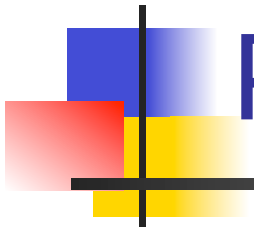
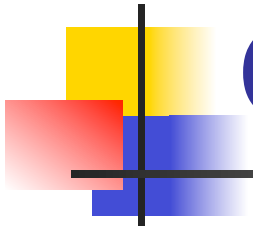


Parallel Image Processing

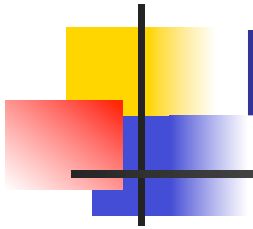
An introductory lecture to the project

Dr. Christos Bouganis



Overview

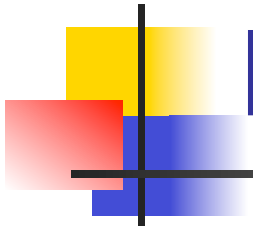
- Project description
- Deadlines
- Image Processing
  - Colour to Black & White
  - Edge Detection
  - Convolution Masks
- Parallelism and Your Project
- Questions



Project Description

- Perform image processing in hardware
- Make efficient use of parallelism
- Use of a C-like language (Catapult-C)

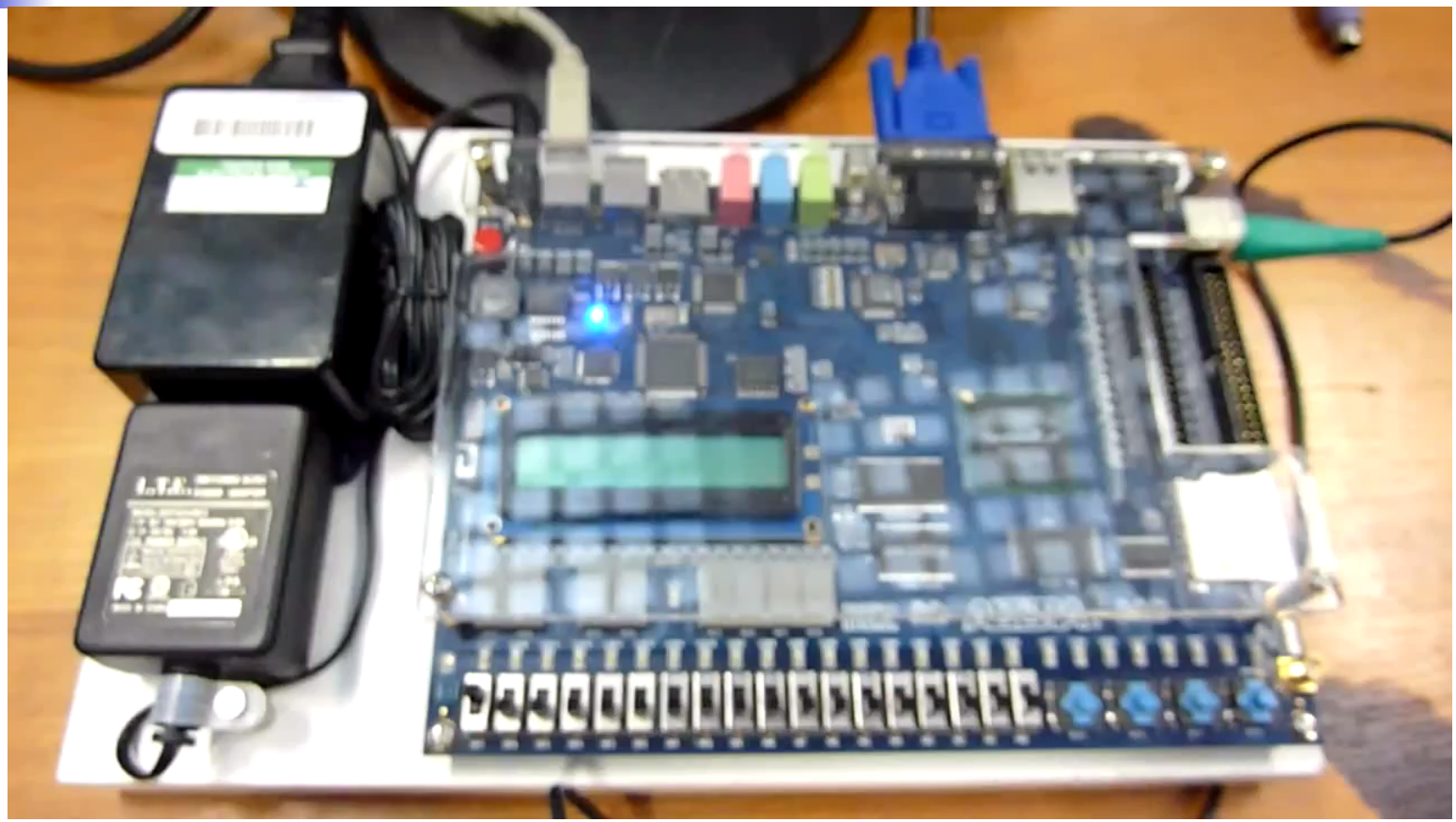
- Extra motivation:
  - prize to the best project

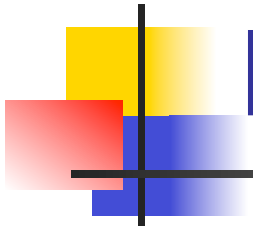


Planning

- This term: planning and learning (Catapult C)
  - Start: 3rd March
  - ~4 weeks part-time (5 Lab sessions)
- Next term: implementation
  - ~3 weeks full-time
- Overall:
  - 10% of year marks
  - something fun – show what you can do.

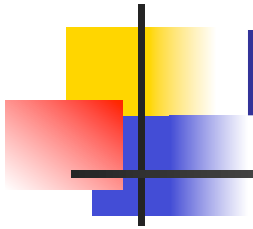
An example of a prize winning project (Handel-C based)





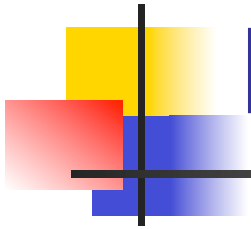
Deadlines

- This term:
  - brief planning document (27th March), one per group. Earlier if possible.
- Next term:
  - working demonstration and oral examination
    - 15th May
  - final report
    - 18th May



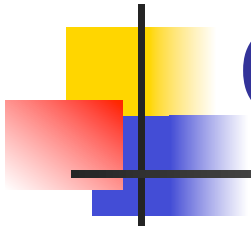
Lab

- Level 1
- The project is based on DE0 board
- Demonstrators will be available to help you.



Objective: Real-time Image Processing

- To "process" an image in real-time:
  - to change it in some way to extract (or suppress) certain features.
  - Work at 25fps
- Examples:
  - colour to B&W conversion
  - edge detection
  - region segmentation



Background
Colour to B&W conversion

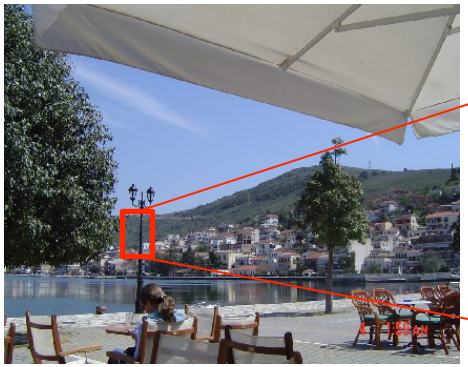
- Colour images typically stored as RGB
- 8-bits per component
- Shades of grey: R = G = B
- Simple colour to B&W:
  - intensity = (R+G+B)/3
  - R = G = B = intensity
- A pixel-by-pixel transformation

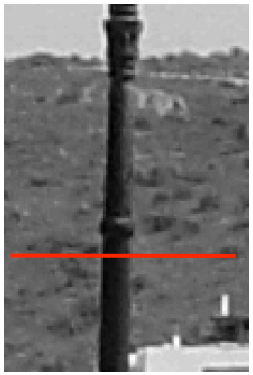
Colour to B&W example

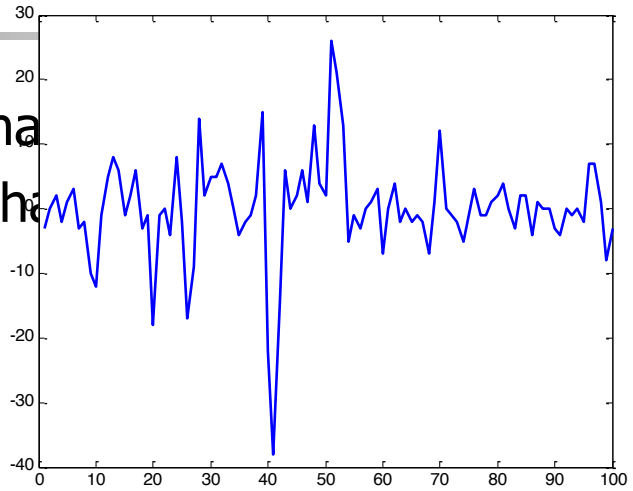
Edge detection

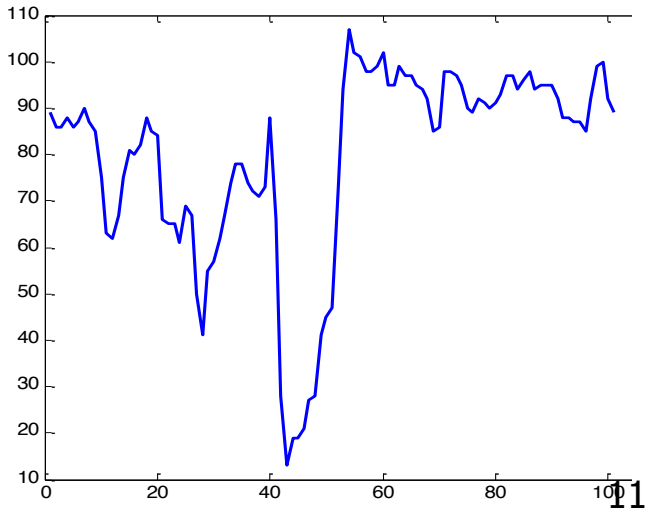
-1	0	+1

- The eye is very sensitive to abrupt cha
- Look for edges by detecting abrupt ch









