Homework 6

Matt Hartnett, ECEN 5863

Q1)

A) For this problem, I created a Verilog module that enables synchronization between different clock domains by coordinating data and signals between a source and sink finite state machine. It includes components for data transfer, request, and acknowledgment synchronization. The module manages an 8-bit data output (Dout) and facilitates communication between the state machines by synchronizing requests and acknowledgments (req_source, ack_source) to their respective clocks (clkB and clkA). I implemented this design as a hierarchy, with a top level module that handles the data source and sink, as well as the synchronization of requests and acknowledgements.

   The finite state machines are separate modules that operate off of similar principals. Each has three states: an IDLE, a SENT, and a WAIT. In IDLE, the state machines wait until a data transfer is ready to occur, then they register the data and move into the SENT state. The SENT state sets the acknowledge and request signals to high and then moves to the WAIT state, which waits until the opposite state machine outputs the correct response.

   The whole module operates with an asynchronous, active high reset and a synchronous active high enable. The request and acknowledge signals are both synchronized to the opposite clock domain using a 2FF synchronizer circuit.

   The fMax for this design is: 410.678MHz (clkA) and 385.505 (clkB), as shown in figure 1. A simulation screenshot showing the five data transfers is shown in figure 2. (Note: the first data transfer is 0x0, which doesn't show up well on the simulation, but does occur as expected)

   Out of the 5 data transfers, 5 of them were correct.
   By constraining the design, I actually got a lower set of fMaxes, presumably since the input/output delay was no longer 0ns.

B) For this section, I duplicated the project from A, and I only had to modify the constraints and simulation testbench to switch the frequencies of the clocks.
   The fMax for this design is: 410.678MHz (clkA) and 385.505 (clkB), as shown in figure 3. A simulation screenshot showing the data transfers is shown in figure 4.

   Out of the 5 data transfers, 5 were correct.
   Constraining the design did not improve performance.

Q2) For this design modification, the first step was finding when the FIFO would be full versus empty. Due to the pointer logic, we can know that the FIFO isn't full or empty when the values of the read and write pointer don't match. Looking at the remaining cases, where they're either both 0 or both 1, it's clear that the FIFO is full if both pointers are 1 and empty if both pointers are 0. The next step is synchronizing the pointers across the clock domains. Since the both signals are required for both full and empty, both pointers need to be synchronized. I placed the synchronizing logic (2FF sync) inside of the processes that calculated the empty and full signals. This was done using the d1 and d2 signals already instantiated in the code.

I then simulated the design, testing the full and empty signals by writing in 2 memory pieces, reading 2 out, writing in 1, and reading that 1 out. This is the full gamut of possible outcomes from this FIFO since it only has 4 possible states.
The simulation that shows the writing of 2 is shown in figure 5.You can observe the empty signal going high once data is written in, then the full signal going high as well once the second piece of data is in.
The simulation that shows the writing of 1 value is shown in figure 6. In this simulation, you can see that the empty signal goes low for the time period where there is data in the FIFO, but when the FIFO is read out, it goes back high. Additionally, the full signal stays low, since 1 data value isn't enough to trigger a full FIFO.

Note: I did not simulate the other behavior of the FIFO, I assumed the rest of the design was created correctly.

Q3) To accurately find the power, I created the verilog design that generated 32 counters, each 16 bits wide. The final bits of all of these counters then fed into a cout register. I verified that the design was built as expected by opening the RTL viewer and checking that none of the design was optimized out. This allowed me to see how many FFs and LUTs were being used, which is necessary to figure out the power consumption from the logic elements. I found that there was 512 FFs (32 counters, at 16 FFs each) and 547 LUTs (32 adders for each counter, 32 for the output, 3 for clock and reset). I then took the Early Power Estimator tool and input the logic and IO to find a total power consumption of 0.253W, as shown in figure 7.

In that same figure, the tool estimates that the junction temperature is 27.4 degrees C, well below the maximum allowed 82.2 degrees. Since the temperature is so low, no heat sink is required.

Appendix

Summary

Clock Domain	Period (ns)	Frequency (MHz)
clkA	2.435	410.678
clkB	2.594	385.505

Fig 1. The fMax for the handshake synchronizer design with a slow clkA.

