TCP/IP in hardware using SME

Mark Jan Jacobi & Jan Meznik

KU

September 19, 2019

TCP/IP in hardware using SME

Mark siger introduktion og 2-3 saetninger "abstrakt"

Table of Contents

Table of Contents

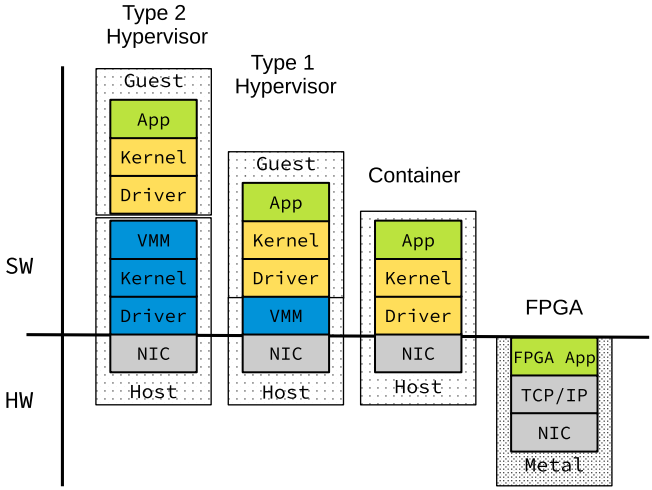
1. Introduction
2. Implementation
3. Evaluation
4. Discussion
5. Conclusion
6. Future Work
7. Questions
8. Demonstration

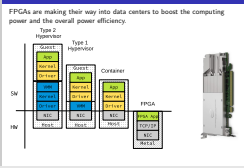
Table of Contents

Table of Contents

Background and Motivation

FPGAs are making their way into data centers to boost the computing power and the overall power efficiency.





Applikationer og Big-Data udregninger flytter til Cloud, drevet af store data centre.

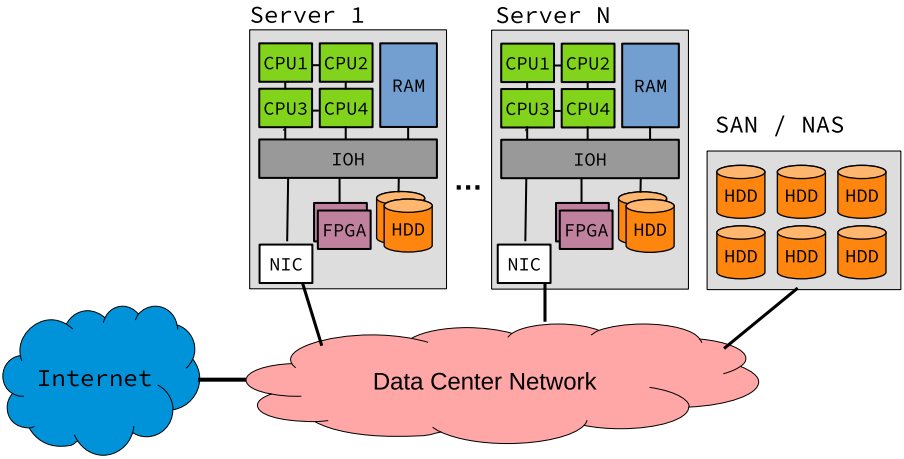
Disse data-centre kraever rigtigt meget plads, store maengder af stroem og er i stigende grad svaere at vedligeholde og udvide.

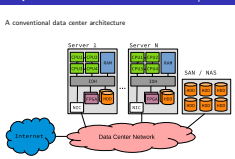
De fleste data-centre er derfor begyndt at aflaste beregningerne til FPGAer, som fjerner meget af overhead til beregningerne

FPGA bruges til at få en computer til at køre hurtigere hvis de mest brugte instruktioner, skrives direkte ned i hardwaren

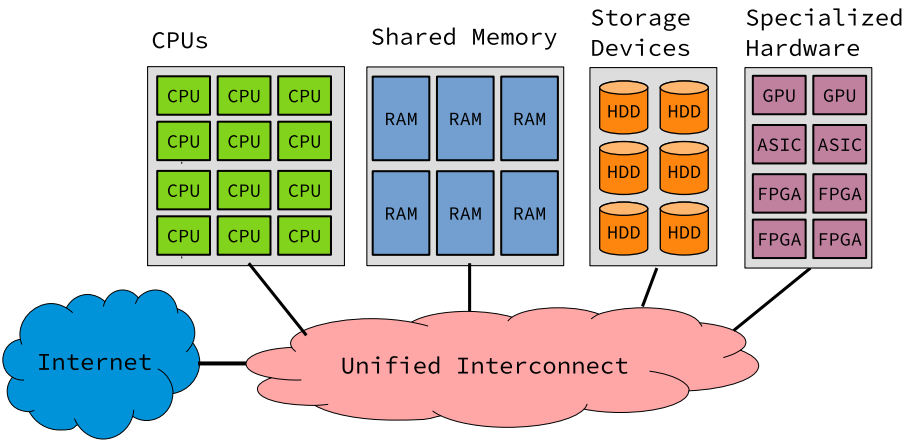
PROBLEMET er at der kun kan vaere en begreanset antal af FPGAer i konventionele servere

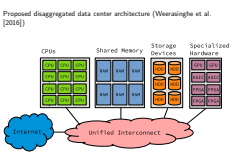
A conventional data center architecture





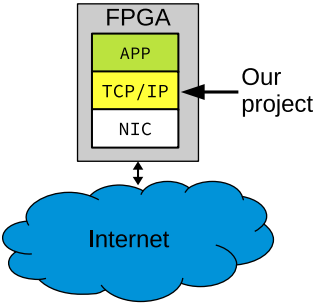
Proposed disaggregated data center architecture (Weerasinghe et al. [2016])





Hvis man splitter resourcerne op, kan man takket været FPGA få bedre ydeevne på det samme areal, samt nemmere håndtering af servere og deres komponenter.