Edge Detection (2D)

-1	0	1
-2	0	2
-1	0	1

mask

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
20	20	20	20	20	20
20	20	20	20	20	20
20	20	20	20	20	20

original image

?	?	?	?	?	?
?	0	40	40	0	?
?	0	30	30	0	?
?	0	10	10	0	?
?	0	0	0	0	?
?	?	?	?	?	?

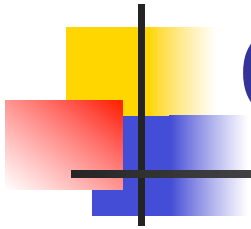
new image

- Sobel edge detection
- smoothing (blurring)

Edge Detection

Before

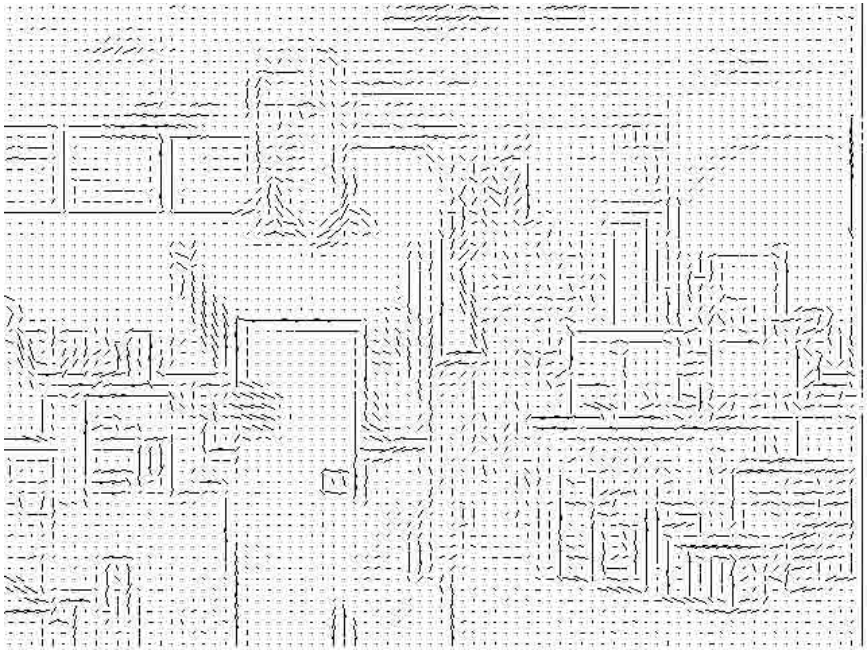
After (Canny)

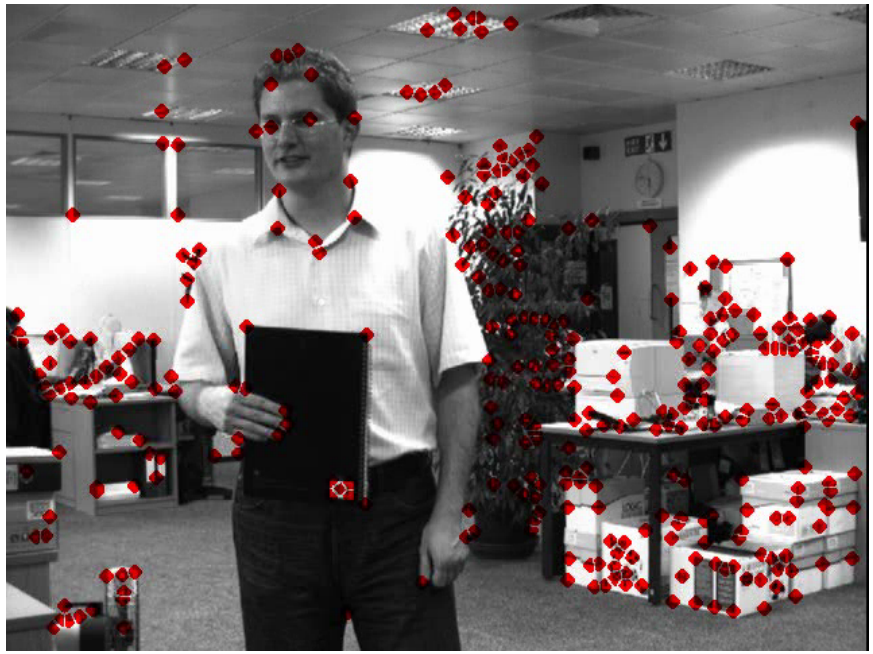


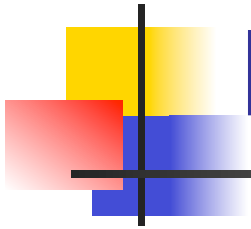
Convolution Masks

- By putting different numbers in the mask, we get different effects:
  - embossing, blurring, etc.
- The general term is "2D convolution"
  - much more next year!
- We could improve the edge detector
  - tune the amount of blurring
  - detect edges in both directions
  - threshold the results

Examples

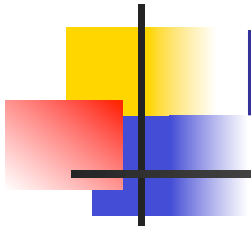






Parallelism and Your Project

- Many image processing tasks can be made to run in *parallel*.
  - Colour to B&W: each pixel is independent.
  - Edge detection: each window location is independent
    - but make sure don't overwrite data!
- In your OS course, you examined threads
  - usually run as time slices on a single processor, or two processors.

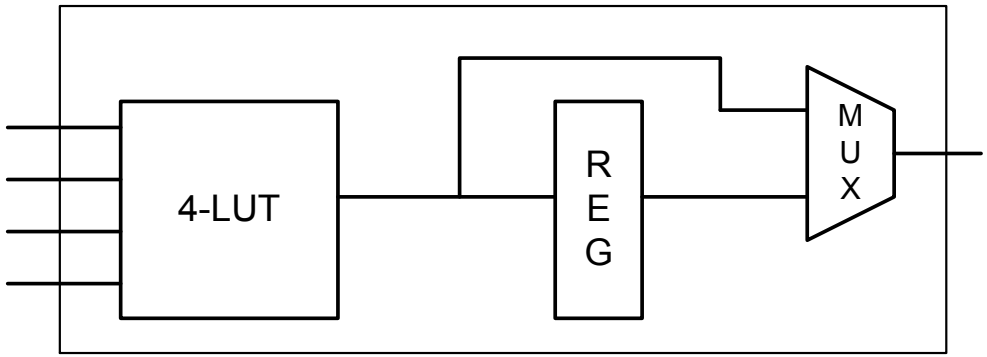


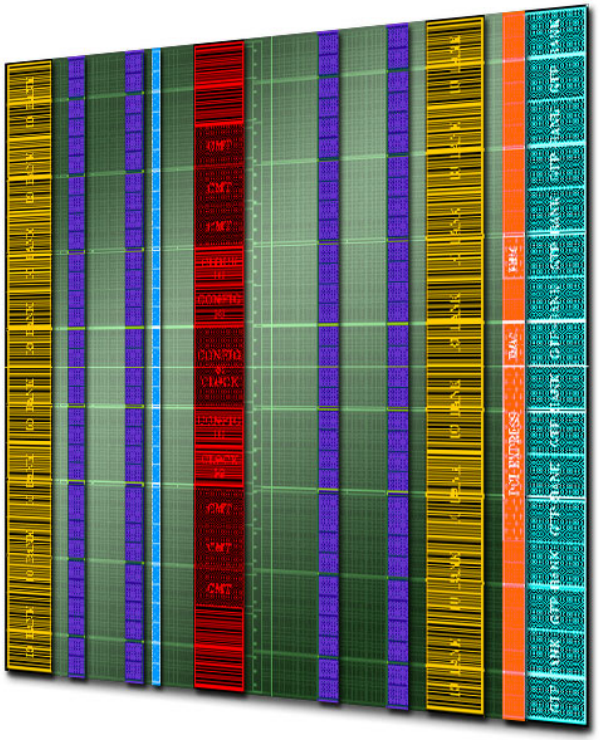
Parallelism and Your Project

- Catapult-C essentially lets you create your own processor.
  - run things *at the same time*.
- The key to a good project
  - *make efficient use of parallelism*.

