Reconfigurable Hardware

Pao-Ann Hsiung

Embedded Systems Laboratory

National Chung Cheng University

Chiayi, Taiwan, ROC.

http://www.cs.ccu.edu.tw/~pahsiung/courses/rc/

pahsiung@cs.ccu.edu.tw

Outline

- Reconfigurable vs. Conventional Hardware
- Hardware Preemption and Relocation
- Area-Time Tradeoff Techniques
- Communication Architectures

Outline

- Reconfigurable vs. Conventional Hardware
- Hardware Preemption and Relocation
- Area-Time Tradeoff Techniques
- Communication Architectures

Reconfigurable vs. Conventional Hardware

- What is reconfigurable hardware?
  - Hardware designs that can be configured for use as and when required
    - Require a well-defined standard interface
    - Require a common communication infrastructure
    - No globally mapped memory address
    - Special case: Swappable Hardware
      - With hardware preemption (context save/restore)
- What is conventional hardware?
  - All other hardware

Reconfigurable vs. Conventional Hardware

Feature	Reconfigurable HW	Conventional HW
Existence	Configurable	Fixed
I/O Interface	Data flow-based	Bus-based
Aspect Ratio	Column	Square
Module Interface	Bus/Slice Macro	Signal
Preemption	Possible	Impossible

Outline

- Reconfigurable vs. Conventional Hardware
- Hardware Preemption and Relocation
- Area-Time Tradeoff Techniques
- Communication Architectures

Hardware Preemption

- Preemptible or Swappable Hardware
  - Hardware with the ability to
    - suspend execution,
    - save and restore its state and context, and
    - resume execution
  - Example
    - A DCT design that can continue transforming the next 8x8 pixel block in an image after being suspended, context saved, and then restored.

Hardware Preemption

- Two methods of hardware preemption
  - Configuration based
    - Uses the configuration readback capabilities of the underlying reconfigurable fabric
  - Design based
    - Hardware design enhanced with preemption capabilities

Configuration based Preemption

- Readback
  - All SRAM configuration bits can be read back through the configuration port by a controller
  - Can be used for
    - Real-time debugging
    - Execution context extraction and saving
    - Fault tolerance

Configuration based Preemption

- Readback
  - Readback data cannot be used directly for reconfiguration
  - Requires time in the same order as configuration
    - 10x to 100x ms
  - Supported currently only by Xilinx Virtex series FPGA
  - Disabled when data is encrypted

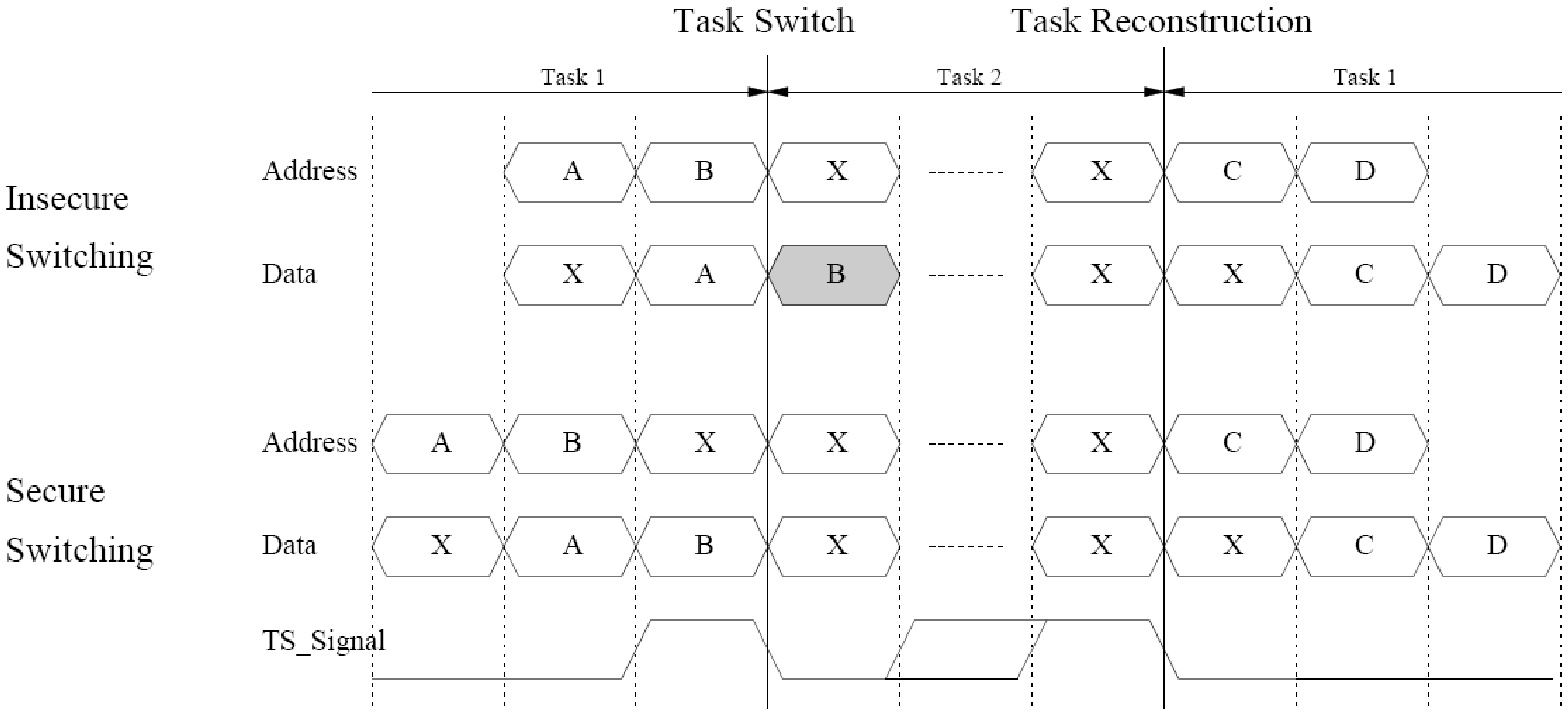
Configuration based Preemption

- Two methods:
  - Full readback [1]
    - All data is read back
      - XC4028EX: 668Kb, 800 ms
      - XCV400: 1.75 Mb, 14.4 ms
  - Partial readback [2]
    - Only frames containing state information are read back (at most 8% of full readback data)
      - XCV2000E: 12.5 ~ 451 Kb, 0.033 ~ 1.2 ms

Full Readback Preemption

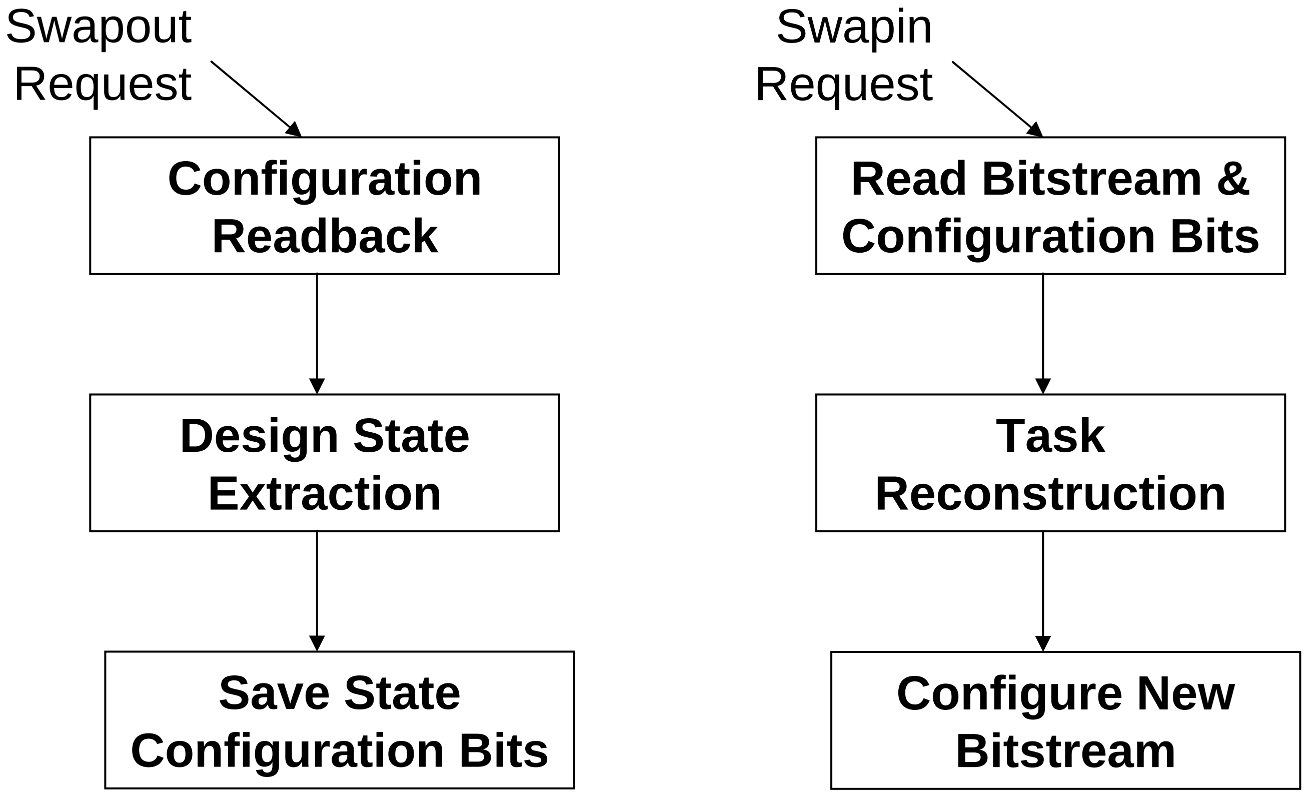
- Requires
  - – Configuration readback support by FPGA
    - Fast readback is desired
  - – Complete control of all clocks in design
    - Stop clocks to freeze design before readback
  - – Signaling of secure switching
    - To avoid task switching between pipelined address and data phases

Insecure vs. Secure Switching



Source: [1]

Full Readback Preemption



Full Readback Preemption

- Design State Extraction
  - Need to know which bits in the readback stream represent status information
  - Filter all status information bits out of the readback stream and store them
  - XC4028: 18 ms
  - XCV400: 48.6 ms

Full Readback Preemption

- Task Reconstruction
  - Direct manipulation of configuration bitstream
    - No change in logic functionality and connections
    - All status information can be coded by single bits
  - In the original bitstream, initialization bits are changed according to the previously extracted register states
  - XC4028: 13 ms
  - XCV400: 7.5 ms

