Reconfigurable Computer For Digital System Ware

1. Introduction

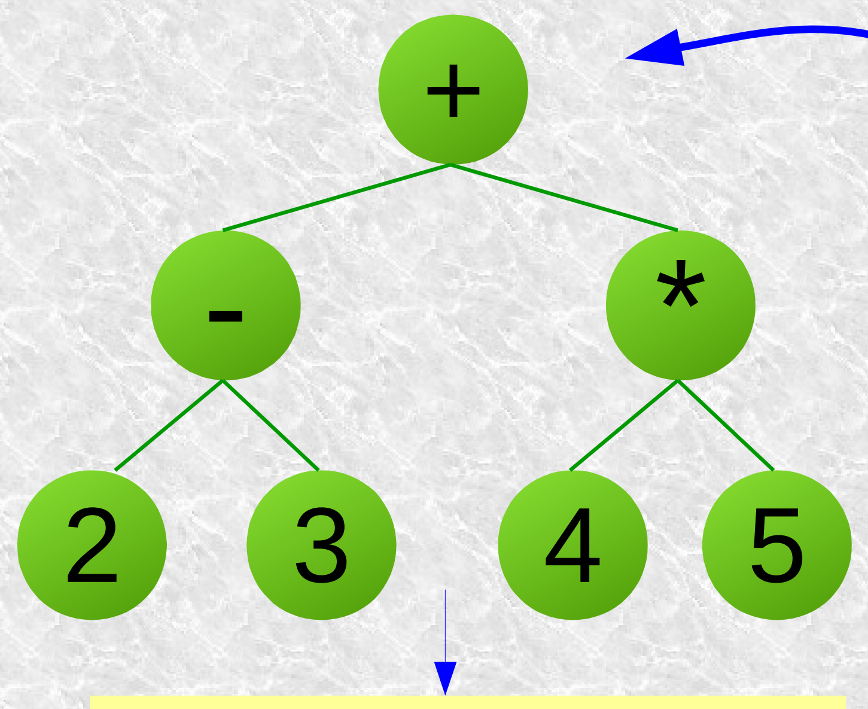
- Since the beginning of digital systems, CPU-based computing systems have been the common-place. However due to their recent decrease in prices, FPGA-based systems are now becoming common too. For this reason, many are exploiting possible applications.
- Indeed the current general purpose computing systems are based solely on the development of computer applications as **sequences of instructions** executed by a processor.
- The aim is to design a computing system where **every** computer application is a digital system logic executed on reconfigurable hardware. Such an application will be called a **Digital System Ware**, or simply, a **disyware**.

2. Background

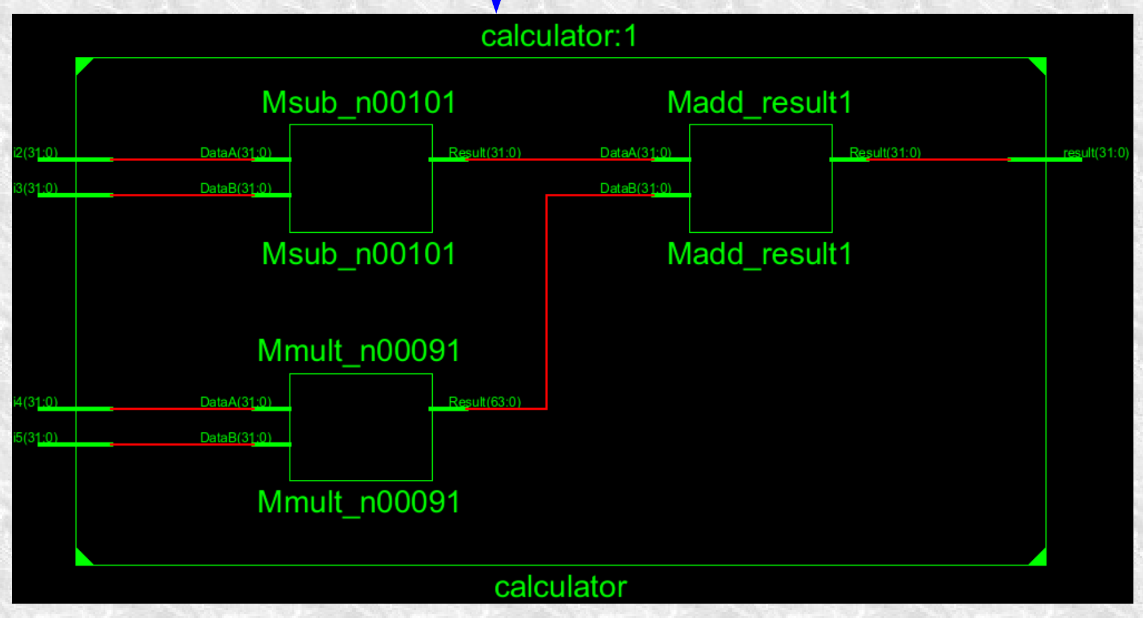
- The field of Reconfigurable Computing already features computer applications developed as digital systems logic. However **only a select few** applications or algorithms are developed, mainly so to achieve high performance co-processing work that the CPU is not suited for.
- Each application gets implemented in its own specific way for its own specific purpose. There is hardly any **generic framework** (or rule of thumb) to ease the development of applications. Indeed this makes reconfigurable computing hard, take long development time, require **high expertise** from the developer, and not welcoming to newbies to the field.