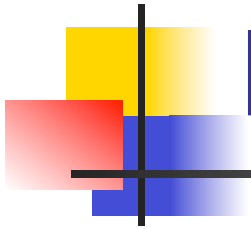
FPGA overview (in 2 min)

4-LUT





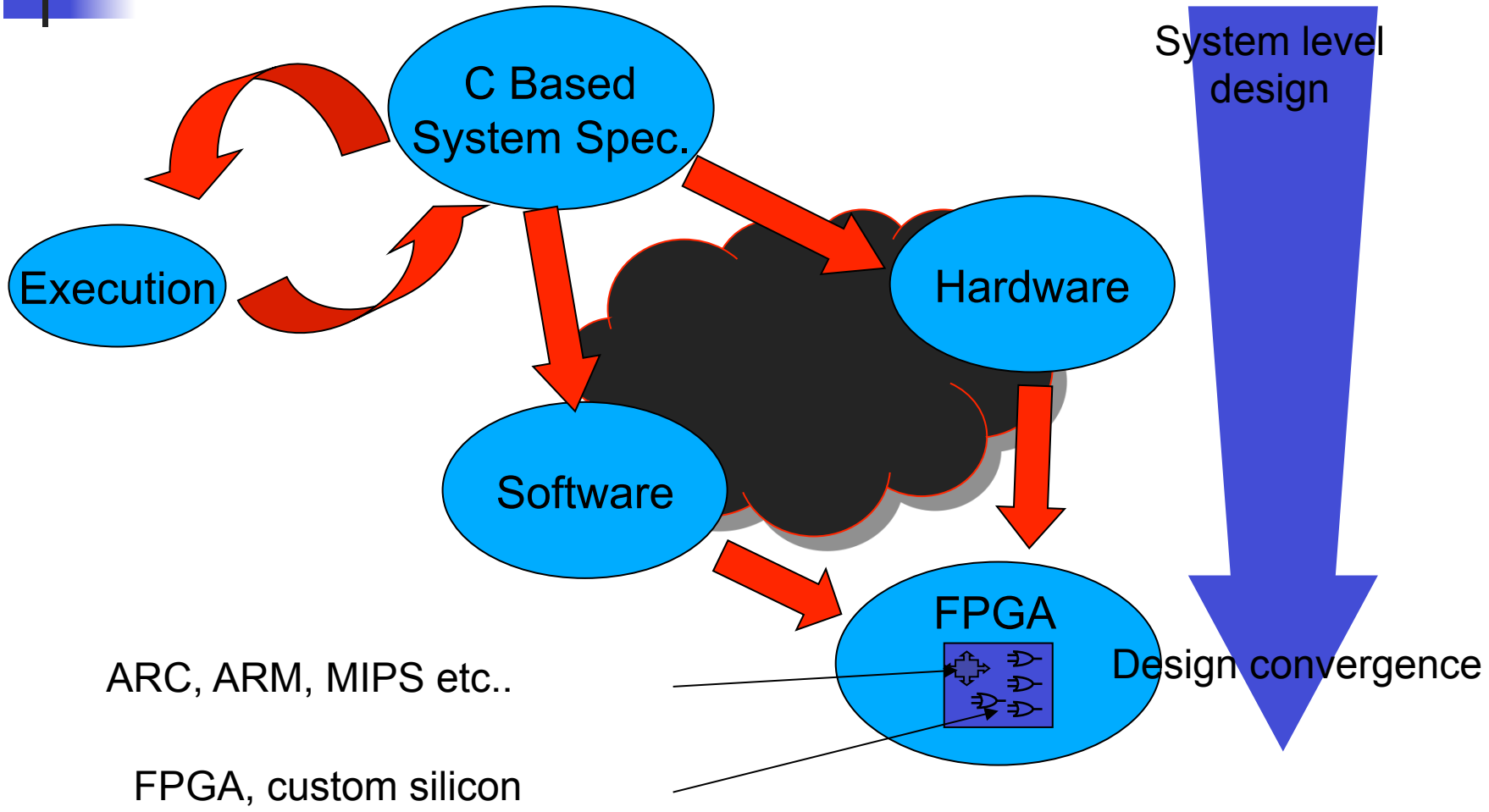
FPGA



High-Level Synthesis (HLS)

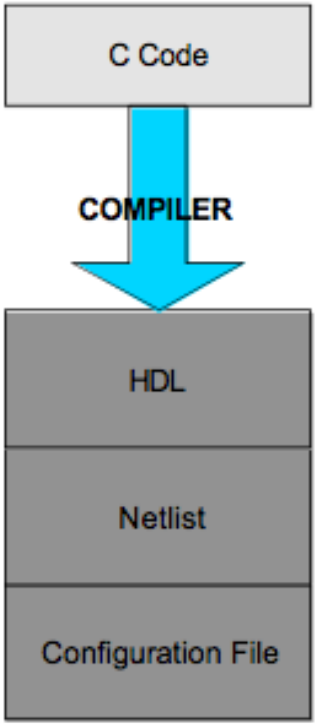
- RTL level description
  - VHDL, Verilog
    - Full-control of generated HW
    - Time-consuming
    - Gap between algorithm and hardware
- High-level Synthesis
  - Usually a C-like language
  - Compiler responsible to extract “good” HW

Hardware and software co-design

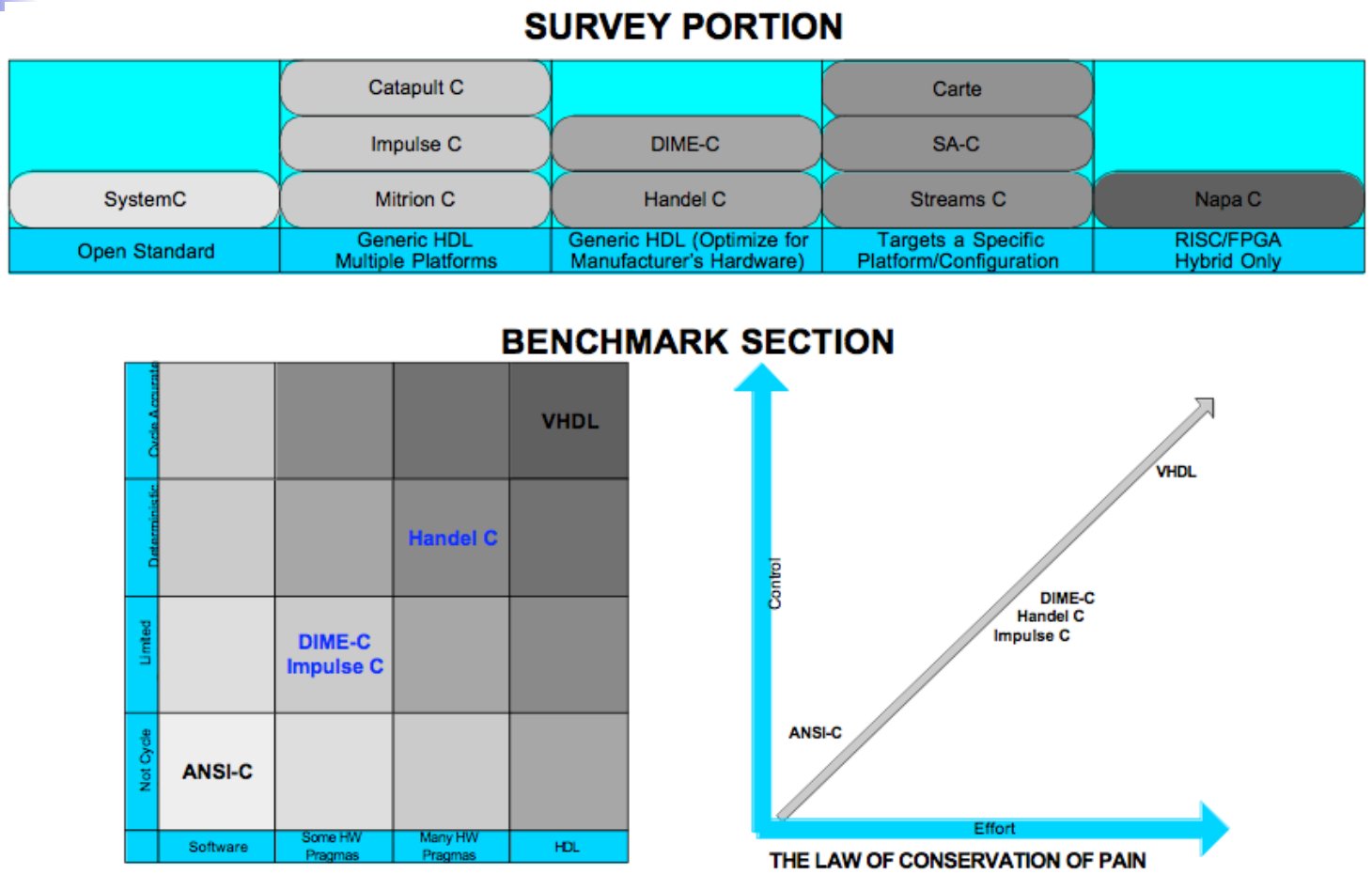


Available tools

- SystemC, ImpulseC, HandelC (not any more), Catalupt C, …
- Aim:

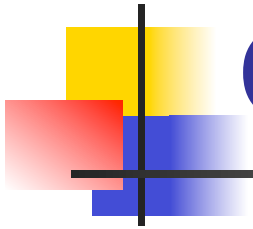


Spectrum of C-based tools



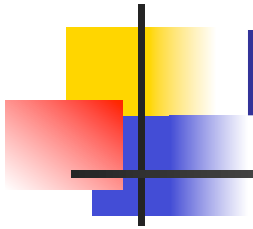
Taken from: **Survey of C-based Application Mapping Tools for Reconfigurable Computing**, Holland et. al.