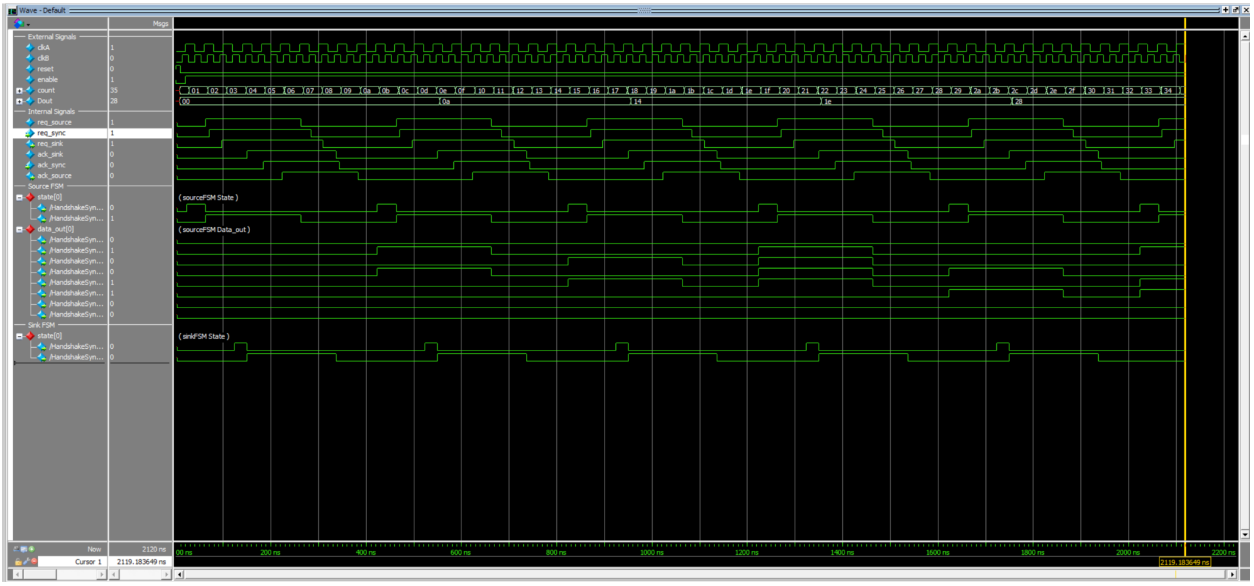

Fig 2. The simulation waveform of the handshake synchronizer design with a slow clkA.

Summary

Clock Domain	Period (ns)	Frequency (MHz)
clkA	2.435	410.678
clkB	2.594	385.505

Fig 3. The fMax for the handshake synchronizer design with a fast clkA.