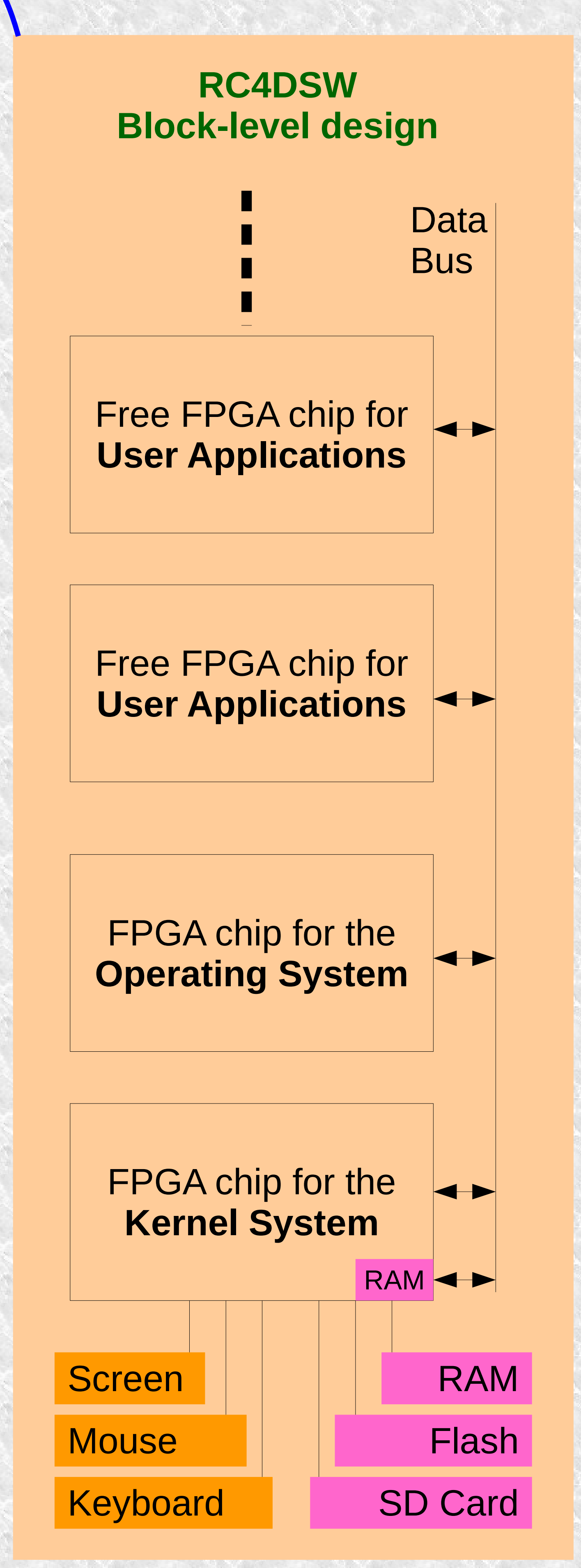
3. Methodology

- Previous research work has been made towards finding a generic framework, but which did not consider 2 fundamental difficulties:
  - 1st is an **inherently parallelised** framework design so to take full advantage of the inherently parallel nature of a digital system logic.
  - 2nd is a framework design that allows for a **large application** to be broken down into portions such that only those portions needing execution get loaded onto the FPGA (and thus be executed).
- Conventional applications development are characterised by what is called ***function-call***. An analogous concept called ***module-call*** is used here. Every hardware module implements several module operations. A module operation is associated with a unique ID used to call that operation, through a generic protocol called the module-calling protocol.



```
/* calculator.v
   This module does not
   need to process call_info.
*/
`timescale 1ns / 1ps
module calculator(
    //input [15:0] call_info,
    //input [31:0] stack_address,
    input [31:0] i2, i3, i4, i5,
    output [31:0] result
    );
assign result = i2 - i3 + i4 * i5;
endmodule
```



