# Class Notes: Reconfigurable Computing Systems

Muhammad Obaidullah
*M.A.Sc. Candidate, Ryerson University, mobaidullah@ryerson.ca*

## I. Lecture 1: $10^{th}$ September 2015

### Definitions & Classification

### A. Computation Process & Architecture

1) **Task:** It is an information object that consists of algorithm and data structure.
2) **Algorithm:** It is formal representation of functions (operations) to be executed in determined sequence according to information dependencies.
3) **Data Structure:** It formal representation of data elements and their inter-dependencies.

**Algorithm** is implemented in a form of control flow.
**Data Structure** is implemented in a form of data flow.

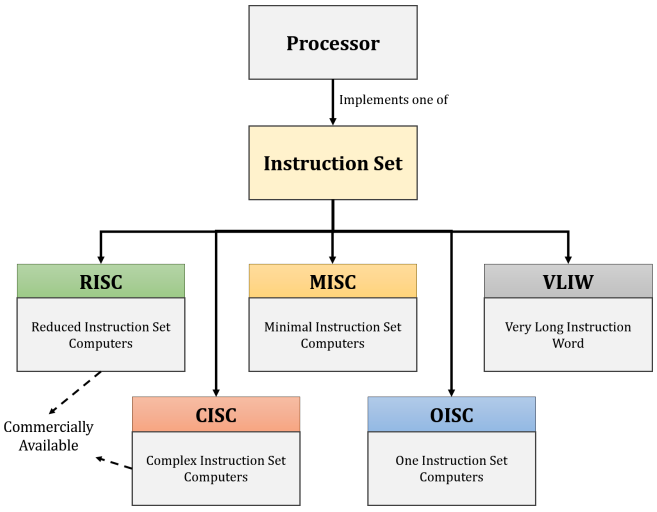



Figure 1: Types of instruction sets implemented by processors. RISC is generally preferred because of pipelining and fixed instruction execution time.

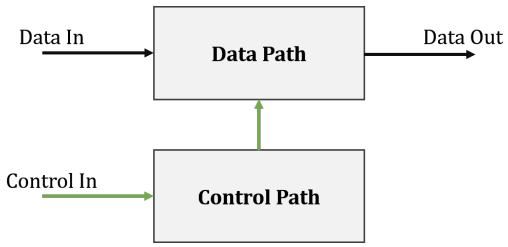



Figure 2: Control manipulates/defines the data path. A processor requires a control input but does not provide a control output.

4) **Computing Architecture:** $A_{CS}$ can be considered as a set of component(C) links(L) between components and functional procedures (P)

$$A_{CS} = \{C,\ L,\ P\} \tag{1}$$

## II. Lecture 2: $17^{th}$ September 2015

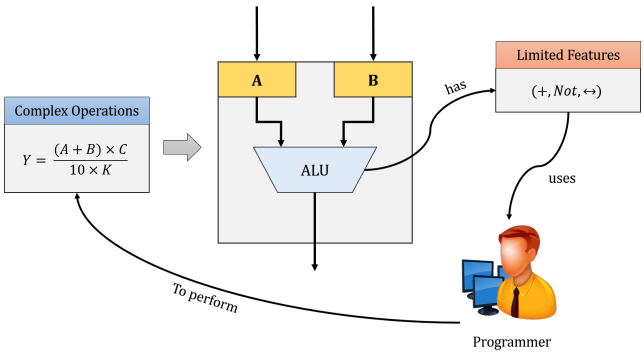



Figure 3: If the hardware is limited in the amount of operation it can perform, programmer can break down the task to implement is small manageable operations that the machine can do. For example if the machine can only do addition operation, the programmer can make it do multiplication by 5. This can be done by commanding the machine to add the number five times to itself.

### A. Correspondence between the task and computing architecture

Task can have different algorithms and data structure. Therefore, operational components, links, and procedure of processor may be different and need to conform the specifics of the tasks and specification constraints.
Thus,

$$Task\ 1 \rightarrow \{C,\ L,\ P_1\} \tag{2}$$

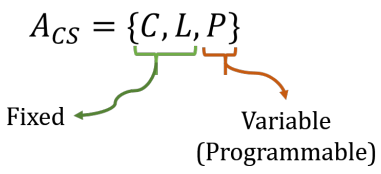
$$Task\ 2 \rightarrow \{C,\ L,\ P_2\} \tag{3}$$

### B. Concept of computing systems with programmable procedure

**Idea:** Maybe it is possible to find a universal set of elementary:

1) Data acquisition
2) Data processing
3) Data storage operations

to execute any algorithm and data structure on universal computing platform.
*In this case:*




**Mapping the task:** It is the process o loading certain control information (eg. sequence of instructions) to the memory control unit according to the structure of the task and mapping

of data to the data memory. The processors with universal data-execution path (data-path) general for any algorithm and universal memory general for any data structure where task-t-architecture adaptation is done by programmer were called computers with programmable procedure.

**Advantages:-**

1) Cheapest hardware implementation in comparison with other concepts.
2) Relatively easy adaptation of a task to computing platform.
3) Easy interfacing to uniformed peripherals.

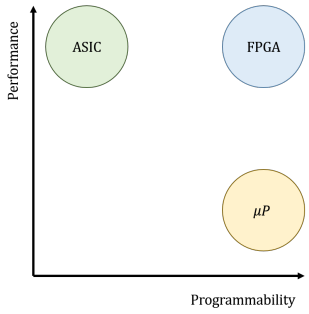
**Disadvantages:-**

1) Limited performance due to sequential nature of execution of control information.

