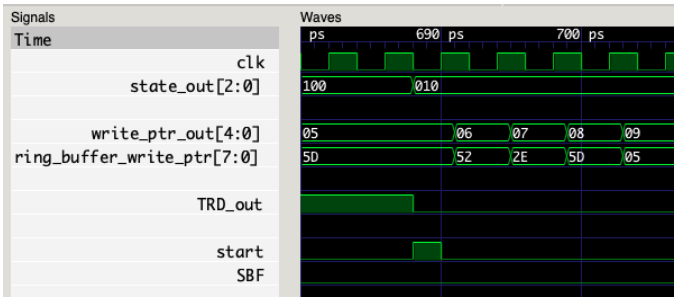

Fig. 8: Starting From Idle gktwave Output

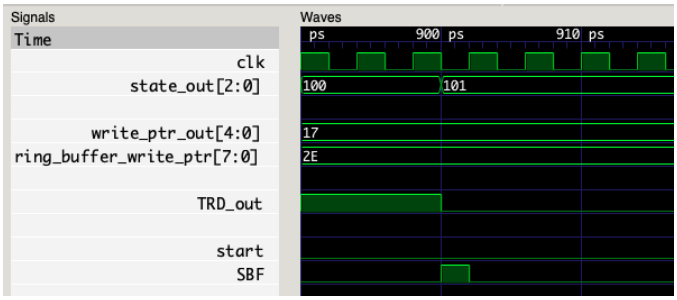

Fig. 9: Transmitting Data From Idle gktwave Output

From the `IDLE` state, the TSC module can either transition back to `RUNNING` state or into `SENDING` state 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

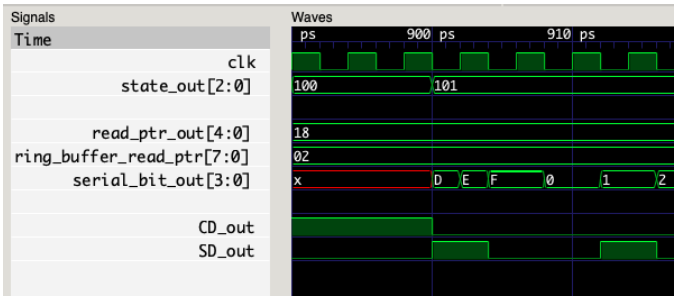

Fig. 10: Starting Data Transmission gktwave Output

As the module transitions into `SENDING` (0b101) state, it immediately pulls CD low and SD high as required; it also sets the serial bit to `WAIT` (0xD). When the serial bit is `WAIT`, it is waiting for the next rising clock edge before it transitions into the `TRANSMISSION_START_BIT` (0xE) which occurs at 902 ps. Once CD has been low and SD has been high for at least one rising clock edge, the first byte can be sent, which is explained in Subsec. III-H.

H. Byte Transmitting

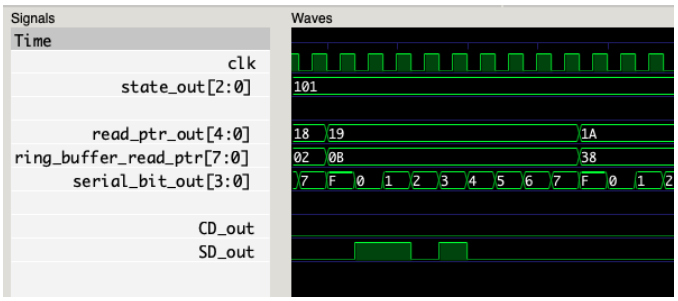

Fig. 11: Byte Transmitting gktwave Output

The HUB module reads data on the rising clock edge, therefore the TSC module must write data on the falling clock edge, which is shown in Fig. 11 by the SD line only changing with the falling edge clock. Additionally, each byte is preceded by a low `START_BIT` (0xF).

Little Endian encoding was chosen for sending the data on the SD line, so reading the data on the rising edge of the clock as the HUB module would give 0-1101-0000. This is equivalent to the value 0b0000-1011 = 0x0B which shows that the TSC module correctly sent the byte in the 19th address of the ring buffer.

I. Ending Data Transmission

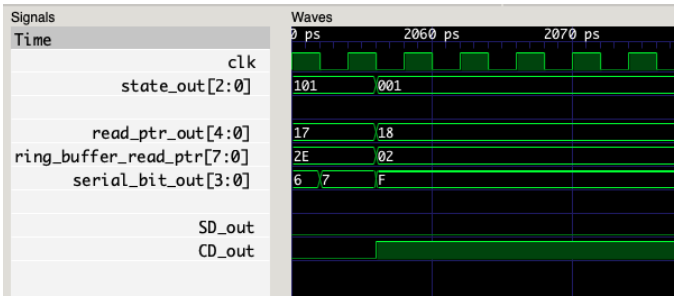

Fig. 12: Ending Data Transmission gktwave Output

Fig. 10 shows that the first byte transmitted was at address 18, and Fig. 12 shows that the byte at address 17 has just been transmitted which means the full 32 byte ring buffer has been transmitted. The TSC module correctly recognises this by dropping the SD line low, pulling the CD line high, resetting the read pointer back to just in front of the write pointer, and finally setting the state back to `READY` (0b001), waiting to receive a start command once again.

IV. Conclusion

The Trigger Surround Cache (TSC) module was successfully implemented and tested. The system was robust and worked as expected in all of the tests conducted. The next steps would be to apply the simulated code and test bench to a physical FPGA and confirm if it works in a real-world application.

V. Recomendations

A few recommendations highlight that there are some code improvements that can be made to maximise efficiency. 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. Further code safety would also be advised to prevent against misuse and edge cases. The tests could also be repeated with differing adc modules.