FPGA usage



FPGA usage

The Internet

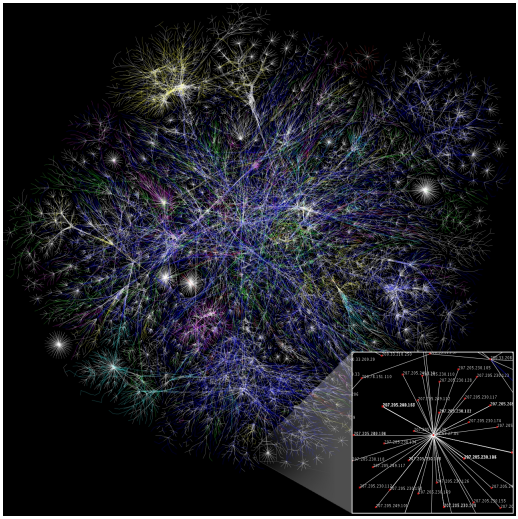


Figure: Map of about 30% of the accessible the endpoints on the Internet

The Internet

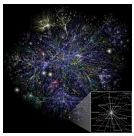
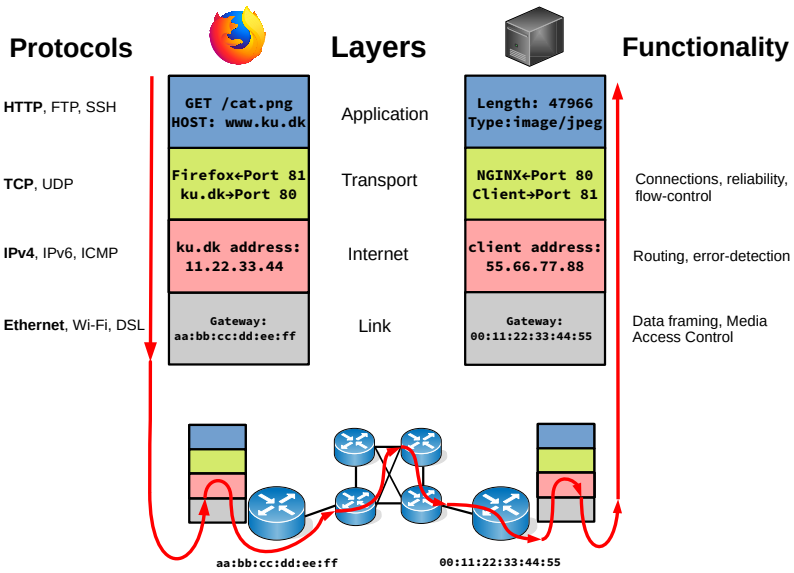
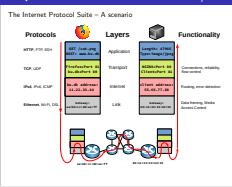


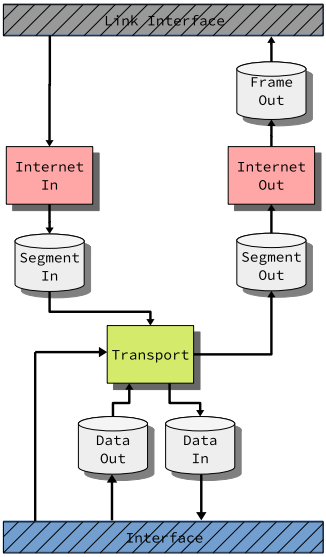
Figure: Map of about 30% of the accessible the endpoints on the Internet

The Internet Protocol Suite – A scenario





Design with the 4 layers in mind



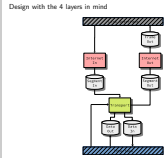
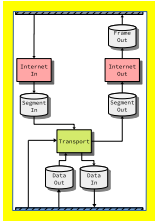


Table of Contents

Table of Contents

1 Introduction
2 Implementation
- SME introduction
- Processes
- Buffers
- Interface signal protocol

3 Evaluation
4 Discussion
5 Conclusion
6 Future Work
7 Questions
8 Demonstration



SME(Synchronous Message Exchange) introduction

- Processes and Busses
- Higher abstraction
- Handling of clocks
- Easy testing
- Not fully feature complete with C#(No threads, no allocation)

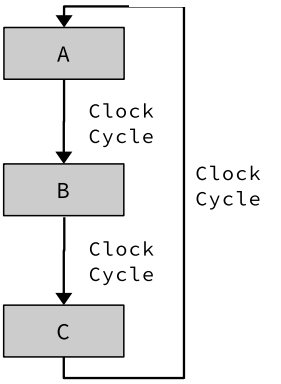
Implementation
SME introduction
SME(Synchronous Message Exchange) introduction
- Processes and Busses
- Higher abstraction
- Handling of clocks
- Easy testing
- Not fully feature complete with C#(No threads, no allocation)