### *C. Concept of processors with control information integrated in data-path: Application Specific Integrated Circuits (ASIC)*

**Idea:** Maybe it is possible to find the mechanism for architecture to task adaptation where integrated control information to the data path would conform task algorithm, data structure and performance constraints simultaneously. Therefore, for each task $T_i \rightarrow \{C_i,\ L_i,\ P_i\}$ associated architecture should be found.

This means that $\{C,\ L,\ P\}$ are variable. This is a concept of Reconfigurable Computing Systems (RCS).

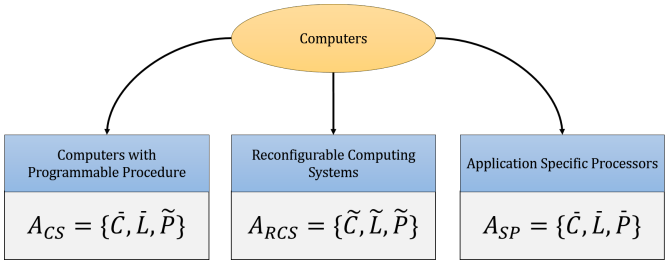



**RCS** is a class of computing systems, where functions of components, topology of links, between components and procedures of data-execution, data-storage, data-input/output transfer can be programmed.

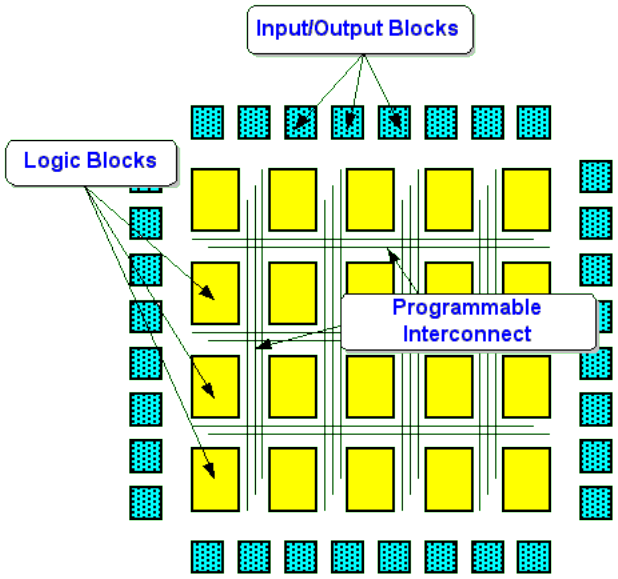
$$A_{RCS} = \{\tilde{C}, \tilde{L}, \tilde{P}\}$$

Variable
(Programmable)

### *D. Classification of Computing Systems*




### *E. General Organization of Programmable Logic Device Hardware platform of RCS*



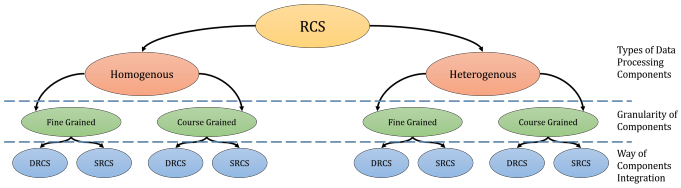

## III. Lecture 3: $24^{th}$ September 2015

**FCR:** Field of Configurable Resources is collection of arrays of :

1) logic resources
2) memory resources
3) interconnect resources
4) interface resources
5) special function resources etc.

FCR must be considered in 3 levels

1) System-on-Chip level (SoC) inside the programmable logic device (PLD)
2) System-on-Board
3) System level (which is multi-board)




**Homogeneous RCS** Consist of identical components communicating over identical information links.

**Heterogeneous RCS** Consist of different types of identical components communicating over different types o links.

**Fine Grained RCS** Consist of components performing elementary operation on 1-bit wide operands.

**Course Grained RCS** Consist of components performing macro-operations on multi-bit operands (words).

**Hybrid RCS** Consist of fine grained and course grained and associated links.

**Statically Reconfigurable RCS** Assumes architecture reconfiguration at start-up of each mode of operation.

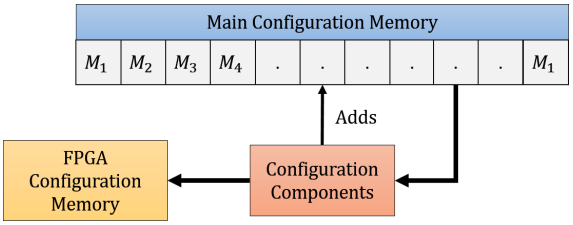



Figure 4: Statically Reconfigurable RCS stops all system and then reconfigures. This is one of the disadvantage.

**Dynamically Reconfigurable RCS** Assumes that part architecture can be reconfigured when the rest of the architecture continues non-stop operation.

**Task Mode** is one of possible combination of the algorithm and/or data structure.

$$720p\Big(1024 \times 720\Big)\ pixels\ \ 60fps/120 \tag{4}$$

$$1080p\Big(1920 \times 1080\Big)\ pixels\ \ 60fps/120 \tag{5}$$

## IV. Lecture 4: $1^{st}$ October 2015

**Task Mode** One of possible variant of algorithm & one of possible variant of data structure. Task modes are smaller tasks that need to be taken in order to complete the given task. For example, if the task is to fly to London from Toronto, the task modes can be driving to airport, boarding, flying, and finally landing. Additionally, task modes taken in order to complete task can change depending upon the required specification. For example, if the task is to fly to London from Toronto in ship. Then, the task modes will change to driving to docks, boarding the ship, cruising, and finally arriving.