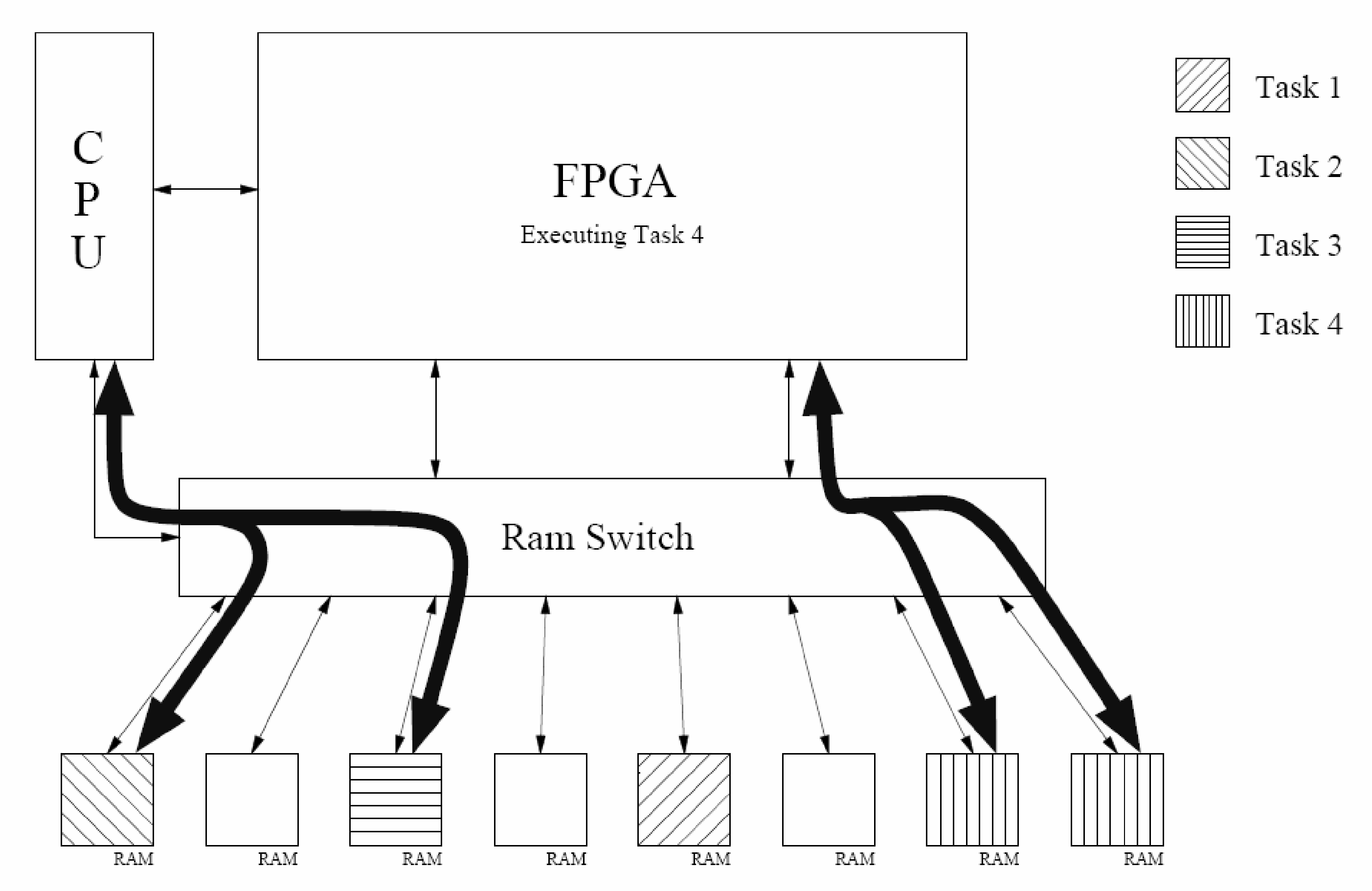
Full Readback Preemption

	XC4028	XCV400
Bitstream size	668 Kb	1.75 Mb
Readback	800 ms	14.4 ms
State extraction	18 ms	48.6 ms
Task reconstruction	13 ms	7.5 ms
Configuration	18 ms	12.4 ms
Relocation	849 ms	82.9 ms

Memory Bottleneck in Full Readback Preemption

- If several preemptions are requested at the same time
  - High bandwidth of external memory accesses is required
  - External memory is usually single-port, hence becomes a memory bottleneck
- Use a new RAM switch scheme [1]
  - A RAM is disconnected/reconnected when its owner task is swapped out/swapped in

RAM Switch Connection Scheme



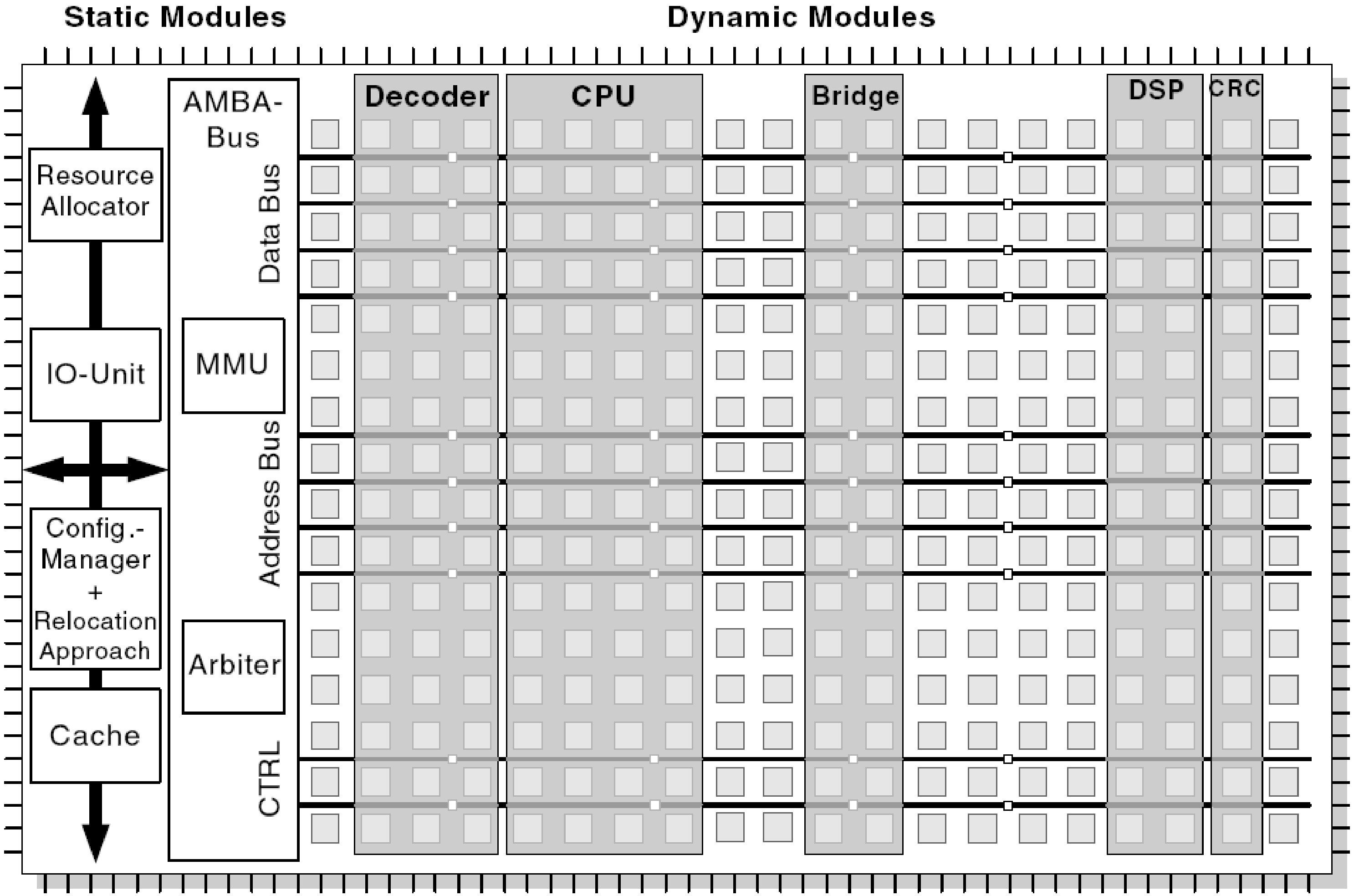
Partial Readback Preemption

- In a Xilinx XCV2000E FPGA
  - a CLB has 48 frames
  - Only 4 out of the 48 frames are contain flip-flops (state information)
- In a Xilinx XCV600E FPGA
  - Readback stream is 483 KB
  - 15,552 FF and 294,912 BRAM bits
  - Using all FF and BRAM, only 38 KB of state information, i.e. 38/483 < 8%
  - In a typical design, only 30% FF is used!

Partial Readback Preemption

- Significantly reduce amount of readback data by reading only those configuration frames containing state information
- Depends on smallest unit of configuration
  - Xilinx: a frame
- Filtering approach
  - State information extracted during readback and not after readback (saves memory)

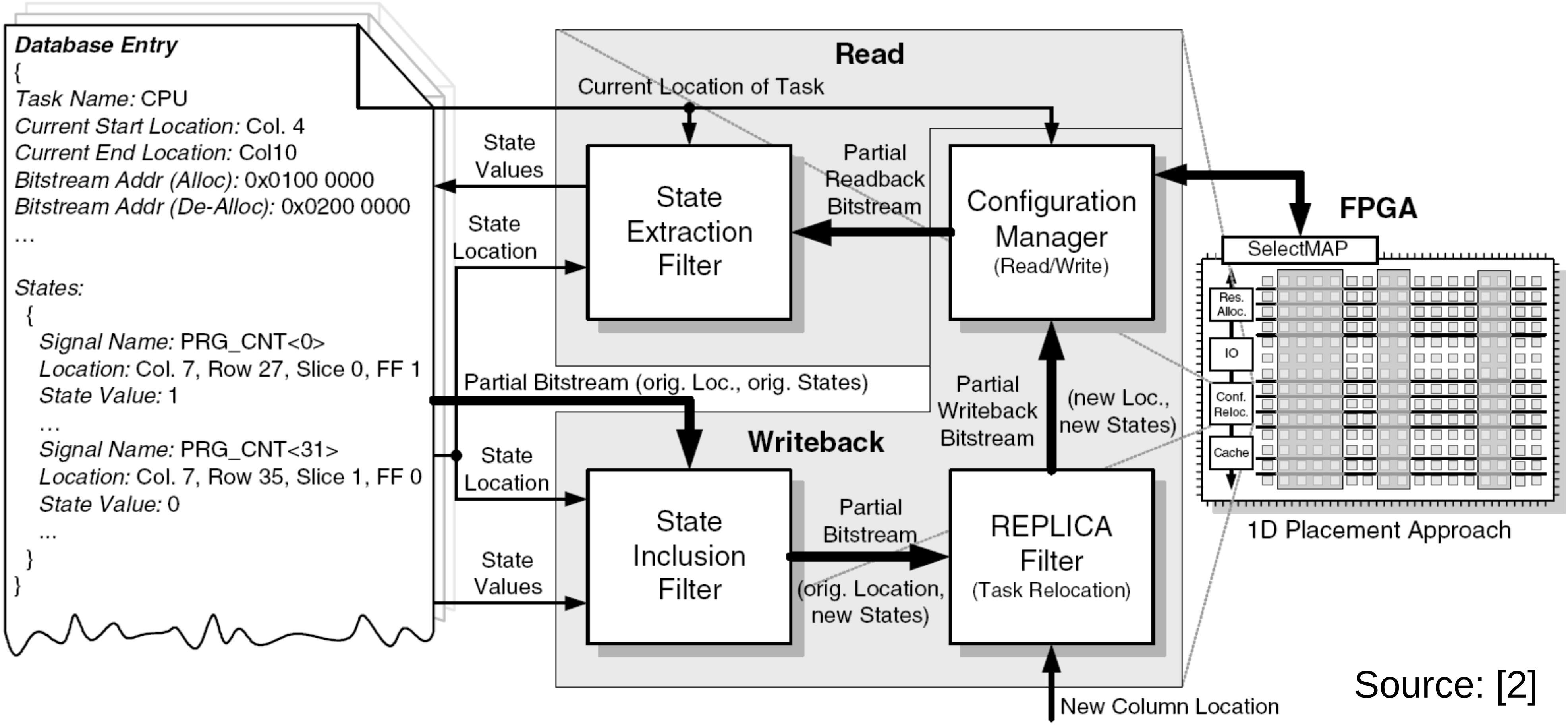
1-D Reconfigurable System



Context Relocation

- Partial readback preemptions
  - Context relocation = swap-out + swap-in
- Four main functions in context relocation
  - Configuration Manager (CM)
  - State Extraction Filter (SEF)
  - State Inclusion Filter (SIF)
  - REPLICA Filter

Context Relocation



Database Entry
{
Task Name: CPU
Current Start Location: Col. 4
Current End Location: Col10
Bitstream Addr (Alloc): 0x0100 0000
Bitstream Addr (De-Alloc): 0x0200 0000
…

States:
{
Signal Name: PRG_CNT<0>
Location: Col. 7, Row 27, Slice 0, FF 1
State Value: 1
…
Signal Name: PRG_CNT<31>
Location: Col. 7, Row 35, Slice 1, FF 0
State Value: 0
...
}
}

Read

Current Location of Task

State Values

State Location

State Extraction Filter

Partial Readback Bitstream

Configuration Manager (Read/Write)

FPGA

SelectMAP

Res. Alloc.

IO

Conf. Reloc.

Cache

1D Placement Approach

Partial Bitstream (orig. Loc., orig. States)

Partial Writeback Bitstream

(new Loc., new States)

Writeback

State Location

State Values

State Inclusion Filter

Partial Bitstream

(orig. Location, new States)

REPLICA Filter (Task Relocation)

New Column Location

Source: [2]

Context Relocation: Swap-out

- Resource allocator initiates context relocation
- Stop clock (implemented by clock gating)
- CM initiates SelectMAP or ICAP to read all frames with state information
- During readback, all frames passed to SEF, which determines the state values
  - Database updated, without storing all readback data

Context Relocation: Swap-in

- SIF integrates task register values from database with original partial bitstream
- Use REPLICA filter [4] to relocate hardware task from its original location to its new column location
- Relocated bitstream downloaded by CM

Relocation Database

- All necessary information about task
  - Current location in terms of CLB columns
  - Memory address of partial bitstreams
    - PARBIT [3] used to generate partial bitstreams from complete ones
  - Memory address of empty bitstreams
  - Location of all state registers
    - XCV200E: 19 bits per state register
      - 8 bit column + 8 bit row + 1 bit slice + 1 bit FF + 1 bit current state value
    - How many bits per state register for Virtex-5?