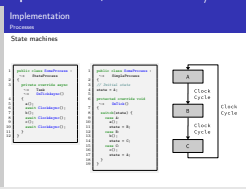
- What is a bus and a process
- No VHDL code
- Clocks abstracted away behind the management of processes and busses
- Testing straight in the simulator, but also in afterwards in the GHDL compiler, via an clock lookup table
- Since not feature complete, only simple structures can be used. We choose state diagrams since they are possible to make, and easy to understand

State machines

```
1  public class SomeProcess :
   ↪    StateProcess
2  {
3    private override async
       ↪    Task
       ↪    OnTickAsync()
4    {
5      a();
6      await ClockAsync();
7      b();
8      await ClockAsync();
9      c();
10     await ClockAsync();
11   }
12 }
```

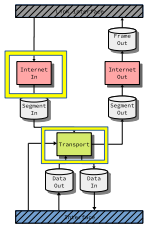
```
1  public class SomeProcess :
   ↪    SimpleProcess
2  {
3  // Initial state
4  state = A;
5
6  protected override void
   ↪    OnTick()
7  {
8    switch(state) {
9      case A:
10       a();
11       state = B;
12     case B:
13       b();
14       state = C;
15     case C:
16       c();
17       state = A;
18   }
19 }
```



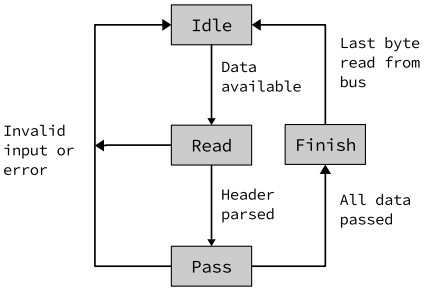


State machines

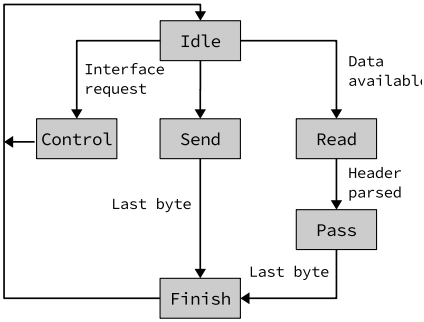
- `StateProcess`
  Eksekvering kan stoppes når som helst(i bidder)
- `SimpleProcess`
  Run er en clock altid, state machine håndteres med en switchcase.
  Algoritme kan splittes op i flere bidder, men kræver en state per bid



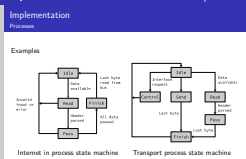
Examples



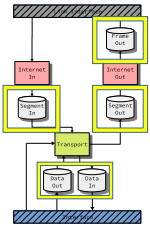
Internet in process state machine



Transport process state machine



- Gå igennem state diagrammer
- Snak om grundlaget for de forskellige typer brug

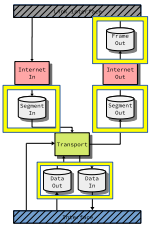


Why buffers?

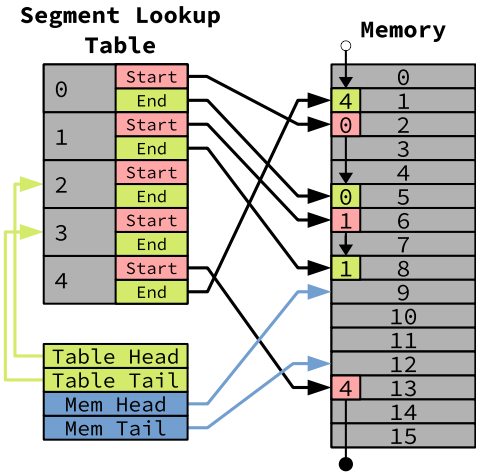
- Fixes segmentation
- Processes can get data at their leisure

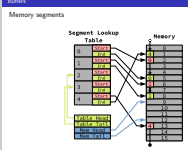
Implementation
Buffers
Why buffers?
- Fixes segmentation
- Processes can get data at their leisure

Hvorfor bruer vi buffers?

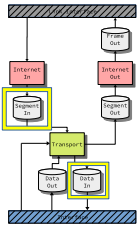


Memory segments

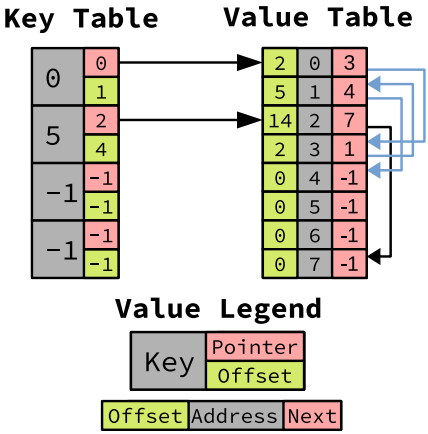




- Reason behind?
  - Segment handling
  - References to other segment to concatting of segments later



Memory dictionary

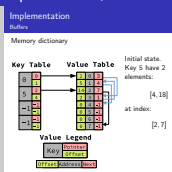


Initial state.
Key 5 have 2 elements:

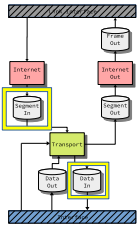
$$[4, 18]$$

at index:

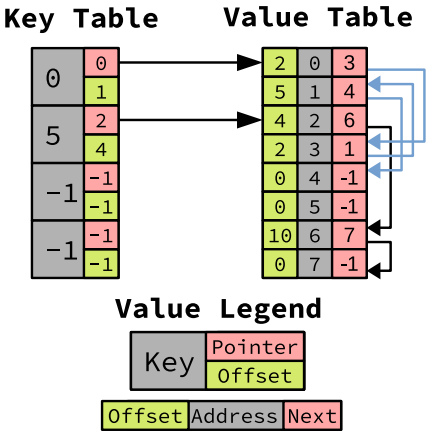
$$[2, 7]$$



Snak om input



Memory dictionary

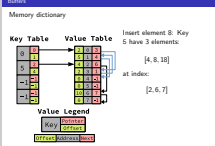


Insert element 8: Key 5 have 3 elements:

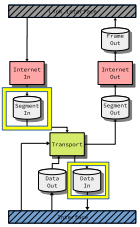
$$[4, 8, 18]$$

at index:

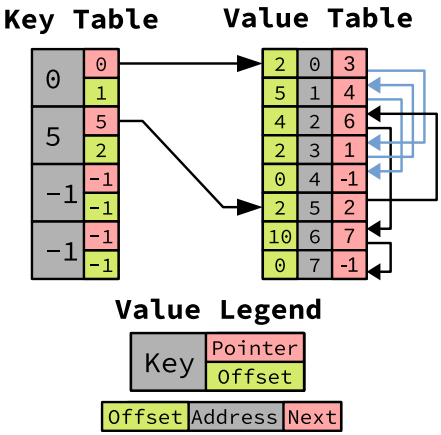
$$[2, 6, 7]$$



Snak om input



Memory dictionary

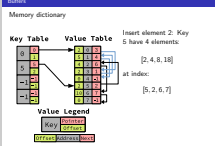


Insert element 2: Key 5 have 4 elements:

$$[2, 4, 8, 18]$$

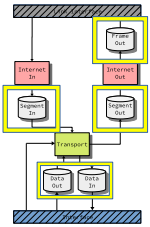
at index:

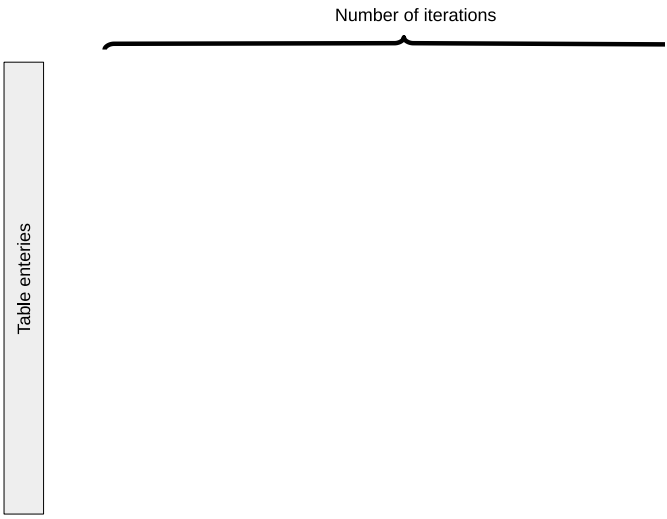
$$[5, 2, 6, 7]$$

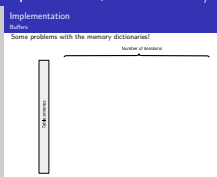


Snak om input

Some problems with the memory dictionaries!



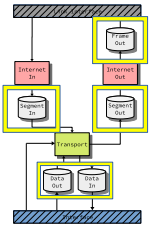


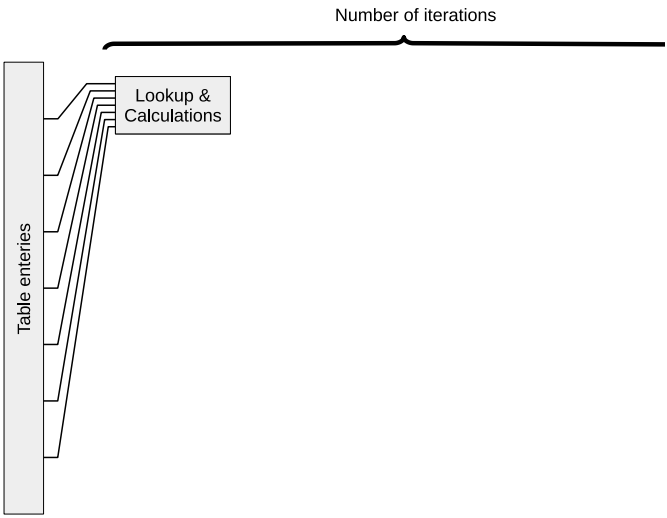


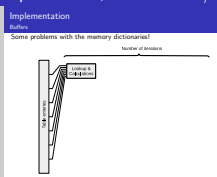
Overflow!
kan læses ved:

- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?

Some problems with the memory dictionaries!