**Task Segment** It is the part of a task or task mode associated with given part of algorithm and data structure initiated by certain event and terminated by completion or termination terminal event. In other words, it is sub-routine.

Task segmentation does not depend on task implementation. Implementation depends on task segmentation and technical specification. As a result , task segmentation is also known as functional specification.

Computers are information processing machines, humans are sense processing machines. Humans retrieve sense out of information. Machines processes sense and information.

### A. *Implementation of a task segment*

It can be implemented using resources associated with functionalities in the segment. It can be done in space **and** in time. It always requires BOTH space and time. The real question is how much of each.

**Temporal Implementation:** Tasks implemented exploiting time division. Tasks are taken in small chunks and executed one-at-a-time. Conventional CPUs perform tasks in time divided chunks one at a time. This time division is called clock cycle.

**Spacial Implementation:** Tasks implemented exploiting spacial division. Large chunks of tasks are implemented in actual hardware and executed in parallel. This kind of implementation is the basics of reconfigurable computing systems.

**Temporal Partitioning:** Temporal Partitioning of resources assumes mapping of different task segments set of resources in different periods of time.

Events occur at moments of time and two events "intitialization" event/moment and "termination" event/moment. Between these two events, is a time define as a period. Initialization and termination events/moments cannot occur at the same time, should occur one after another, and always in per-determined order.

**Spacial Partitioning:** Spacial Partitioning of resources is distribution of different parts of resources between different task segments in spacial domain.

### B. *Performance Acceleration*

*1)* ***Instruction Level Parallelism (ILP)****:* where the control information can be executed in parallel to data execution.

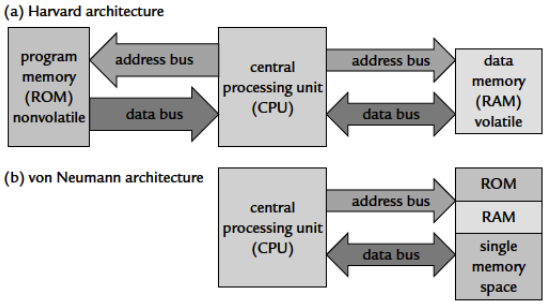



Figure 5: Differences between Harvard architecture and von Neumann architecture showing requirement of additional buses

*Results in architecture:* 2 types of memory (Instruction and data) and two associated buses.

*2)* ***Data Level Parallelism (DLP)****:* where data structure reflects natural parallelism and therefore can be divided on several segments to be processed simultaneously.

**SIMD** Single Instruction Multiple Data architecture utilizes data parallelism.

*3)* ***Branch Parallelism****:* where several branch of algorithm task segments can be executed in parallel simultaneously.

### C. *Example*

Let's take the following function to be executed by the processor:

$$Y = \sum_{i=1}^{n} \Big( \big(a_i + b_i\big)^2 \times K_1 + \big(c_i + d_i\big)^2 \times K_2 \Big) \tag{6}$$

where i = 1,2,3,....1024

The assembly code for performing this function in processor looks something like the following:

**PROGRAM**

```
Clear R5; \\(Result Accumulation)
Clear i = 0; \\(Clear pointer i)
Move a_i, R1;
Move b_i, R2;
Move c_i, R3;
Move d_i, R4;
Add R1, R2;
Add R3, R4;
Add R2, R2;
Add R4, R4;
Add R1, R2;
Add R2, R4;
Add R2, R4;
Add R4, R5;
Increment i;
if i < 1025 goto line 3;
```

Let's calculate how much time a normal micro-processor takes to calculate result:

$$T_{exe}(CSIS) = 2\, i_{latency} + \Big[10\, i_{simple} + 4\, i_{result}\Big] \times 1024 \quad (7)$$

where $i_{latency}$ is instruction latency (time taken) to clean up previous instructions and pointers, $i_{simple}$ is the number of clock cycles taken for executing a simple instruction, and $i_{result}$ is the number of clock cycles taken to store the result. Therefore for this example,

$$T_{exe}(CSIS) = 2 \times \Big[10 \times 5c.c. + 4 \times 6c.c.\Big] \times 1024 = 75786c.c. \quad (8)$$

## V. Lecture 5: $8^{th}$ October 2015

### A. Potential speedup of pipelined data instruction execution

**Data Level Parallelism (DLP)** Assuming that data structure is an array of **independent data elements**. Each of which should be executed according to set of functions $f_1, f_2, f_3 ... f_N$, the speedup can be determined as follows:

| Function | F1 | F2 | F3 | F4 | F5 | F6 | F1 | F2 | F3 | F4 | F5 | F6 | F1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Data Element | | | | | | | | | | | | | |
| D1 | F1 | F2 | F3 | F4 | F5 | F6 | | | | | | | |
| D2 | | F1 | F2 | F3 | F4 | F5 | F6 | | | | | | |
| D3 | | | F1 | F2 | F3 | F4 | F5 | F6 | | | | | |
| D4 | | | | F1 | F2 | F3 | F4 | F5 | F6 | | | | |

Latency

Cycle Time

Figure 6: Data level parallelism. The time it takes for the first result to come out is called latency. But after that time the results are pushed out within one clock cycle.

$$T_{exe.} = T_{latency} + (n-1) \times \tau_{cycle} \quad (9)$$

where $n$ is number of data elements in array

### B. Speedup of pipelined data path over non-pipelined (sequential) data-path

$$T_{exe.}^{non-pipelined} = n \times m \times \tau_{cycle} \quad (10)$$

where $n$ is number of data elements in array, $m$ is number of functions to be executed for each element and $\tau_{cycle}$ is the cycle time for a function.

$$T_{exe.}^{pipelined} = m \times \tau_{cycle} + \Big(n-1\Big) \times \tau_{cycle} \quad (11)$$

$$Speedup = \frac{n \times m \times \cancel{\tau_{cycle}}}{\Big(m+n-1\Big) \times \cancel{\tau_{cycle}}} = \frac{n \times m}{m+n-1} \quad (12)$$

In case when n >> m,

$$= \frac{n \times m}{m+n-1} \approx \frac{n \times m}{n} = m \quad (13)$$

$\therefore$ Speedup is equal to number of functions deployed in the pipeline.