Configuration Manager

- Only frames with state information are read back by CM
- Generate frame address from
  - CLB column #, Slice #, FF #
- Frame address = (MJA, MNA)
  - Major/Minor Addresses for XCV200E
  - MJA = Chip_Cols – Col x 2 + 2
  - MNA = Slice x (12 x FF – 43) – 6 x FF + 45
    - Slice = 0, 1,   FF = 0, 1
  - MNA = 45, 39, 2, or 8

State Extraction Filter

- Reads frame data (readback stream) from CM
- Calculates bit index
  - bit_idx = (18 x row) + 1
- Extracts state values
- Updates database

State Inclusion Filter

- MJA and bit index are same as for SEF
- MNA differs
  - 41 if (Slice, FF) = (0, 0)
  - 35 if (Slice, FF) = (0, 1)
  - 6 if (Slice, FF) = (1, 0)
  - 12 if (Slice, FF) = (1, 1)

REPLICA Filter

- Relocating tasks from original location to new column location [4]
- Update of CRC values

Relocation Time

4 frames/task column

First frame is a pad frame for each new access

State Capture Time

Allocation + Deallocation Time

$$T_{Reloc} = 4N_{Cols}\frac{N_{Init} + 2N_{Byte/Frame}}{f_{SelectMAP}} + 2\frac{N_{BitstreamSize}}{f_{SelectMAP}}$$

$$\text{with } N_{Init} = 7 \cdot 4 \text{ Byte}$$

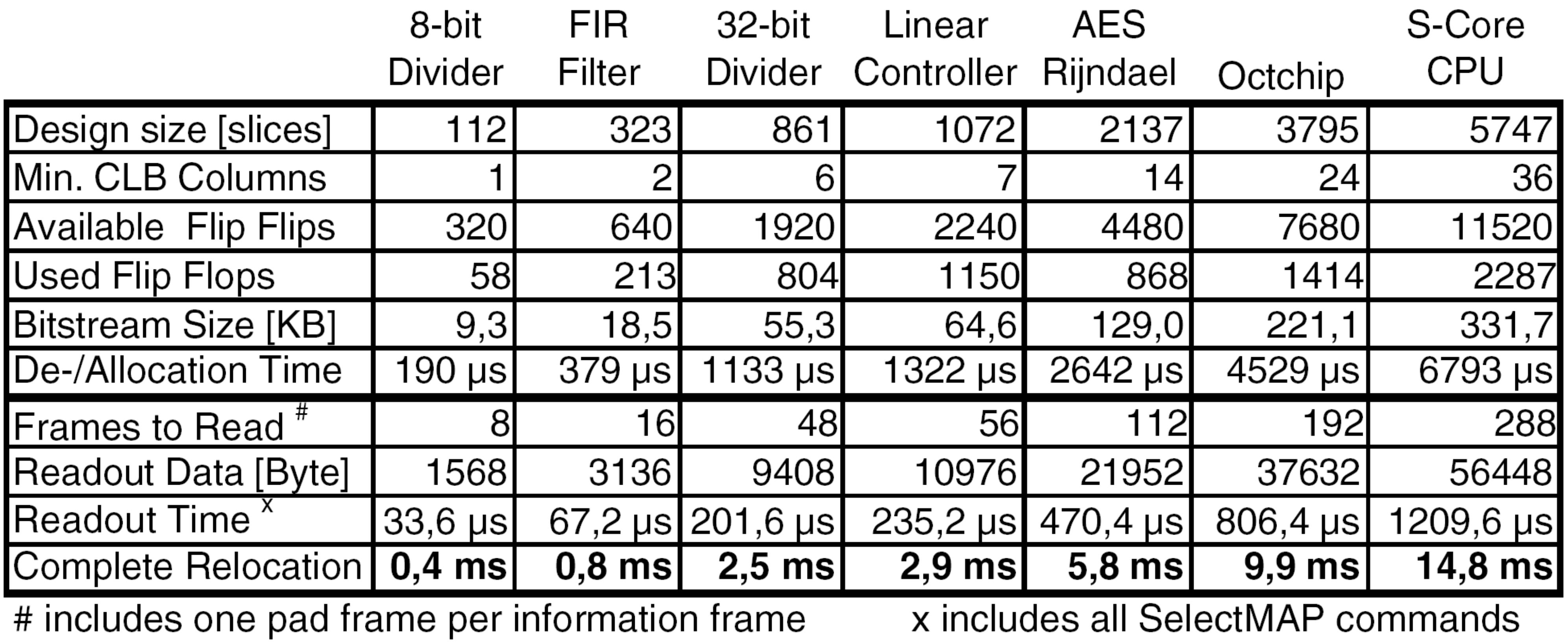
48 frames per CLB column

$$T_{Reloc} \approx 4N_{Cols}\frac{2N_{Byte/Frame}}{f_{SelectMAP}} + 2\frac{48N_{Cols}N_{Byte/Frame}}{f_{SelectMAP}}$$

$$= N_{Cols}\frac{104N_{Byte/Frame}}{f_{SelectMAP}}$$

SelectMAP frequency

Relocation Times



	8-bit Divider	FIR Filter	32-bit Divider	Linear Controller	AES Rijndael	Octchip	S-Core CPU
Design size [slices]	112	323	861	1072	2137	3795	5747
Min. CLB Columns	1	2	6	7	14	24	36
Available Flip Flips	320	640	1920	2240	4480	7680	11520
Used Flip Flops	58	213	804	1150	868	1414	2287
Bitstream Size [KB]	9,3	18,5	55,3	64,6	129,0	221,1	331,7
De-/Allocation Time	190 µs	379 µs	1133 µs	1322 µs	2642 µs	4529 µs	6793 µs
Frames to Read #	8	16	48	56	112	192	288
Readout Data [Byte]	1568	3136	9408	10976	21952	37632	56448
Readout Time x	33,6 µs	67,2 µs	201,6 µs	235,2 µs	470,4 µs	806,4 µs	1209,6 µs
Complete Relocation	**0,4 ms**	**0,8 ms**	**2,5 ms**	**2,9 ms**	**5,8 ms**	**9,9 ms**	**14,8 ms**

includes one pad frame per information frame x includes all SelectMAP commands

Source: [2]

Design based Preemption

- Hardware design enhanced with preemption capability
  - – Interruptible state identification and selection
  - – Stopping execution at an interruptible state
  - – Read-write access to state and context registers
  - – Swap control interface

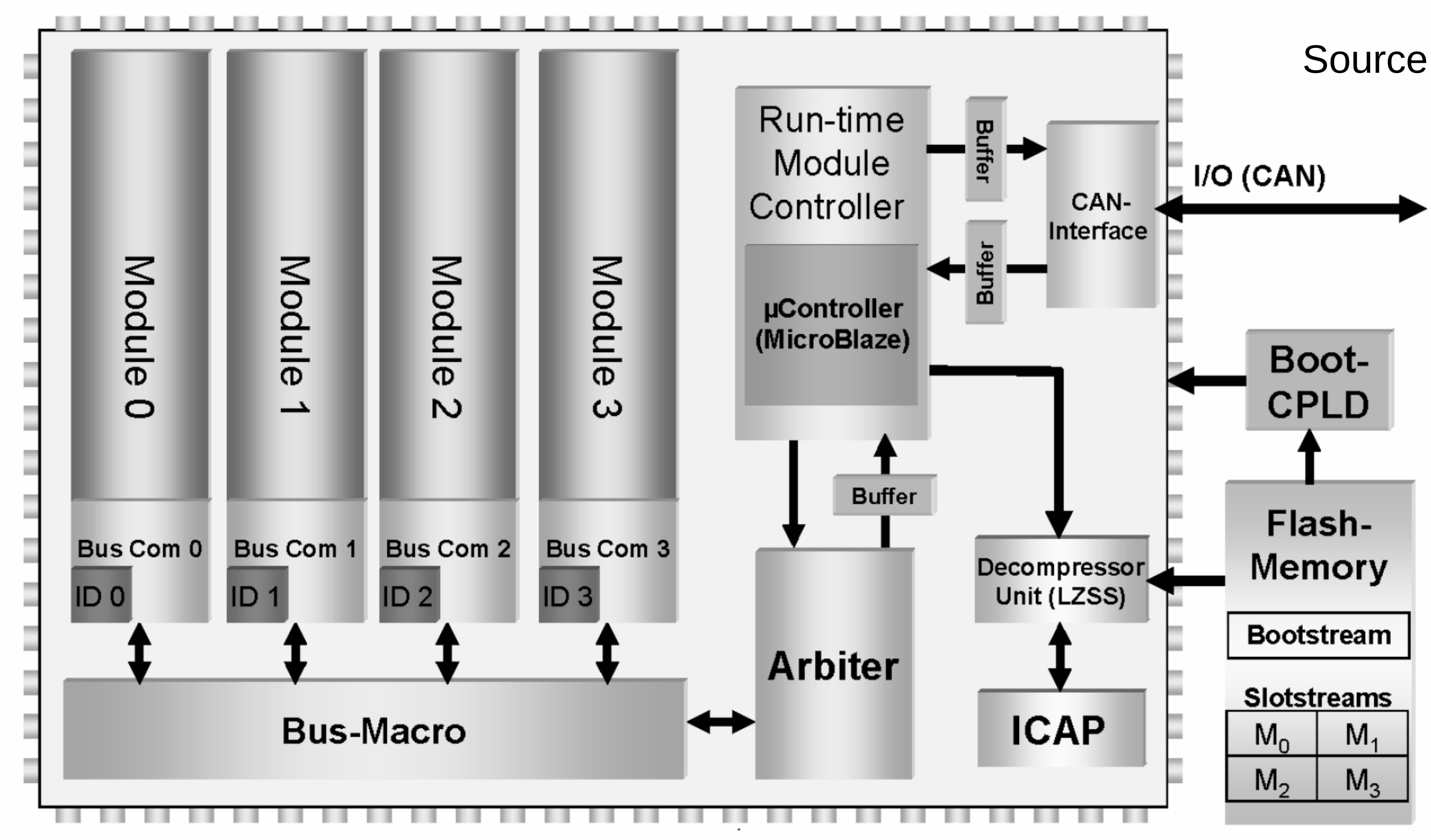
Design based Preemption

- Three types of preemption architecture:
  - Software Run-Time Controller [5]
  - Hardware Task Interface and Wrapper [6]
  - Operating System for Reconfigurable Systems (OS4RS) [7]
- Two types of task relocations:
  - Hardware-hardware [5, 6]
  - Hardware-software [7]

System Architecture for Software Run-Time Controller

- Slot-based modules
- Bus-macros connect all slots
- Arbiter controls communications on bus macros
- Run-time controller runs on Xilinx MicroBlaze and allocates/deallocates module slots
- Reconfiguration performed using ICAP