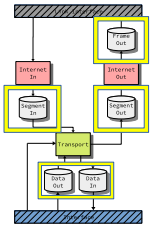




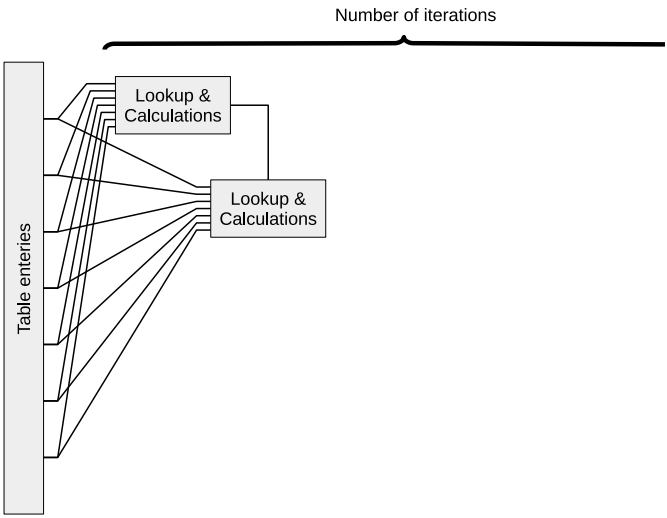


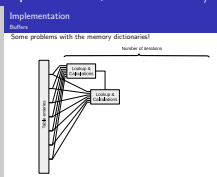
Overflow!
kan læses ved:

- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?



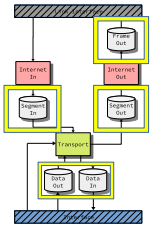
Some problems with the memory dictionaries!



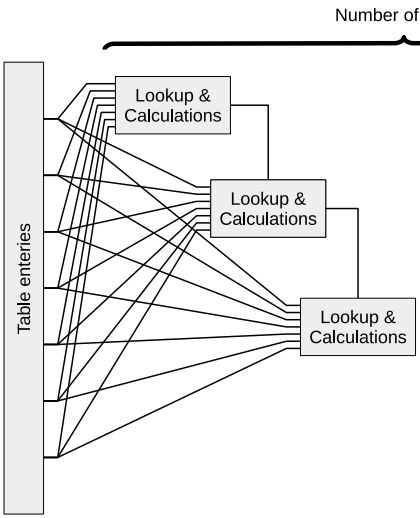


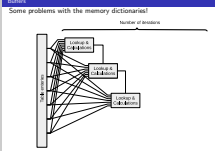
Overflow!
kan læses ved:

- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?



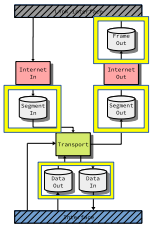
Some problems with the memory dictionaries!



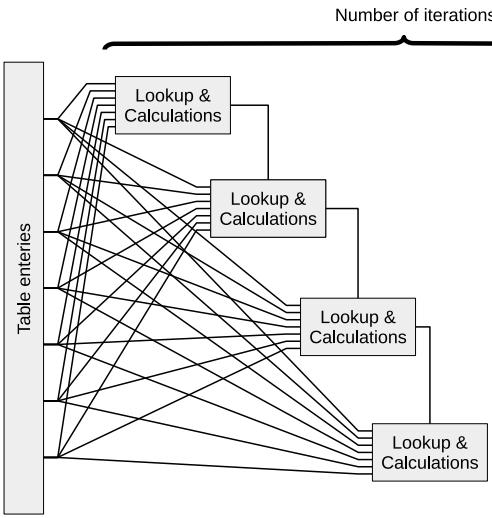


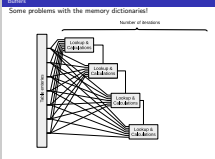
Overflow!
kan læses ved:

- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?



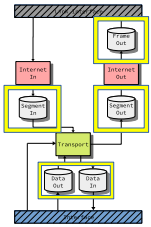
Some problems with the memory dictionaries!



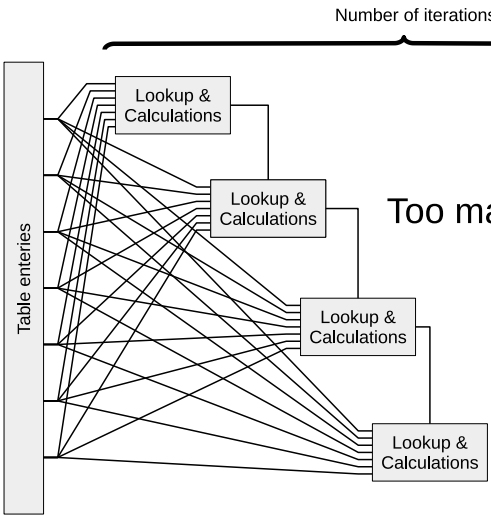


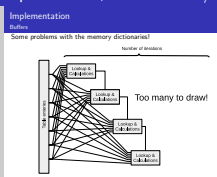
Overflow!
kan læses ved:

- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?



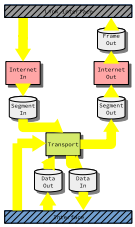
Some problems with the memory dictionaries!



