Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

# Numerical mathematics on FPGAs using CλaSH

## From Haskell to a hardware accelerator

Martijn Bakker

Computer Architecture for Embedded Systems (CAES)
University of Twente

July 1, 2015

# Overview

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

1. Functional programming
2. FPGA
3. CλaSH
4. Problem definition and breakdown
5. CλaSH project overview
6. Results
7. Performance
8. Demo
9. Discussion
10. Conclusion

# Properties

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Main building block: functions

- No assignments, only *unchangeable* definitions
- No statements, only expressions
- ‘Variables‘ that cannot vary

- The execution of a program is a function evaluation

# Resulting features

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- No global, mutable state
- No side effects
- Pure functions

- Lazy evaluation
- Higher-order functions
- Strong type system
- Partial function application
- Function composition

- Clear structure of the program, similar to mathematics

# Example of functional programming

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 1: A very short introduction to functional programming

```
fib :: Integral a => a -> a
fib n | n == 0    = 1
      | n == 1    = 1
      | otherwise = fib (n-2) + fib (n-1)

fib_list :: Integral a => a -> [a]
fib_list n = map fib [0..n]

choose_list :: Integral a => (a -> a) -> [a]
choose_list function = map function [0..]

sum_list :: Integral a => [a] -> a
sum_list list = foldl (+) 0 list
```

# The Field-Programmable Gate Array

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- 'Programmable hardware'
- Create your own circuit instead of a list of instructions

Figure: FPGA structure

# Why would you want to use an FPGA?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Advantages over CPUs

- High throughput
- Low *guaranteed* latency
- Low power use
- Support for new "instructions"

Advantages over ASICs

- Reconfigurability

# Drawbacks

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 2: A single AND-gate specified in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164. all ;

ENTITY andgate IS
PORT (
  input_0    : in  std_logic ;
  input_1    : in  std_logic ;
  output_0   : out  std_logic
  );
END ENTITY andgate;

ARCHITECTURE a OF andgate IS
BEGIN
  output_0 <= input_0 and input_1;
END ARCHITECTURE a;
```

# Drawbacks

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

1. FPGA development is **hard**
   - Vendor-specific, closed-source tools
   - Small communities
   - Ancient HDLs
   - Synthesis, debugging and verification is a slow process
2. FPGAs cannot be reconfigured as quickly as CPUs
3. FPGAs run at lower clock speeds than ASICs

# CλaSH - What?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- CAES Language for Synchronous Hardware

- A library for the specification of hardware in Haskell.
- Includes a compiler: generation of VHDL and Verilog
- Written by Christiaan Baaij at CAES

- www.clash-lang.org

# CλaSH - Why?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Similarities between hardware and functional programming

- Hardware consists of largely combinatorial parts: pure functions
- Combinatorial circuits contain a dependency tree: function dependencies
- Higher-order functions

# CλaSH - Apparent issues?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Hardware has a mutable state
- Functional programming has no mutable states

Solution: the Mealy machine.

# CλaSH - Apparent issues?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Hardware has a mutable state
- Functional programming has no mutable states

Solution: the Mealy machine.

# The Mealy machine

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: Block diagram describing a Mealy machine. The loop gets executed once every clock cycle.

# Example

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 3: A basic multiply-accumulate circuit specification in CλaSH

```
module MAC where
  import CLaSH.Prelude

  topEntity :: Signal (Signed 9, Signed 9) -> Signal (Signed 9)
  topEntity = mealy mac 0