System Architecture for Software Run-Time Controller

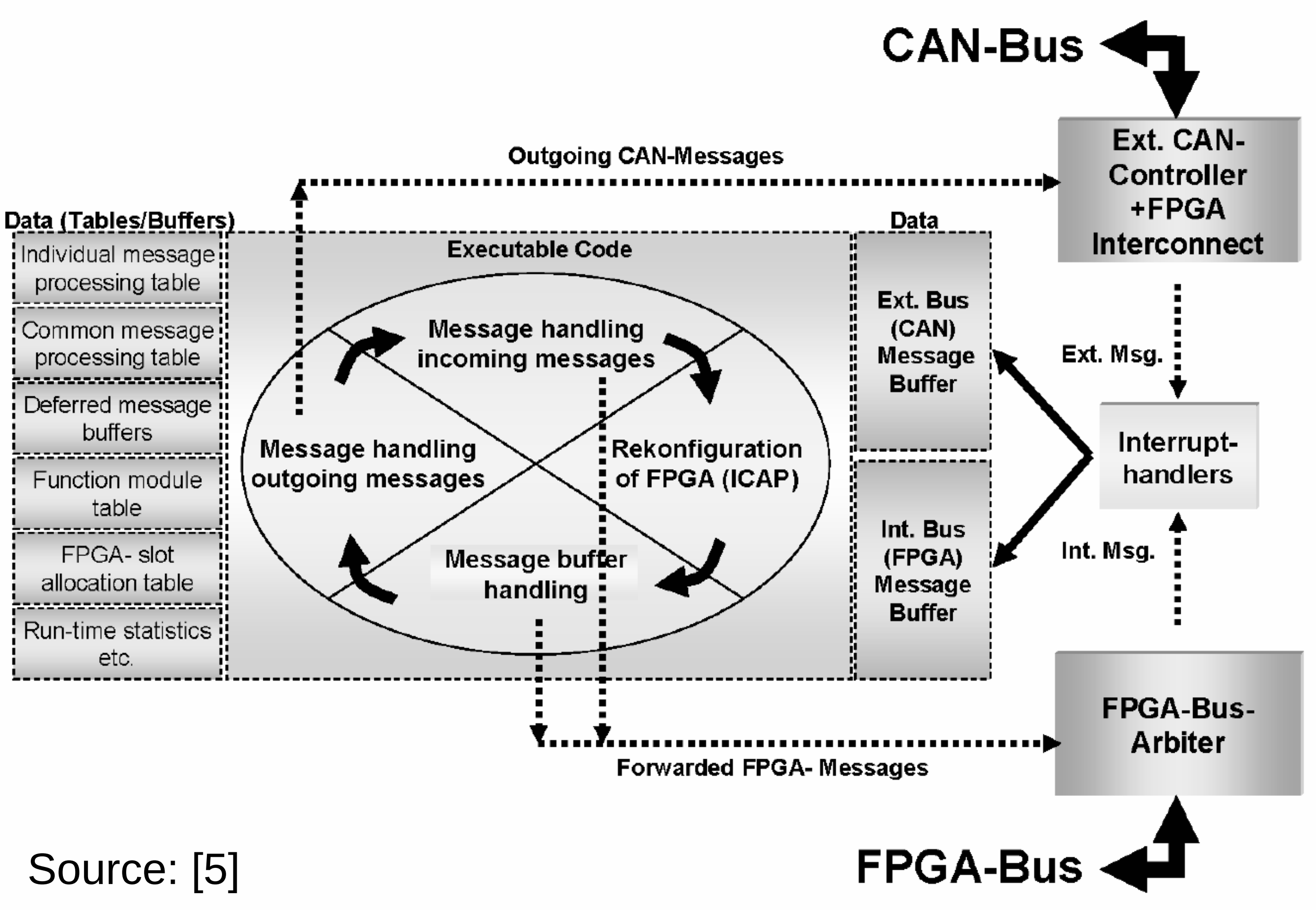


Source: [5]

Software Run-Time Controller

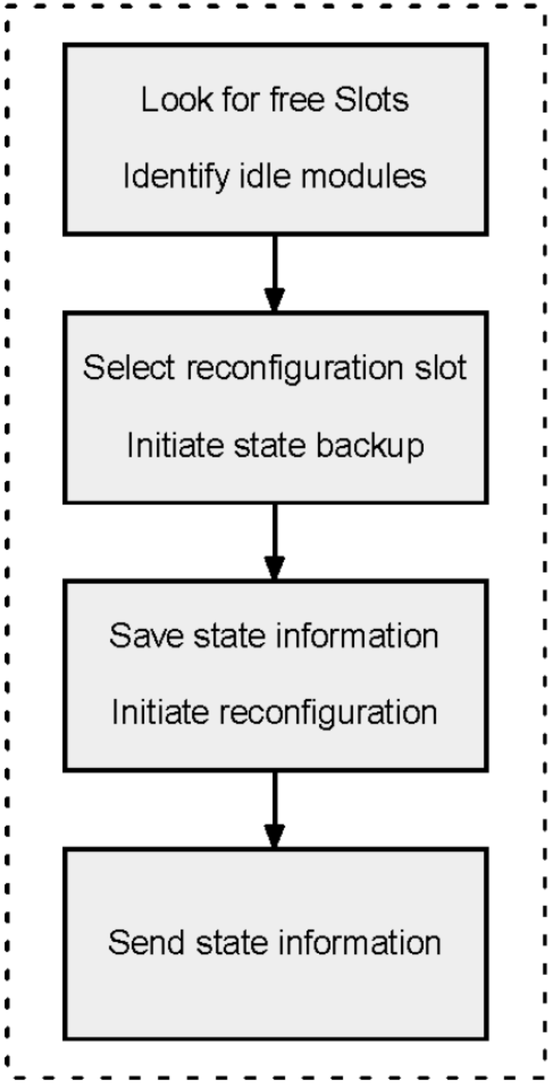
- Four major modules in controller
  - – Management of incoming messages
  - – Reconfiguration process
  - – Message buffer management unit
  - – Management of outgoing messages

Software Run-Time Controller


CAN-Bus
Ext. CAN-Controller +FPGA Interconnect
Outgoing CAN-Messages
Data (Tables/Buffers)
Individual message processing table
Common message processing table
Deferred message buffers
Function module table
FPGA- slot allocation table
Run-time statistics etc.
Executable Code
Message handling incoming messages
Message handling outgoing messages
Rekonfiguration of FPGA (ICAP)
Message buffer handling
Data
Ext. Bus (CAN) Message Buffer
Int. Bus (FPGA) Message Buffer
Ext. Msg.
Interrupt-handlers
Int. Msg.
FPGA-Bus-Arbiter
Forwarded FPGA- Messages
FPGA-Bus

Source: [5]

Reconfiguration Process in Software Run-Time Controller



Look for free Slots

Identify idle modules

Select reconfiguration slot

Initiate state backup

Save state information

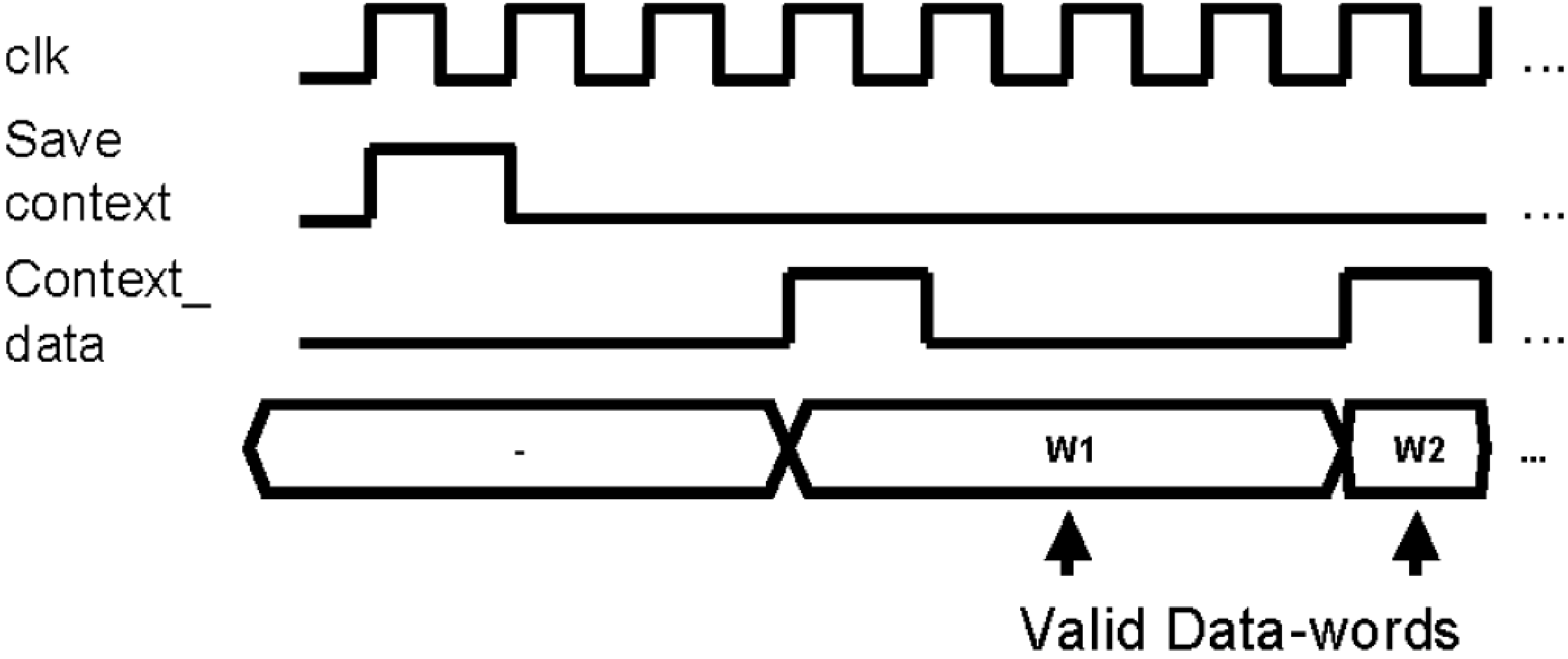
Initiate reconfiguration

Send state information

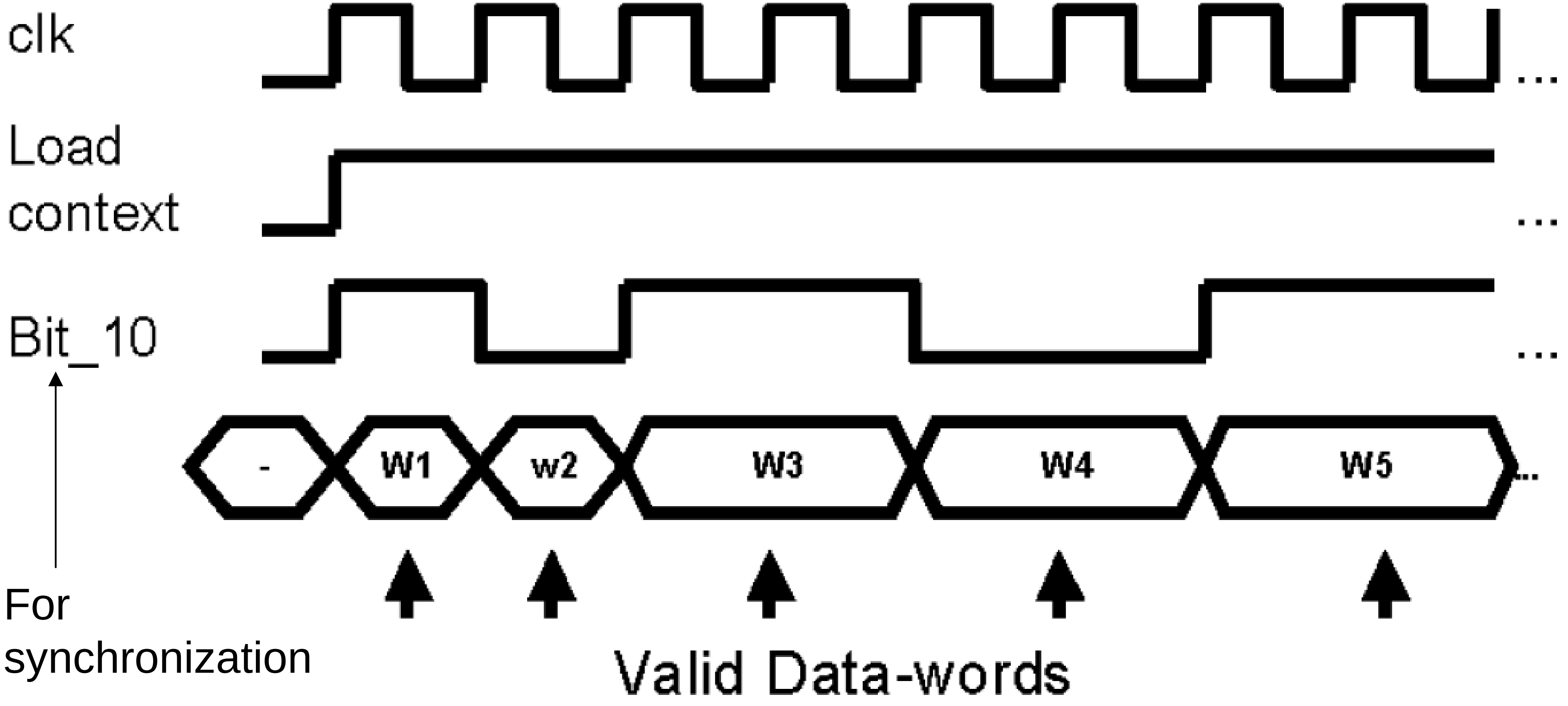
Swapping Process

- Module prompted to send state information to run-time system
- Context save-restore mechanism is part of the function’s model description
  - A parallel running state machine responsible for re-initializing and saving state information