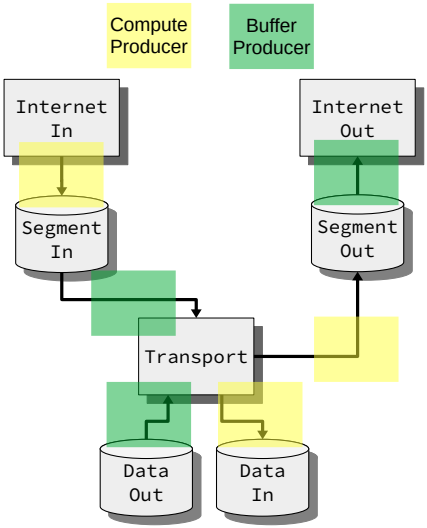


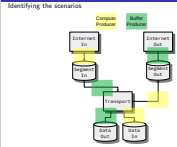
Overflow!
kan læses ved:

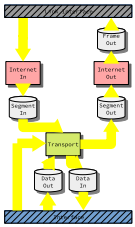
- Kør løkken en gang per clock
- Brug en anden model end en linked list, måske et fast offset?

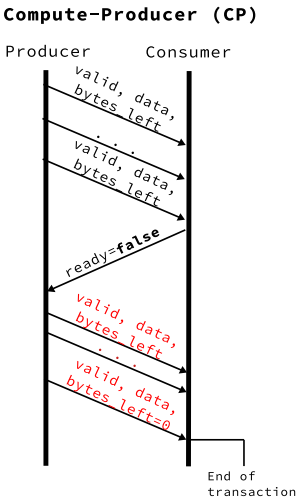


Identifying the scenarios

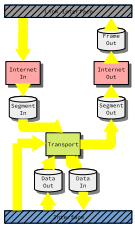






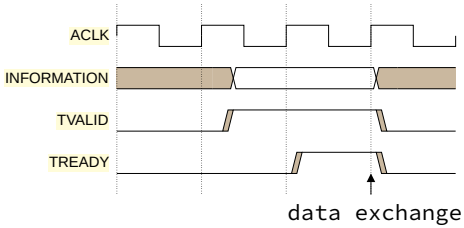


Implementation
Interface signal protocol

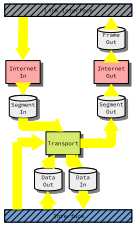


Buffer-Producer: Inspired by AXI4

- Single clock offset when sending data.
- Indicate end of stream with `bytes_left`.



Implementation
Interface signal protocol
Buffer-Producer: Inspired by AXI4
Single clock offset when sending data.
Indicate end of stream with bytes_left.
ACLK
INFORMATION
TVALID
TREADY
data exchange



Buffer-Producer (BP)

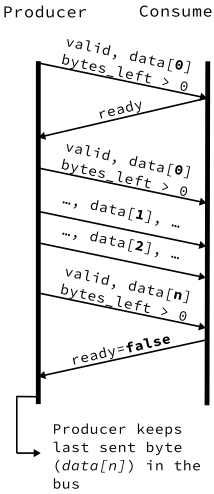
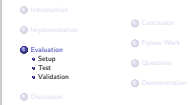


Table of Contents

Table of Contents



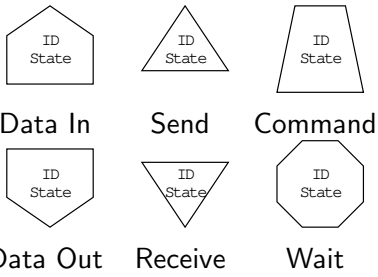
Graph file simulation

- Full input - output
- Does not take latency between packets into account
- Simplifies test cases

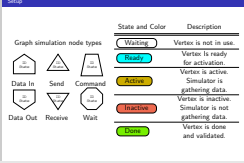
Evaluation
Setup
Graph file simulation
- Full input - output
- Does not take latency between packets into account
- Simplifies test cases

Definer send og receive bedre

Graph simulation node types



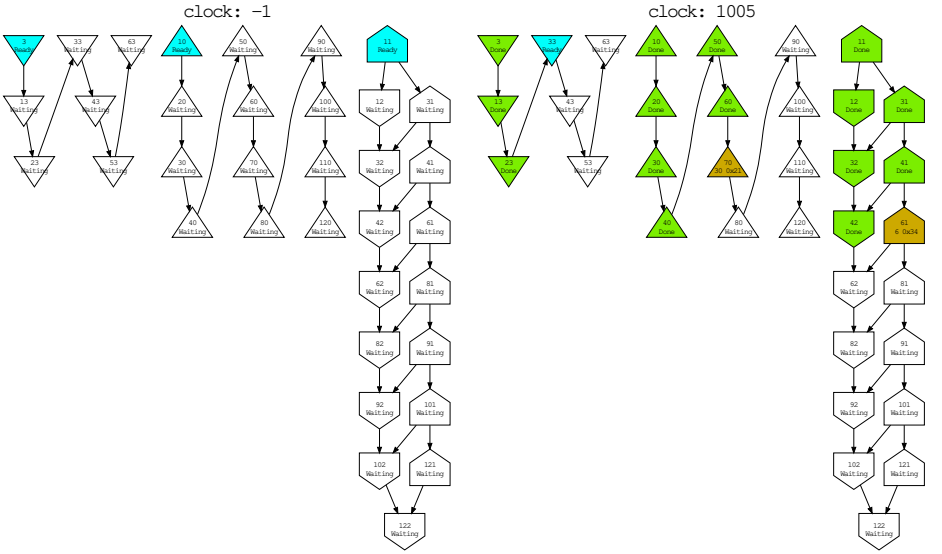
State and Color	Description
Waiting	Vertex is not in use.
Ready	Vertex Is ready for activation.
Active	Vertex is active. Simulator is gathering data.
Inactive	Vertex is inactive. Simulator is not gathering data.
Done	Vertex is done and validated.