### C. Side Topic: FPGA vs CPLD

FPGA has arrays of universal elements (implemented using LUTs) which can be programmed to act as any gate or memory elements whereas CPLD has arrays of different gates and a circuit is programmed of how these are connected.

CPLDs have timing issues whereas in FPGA, this problem is being solved nowadays by internal IPs in FPGA. Therefore, there is a minimum clock cycle requirement in FPGAs but not in CPLDs.

**The concept of LUT, Look-up-table as universal logic element**

At the time of programming, the LUTs are fed with the truth table of what needs to implemented and the multiplexers just forward that to output at run time.

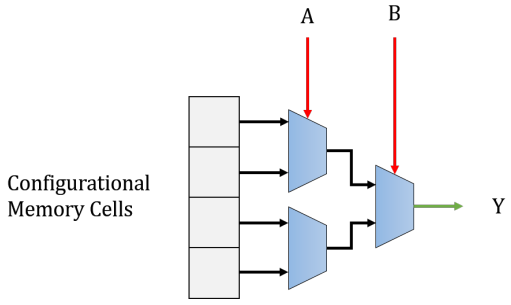


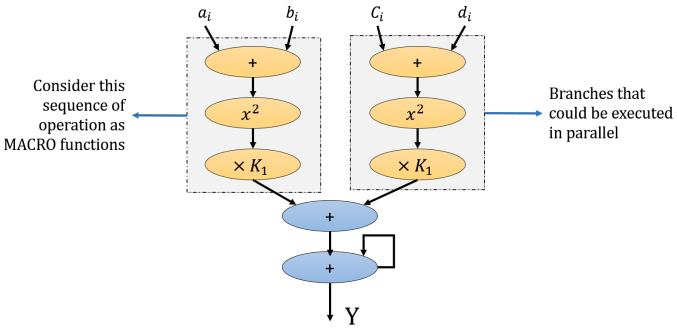

Figure 7: Any gate (operation) can be implemented using LUT. This is the basic building block of FPGA.

### D. Acceleration by using spatial parallelism in algorithm structure (Branch Parallelism)

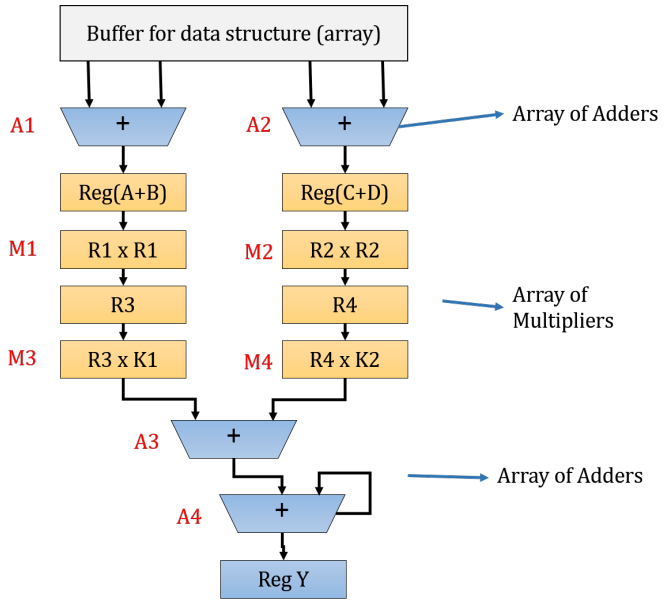
Very often in the algorithm, there are several operations which are independent of each other and can be executed in parallel. These kinds of operations can be speedup by creating parallel branches.

$$Y = \sum_{i=1}^{n} \Big( \big(a_i + b_i\big)^2 \times K_1 + \big(c_i + d_i\big)^2 \times K_2 \Big) \quad (14)$$

Operation $a_i + b_i$ can be performed in parallel with operation $c_i + d_i$. Branch level parallelism makes use of both spatial and temporal parallelism.




*E. Speedup by exclusion of control information execution. Creation of function specific circuit (ASIC)*




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A1 | A+B | | A+B | | | | | | |
| A2 | C+D | | C+D | | | | | | |
| A3 | | | | | Sum 1 | ✪ | Sum 2 | | |
| A4 | | | | | | | Sum 1 | ✪ | Sum 2 |
| M1 | | $(A+B)^2$ | | $(A+B)^2$ | | | | | |
| M2 | | $(C+D)^2$ | | $(C+D)^2$ | | | | | |
| M3 | | | $(A+B)^2\times K_1$ | | $(A+B)^2\times K_1$ | | | | |
| M4 | | | $(C+D)^2\times K_2$ | | $(C+D)^2\times K_2$ | | | | |

First Operation

Second Operation

Figure 8: Whenever each process is finished, a flag is raised and the next operation starts only when the previous operation is done and clock cycle is received.

