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 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.

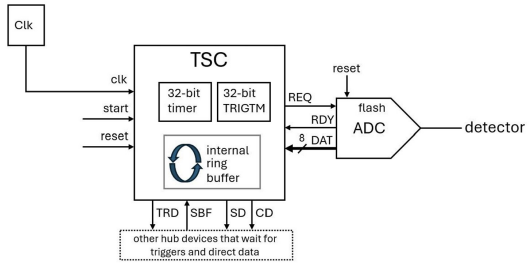

Fig. 1: Block diagram of the TSC

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.

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

- STOP (0b000) : State when the machine is powered on and has not been reset yet.
- READY (0b001) : State that is enter 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 hight. It incrementing the timer and wright 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 it pulled 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 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 posedge 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 1 and 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
```

5) 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 (state != `TRIGERED) begin  //if it hasn't been triggered already, check for a valid trigger
    if (adc_data > TRIGVL) begin
      state = `TRIGERED;
      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

```
  `TRIGERED: begin
    timer++;
    if(remaining_values--) begin //check that there are remaining values left to capture and decrement.
      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
```

6) Sending data to to HUB devices: If the TSC module is in the IDLE state and the SBF line is pulled hight the TSC module will transition to the SENDING state. durint this transition it will set SD hight and CD low

III. Testing and Validation

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.