

UNIVERSITY OF YORK

EMBS

EMBEDDED SYSTEMS DESIGN AND IMPLEMENTATION

Open Examination 2

Examination number:

Y3606797

Contents

1	Design	2
2	Parallelism	4
3	Testing	5
4	Evaluation	5

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 loops 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.

Due to my unsuccessful attempts to implement my designs, I don't have much functionality implemented in hardware. An initial problem was that my data was too large for the available BRAM on the board. To get around this I attempted a design which would process blocks of particle data for all the attractor data, thus ensuring that at any time there is enough BRAM to store the data being processed. I believe this design would have worked with the correct checks in place in the SDK to ensure the correct data is updated.

The same principles of pipe-lining would be applied to the processing of the blocks with the aim to maximise the performance of the system.

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		Type
min	max	min	max	
3	111780003	4	111780004	none

[-] Detail

[+] Instance

[-] Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- 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		Type
min	max	min	max	
3	6576003	4	6576004	none

[-] Detail

[+] Instance

[-] Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- 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.

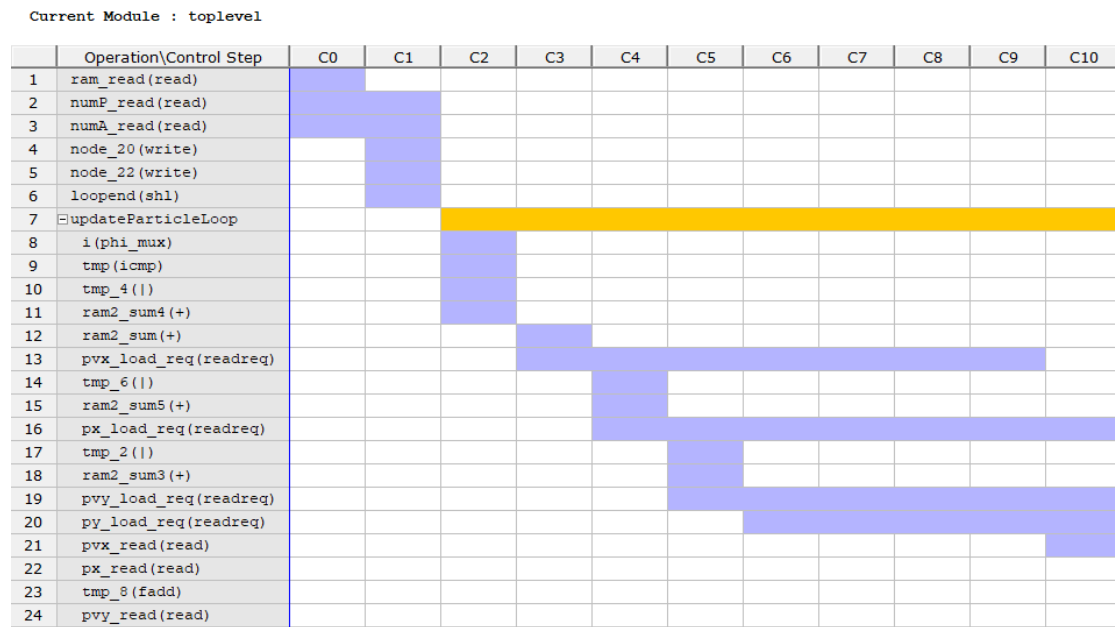


Figure 3: After synthesis with pipeline directives

3 Testing

As far as testing the hardware goes, I implemented a simple testbench to verify that my code was performing as expected. I populate dummy arrays with dummy data to feed into the toplevel function, then checking that the results are as expected. That formed the majority of the testing with regards to the hardware elements of my system.

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 was a poor performance decision overall. The other designs I have discussed were designed to avoid this occurrence.

The LUT utilisation was low enough that I had a lot more room for moving functionality from software to hardware, or I could introduce some parallelism by using multiple IP of the same type of different parts of my particle system data.

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