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 ans Software

This was run on a MacBook Pro computer using Iverilog. Additionally, gtkwave was used to monitor the wires.

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.

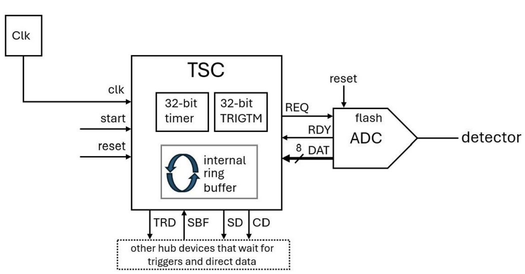

Fig. 1: Block diagram of the TSC

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.

1) State register: The state register is a 3-bit register that has the folowing states:

- 000 Stop : State when the machine is powered on and has not been reset yet.
- 001 ready : State that is enter on the reset pin rising edge and it waits for the start pin rising edge.
- 010 Running : State that is entered from ready or Idle state once the start pin rising edge is pulled hight. It incrementing the timer and wright adc values to the ring buffer.
- 011 Triggered : State entered when the value read from the ADC is greater than the predetermined trigger value (TRIGVL). It capturing the next 16 values.
- 100 IDLE ; This state is entered when a trigger event has occurred and the TSC is waiting for the start pin or SPF pins rising edge.
- 101 SENDING : This state is entered from the IDLE state when the SFB gose high. It indicates that data is being sent on the SD line.

2) 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 store in the TRIGTM register the timer is multiplied by the clock period (4 ps).

3) 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_prt and is Initial set to 5'b11111. and the head pointer is named read_ptr and is initial set to 5'b00000. The 5 bit format for the head and tail index is used to induce role offer 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 proses is repeated until the read_prt = wright_ptr. Indicates that all the values have been read.

4) How the TSC intervacec with the ADC: The ADC is initialized in the TSC module. this is creates the adc_request, adc_request, adc_ready, and DAT wires. The adc_request line is to the main reset line this mean that the ADC is reset when the TSC_td module pulls the reset pin hight. once the ADC is reset the adc_ready line is pulled hight to indicate that the ADC is ready to send data. 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 posedge of the clk line for 1 ps to request data from the ADC. This can be seen in the two code section below.