FROM C PROGRAMS TO HARDWARE: CATAPULT C



Core Language Features

- Standard ISO-C (ANSI-C) … if, while, switch etc including
  - Functions, Structures, Pointers
- Extensions for hardware implementation
  - Arbitrary widths on variables
  - Untimed (User is not aware about the timing)
  - Bit manipulation operators
  - RAMs/ROMs and external pin connections

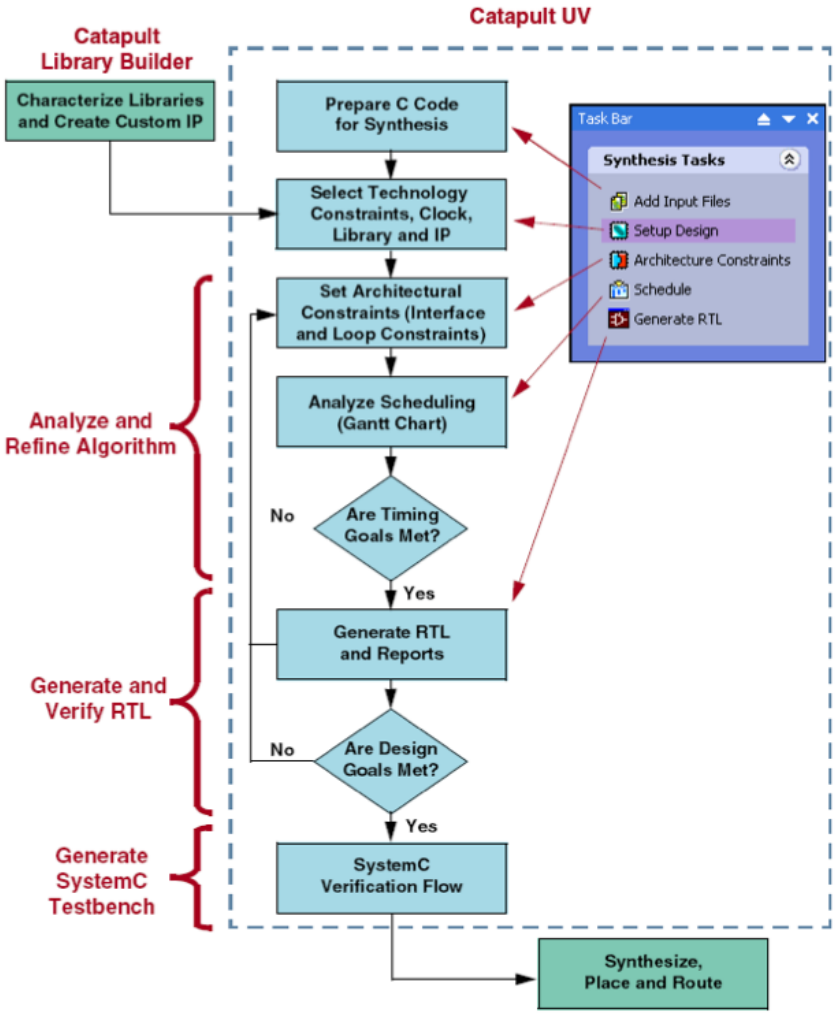


But remember

Design with HLS => describe HW using C

Design with HLS !=> code in C and compiler generates HW

The Catapult C Flow



Bit Accurate Data Types

- Bit accurate data types model true HW beh.
- Need to simulate fast => Algorithmic C data types (by Calypto Design Systems)
- For fixed point user needs to include

```
#include <ac_int.h>
#include <ac_fixed.h>
```

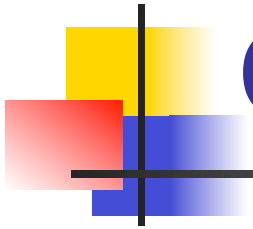
- Unsigned integer:

```
ac_int <8,false> temp;
```

8 bits, unsigned

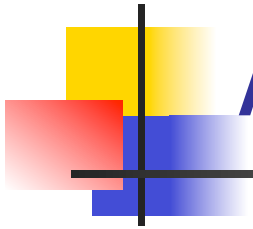
- Unsigned Fixed Point:

```
ac_fixed <W,I,false>
W:width, I:location of decimal point relative to MSB
```



Catapult C has no timing

- No concept of timing -> no clocks, resets, enables
- These signals are added by the synthesis tool