Module Context Data Save



Module Context Data Restore

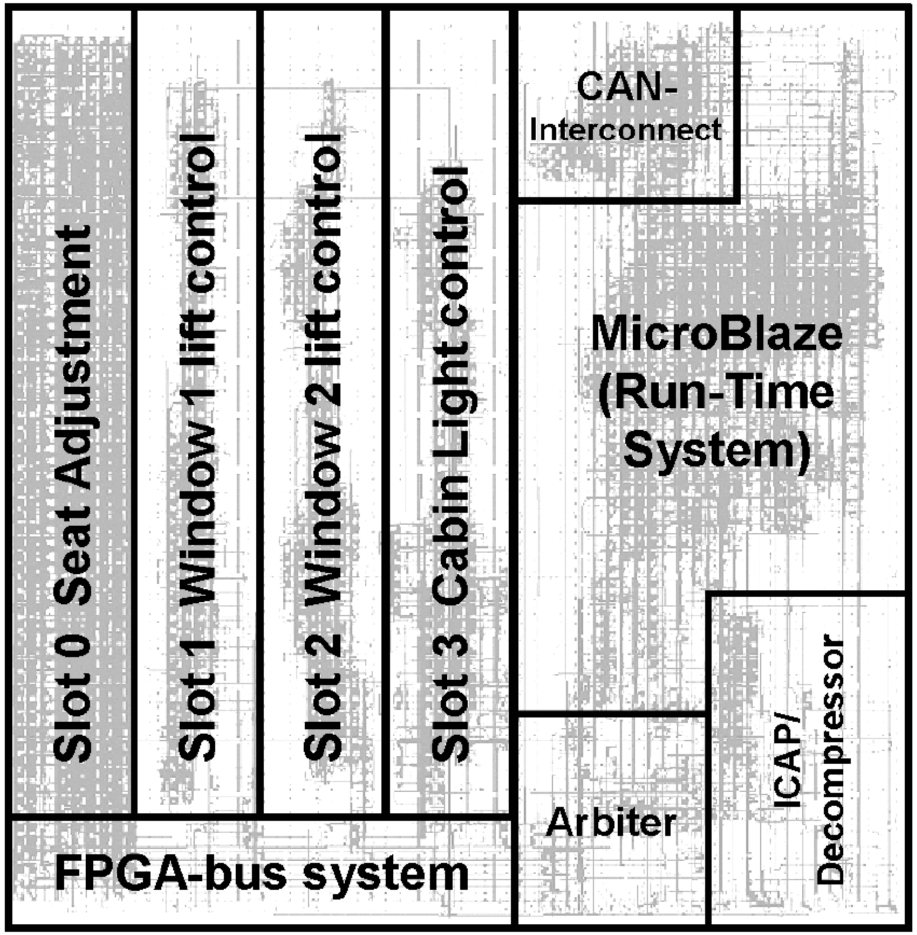


Example Implementation in Automobiles

- CAN-bus connected ECU functions
  - CAN: Controller Area Network
  - ECU: Engine Control Unit
    - Central Locking
    - Cabin Compartment Lighting
    - Seat Adjustment
    - Power Window
    - Rear-view Mirror
    - Sunroof

Example Implementation

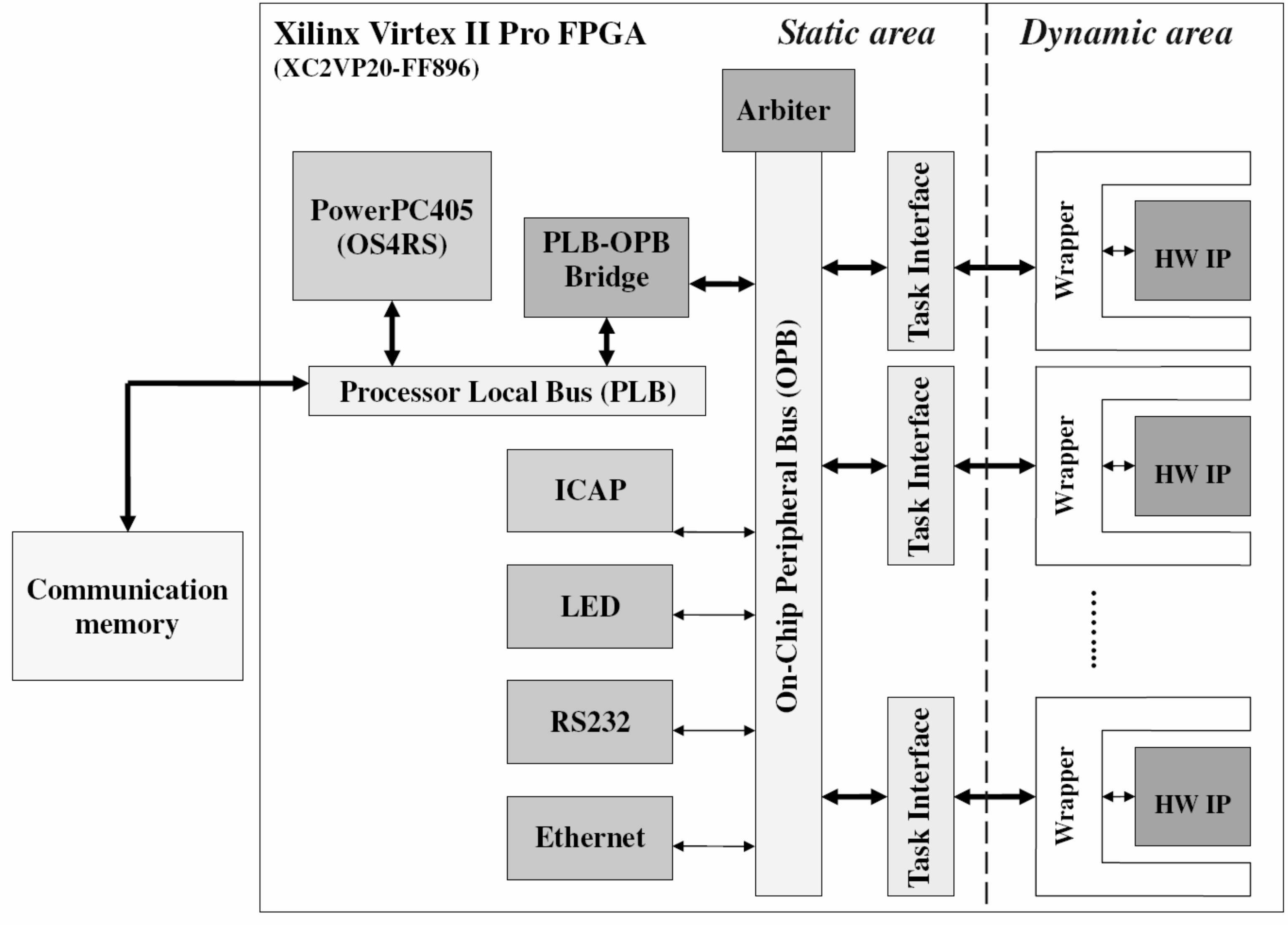
- Xilinx XC2V3000 FPGA
  - 64x56 = 3584 CLBs
- Xilinx MicroBlaze softcore
  - 32 bit RISC, 125 MHz
  - 950 CLB (26.5%)
- LZSS decompression unit
  - 134 CLB (3.7%)
- Arbiter
  - 85 CLB (2.4%) 180 MHz (delay < 5.5 ns)
- Bi-directional bus macros
  - 120 CLB (3.4%) 180 MHz (delay < 5.5 ns)


Slot 0 Seat Adjustment
Slot 1 Window 1 lift control
Slot 2 Window 2 lift control
Slot 3 Cabin Light control
CAN-Interconnect
MicroBlaze (Run-Time System)
Arbiter
ICAP/ Decompressor
FPGA-bus system

Example Implementation in XC2V3000

- Average response time < 1ms
- Average reconfiguration time per slot = 15 ms
- Demanded response time of 100 ms
- Hence, feasible!!!
- 8 control functions implemented
- Each partial bitstream is 118 KB
- Bitstream compression rate is 60%
- No dynamic relocation employed!
  - Four partial bitstreams generated, one for each slot

System Architecture for Hardware Task Interface and Wrapper



Hardware Task Interface and Wrapper

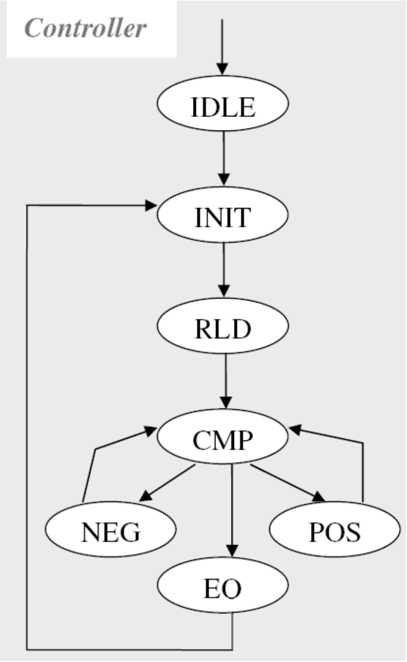
- How to make a HW IP reconfigurable?
  - Must be enhanced with swap capabilities
    - Stop execution at some interruptible state in FSM
    - Access to state and context registers
  - Must be interfaced with
    - Task interface
      - For interfacing with a standard bus such as OPB
    - Wrapper
      - For swap control, task shutdown, context data save/restore, data (un)packing, …

Interruptible State

- Not every state of a HW FSM is interruptible because
  - Communication undergoing
  - Pipeline not flushed
  - Register values do not represent complete state
  - Not possible to resume from a state

Interruptible State

- A state is said to be interruptible if the hardware task can resume execution from that state after restoring the task context, either partially or fully.
- GCD Example
  - Interruptible states:
    - INIT, RLD, CMP
  - Non-interruptible states:
    - NEG, EQ, POS
    - Comparator results not saved, hence cannot resume



Access to Registers

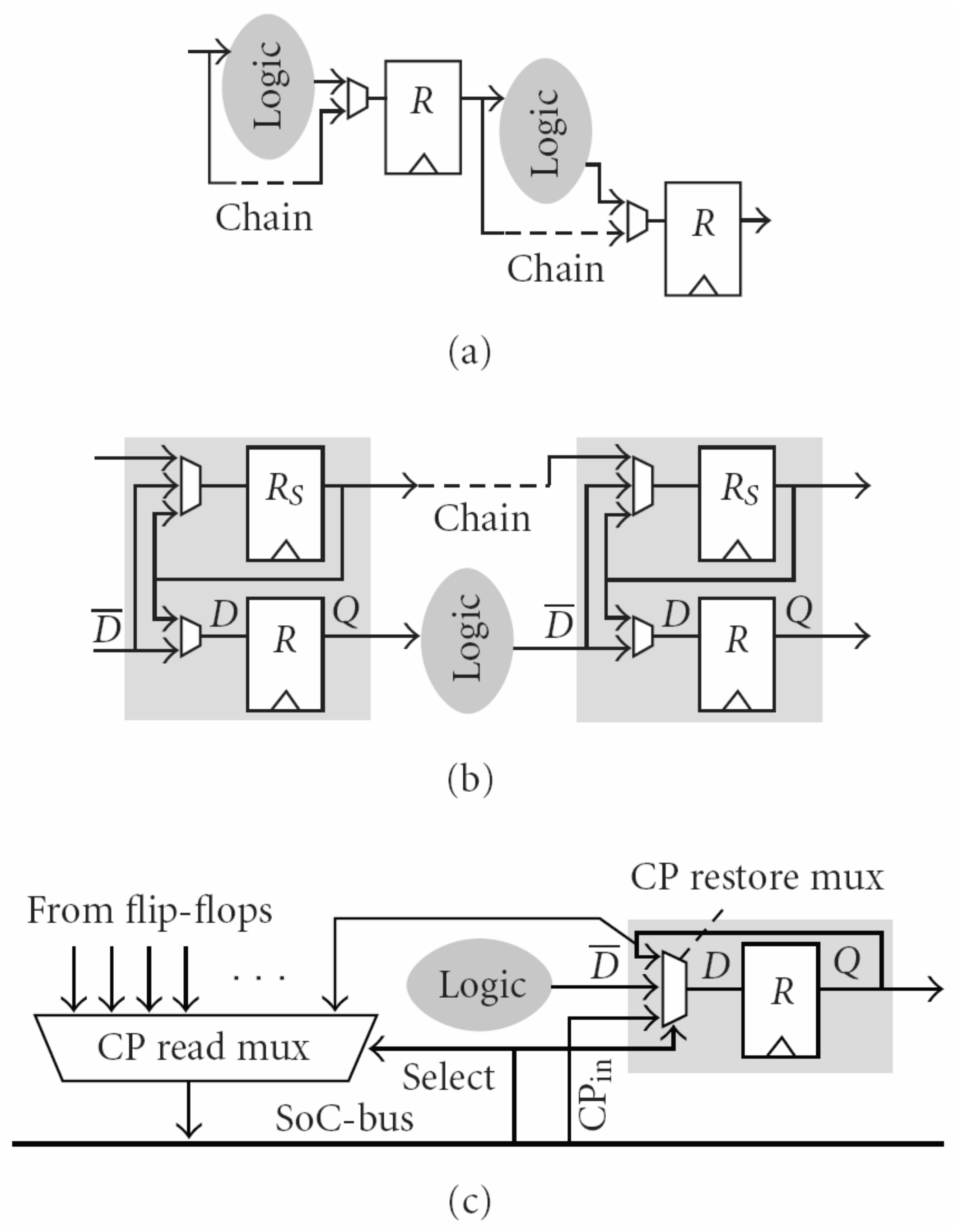
- Scan Chain
  - Scan multiplexer in front of each FF
    - Regular execution mode vs. Scan mode
- Shadow Chain
  - Each FF is duplicated and connected to chain
  - Store/restore/swap within a single cycle
- Memory Mapped
  - CPU can access directly using address and data bus

