University of York

EMBS

Embedded Systems Design and Implementation

Open Examination 2

Examination number:

Y3606797

Contents

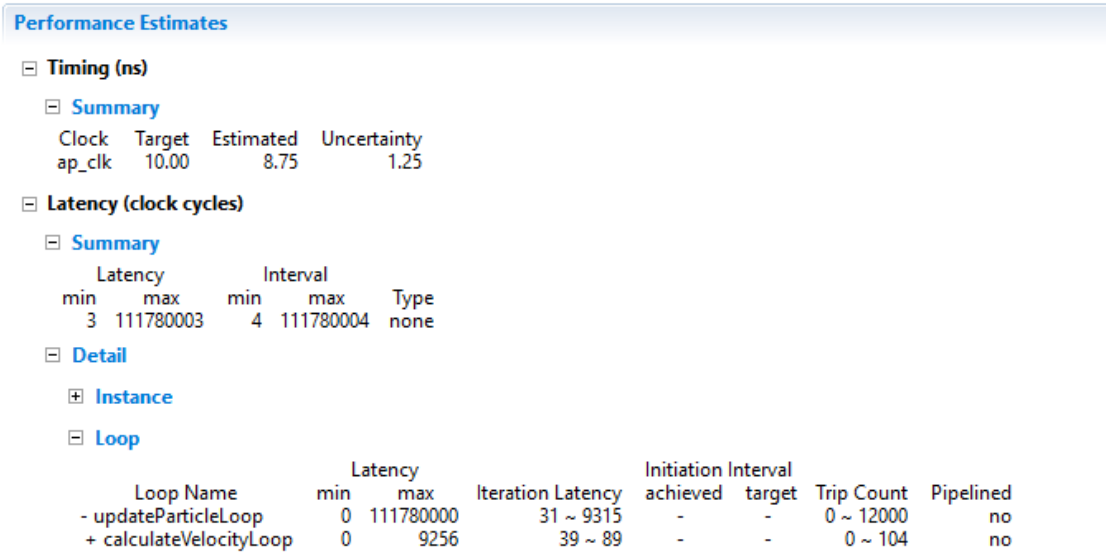
1 Design

The main feature of my design is to split the loops responsible for updating the particles into a piece of hardware using HLS. This loop is the main computation for the simulation and thus it makes sense to optimise it's calculation through custom hardware. Which is what I intended to do with my design iterations. I introduced pipe-lining directives to the inner loop in order to cut the number of clock cycles needed to perform the particle updates. I implemented the pipelined structure within HLS and the result was to cut the maximum latency by a factor of 16, as seen in fig. 1 and fig. 2.

This design was exported to Vivado and used within the SDK but unfortunately I was unable to get the desired performance from the custom IP, software was outperforming it. Therefore I stuck with the software calculations for the particle updating as I could not put together a working hardware accelerator that actually made a difference in time.

Another loop to optimise within the simulation would be the particle collisions with walls, which I haven't implemented. Again I would attempt to extract some parallelism from the loop by pipe-lining or unrolling.

Other design decisions intended to increase performance include: using a fast approximation of the square root function, optimising the length of my loops and other small optimisations.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.75	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
3	111780003	4	111780004	none

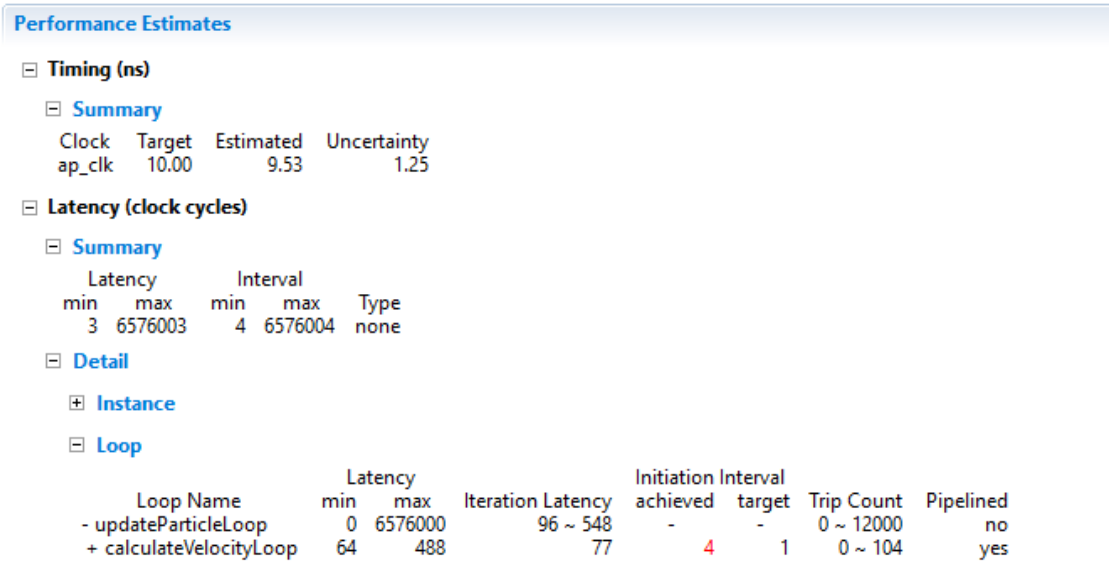
Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- updateParticleLoop	0	111780000	31 ~ 9315	-	-	0 ~ 12000	no
+ calculateVelocityLoop	0	9256	39 ~ 89	-	-	0 ~ 104	no