*F. Project*

Write literature observation project. Use these big places:

1) IEEE Xplore
2) ACM
3) US Patents

**Topic:** Reconfigurable architecture of ..... application.
**Keywords:**

1) FPGA
2) Computation
3) Adaptation

**Abstract:**

- **Motivation** (problem statement)
- **Objective** (What is the goal of the project)
- **Results** (What you did)

**Introduction:**

- **Background** (try to analyze about 10-15 abstracts or as many as possible)

**Analytical Part**
**Summary**

## VI. Lecture 7: $29^{th}$ October 2015

*A. Concept of macro-function and functional units*

**Macro-function (Macro-operation)** is function which encapsulates relatively *complex part of algorithm* commonly used in particular class of applications associated with per-determined data structure.

$$MF_i => AS_i \& DS_i \tag{15}$$

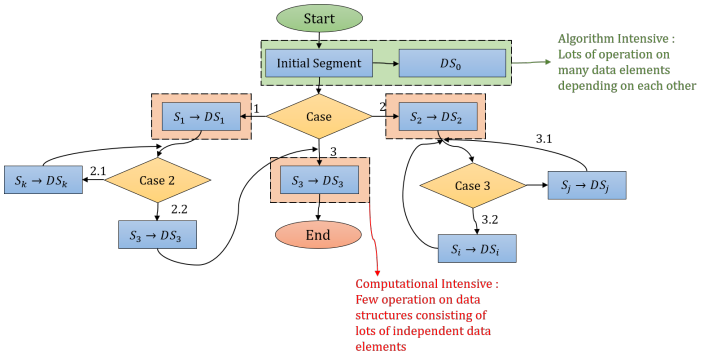
Examples

1) IR, FIR FFT etc. in DSP
2) Edge Detection, Color Conversion (eg. Sobel algorithm) $(Y^{'}C_BC_R)$

**Functional Unit** is the function-specific computing circuit *optimized for given function* or set of functions in such a way that it satisfies set of performance constraints and minimizes/-maximizes one parametric objective.

$$FV_i => MF_i => \begin{cases} FV_{i1} \rightarrow \{PC_1\} \\ FV_{i2} \rightarrow \{PC_2\} \\ ..... \\ FV_k \rightarrow \{PC_k\} \end{cases}$$

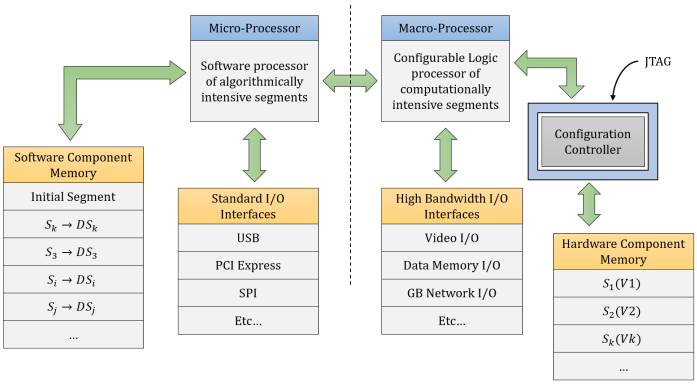
**Idea:** It may be cost effective to couple micro-processing module (unit) and reconfigurable logic block (unit), which can be configured to macro-function specific unit and thus, accelerate the entire data execution process. This reconfigurable logic block was called reconfigurable hardware accelerator.




Any task in application may have number of algorithmic intensive segments and number of architectural intensive segments.

### B. General architecture of Reconfigurable Computing Systems

**Component** is an information object which represents the information of associated functional unit in form acceptable for data-execution platform.




## VII. Lecture 8: $5^{th}$ November 2015

### A. Virtualization of Computing Architecture

**Task segment implementation according to general RCS architecture**

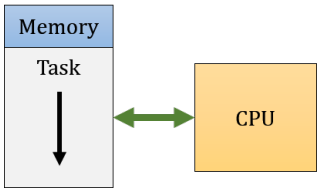
1) Task segment can be implemented in different forms according to set of constraints as:
   - **Software** (procedural) components (eg. routing, driver, etc.)
   - **Hardware** (dedicated to segment circuit)
2) Each segment can be implemented in software or hardware form depending on specification constraints. Therefore there could be a set of components associated wit given segment

$$S_i => \begin{cases} C_{i1} \rightarrow set\ of\ constraints\ set\ i\ 1 \\ ..... \\ C_{in} \rightarrow set\ of\ constraints\ set\ i\ n \end{cases}$$

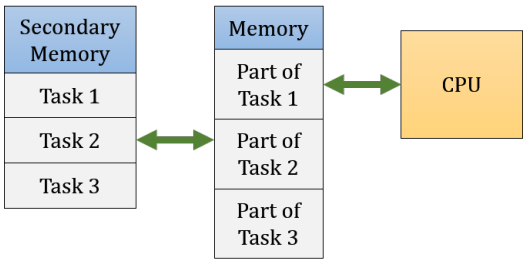
**Virtual Component:** It is information object representing given segment (function and data structure) in form which satisfies all specification constraints. Thus, there can be:

1) Virtual software component (VSC) $\rightarrow$ software routine, interrupt service routine, device driver, etc.
2) Virtual Hardware Component (VHC) $\rightarrow$ configuration bit file or configuration bit stream.