Listing 1: Code for storing data and moving pointers i 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
```

III. Testing and Validation

This section details the tests that were conducting 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

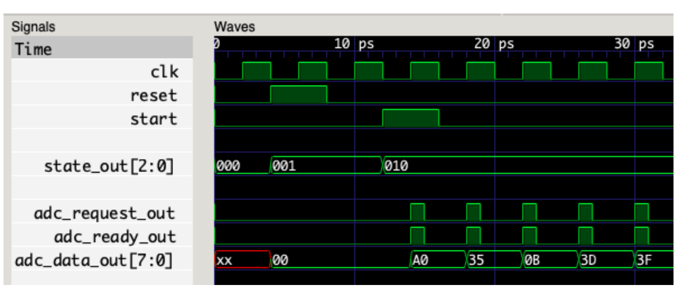

Fig. 2: Resetting and Starting gktwave Output

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 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 begin 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

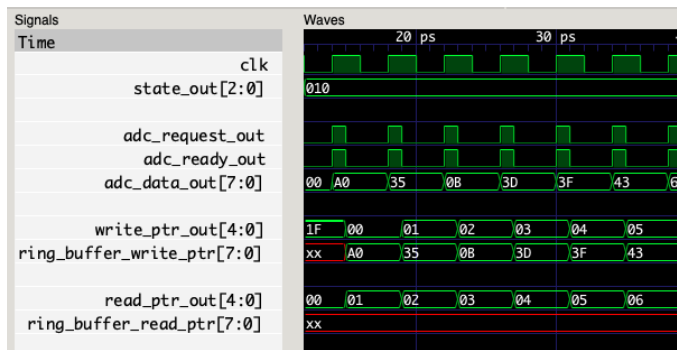

Fig. 3: Ring Buffer Writing gktwave Output

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

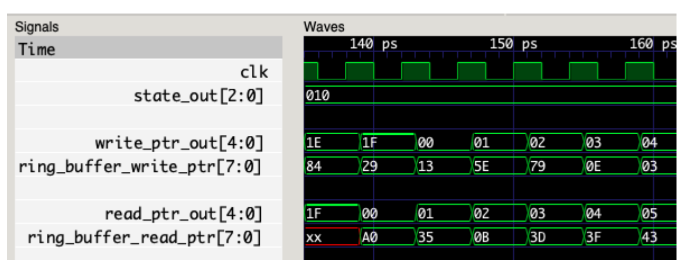

Fig. 4: Ring Buffer Reading gktwave Output

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

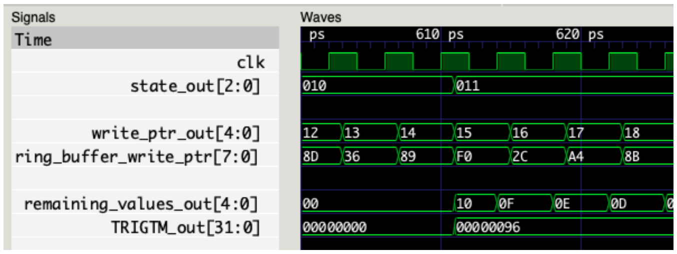

Fig. 5: Triggering gktwave Output

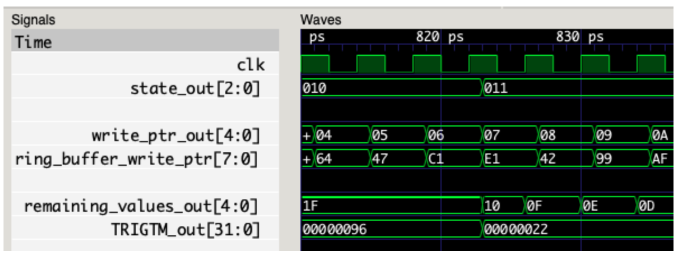

Fig. 6: Re-triggering gktwave Output

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:

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

E. Raising TRD Line

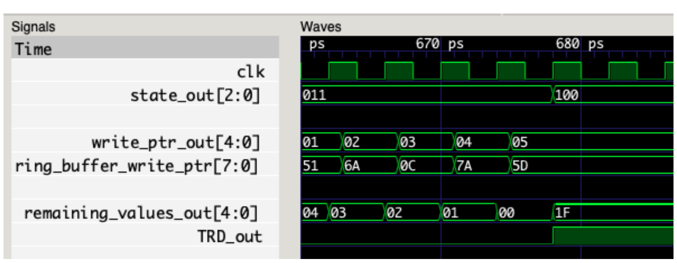

Fig. 7: Raising TRD Line gktwave Output

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

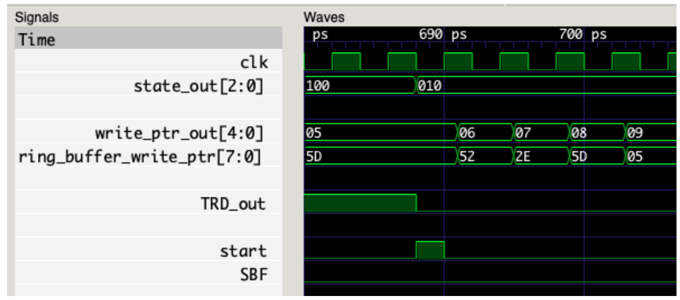

Fig. 8: Starting From Idle gktwave Output

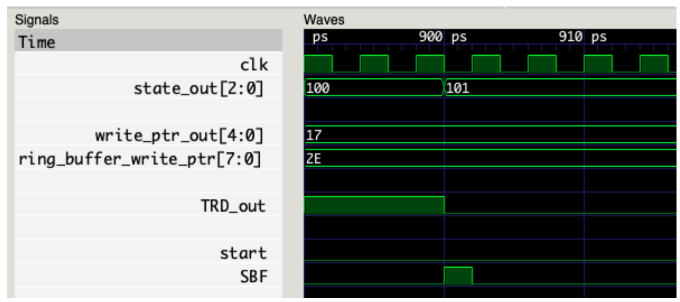

Fig. 9: Transmitting Data From Idle gktwave Output

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

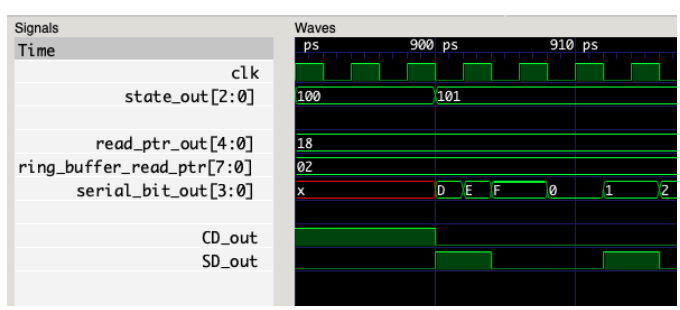

Fig. 10: Starting Data Transmission gktwave Output

H. Byte Transmitting

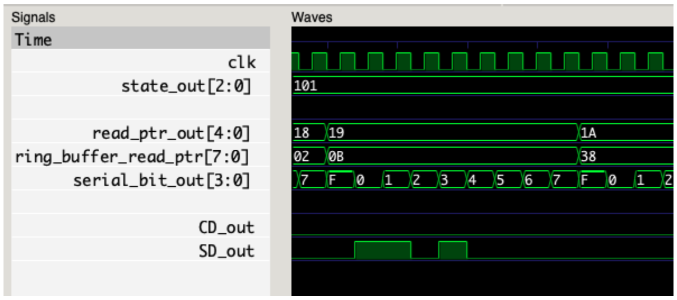

Fig. 11: Byte Transmitting gktwave Output

I. Ending Data Transmission

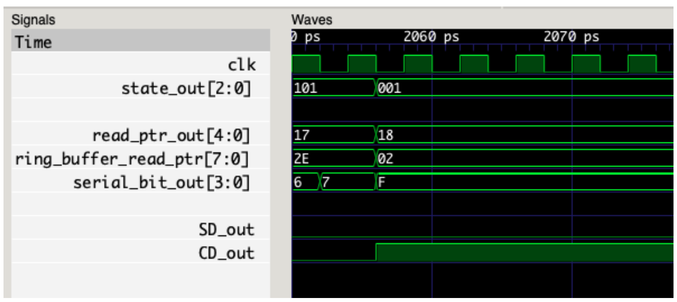

Fig. 12: Ending Data Transmission gktwave Output

IV. Conclusion

This report shows that for multiplying small matrix sizes and counts, single-threaded matrix multiplication is faster, however, when the matrix sizes and counts are large enough, parallelized matrix multiplication becomes faster due to its overhead becoming negligible relative to its computation time.

The overheads for the parallel matrix multiplication program are the OpenCL setup overhead and kernel startup overhead, with the former being constant and the latter increasing linearly with matrix count.

Lastly, there is an unexplained decrease in speed up for small matrix sizes and matrix counts over 60 which is theorised to be caused by the matrix arrays allocation and transfer between the heap, stack and cache, however this hypothesis still requires further testing to confirm.