  mac :: Num t => t -> (t, t) -> (t, t)
  mac state input = (state', output)
    where
      (x,y)    = input          -- unpack the two inputs
      state'   = state + x*y    -- the new state
      output   = state'         -- output the new state
```

# Wrap-up of introduction

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Imperative languages:
- C / C++
- Matlab

Consist of:
- Assignments and statements
  i = i + i

compile

Sequence of instructions

execute (CPU)

Result

Functional languages:
- Haskell (CλaSH)

Consist of:
- Function definitions
  fac 0 = 1
  fac n = n * fac (n-1)

compile (CλaSH)

Hardware Description Language

synthesis
deploy (FPGA)
run

Result

# Definition

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

The **feasibility** and **performance** of performing **numerical approximations** to **ODEs** on an FPGA with hardware specified in CλaSH.

1. **Feasibility**: does the most simple version work?
2. **Performance**: comparison with a desktop CPU
3. **Numerical approximations**: simple integration schemes (Euler, RK2, RK4)
4. **ODEs**: 4 coupled first order differential equations with constant coefficients (4x4 matrix)
5. **FPGA**: a Terasic SoCKit development board, containing an Altera Cyclone V SoC FPGA

# Breakdown - requirements for the system

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Number representation
- Number storage
- Control protocol
- Supplying a suitable clock frequency

1. Programming
2. Getting matrix constants and initial values in
3. Controlling and monitoring the state
4. Performing the actual computation
5. Getting results out of the FPGA (go to 3)

# Number representation

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Floating point or fixed point?

- CλaSH only supports fixed point

- Signed fixed point with 8 integer bits and 24 fractional bits
- Total width: 32 bits
- Integer range: [-128 .. 127]
- ULP[1]: $2^{-24} \approx 6 \times 10^{-8}$, roughly 7 decimal digits.

[1] Unit of least precision

# Number representation

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Floating point or fixed point?
- CλaSH only supports fixed point
- Signed fixed point with 8 integer bits and 24 fractional bits
- Total width: 32 bits
- Integer range: [-128 .. 127]
- ULP[1]: $2^{-24} \approx 6 \times 10^{-8}$, roughly 7 decimal digits.

[1] Unit of least precision

# Number representation

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Floating point or fixed point?
- CλaSH only supports fixed point
- Signed fixed point with 8 integer bits and 24 fractional bits
- Total width: 32 bits
- Integer range: [-128 .. 127]
- ULP[1]: $2^{-24} \approx 6 \times 10^{-8}$, roughly 7 decimal digits.

[1] Unit of least precision

# Number storage

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Registers[2], SRAM[3] or SDRAM[4]?

- All variables get updated every single cycle.
- All data fits in the registers.
- No need for SRAM or SDRAM.

[2] Direcly accessible on-chip: latches
[3] Static RAM, on-chip, single cycle access delays
[4] Synchronous Dynamic RAM, off-chip, 'high' latency

# Number storage

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Registers[2], SRAM[3] or SDRAM[4]?

- All variables get updated every single cycle.
- All data fits in the registers.
- No need for SRAM or SDRAM.

[2] Direcly accessible on-chip: latches
[3] Static RAM, on-chip, single cycle access delays
[4] Synchronous Dynamic RAM, off-chip, 'high' latency

# Supplying a suitable clock frequency

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- A higher* clock frequency means higher throughput

- *up to a certain limit, otherwise your system fails.
- Output stabilization of the combinatorial circuits takes time.

- FPGA contains a crystal (50 MHz) and PLL[5] circuits.
- CλaSH + IO + Altera PLL = non-deterministic behaviour.
- Solution: a simple integer frequency divider.

[5] Phase-Locked Loop

# Supplying a suitable clock frequency

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- A higher* clock frequency means higher throughput
- *up to a certain limit, otherwise your system fails.
- Output stabilization of the combinatorial circuits takes time.
- FPGA contains a crystal (50 MHz) and PLL[5] circuits.
- CλaSH + IO + Altera PLL = non-deterministic behaviour.
- Solution: a simple integer frequency divider.

[5] Phase-Locked Loop

# Supplying a suitable clock frequency

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- A higher* clock frequency means higher throughput

- *up to a certain limit, otherwise your system fails.
- Output stabilization of the combinatorial circuits takes time.

- FPGA contains a crystal (50 MHz) and PLL[5] circuits.
- CλaSH + IO + Altera PLL = non-deterministic behaviour.
- Solution: a simple integer frequency divider.

[5] Phase-Locked Loop

# Actual computation - External Types

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 4: CλaSH topEntity for the ODE solver

```
topEntity :: Signal InputSignals -> Signal OutputSignals
topEntity = mealy solveODE initialState

solveODE state input = (state', output)
  where
    state  = (systemState,systemConstants,oul,block)
    state' = (systemState',systemConstants',oul',block')
```

# Actual computation - Internal Types

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 5: Internal types for the ODE solver

```
type Data = SFixed 8 24
type UInt = Unsigned 32
type ValueVector = Vec 5 Data
type ConstantVector = Vec 20 Data

data ODEState = ODEState { valueVector :: ValueVector
                         , time  ::  Data
                         } deriving (Show)

type Equation = (ODEState, ConstantVector) -> ValueVector
type Scheme = SystemConstants -> Equation -> ODEState -> ODEState
```

# Actual computation - Equation

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

$$\mathbf{x}' = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} \mathbf{x} \tag{1}$$

Listing 6: Computing the derivative