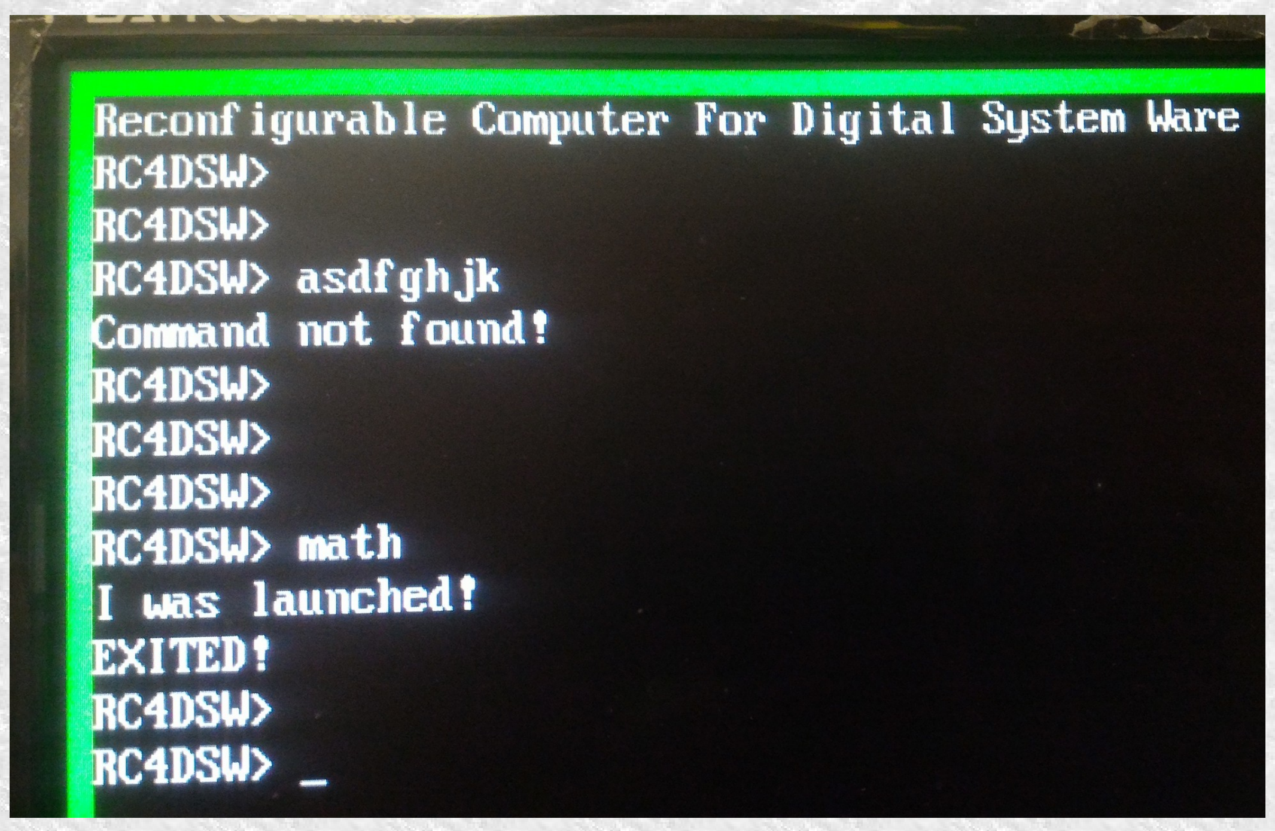

RC4DSW
Block-level design
Data Bus
Free FPGA chip for User Applications
Free FPGA chip for User Applications
FPGA chip for the Operating System
FPGA chip for the Kernel System
RAM
Screen
Mouse
Keyboard
RAM
Flash
SD Card

4. Design

- A large portion of the source code of an application is implemented in the usual way. The module-calling protocol is essentially just an **addition** to encapsulate module operations.
- A **caller** module calls a **callee** module by sending the (call_info, stack_address) data pair to the operating system (OS). The OS then forwards this data pair to the callee. The callee detects which operation to perform from the info.
- The call_info contains all the information needed to locate the callee module. When no FPGA chip implements the callee module the OS first replaces the bitstream of a **non-busy** FPGA chip with that which implements the callee module.
- The protocol uses the stack to store information as well as to pass module-call arguments. This in particular enables **recursive** module calls as well as **parallel** execution of module operations.
- The entire system is based on a **64-bit bus.**