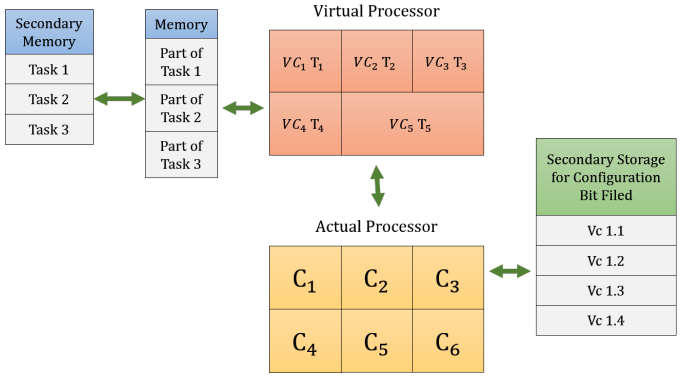
**Actual Systems**




**Actual Processor & Virtual Memory**




**Virtual Processor & Virtual Memory**




**Application Specific Processor (ASP)** is the circuit optimized for ASP can be implemented in different form depending on:

1) Variations in work load
2) Variations in environmental parameters
3) Ratio between algorithmically intensive and computationally intensive segments of the task

as:

1) **ASIC - Application Specific Integrated Circuit** (one hardware component) for cases when no variations of workload exist (mono-task) and most of task segments are computationally intensive.
2) **ASIP - Application Specific Instruction Processor** (CPU + hardware accelerator) for applications where workload is dynamic but most of segments are algorithmically intensive.
3) **ASVP - Application Specific Virtual Processor** general form of any processor architecture for cases when workload and environmental constraints are dynamic.

ASVP $\rightarrow$ How to integrate VHCs and VCSs ?

## VIII. Lecture 9: $12^{th}$ November 2015

### A. Architecture Organization of ASVP in on-chip level of RCS

**DPR (Dynamically Partially Reconfigurable) Systems**

*1) **Statically reconfigurable SoPC (System-on-Programmable Chip)**:* Applicable (effective solution) in case when:

1) Number of modes are in range of 4-5 to 40-50.

2) Period of time allowed for mode switching is greater than complete reconfiguration time for target FPGA(s).

**Why is it effective ?**

Let's take the following application example:-
Application ==========
2000 system gates are static part in all application modes. (Example system Clock distribution circuit and other common stuff)
Mode 1 10,000 requires system gates
Mode 2 8,000 requires system gates
Mode 3 9,500 requires system gates
Mode 4 12,000 requires system gates
Mode N 6,500 requires system gates
==========

System Gates Required to implement system ==========
10K - 2K = 8K
8K - 2K = 6K
9.5K - 2K = 7.5K
12K - 2K = 10K
6.5K - 2K = 4.5K
———————
$\sum$ = 36K
+ 2K
———————
38K
==========

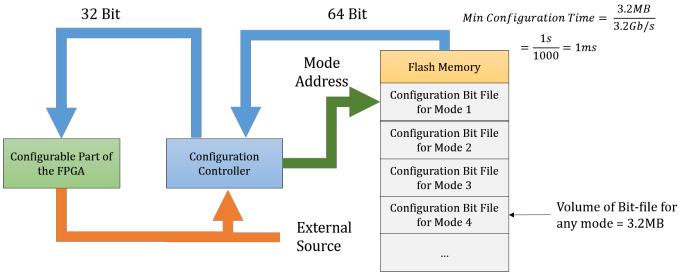



Figure 9: Offline reconfiguration of the FPGA

*2) Dynamically reconfigurable SoPC (System-on-Programmable Chip):*

1) With temporally reconfigurable resources
2) With spatially reconfigurable resources

**Temporally run-time reconfigurable RCS**

$$\tau_{cycle} = \frac{1s}{10\ frames} = 16.66ms \quad (16)$$

Example:

$$XGA\ Resolution = 1024 \times 768 \approxeq 0.8Mpixels \quad (17)$$

$$Execution\ Time = 0.8M \times 1pixel/cc. \times 5ms/cc. \quad (18)$$

$$= 0.8 \times 10^6 \times 5 \times 10^{-9}s = 4 \times 10^{-3}ms \quad (19)$$

**Timing diagram for temporally reconfigurable RCS**

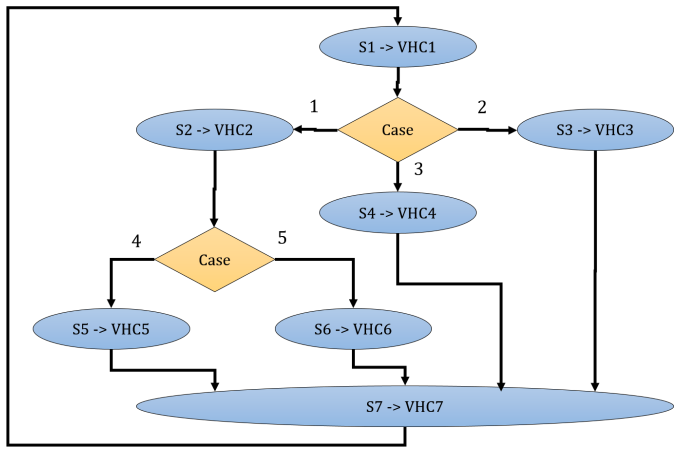
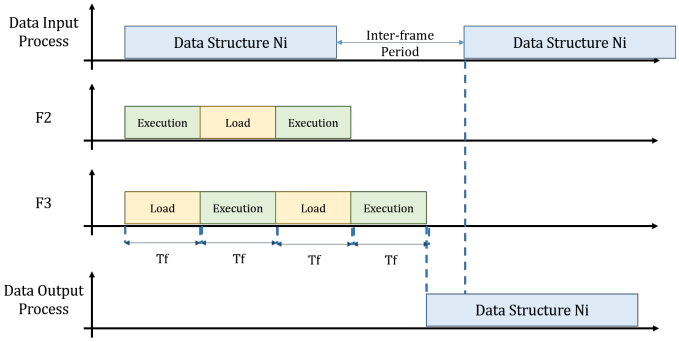



Figure 10: Example Application of Temporally run-time reconfigurable RCS




**Spatially reconfigurable RCS**

The entire space of resources (Field of Configurable Resources) is divided on slots (identical or non-identical) each of which is dedicated on segment specific component configured by respected VHC.