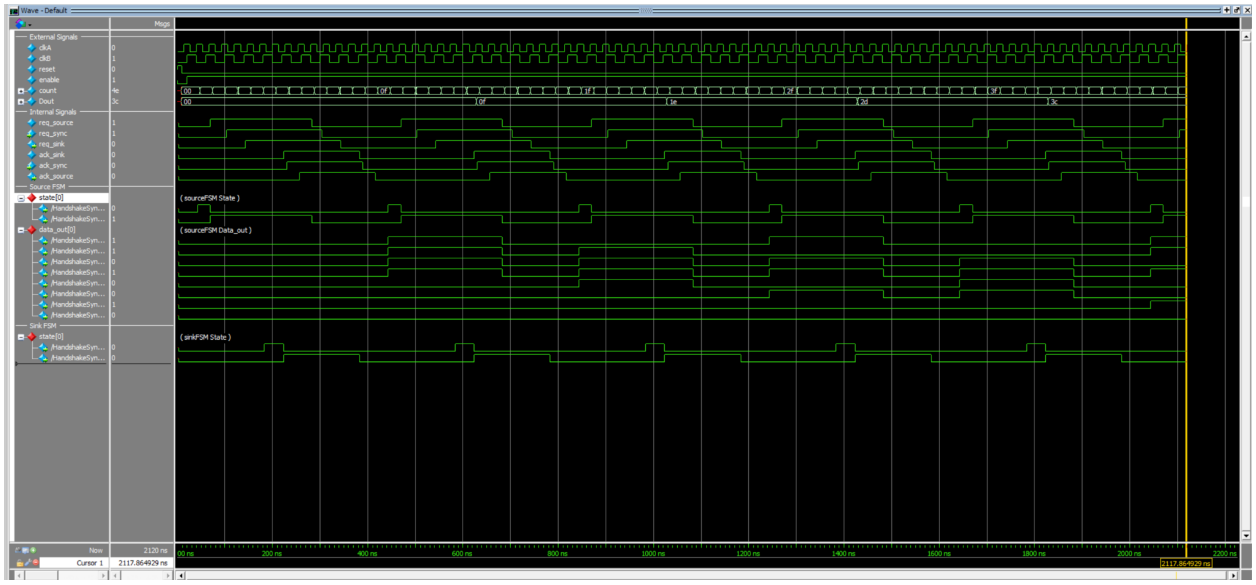

Fig 4. The simulation waveform of the handshake synchronizer design with a fast clkA.