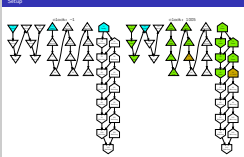


Hop til illustrationen på næste slide nå du snakker om det!

Evaluation

Setup





Evaluation
Test

Senario

- Real life scenario
- Test at high workloads
- Remove garbage
- Respond to packet
- Differ between concurrent connections

Evaluation
Test
Senario
- Real life scenario
- Test at high workloads
- Remove garbage
- Respond to packet
- Differ between concurrent connections

Fortæl kun om hvad vi vil have, ikke hvad vi har lavet af test

Evaluation
Test

The test

- 17283 packets in total
- Two "sessions"
- 640*2 UDP packets that needs a response
- 640 well formed UDP packets with no session (discard)
- Rest of data is "background noise" (TCP packets with state, data, etc)
- Total data sent through: 1832958 bytes
- 1.83 Million clocks used

Evaluation
Test
The test
17283 packets in total
Two "sessions"
640*2 UDP packets that needs a response
640 well formed UDP packets with no session (discard)
Rest of data is "background noise" (TCP packets with state, data, etc)
Total data sent through: 1832958 bytes
1.83 Million clocks used

Latency calculations:

n_D : The number of bytes in the data part of the protocol. This excludes both headers from transport and internet.

n_I : The internet header size.

n_T : The transport header size.

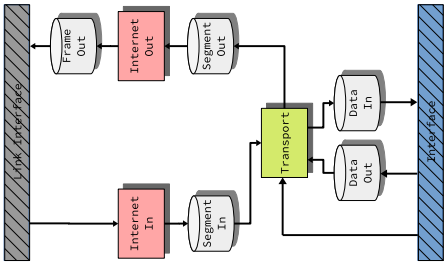
n : The total packet size.

From packet to user

$$6 + n_I + 2n_T + 3n_D$$

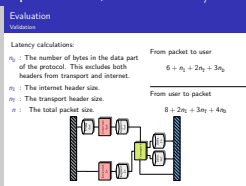
From user to packet

$$8 + 2n_I + 3n_T + 4n_D$$



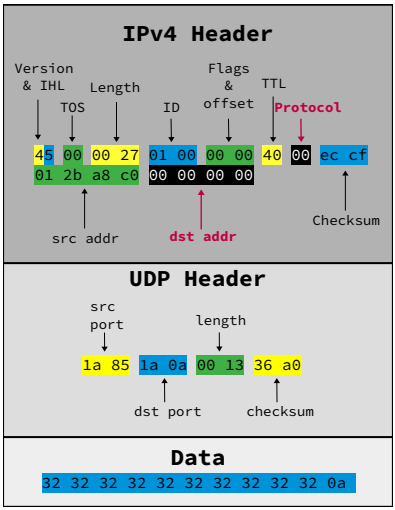
2019-09-19

TCP/IP in hardware using SME
└Evaluation
└Evaluation

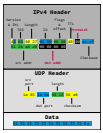


Bufferen kan ikke videresende data dirrekte, da den skal gemme segmentet først

Outgoing packet validation:



Evaluation
Validation
Outgoing packet validation:



Protocol ikke sat korrekt, destination ip ikke sat korrekt

Internet Protocol Suite compliancy as per RFC 1122

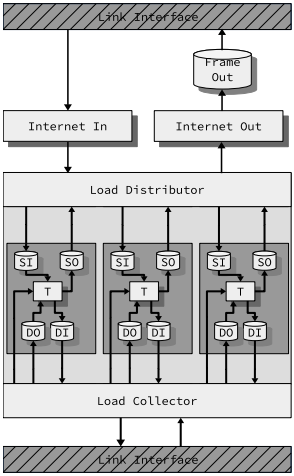
Evaluation
Validation
Internet Protocol Suite compliancy as per RFC 1122

Ikke testet helt igennem, mem felterne er generelt sat

Table of Contents

Discussion

Improving the performance:



Estimated performance:

$$1\ \textbf{Byte} * 10\ \textbf{MHz} = 80\ \textbf{Mbps}$$

Discussion
Improving the performance:
Estimated performance:
1 Byte * 10 MHz = 80 Mbps