An example

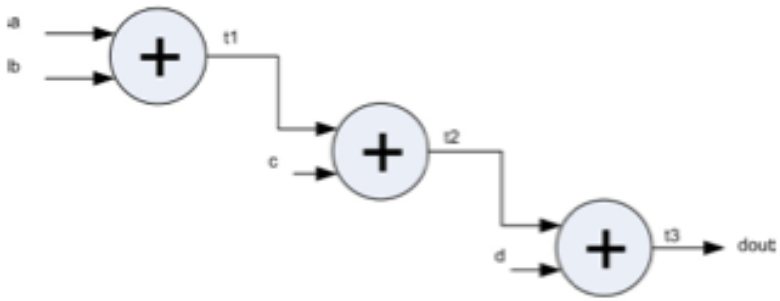
- Simple C++ accumulate

```
#include "accum.h"
void accumulate(int a, int b, int c, int d, int &dout){
  int t1,t2;

  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```

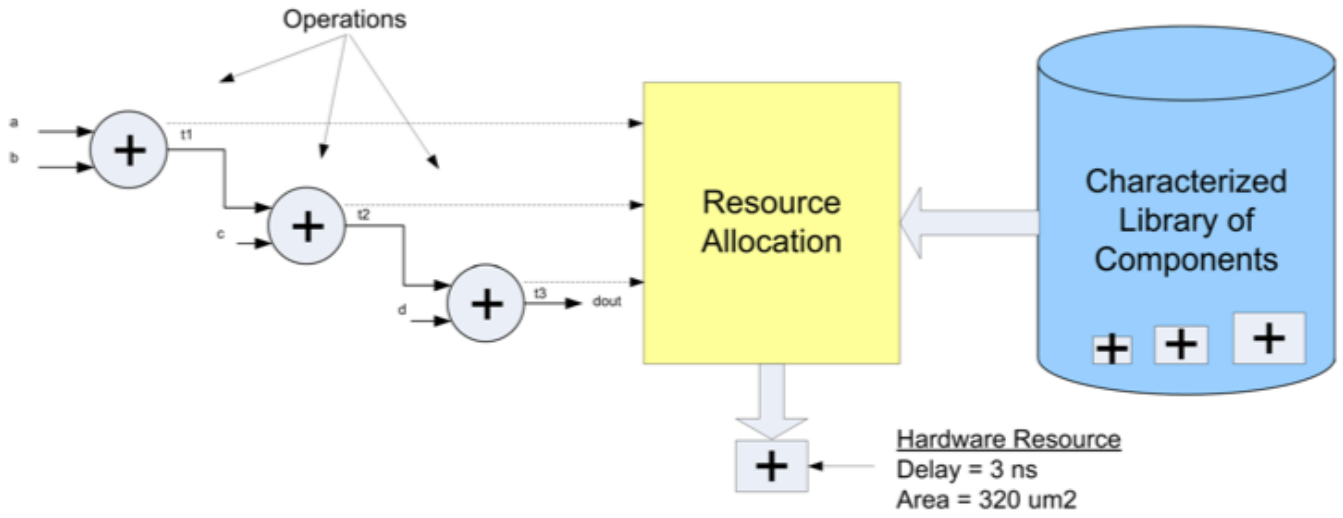
Step 1. Data Flow Graph Analysis

- Analysis of the dependencies
- DFG
- Each node is an operation



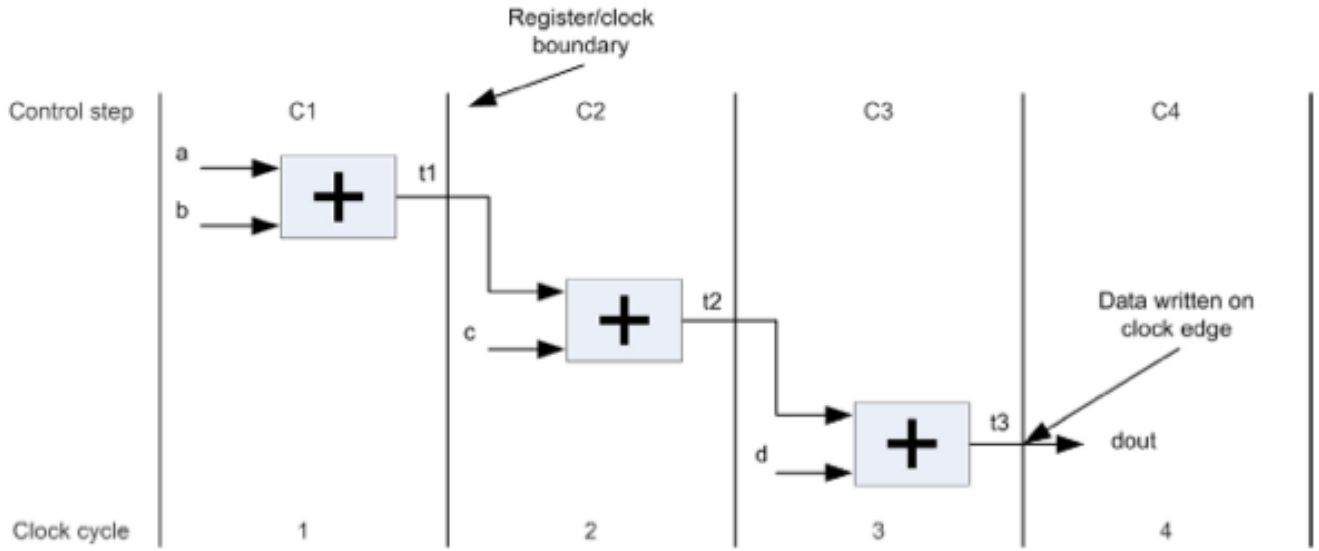
Step 2. Resource Allocation

- Each operation is mapped into a HW resource
- Annotate with timing and area info



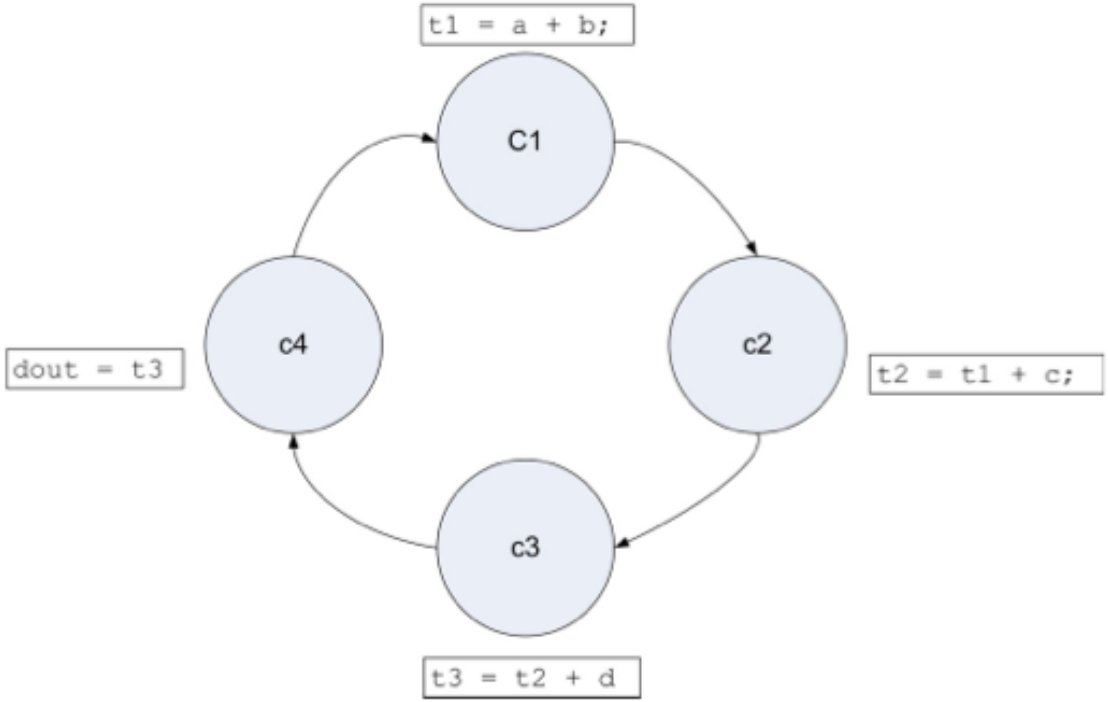
Step 3. Scheduling

- HLS adds "time" => schedule of operations
- Decides when (clock cycle) each operation is performed



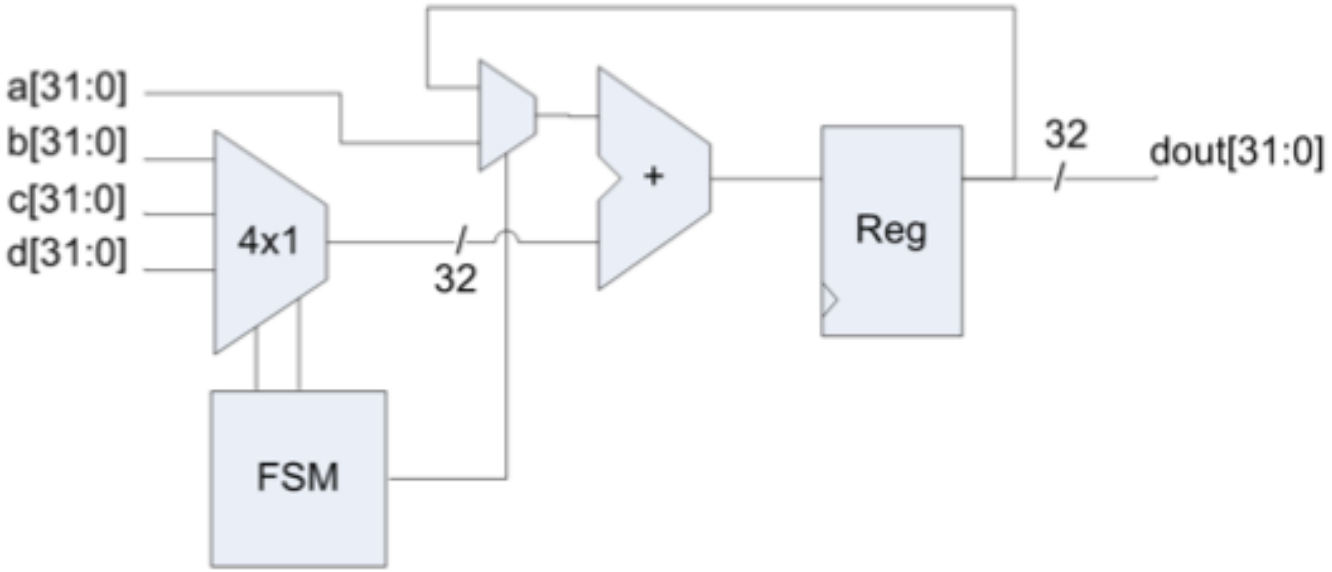
Step 3. Scheduling (cont')

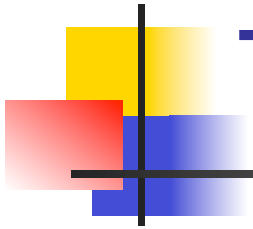
- Data Path State Machine
  - 4 states, (4 clock cycles)



Step 4. Generate HW

- Assume "unconstraint"
  - Use minimum resources if sharing save area



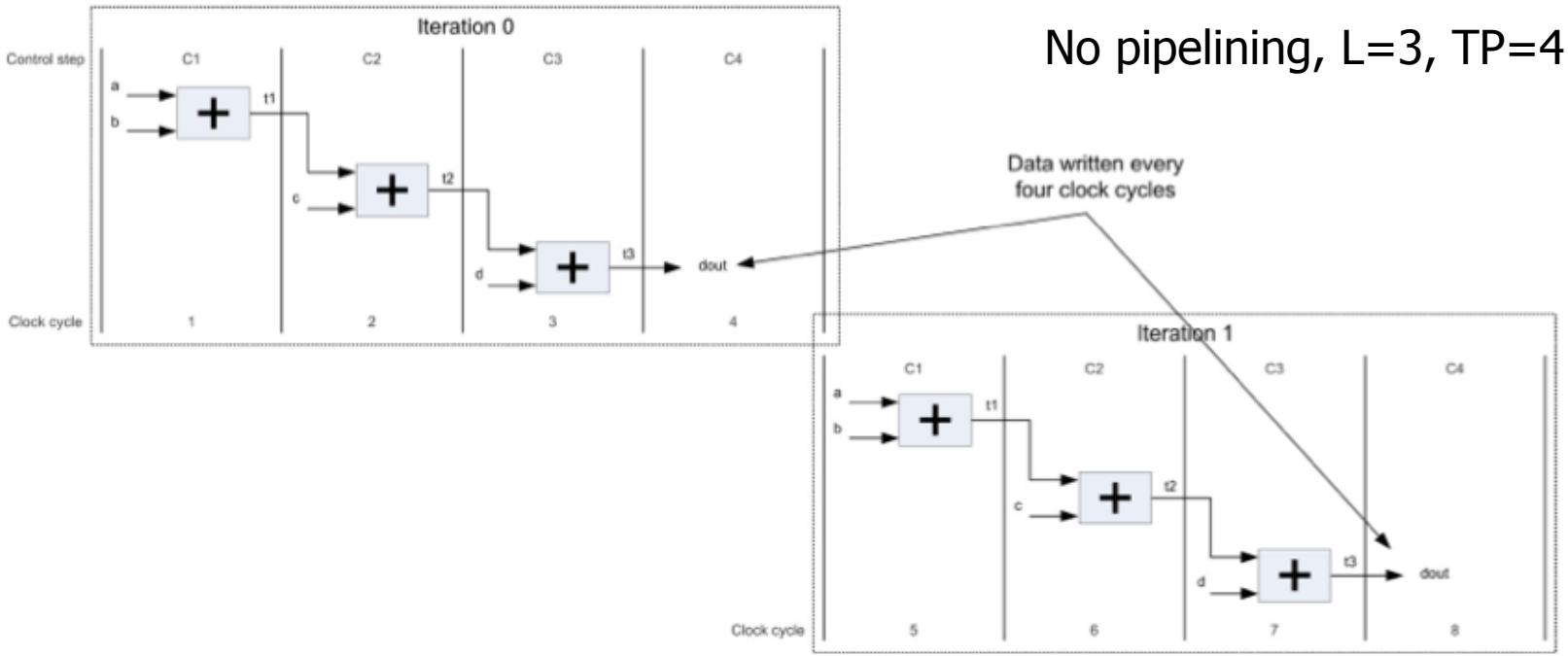


Tuning Design Performance

- Loops
  - Pipelining -> How often a loop can start
  - Unrolling -> multiple loop iterations in parallel