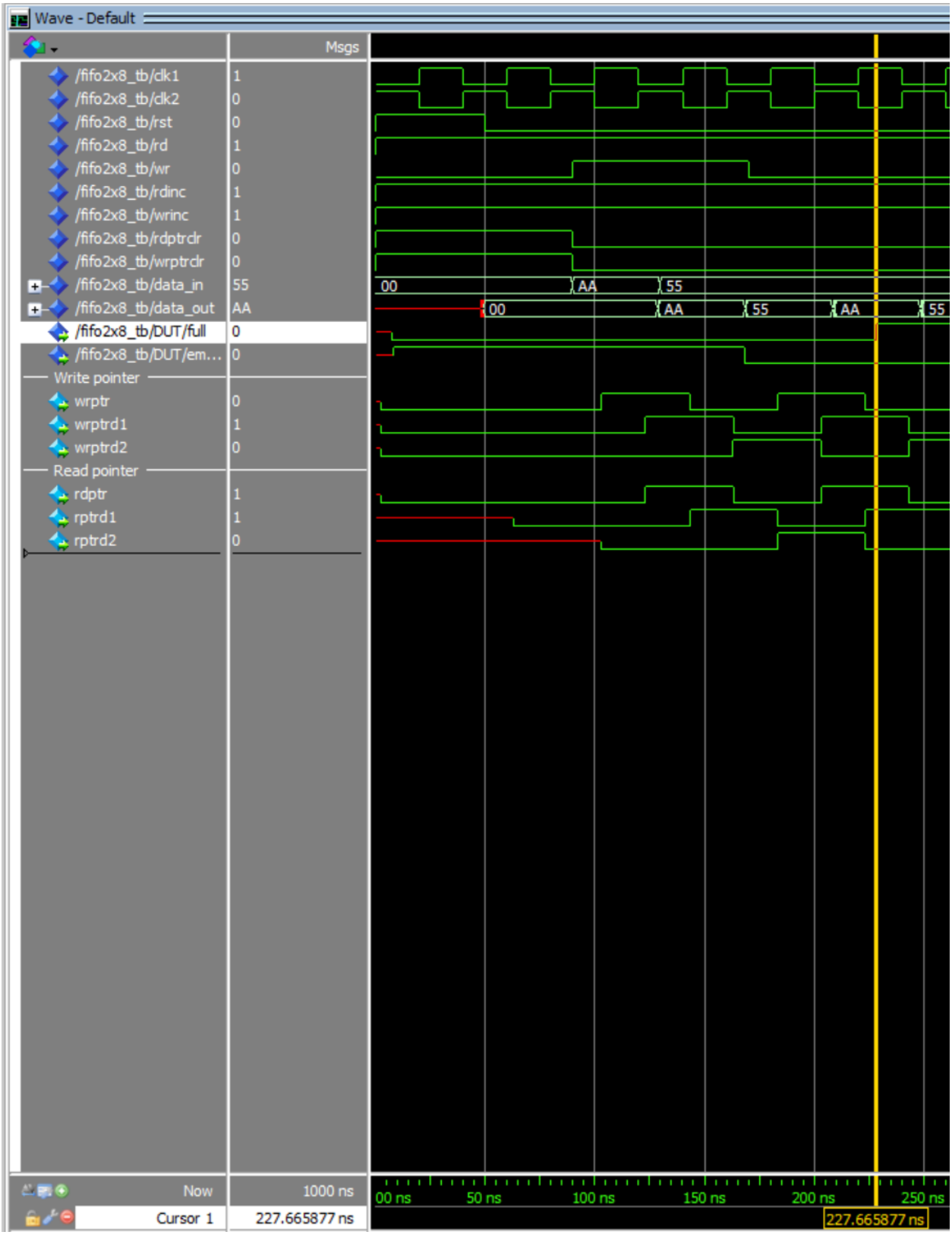

Fig 5. The simulation waveform of the 2x8 FIFO showing the 2 data values being written in.

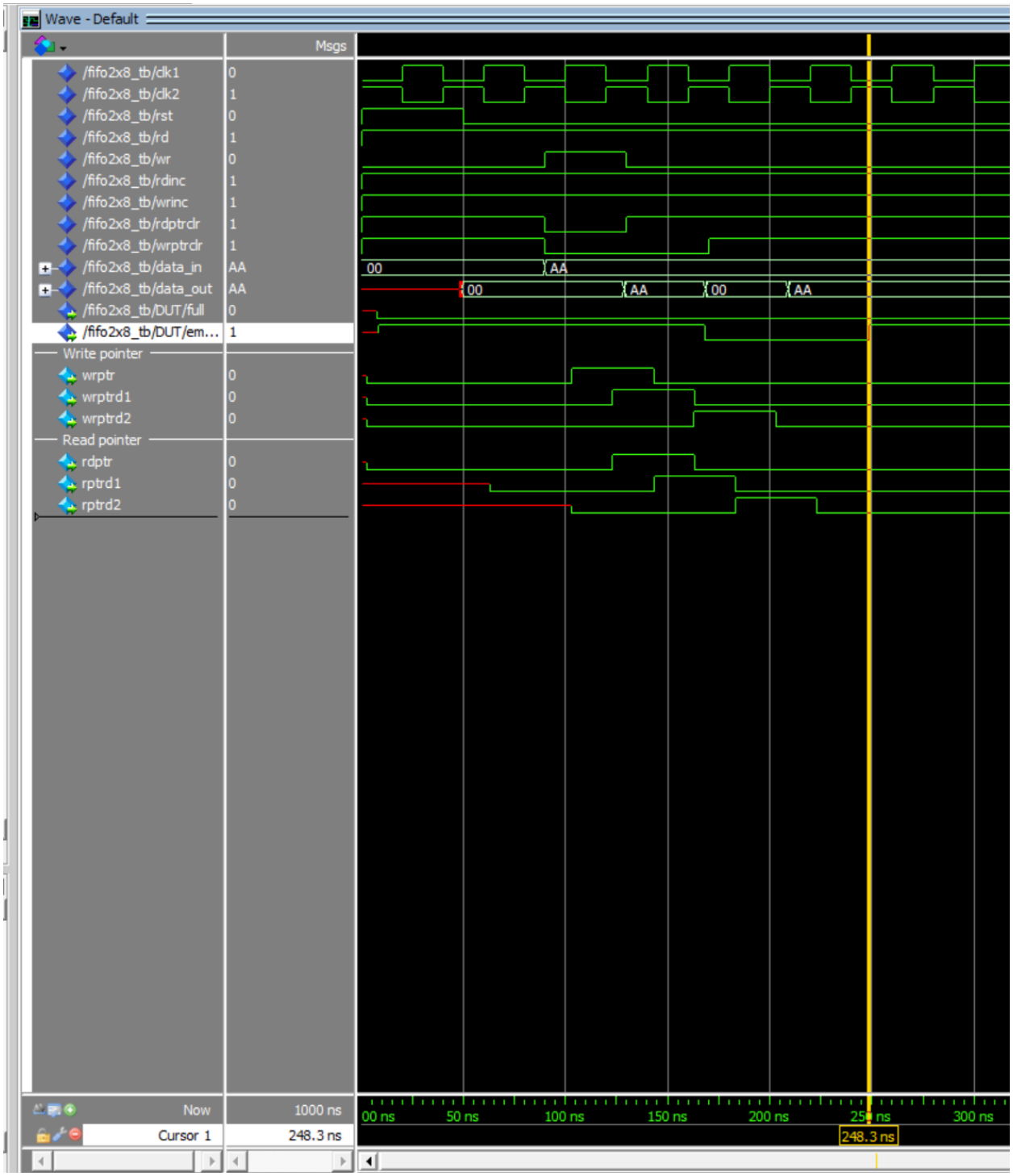

Figure 6. The simulation showing 1 data value being written in.