Figure 1: Before adding any directives



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.53	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
3	6576003	4	6576004	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- updateParticleLoop	0	6576000	96 ~ 548	-	-	0 ~ 12000	no
+ calculateVelocityLoop	64	488	77	4	1	0 ~ 104	yes

Figure 2: Examples of parallelism within HLS component

2 Parallelism

The design makes use of parallelism through the use of pipeline directives in the HLS component. A pipeline allows for parallel execution of what would be otherwise sequential operations in some cases. This can be seen in fig. 3, the blocks in the same column are operations which are happening in parallel. My two main loops in the particle simulation code do not execute in parallel because their loops do not overlap. Parallelism is still exploited within the loops, but the two loops have a dependence which limits the extent of their parallelism.

Further from this, if we have enough space on the hardware, we could introduce multiple IP blocks to perform a function on different parts of the data in parallel. I explored this option by making two blocks and splitting the memory between them but I couldn't get a working implementation in time.

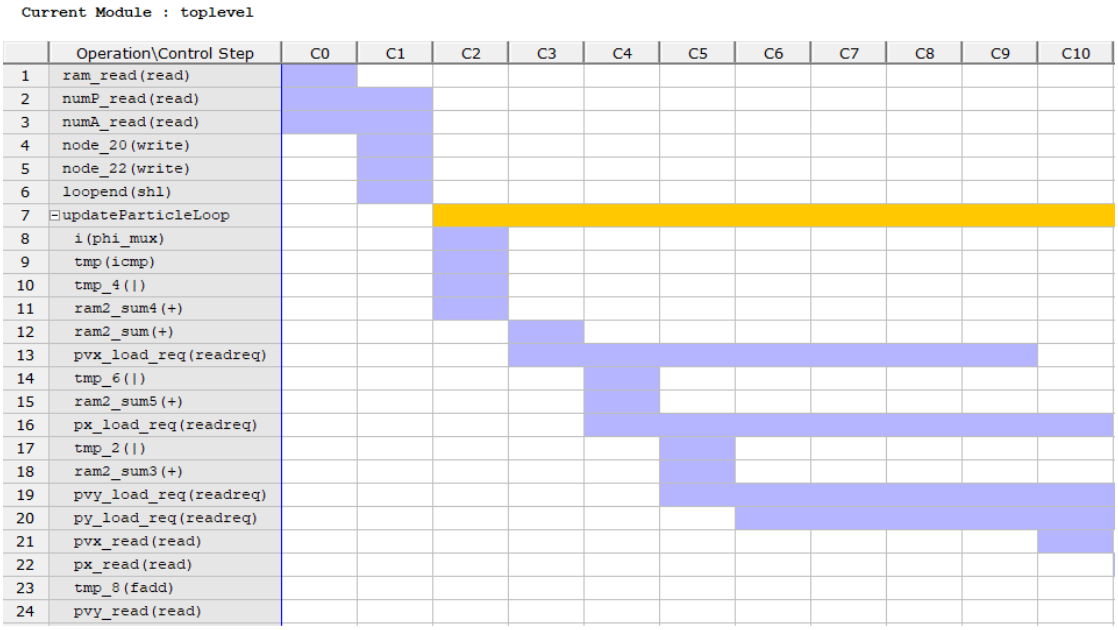


Figure 3: After synthesis with pipeline directives

3 Testing

As far as testing the hardware goes, I implemented a simple testbench to verify that my code was performing as expected. In the test bench I populate dummy arrays with dummy data to feed into the toplevel function, then I check that the results are as expected. That formed the majority of the testing with regards to the hardware elements of my system, it is important to test the functionality of your toplevel function because the synthesis of the hardware takes a long time.

To test the rest of the system, I implemented various short-cut keys which can be sent through the serial port to the processor. These short cuts perform the various functions of the particle accelerator fig. 4.

Char input	Program function
e	Start Ethernet prompt
r	Start random scenario prompt
f	Toggle FPS output to uart

Figure 4: Short-cut keys and functions

4 Evaluation

The design decisions discussed and attempted in my implementation are focused on pure performance enhancements to the particle system, disregarding the usage of memory or hardware on the board in favour of performance.

The utilisation of the board is shown in fig. 5, this is the utilisation for the above pipelined examples fig. 2. My data array was too big to be stored directly in BRAM so I was directly accessing the memory via the port, which results in slower performance overall. To avoid this I could feed in blocks of data at a time to process and return, but I didn't have the time to attempt this for the assessment.

The LUT utilisation was low enough that I had a lot more room for moving functionality from software to hardware, such as wall collisions or perhaps drawing to the frame buffer for the VGA output. Or I could explore using multiple blocks of IP to extract more parallelism from my hardware.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	621
FIFO	-	-	-	-
Instance	2	10	2590	4082
Memory	-	-	-	-
Multiplexer	-	-	-	1381
Register	-	-	1854	99
Total	2	10	4444	6183
Available	120	80	35200	17600
Utilization (%)	1	12	12	35

Figure 5: Board resource utilisation