  - Merging

Loop Pipelining

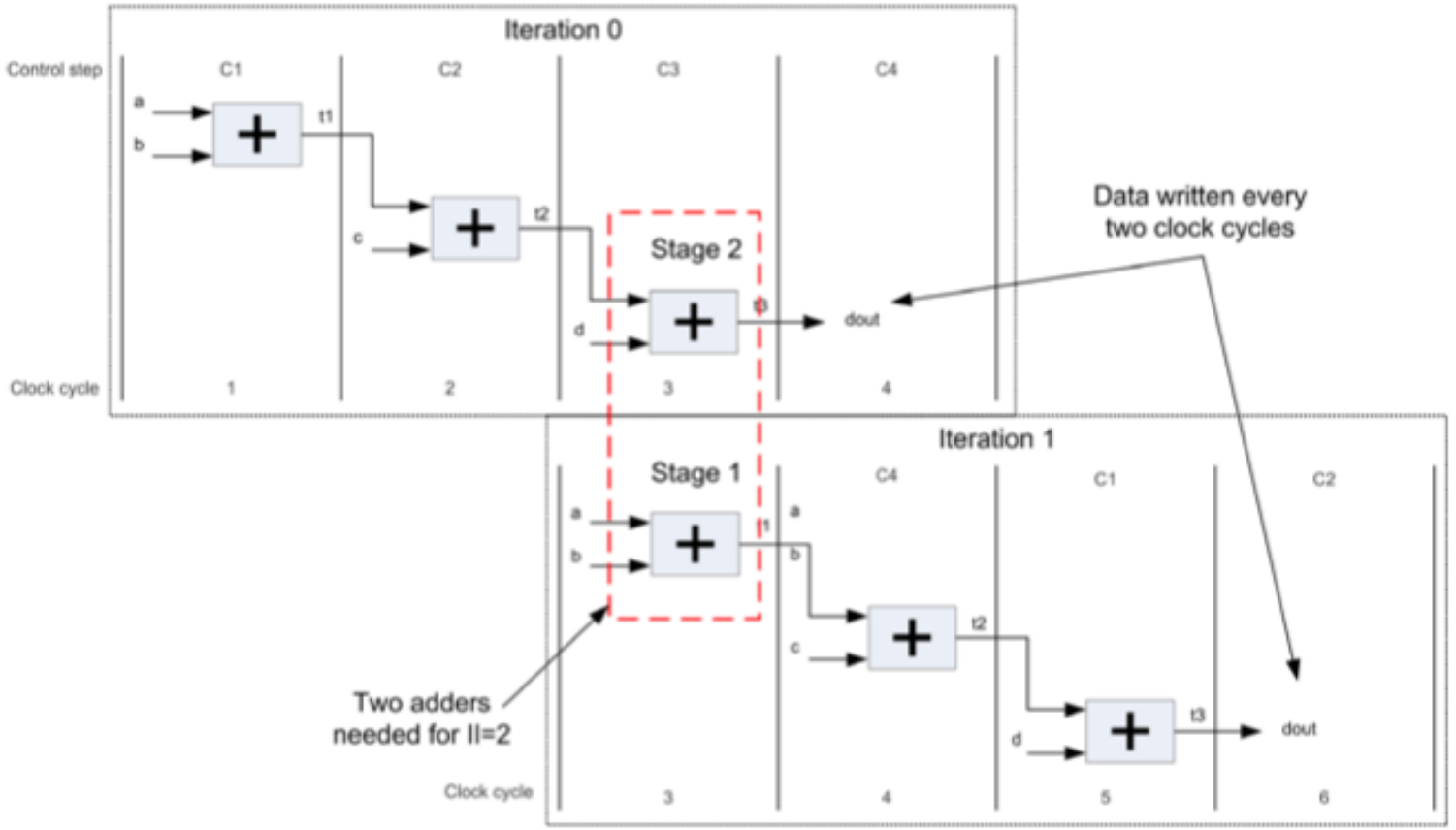
- Loop is implicit in the top-level function
  - Initiation Interval (II)
  - Latency (L)
  - Throughput (TP)

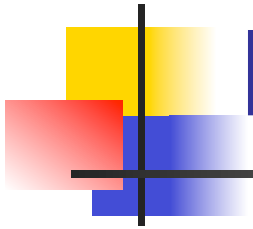


No pipelining, L=3, TP=4

Loop Pipelining (2)

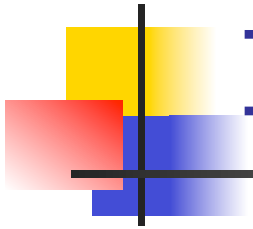
II=2, L=3, TP=2





Function Call

- Each function call, generates a new HW (It depends)
- Best to have a single “return”
  - Leads to sharing of resources.
  - Multiple “returns” may lead to undefined behavior (design error)



IO and Memories

- Unconditional IO (no handshake – wire type)
  - Pass by reference
    - Data is expected every clock cycle
    - Off-chip storage (memory or registers)
  - Pass by value
    - Register the data internally (similar to C)
    - Reduces throughput requirements
- Conditional IO (handshake in place)
  - Similar, but now more control of when the signal is available

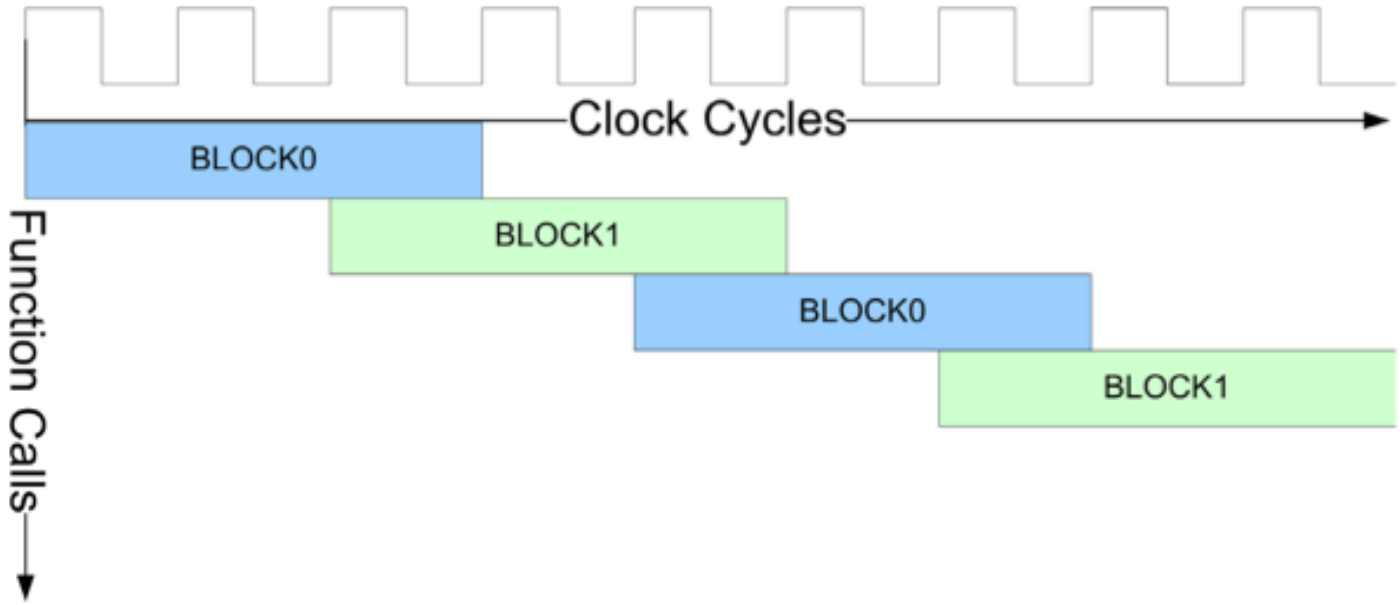
Hierarchical Design

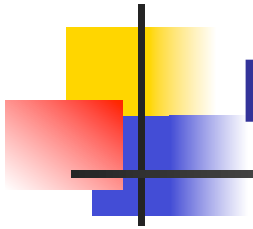
- Explicit Hierarchy
  - Two blocks write the same array