```
matrix2d :: Equation
matrix2d (odestate, constants) = dxs
  where
    xs = valueVector odestate

    c1 = constants !! 4
    c2 = constants !! 5
    c3 = constants !! 6
    c4 = constants !! 7

    x0 = c1 * (xs !! 0) + c2 * (xs !! 1)
    x1 = c3 * (xs !! 0) + c4 * (xs !! 1)

    dxs = fst $ shiftInAt0 xs (x0 :> x1 :> Nil)
```

# Actual computation - Scheme

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + h\mathbf{x}'_k & \text{if } t_k < t_{max} \\ \mathbf{x}_k & \text{otherwise} \end{cases} \tag{2}$$

Listing 7: Eulers method

```
euler :: Scheme
euler constants equation state = state'
  where
    c_user = userconstants constants
    c_maxtime = maxtime constants
    h = timestep constants
    t = time state
    xs = valueVector state

    --Apply Euler's integration scheme
    eulerxs = zipWith (+) xs $ map (*h) (equation (state, c_user))
    (xs', t') = if t < c_maxtime then (eulerxs, t + h)
                                 else (xs, t) -- already at maximum time

    state' = ODEState {valueVector = xs', time = t'}
```

# Actual computation - Control

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Listing 8: Controlling the ODE solver (important lines start with '!')