Access to Registers

(a) Scan Chain

(b) Shadow Chain

(c) Memory Mapped



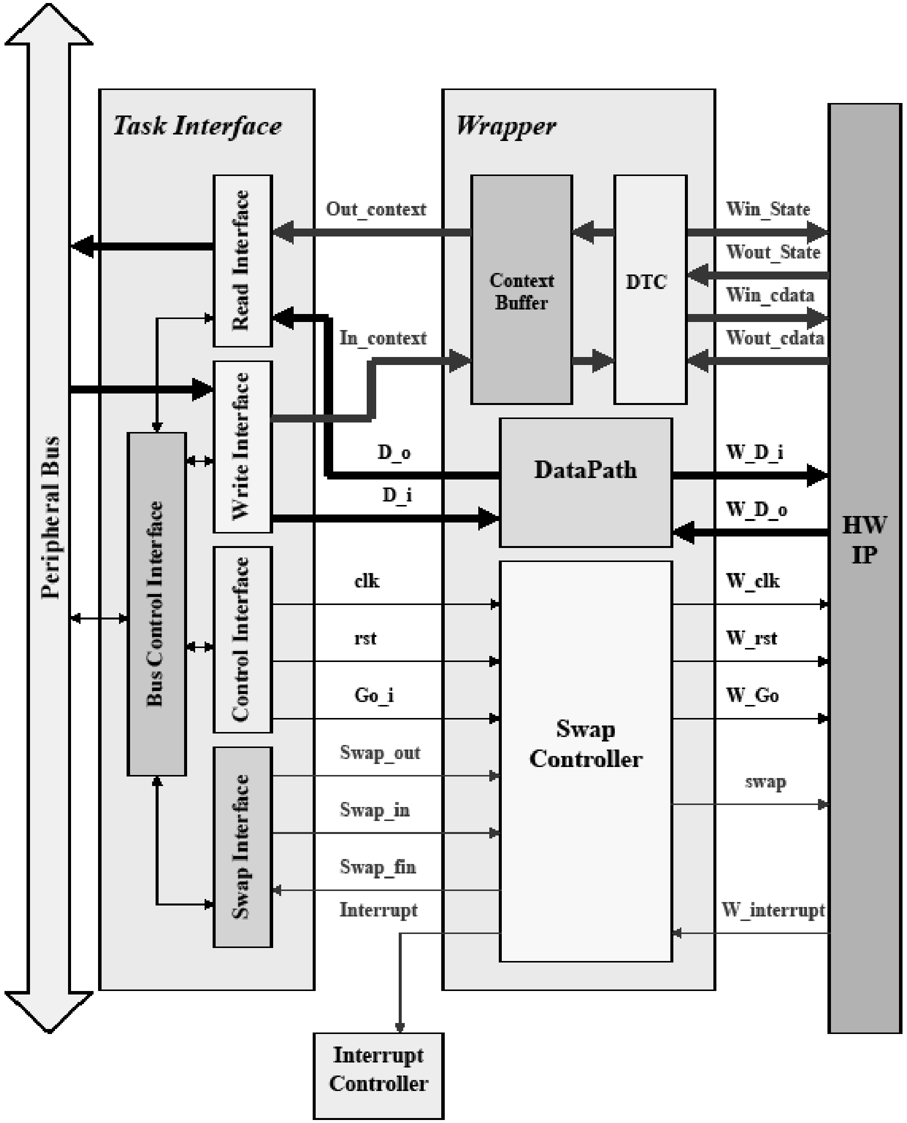
(a)

(b)

(c)

Source: [8]

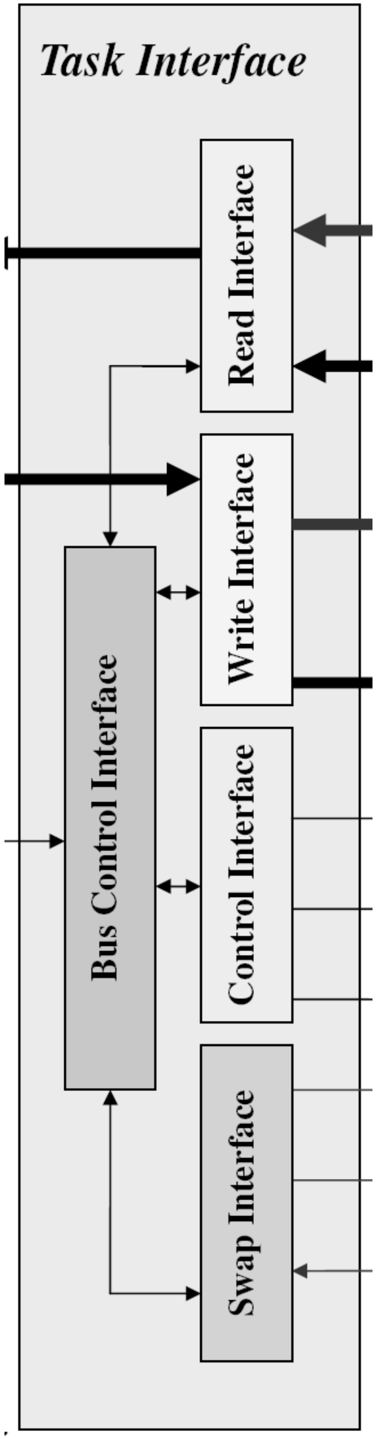
Hardware Task Interface and Wrapper



Source: [6]

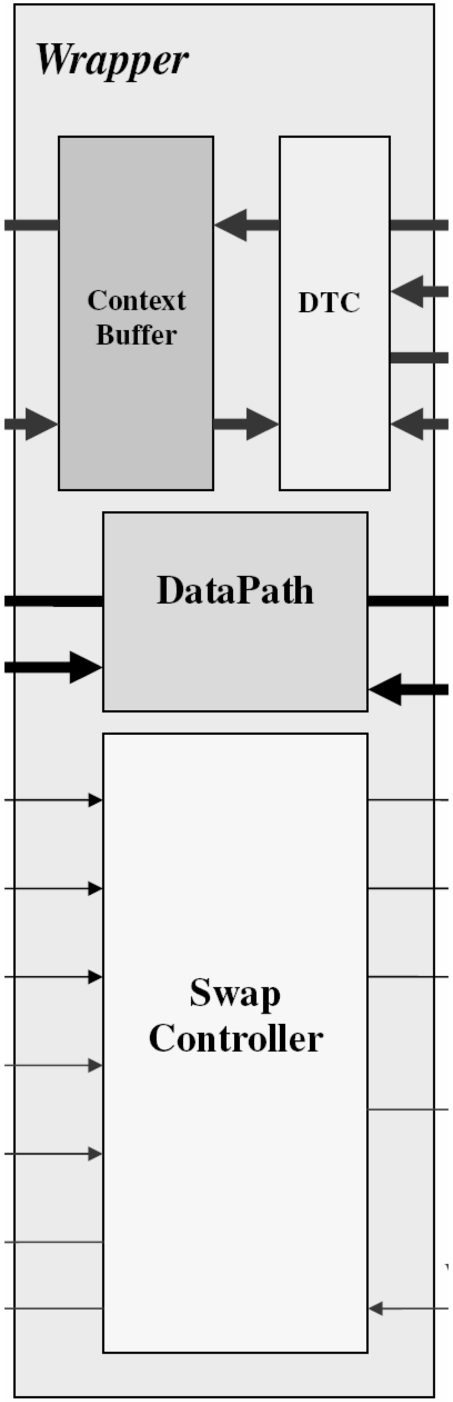
Task Interface

- Read/Write Interface
  - – Normal bus read/write transactions
- Control Interface
  - – IP control interface:
    - reset, clock, done, go
- Swap Interface
  - – Swap out/in requests, interrupts

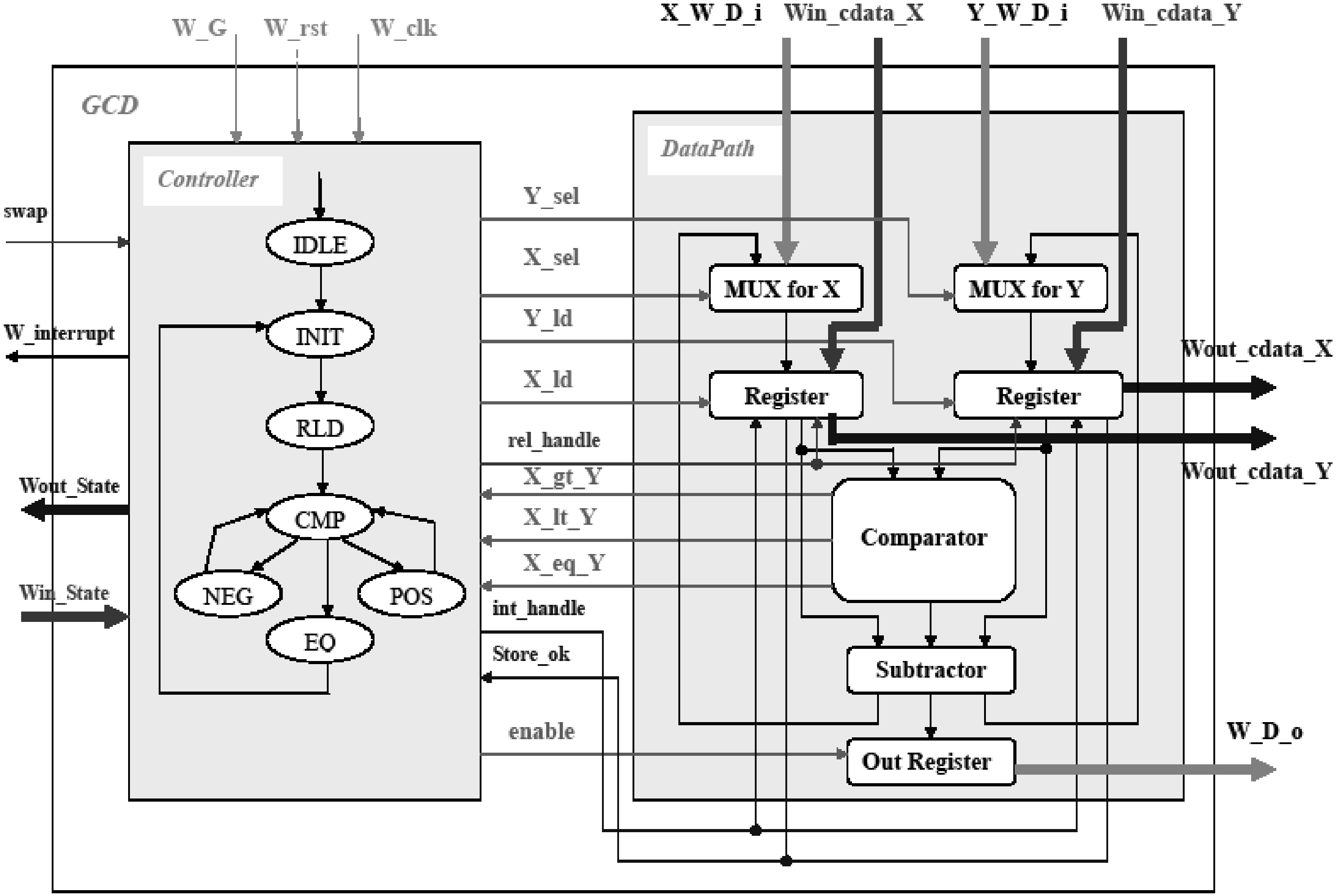


Generic Wrapper

- Context Buffer
  - – For storing data context of HW IP
- Data Transformation Component (DTC)
  - – For packing/unpacking of data context into 32 bits (bus width)
- Datapath
  - – For data transfer
- Swap Controller
  - – For controlling swap out/in and reconfiguration