```
 1  void BLOCK0(int din[3],int dout[3]){
 2   WRITE:for(int i=0;i<3;i++){
 3      dout[i] = din[i];
 4    }
 5  }
 6  void BLOCK1(int din[3],int dout[3]){
 7   READ:for(int i=2;i>=0;i--){
 8      dout[i] = din[i];
 9    }
10  }
11  void top(int din[3],int dout[3]){
12    int tmp[3];
13    BLOCK0(din,tmp);
14    BLOCK1(tmp,dout);
15  }
```

Design Constraints
Main loop pipelined with II=1
All loops left rolled
All arrays mapped to registers

Out of Order Array Accesses in a Flat Design



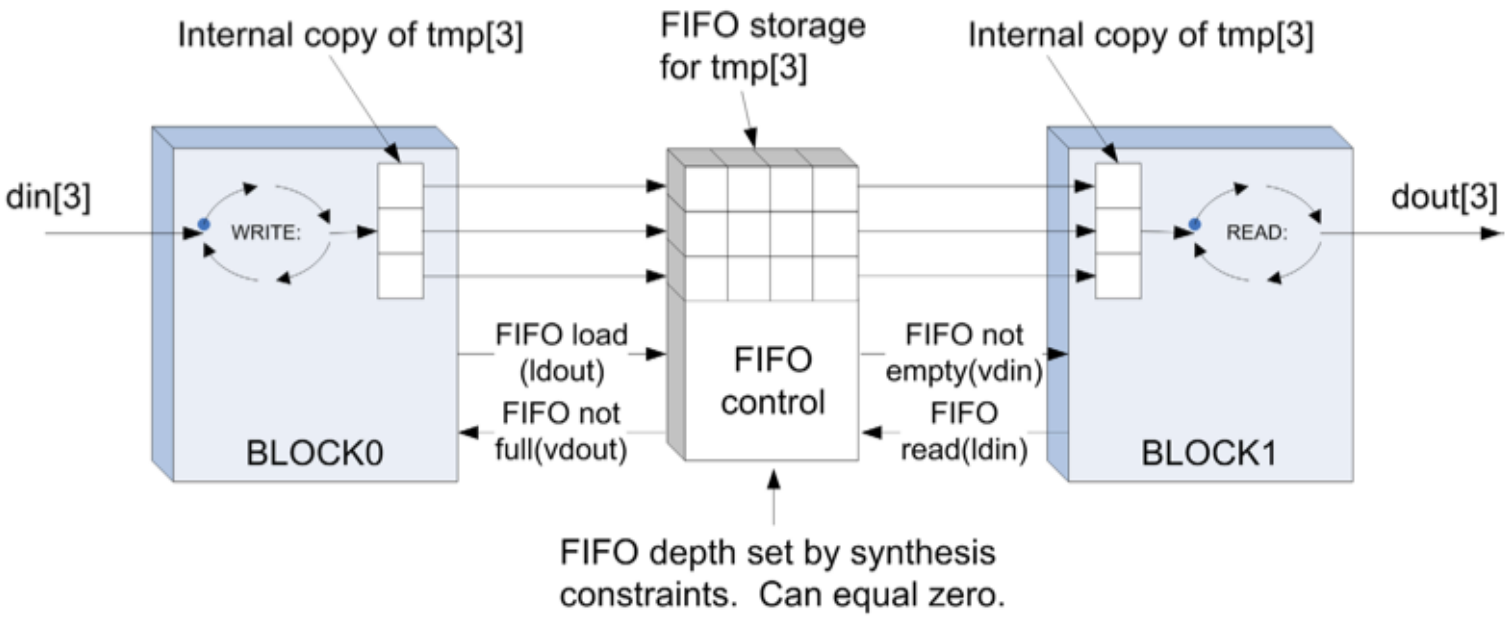


Functions constrained to be hierarchical

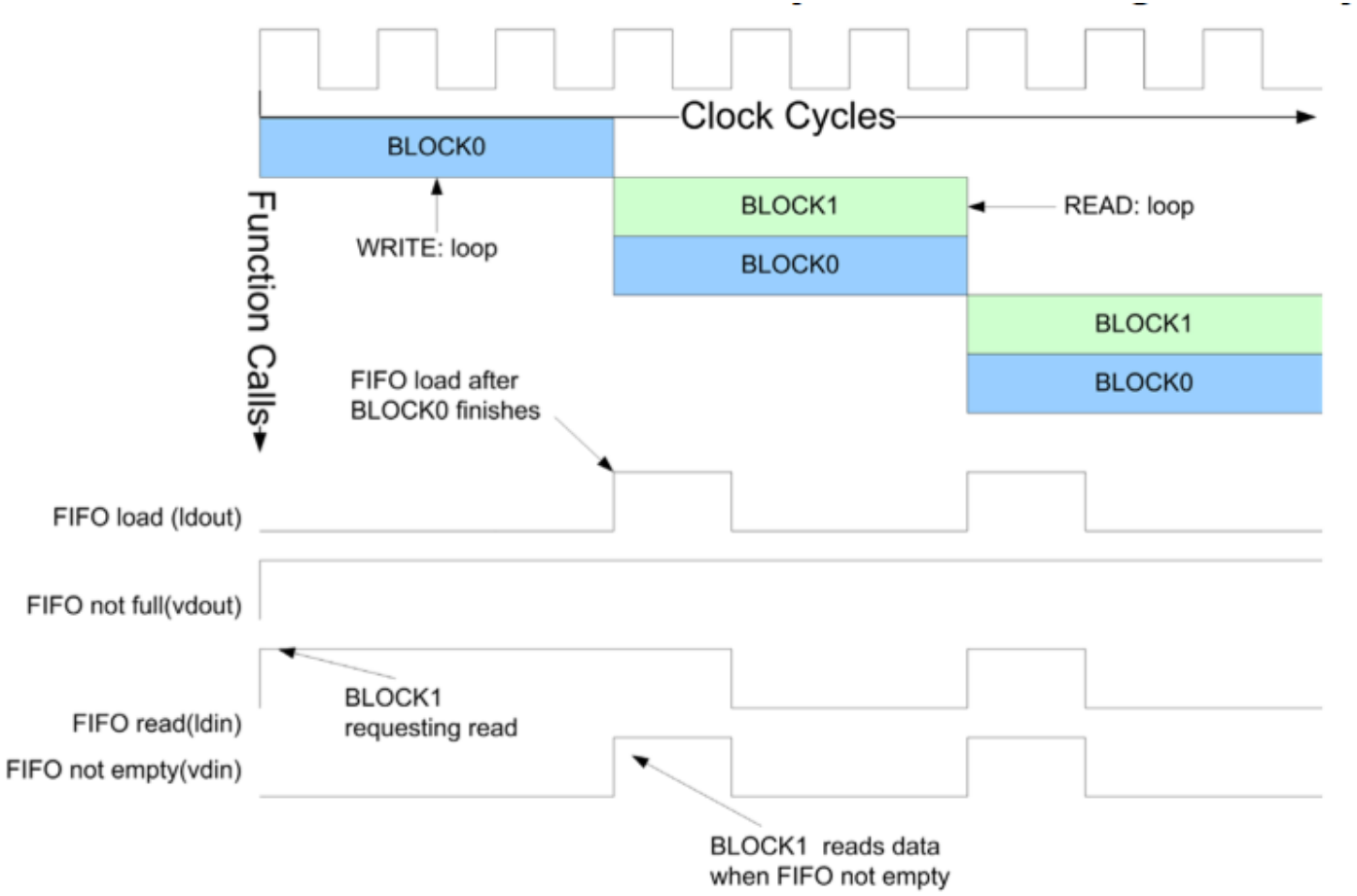
- Functions => Individual blocks
- Need for communication channels
  - Registers or memories

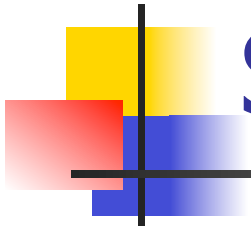
```
Design Constraints
BLOCK0 and BLOCK1 mapped to hierarchy
BLOCK0 and BLOCK1 pipelined with II=1
All loops left rolled
All arrays and channels mapped to registers
```

Array Accesses using Hierarchy and Registers



New schedule

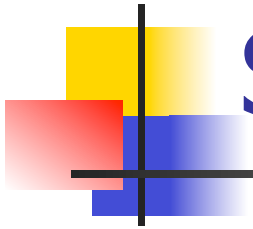




Streaming

- Previous problem
  - Indexing was in opposite order
  - Fixed by hand
    - FIFOs, ping-pong memories
- If index in the same order and index computation is unconditional
  - Automatic streaming
- In many designs, the streaming behavior needs to be coded into the algorithm
  - Use of Algorithmic C channel class (ac_channel)