5. Results (so far)

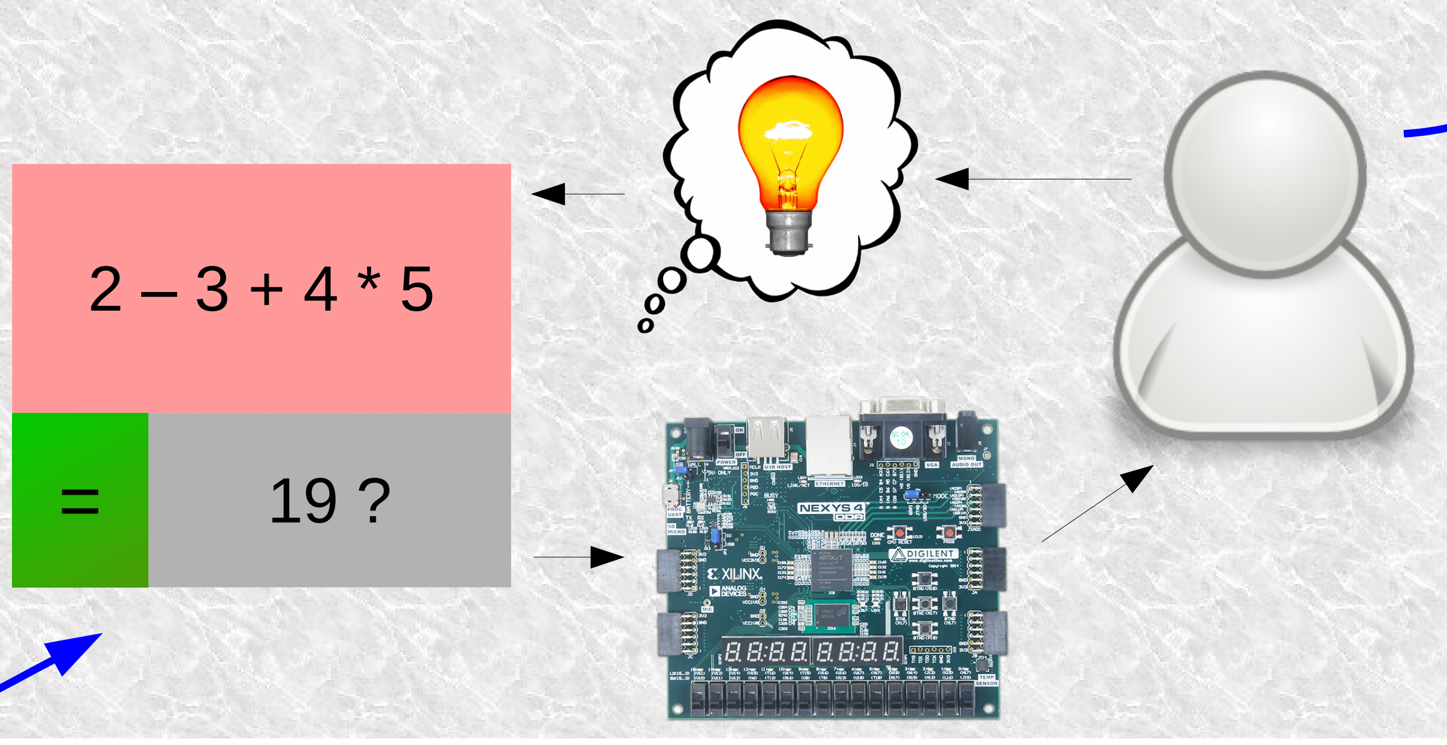
- Module calls introduce performance overhead. But this is relatively small compared to the work done per module operation. Precisely, the usual case where only one module-calling at-a-time is performed takes at most **10 clock cycles**.
- The figure below shows the screen output of the reconfigurable computer under execution. The GUI-based command line interface is used to execute other applications. This is similar to our current general purpose computers. Therefore indeed any application can be featured here.



```
Reconfigurable Computer For Digital System Ware
RC4DSW>
RC4DSW>
RC4DSW> asdfghjk
Command not found!
RC4DSW>
RC4DSW>
RC4DSW>
RC4DSW> math
I was launched!
EXITED!
RC4DSW>
RC4DSW> _
```

6. Way Forward

- The prototype deliverable will be a ready-to-use end-user **reconfigurable computer** that can load several end-user applications and execute all at the same time. Expected applications are:
- A simple graphical **calculator** (figure below) to illustrate GUI integration, a simple **synthesizer** to illustrate embedded system integration, and a simple **DSP tool** to illustrate high performance.



7. Applications

- With a new **high-level HDL** designed so that it inherently implements the framework, current areas with high usage of reconfigurable computing (such as Radio Astronomy and Digital Signal Processing) will have an increase in effectiveness and number of people involved.
- The merging of high performance computing with the ease of developing any application will bring reconfigurable computing closer to **completely replacing** the processor-based sequence of instructions computing paradigm.
- Yes indeed, the future of **general purpose** computing systems lies in the development of computer applications as digital systems logic executed on reconfigurable hardware.

By: Cyrille Yemeli Tasse , ymlcyr001@myuct.ac.za
Supervisor: Simon Winberg , simon.winberg@uct.ac.za