GCD Example Implementation



Advantages

- Better real-time response
  - 10x ~ 100x ms time save
- Standardization of hardware IP
  - Use scan chain, shadow registers, memory mapping for register access
- Generic wrapper design
  - Uniform design, little resource and time overheads

Swap Time

- D_C: context data (bits),
- D_B: context buffer (bits),
- R_T: data transformation rate (bits/cycle),
- R_B: buffer data load rate (bits/cycle),
- R_P: peripheral bus data transfer rate (bits/cycle),
- T_A: peripheral bus access time (cycles),
- T_I: transition time to go to an interruptible state (cycles) , and
- T_R: reconfiguration time (cycles),

Swap Time

- T_{SO}: Swap-out time
- T_{SI}: Swap-in time

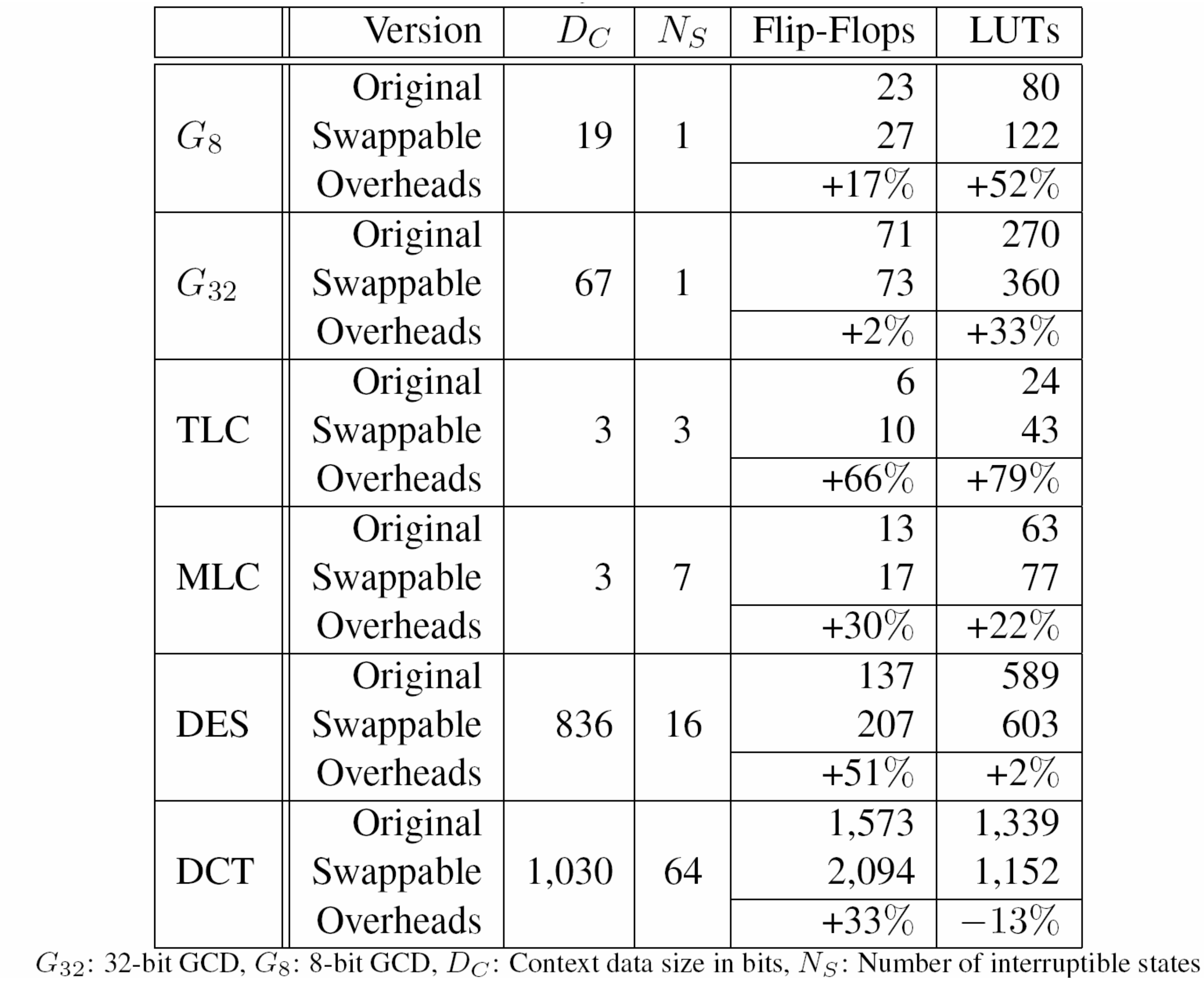
$$T_{SO} = T_I + \left\lceil \frac{D_C}{D_B} \right\rceil \times \left(\frac{D_B}{R_T} + \frac{D_B}{R_B} + T_A + \frac{D_B}{R_P} \right) + T_R$$

$$T_{SI} = T_R + \left\lceil \frac{D_C}{D_B} \right\rceil \times \left(\frac{D_B}{R_T} + \frac{D_B}{R_B} + T_A + \frac{D_B}{R_P} \right)$$

Experiments

- Xilinx Virtex II Pro XC2VP20-FF896 FPGA
- 56 x 46 CLB matrix
- 18,560 LUTs, 18,560 FFs
- 32-bit CoreConnect OPB at 133 MHz

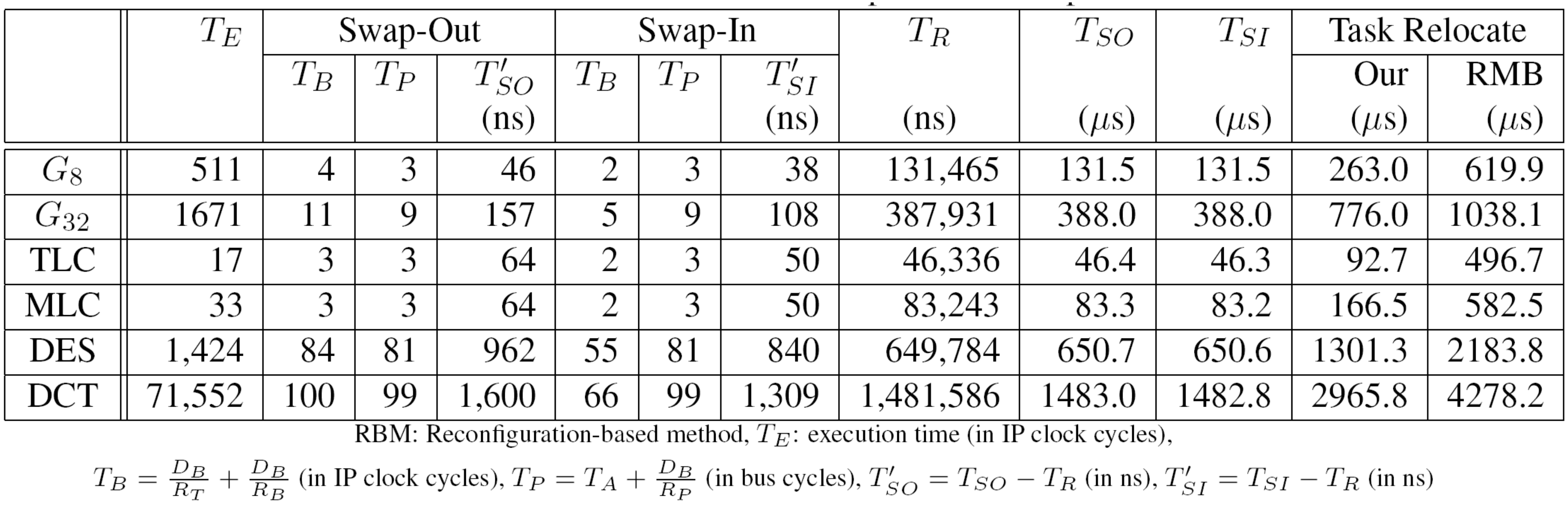
Area Overheads



	Version	D_C	N_S	Flip-Flops	LUTs
G_8	Original	19	1	23	80
	Swappable			27	122
	Overheads			+17%	+52%
G_{32}	Original	67	1	71	270
	Swappable			73	360
	Overheads			+2%	+33%
TLC	Original	3	3	6	24
	Swappable			10	43
	Overheads			+66%	+79%
MLC	Original	3	7	13	63
	Swappable			17	77
	Overheads			+30%	+22%
DES	Original	836	16	137	589
	Swappable			207	603
	Overheads			+51%	+2%
DCT	Original	1,030	64	1,573	1,339
	Swappable			2,094	1,152
	Overheads			+33%	−13%

G_{32}: 32-bit GCD, G_8: 8-bit GCD, D_C: Context data size in bits, N_S: Number of interruptible states

Time Overheads



	T_E	Swap-Out			Swap-In			T_R	T_{SO}	T_{SI}	Task Relocate	
		T_B	T_P	T'_{SO}	T_B	T_P	T'_{SI}				Our	RMB
				(ns)			(ns)	(ns)	(μs)	(μs)	(μs)	(μs)
G_8	511	4	3	46	2	3	38	131,465	131.5	131.5	263.0	619.9
G_{32}	1671	11	9	157	5	9	108	387,931	388.0	388.0	776.0	1038.1
TLC	17	3	3	64	2	3	50	46,336	46.4	46.3	92.7	496.7
MLC	33	3	3	64	2	3	50	83,243	83.3	83.2	166.5	582.5
DES	1,424	84	81	962	55	81	840	649,784	650.7	650.6	1301.3	2183.8
DCT	71,552	100	99	1,600	66	99	1,309	1,481,586	1483.0	1482.8	2965.8	4278.2

RBM: Reconfiguration-based method, T_E: execution time (in IP clock cycles),

$T_B = \frac{D_B}{R_T} + \frac{D_B}{R_B}$ (in IP clock cycles), $T_P = T_A + \frac{D_B}{R_P}$ (in bus cycles), $T'_{SO} = T_{SO} - T_R$ (in ns), $T'_{SI} = T_{SI} - T_R$ (in ns)

Operating System for Reconfigurable Systems

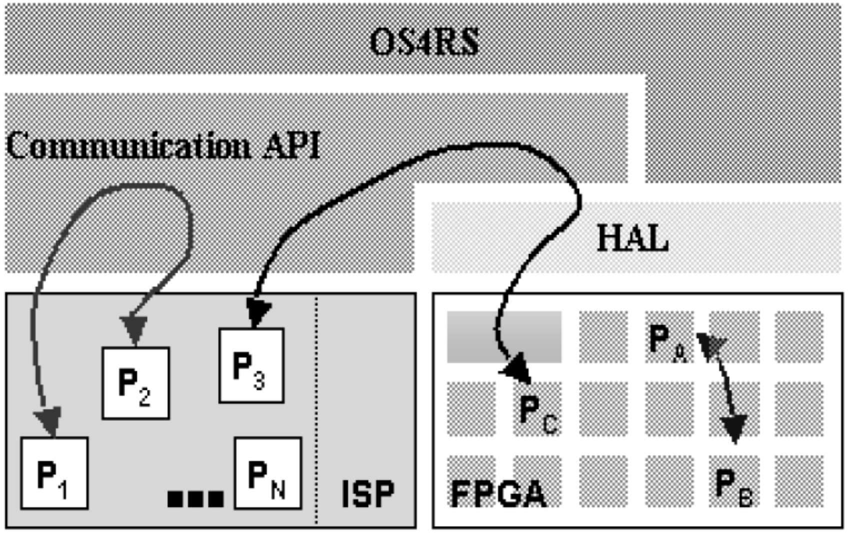
- OS4RS
  - Implements a hardware abstraction layer (HAL) over the FPGA
  - Schedules tasks on processor and FPGA
  - Provides uniform communication for software and hardware tasks to send/receive messages