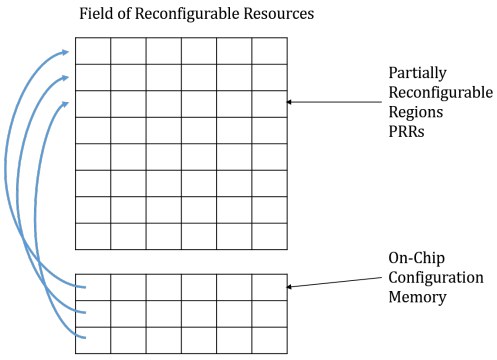
## IX. Lecture 10: 19$^{th}$ November 2015

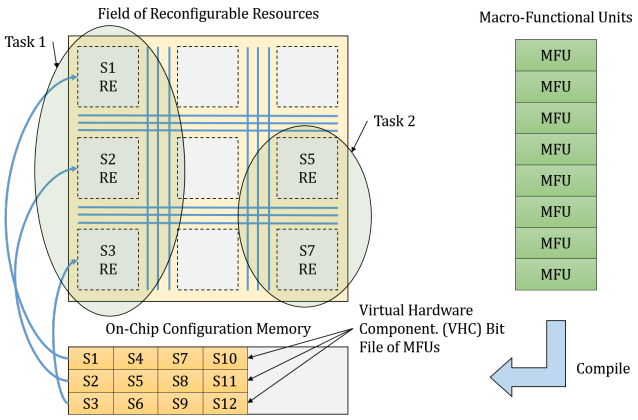
### A. Important Dates

- November 19
- November 26 0.5 + 0.5 Review
- December 3 Final Exam

### B. Spatially Reconfigurable RCS

Each task is considered as segment. Macro-Function Unit (MFU).






### C. RCS Development Process

Is divided on:

1) **Analysis of functional and technical specification**
   *Analysis of Functional Specification:* Description of workload: algorithm and data-structure segmentation according to events(modes of operation) and data-structure associated with algorithm segments.
   *Analysis of Technical Specification:* Determination of performance constraints for each segment multi-objective.
   Sets of Components and determination SW or HW implementation.

2) **Design of Hardware Components**
   *Input is:* Functional specification for Task segmentation associated with this component. Technical Specification for Task segmentation associated with this component.
   *Output is:* **High level** $\rightarrow$ symbol and scheduled sequencing graph binded with resources.
   **Low level** $\rightarrow$ configuration bit-file for the component

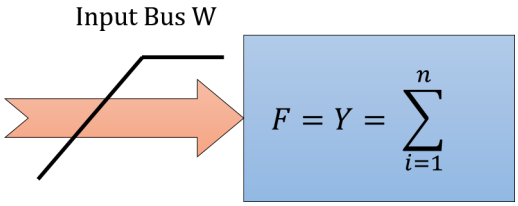
3) **Component Design Process**
   a) Determination of component's bandwidth: Input bandwidth & Output bandwidth.
   b) Determination of component's synchronization and control procedures.

4) **Component's Interface Determination**

5) **Symbol of Component**

6) **Entity Part of HDL Code**

### D. Example




Cycle Time 4c.c.
$a_i, b_i, c_i, d_i$ each 8 bits

$$Input\ Bandwidth = \frac{4 \times 8\ bits}{10\ ns} = \frac{32\ bits}{10 \times 10^{-9}\ s} \quad (20)$$

$$= \frac{32 \times 10^9}{10} = 32 \times 10^8 bits/sec \quad (21)$$

$$= \frac{32 \times 10^8 bits/sec}{1024 \times 1024 \times 1024} = \frac{4 \times 10^8}{1024 \times 1024} = 381.46\ MB/sec \quad (22)$$

$$Clock\ Frequency = 400\ MHz \quad (23)$$

$$\tau_{c.c.} = 2.5ns \quad (24)$$

$$Cycle\ Time = 4c.c. \times \tau_{c.c.} = 10\ ns \quad (25)$$

$$XGA\ Video\ Frame = 1024 \times 768 \times 24 bits/pixel \quad (26)$$

$$@60fps : t_{frame} = \frac{1}{60} = 16.66\ ms/frame \quad (27)$$

$$= \frac{16.6\ ms}{1024 \times 768 \times 24\ bits} = 1.13 bits/ns = 6 bits/c.c \quad (28)$$

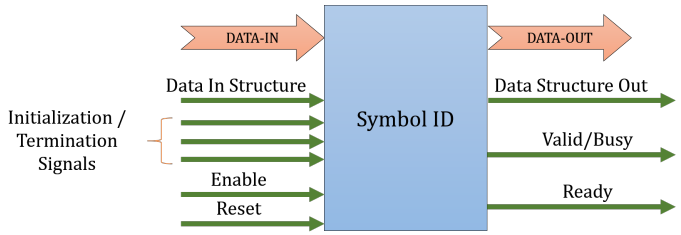
$$t_{c.c} = 200\ MHz, \tau_{c.c.} = 5ns \quad (29)$$

$$Cycle\ Time = 4c.c. = 4 \times 5\ ns = 20\ ns \quad (30)$$

$$20\ ns \times \frac{1.13\ bits}{ns} = 20.26 = 21 bits/cycle \quad (31)$$

### E. Some points about project

1) Initiation (Enable/Set) and Termination (Disable/Reset)
   Example: For video processing, vertical synchronization, horizontal synchronization are initiation(low-level). Termination (High-Level) Signals and symbols of component.
2) Data-in Synchronization
3) Data-out Synchronization




## X. Lecture 11: $26^{th}$ November 2015

### Function-specific Architecture Design

**Example of function:** Video-frame color compaction (R, G, B $\rightarrow$ enhanced B/W image)

$$Y_i = k \times (R_i + B_i + G_i)^2 \quad (32)$$

**Example of Data Structure:** Video-frame XGA $\rightarrow$ 1024 pixels/row $\times 768\ rows$.
R, G, B values $\rightarrow$ 8 bits/pixel in Bayer pattern.