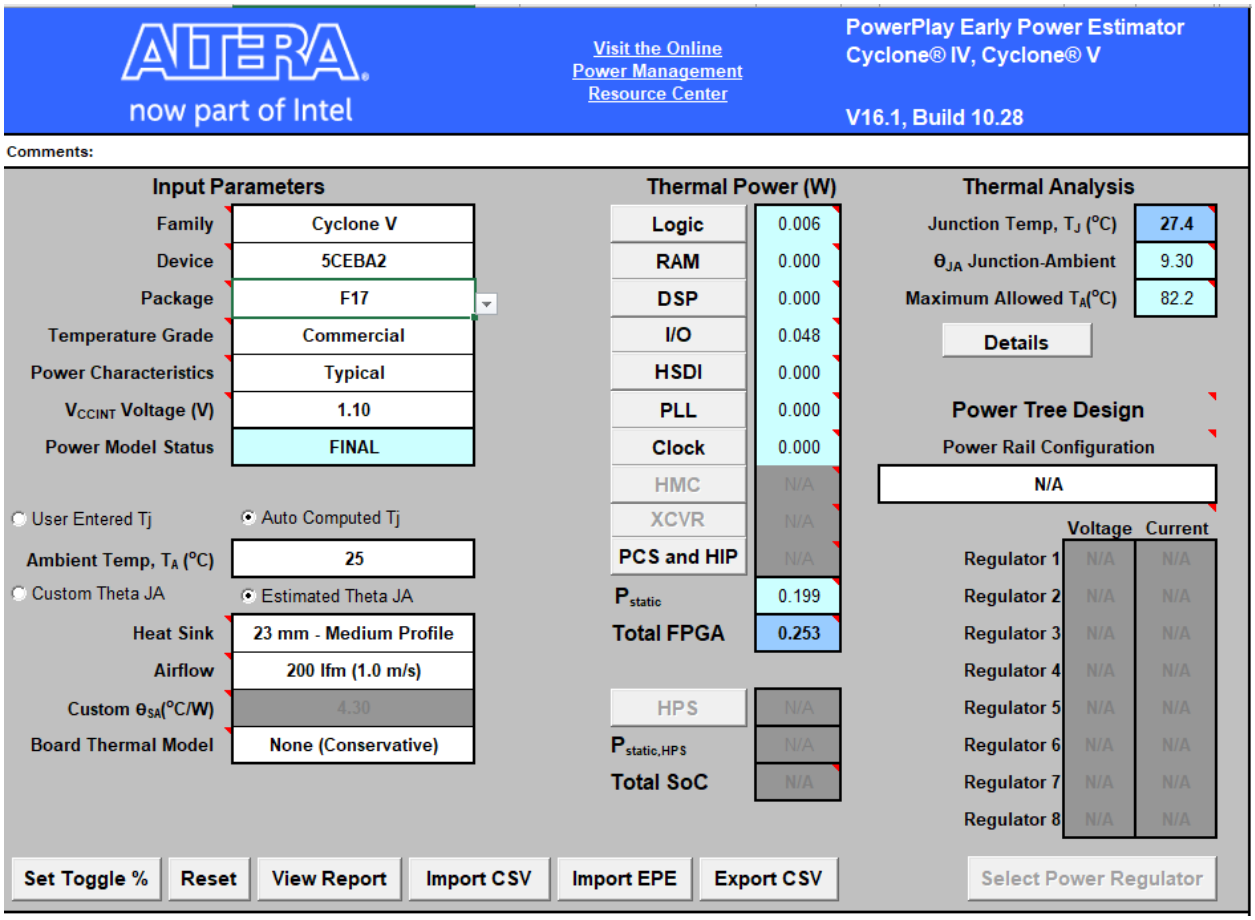

Fig 7. The Altera power estimator tool summary with the correct values for the design.