```
ac_fixed<12,6> tmp = my_channel.read();
my_channel.write(tmp);
```

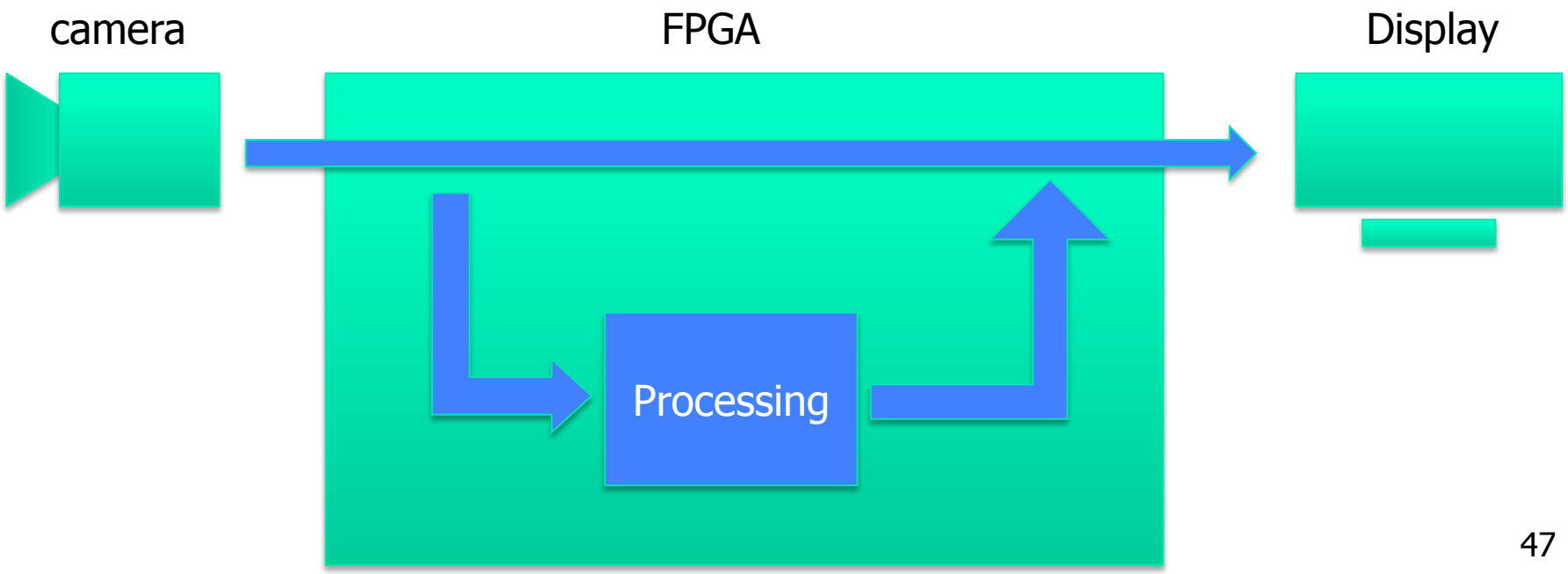


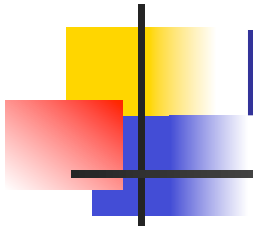
Some project ideas

- Image processing under the mouse
  - Edge detection, BW, zoom
- Fractals
- Image registration
- Ripple effect
- Rotate an image
- Image morphing
- ...

Restrictions

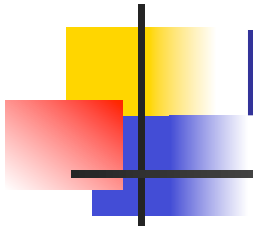
- Clock frequency @ 50Mhz
  - Time to go through the frame only once
  - streaming





Resources

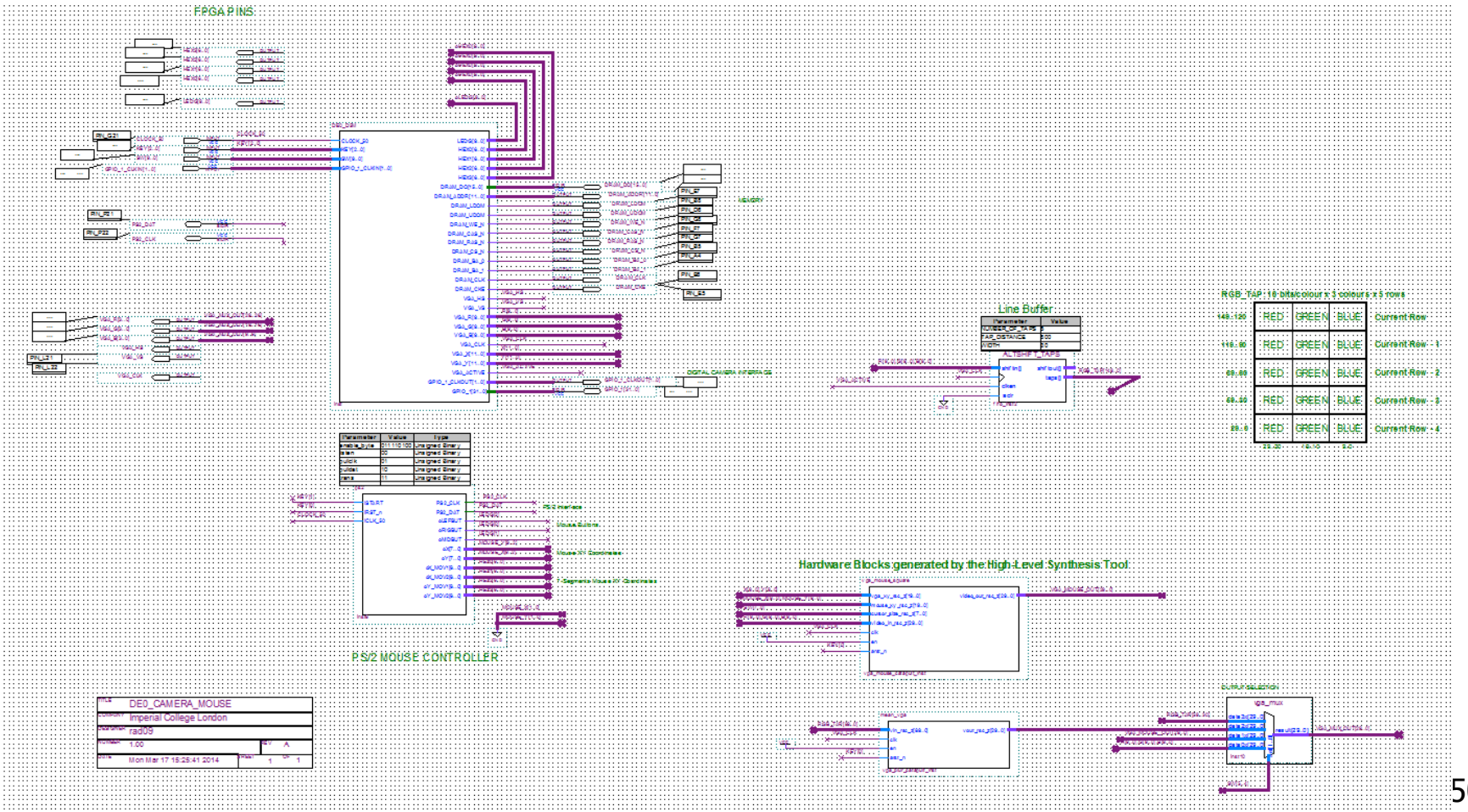
- Part 1
  - Simple dot product
- Part 2
  - Two demos on image processing
- Testbench + design on 2D filtering
  - vga_blur

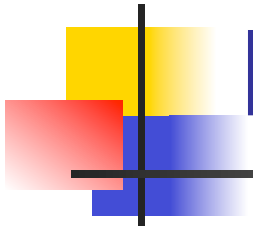


Design approach

- Build the whole project in Catapult-C
  - Doable
  - Needs experience
- Build subcomponents in Catapult-C, connect them in Quartus
  - Easier to implement and debug
  - Streaming

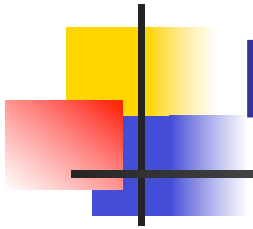
Available design





Reading material

- Catapult C Synthesis User's and Reference Manual
- High-Level Synthesis Blue Book



Examples you MUST go through

- High-Level Synthesis Blue Book
  - 2-D Windowing (pp. 184)
  - Digital Filters (pp. 229)

    (and maybe this one)
  - Hierarchical Design (pp. 191)