**Constraints:** 120 fps

### A. Creation of Sequencing and Scheduling Graph

Sequencing graph is not interruptible. In other words, when you started instruction execution, nothing can interrupt in between.

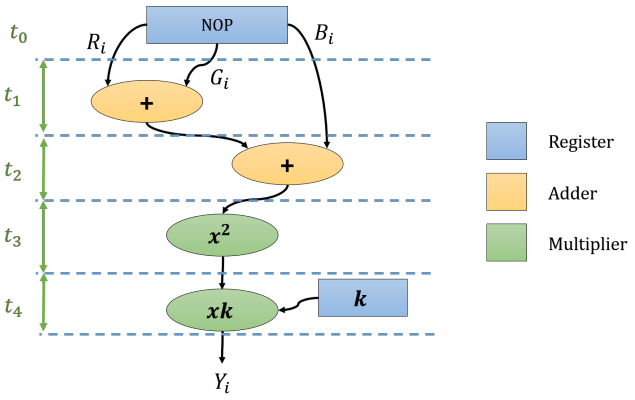



Figure 11: This type of sequencing graph should be drawn for application before implementation to analyze the specifications of design and finalizing implementation.

### B. Creation of Timing(Scheduling) Diagram

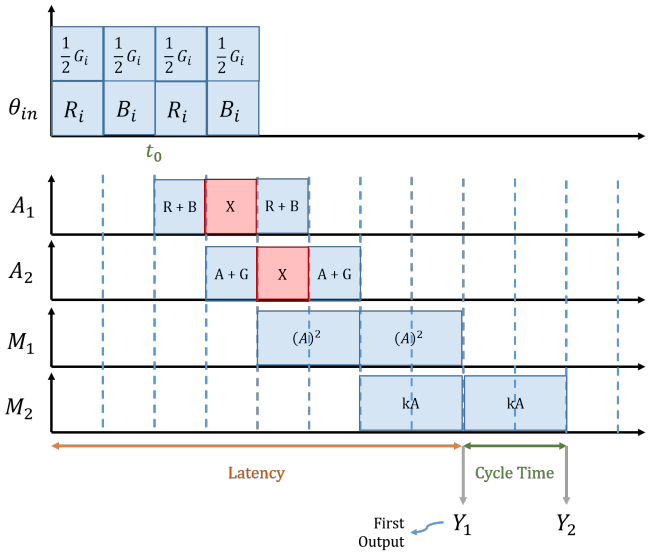



Figure 12: In this type of implementation The adders are having some free time. It may be a good idea to use one adder and do both of connection. However, this immediately means multiplexing the inputs and outputs in time to prevent short circuits. Eg. In one cycle, the output of the adder is connected to a register to store the intermediate value and in another cycle, the output is connected to M1. (Multiplexing output)

Play Tetris with the timing graph to adapt and fit as many operations as possible to get fastest possible output.

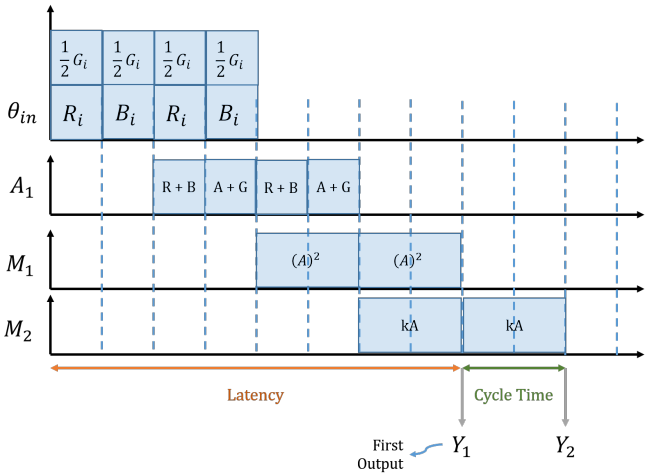



Figure 13: A Single adder can replace A1 and A2 and do the job of both adders. However there is addition of multiplexer hardware to multiplex input and output.

### C. Creation of Block Diagram

Then a block diagram is drawn according to scheduled and binded sequencing graph.

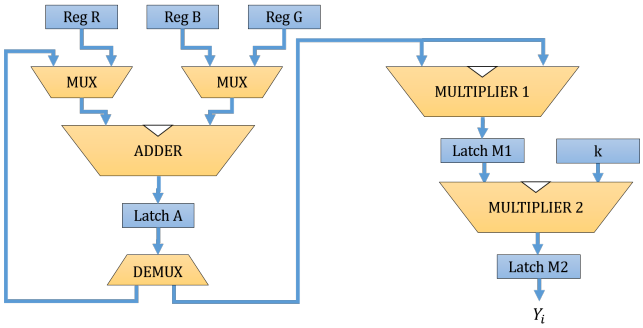



Figure 14: The scheduling diagram does not contain time consumed by wrappers which are used to latch or control data such as a Latch. These times are revealed after drawing a block diagram.