Usability

SOMETHING

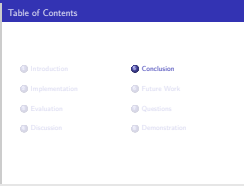
Usability SOMETHING

Using C#

State modelling
Simulation
Concurrency

Using C#
State modelling
Simulation
Concurrency

Table of Contents



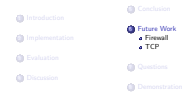
Conclusion

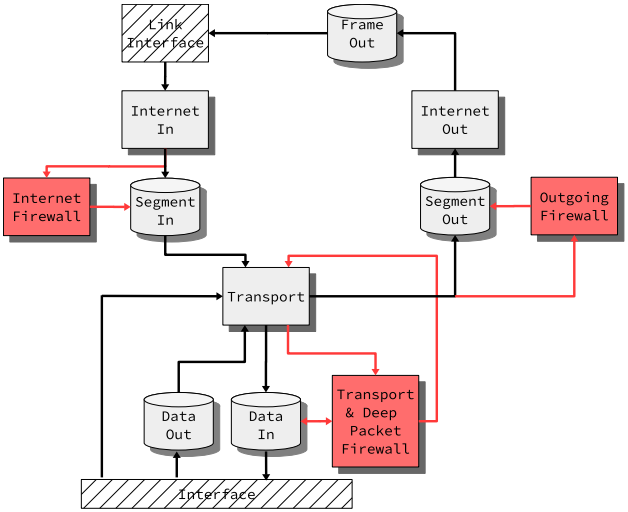
- The design underwent many alternations, but the final layered design has proven to work great
- In 1.83 mio. simulated clock cycles, all of 17283 packets were handled correctly
- Errors in the outgoing packets, but they should be easily fixable
- SME was of great help for the implementation, albeit with a few errors and bugs

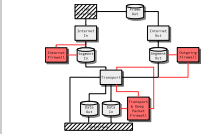
Conclusion
- The design underwent many alternations, but the final layered design has proven to work great
- In 1.83 mio. simulated clock cycles, all of 17283 packets were handled correctly
- Errors in the outgoing packets, but they should be easily fixable
- SME was of great help for the implementation, albeit with a few errors and bugs

Table of Contents

Table of Contents

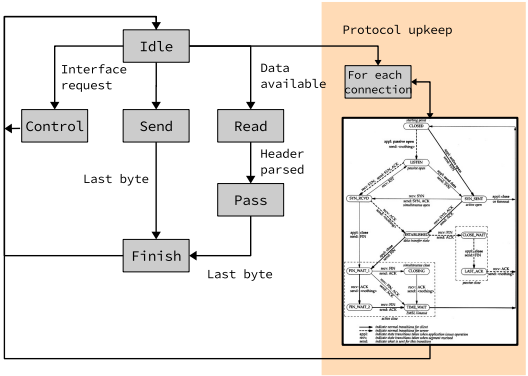






Integration med buffere. Hvad ville det indebære

Implementing TCP



Future Work
TCP
Implementing TCP

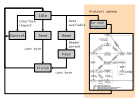


Table of Contents

Table of Contents
1 Introduction
2 Implementation
3 Evaluation
4 Discussion
5 Conclusion
6 Future Work
7 Questions
8 Demonstration

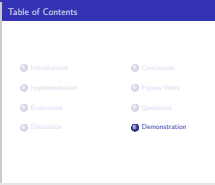
Bibliography

[1] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. Disaggregated fpgas: Network performance comparison against bare-metal servers, virtual machines and linux containers. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 9–17, Dec 2016. doi: 10.1109/CloudCom.2016.0018.

Bibliography

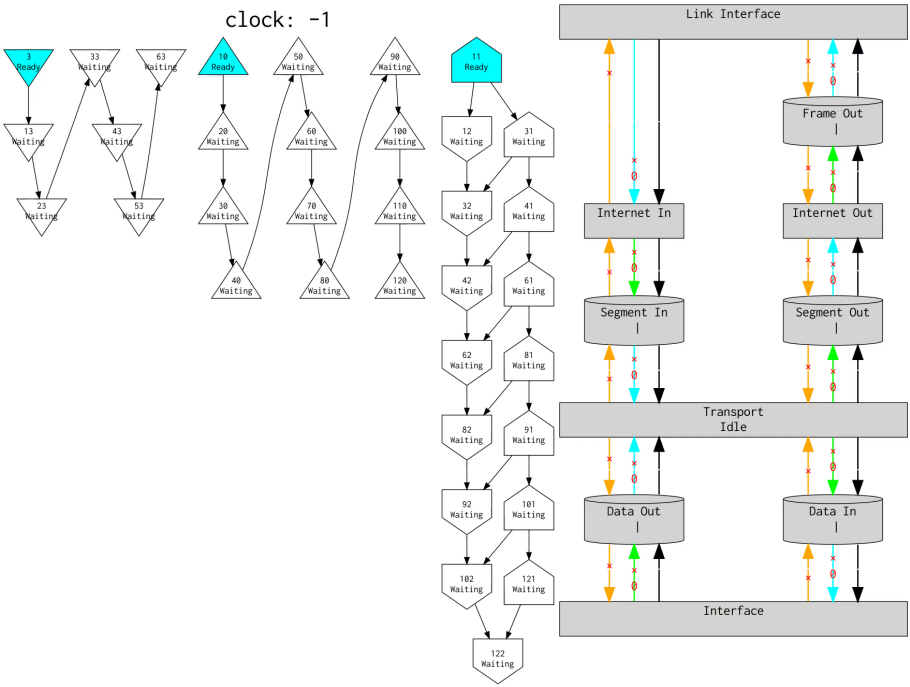
[1] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. Disaggregated fpgas: Network performance comparison against bare-metal servers, virtual machines and linux containers. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 9–17, Dec 2016. doi: 10.1109/CloudCom.2016.0018.

Table of Contents



Demonstration

Demonstration



2019-09-19

TCP/IP in hardware using SME
└Demonstration