```
1  ! scheme = euler
2  ! equation = matrix4d
3
4  (systemState', oul', block')
5    --Handle the setup (reset the state, insert input values, start the computation)
6    | i_c == 2 && i_cs == 1 && i_ca == 0 = ( initialSystemState, 0, 0)
7    | i_is == 1                          = ( systemState{ odestate = s_odestate_in' }, 0, 1)
8    | i_c == 1 && i_cs == 1 && i_ca == 0 = ( systemState{ step = 0 } , 0, 0)
9
10   --Handle the computation and output:
11 ! | block == 1 && i_os == 1            = ( systemState, pack (xs !! i_oa ), block)
12 ! | block == 0 && s_step < c_maxstep   = ( systemState_up' , 0, block)
13 ! | block == 0 && s_step >= c_maxstep = ( systemState{ step = uIntMax}, pack uIntMax, 1)
14
15   --Default, do nothing
16   | otherwise                          = ( systemState, oul, block)
17   where
18     s_odestate_in'  = s_odestate {valueVector = replace i_ia (unpack i_i :: Data) xs}
19
20 !   s_odestate_up   = scheme systemConstants equation s_odestate
21     valueVector_wt  = replace 4 (time s_odestate_up) (valueVector s_odestate_up)
22     s_odestate_up'  = s_odestate_up {valueVector = valueVector_wt }
23     s_step'         = s_step + 1
24
25     systemState_up' = systemState{ odestate = s_odestate_up', step = s_step'}
```

# An overview of the entire system


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

# Results

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Each solution plot contains 3 curves
  1. FPGA-approximation
  2. MATLAB-approximation (same functionality as FPGA)
  3. Analytical or obtained from ode45(), the real solution
- Each error plot contains at least 2 curves
  1. FPGA - real
  2. FPGA - MATLAB approximation

- MATLAB uses IEEE 754 double precision floating point
- FPGA uses 8/24 signed fixed point.

# Results

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Each solution plot contains 3 curves
  1. FPGA-approximation
  2. MATLAB-approximation (same functionality as FPGA)
  3. Analytical or obtained from ode45(), the real solution
- Each error plot contains at least 2 curves
  1. FPGA - real
  2. FPGA - MATLAB approximation

- MATLAB uses IEEE 754 double precision floating point
- FPGA uses 8/24 signed fixed point.

# Simple oscillation - I $x(t) = 50\cos(t)$


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: The final error is $\approx 30$ (h $= 0.01$)

# Simple oscillation - II $x(t) = 50\cos(t)$


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: The final error decreased by a factor 150 (h $= 1\times10^{-4}$)

# Simple oscillation - III $x(t) = 50\cos(t)$


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: Breakdown of the number representation (h $= 1\times10^{-7}$)

# Switching equations

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Eulers method (first order) to RK2 (second order)
- Simple oscillations to 4 coupled equations
- Matrix chosen such that all eigenvalues negative

$$\mathbf{x}' = \begin{bmatrix} 2 & 3 & 2 & 0 \\ -5 & -5 & -3 & 1 \\ 3 & -1 & -2 & -3 \\ 4 & 2 & 2 & -3 \end{bmatrix} \mathbf{x} \quad \text{with} \quad \mathbf{x}(t_0) = \begin{bmatrix} 7 \\ 5 \\ 7 \\ 5 \end{bmatrix} \tag{3}$$

# RK2 - I


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: A high value of h, maximum error $\approx 0.3$ (h $= 0.01$)

# RK2 - II


Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: A very low value of h, maximum error $\approx 0.2$ (h $= 5\times10^{-7}$)

# RK4

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: For all tested values of h, ($h = 1 \times 10^{-5}$)

# Euler revisited

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Figure: Euler outperforming both RK2 and RK4, ($h = 1 \times 10^{-4}$)

# Results

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

Table: Eulers method, sorted by iterations per second.

| Device | Iterations | time (s) | Output interrupts | Iterations per second ($\times 10^6$) |
|---|---|---|---|---|
| CPU - C++ - int | $1\times10^8$ | 1,35 | 1 | 74,1 |
| FPGA | $1\times10^8$ | 2,18 | 1 | 45,8 |
| FPGA | $1\times10^8$ | 2,25 | $1\times10^3$ | 44,4 |
| FPGA | $1\times10^8$ | 2,30 | $1\times10^4$ | 43,4 |
| FPGA | $1\times10^8$ | 3,27 | $1\times10^5$ | 30,6 |
| FPGA | $1\times10^7$ | 0,39 | 1 | 25,6 |
| FPGA | $1\times10^6$ | 0,21 | 1 | 4,76 |
| CPU - C++ - float | $5\times10^7$ | 58,1 | 1 | 0,86 |
| FPGA | $1\times10^5$ | 0,19 | 1 | 0,53 |
| CPU - Haskell - float | $1\times10^6$ | 3,23 | 1 | 0,31 |
| CPU - MATLAB - float | $1\times10^7$ | 304 | 1 | 0,03 |

# Notes

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- FPGA for "low-power, cost-sensitive design needs"
- No automatic derivation of the optimal clock frequency
- 4x4 matrix equation only used ~10% of the FPGA space
- Power usage is ~ 2 orders of magnitude lower than the CPU
- CPUs are heavily optimized for fast integer arithmetic

# The entire process takes ~10 minutes

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

1. CλaSH - Windows/Linux
2. Synthesis (Quartus) - Windows
3. Driver compilation - (arm-linux-gnueabihf-g++) - Linux
4. Run and extract results - Windows/Linux
5. Verification of results (MATLAB) - Windows

- Normal work flow - Use two machines
- Linux for compiling and deploying the driver
- Windows for the other steps

# Discussion

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Accuracy
  - Equality between the FPGA and MATLAB implementations
  - Limitations of the number representation and CλaSH
- Functionality
  - 4x4 matrix with constant coefficients
  - Variety of basic solver schemes
  - Limitations of the number representation and CλaSH
- Performance
  - Sub-par development FPGA is close to a modern CPU core
  - Trade-off between power usage and performance

# What has been shown?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Hardware acceleration of numerical solvers is feasible
- Number representations are important
- CλaSH is usable for projects with complex IO by use as a module
- FPGAs are fast: subpar FPGA close to a modern CPU core

# What has been created?

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Single-command toolchain integration for deploying CλaSH on an FPGA
- IO system of sufficiently high performance
  - Configuration of the Avalon bridges
  - C++ library for easy FPGA accessibility

# Acknowledgements

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
Additional stuff

- Jan Kuper - Introducing me to functional programming, CλaSH and giving feedback
- Christiaan Baaij - Creating CλaSH and answering related questions
- Ruud van Damme & Jan Broenink - Feedback
- Rinse Wester - Input on configuring and using the Avalon bridges

# Frequency too high? Random number generator.

Functional programming
FPGA
C$\lambda$aSH
Problem definition and breakdown
C$\lambda$aSH project overview
Results
Performance
Demo
Discussion
Conclusion
**Additional stuff**

Figure: The main source of failure after overclocking.

# Nondeterministic behaviour

Functional programming
FPGA
CλaSH
Problem definition and breakdown
CλaSH project overview
Results
Performance
Demo
Discussion
Conclusion
**Additional stuff**

Figure: Only when using the Altera PLL

# The core of the circuit: the matrix multiplier


Functional programming

FPGA

CλaSH

Problem definition and breakdown

CλaSH project overview

Results

Performance

Demo

Discussion

Conclusion

**Additional stuff**

Figure: A 4x4 matrix vector multiplication consists of $4 * (4 + 3) = 28$ operations

# Synthesized hardware


Functional programming

FPGA

CλaSH

Problem definition and breakdown

CλaSH project overview

Results

Performance

Demo

Discussion

Conclusion

**Additional stuff**


Date: June 30, 2015

Project: sockit

# But do not zoom in!

Functional programming
FPGA
C$\lambda$aSH
Problem definition and breakdown
C$\lambda$aSH project overview
Results
Performance
Demo
Discussion
Conclusion
**Additional stuff**