OS4RS: Uniform Communication

- Message passing based
  - – Messages have common format for both hardware and software tasks
- Each task has a logical address and a physical address when configured
  - – OS4RS translates logical/physical addresses

OS4RS: Uniform Communication

- Message passing API
  - Both software tasks
    - Routed using logical addresses
  - A software and a hardware task
    - Logical/physical address translation, uses HAL
  - Both hardware tasks
    - Packet-switched interconnection



OS4RS: Placement

- Tile-based placement
  - – Partial bitstreams are configured directly into tiles by OS4RS
  - – High fragmentation
  - – Low latency (tile availability check only)
- Packet-switched interconnection
  - – Controlled by OS4RS

OS4RS: Hardware Relocation

- Two approaches
  - A partial bitstream for every tile [7]
  - Use Jbits [9] to manipulate a single bitstream at run-time

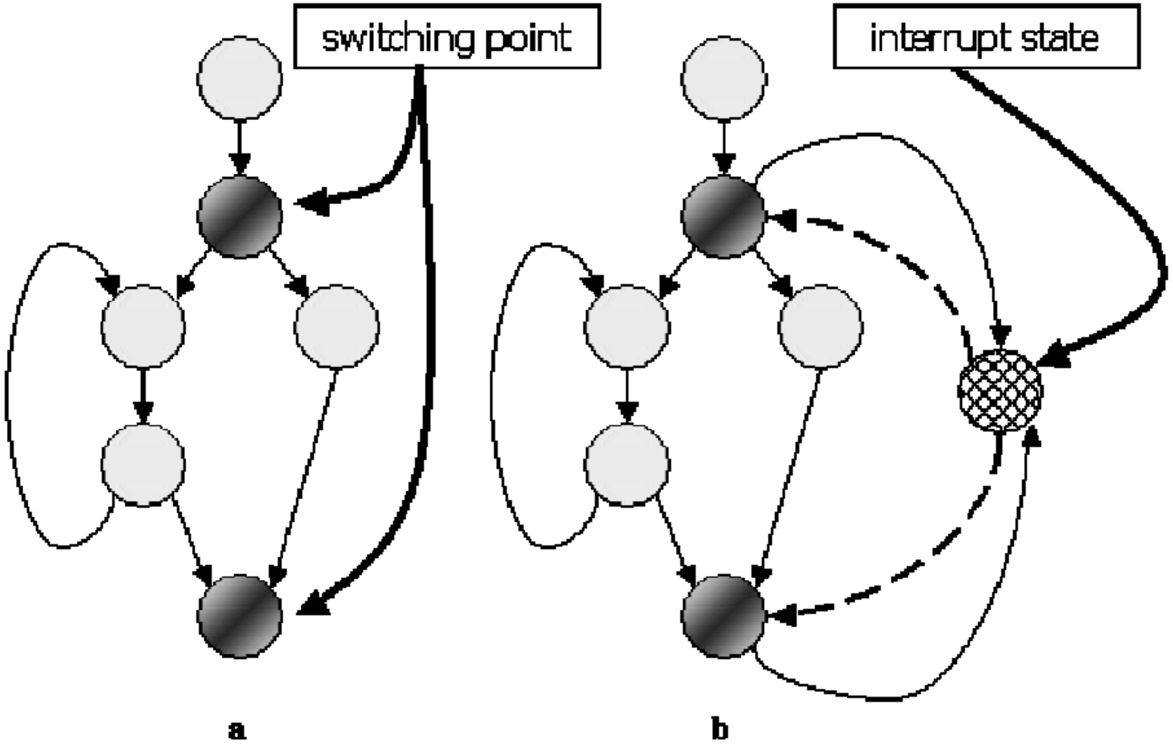
OS4RS: Routing

- Fixed communication infrastructure inside the interconnection network
- Routing is performed by OS4RS through routing tables
  - No complex algorithms, only update of tables when a task is inserted/removed

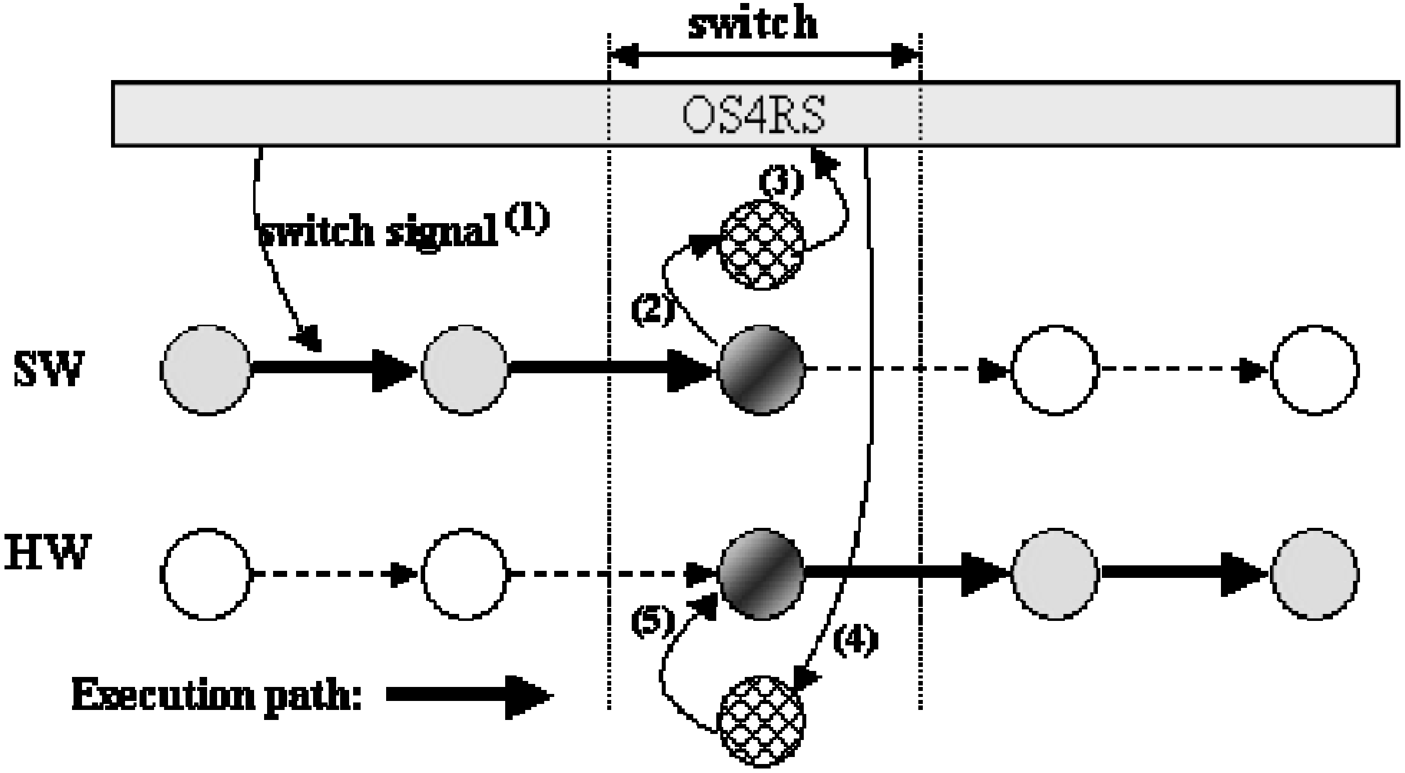
OS4RS: HW-SW Relocation

- Software task
  - – Context defined by processor registers and task memory
- Hardware task
  - – No universal representation of hardware states
  - – Two ways to extract states
    - Readback data contains state information
    - High-level interruptible states

Switching Point and Interrupt State



Task Switching from SW to HW

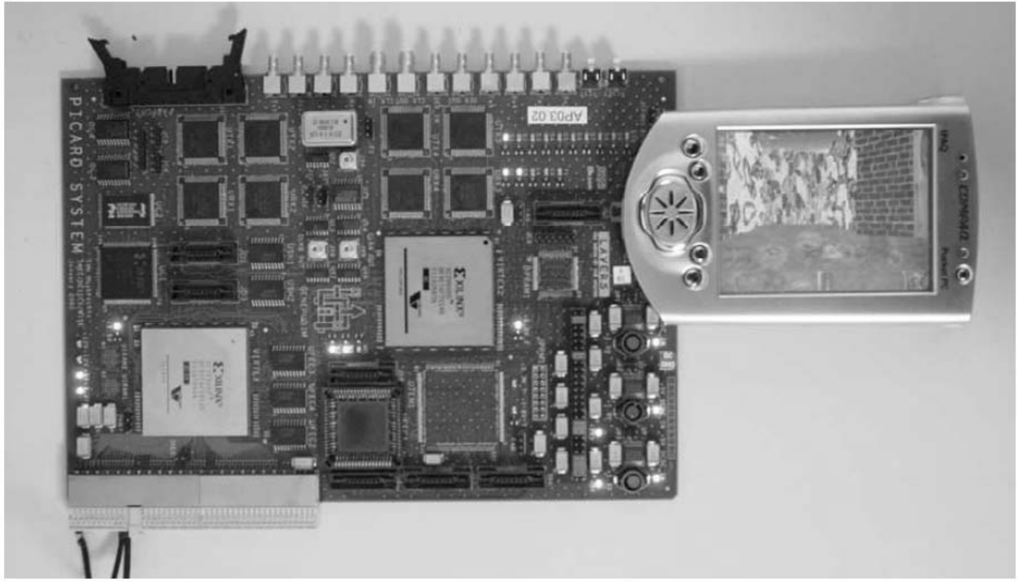


Switching Point

- Switching Point
  - “Low overhead” interruptible states
    - Contains “no state information”
    - Architecture dependent
    - Data transfer
      - Shared Memory: pass a pointer
      - Distributed Memory: copy data
  - Example
    - End of a frame computation

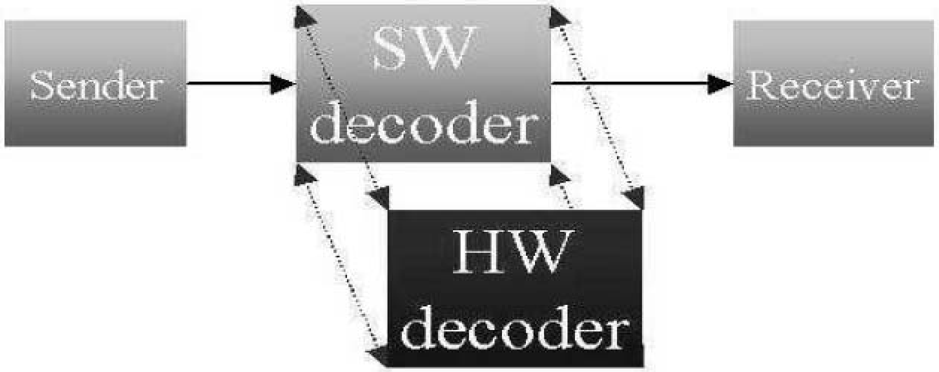
Example Implementation

- Relocatable Video Decoder [7]
  - T-ReCs Gecko demonstrator
    - Compaq iPAQ 3760 (StrongARM SA-1110)
    - Xilinx Virtex 2 XC2V6000 FPGA



Relocatable Video Decoder

- Motion JPEG frame decoder can be scheduled to run in HW or in SW
- Send and receive threads run on iPAQ
- Switch point: end of frame (no state information)



Relocatable Video Decoder

- Results with Synplify Pro
  - HW decoders
    - 23 frames per second (fps)
    - Clock: 40 MHz
    - Specific: 9570 LUTs
    - General: 15901 LUTs
  - SW decoder
    - 6 frames per second (fps)
    - CPU load = 95%
  - Communication
    - BlockRAM: 20 MHz
    - CPU memory access: 103 MHz
      - synchronous RAM, need to insert wait states

Relocatable Video Decoder

- Decoder relocated from SW to HW
  - 6 fps → 23 fps
  - CPU 95% → 95% (Why?)
    - Send and receive threads heavily load processor
    - Memory accesses (103 MHz → 20 MHz) need wait-states (CPU is thus idle in these states)
  - OS4RS overhead is 100 microseconds
  - Total latency is 108 ms
    - Mainly partial reconfiguration
      - Theoretical latency 11 ms, why such a large difference?
        - » Slow CPU-FPGA interface!!!

Hardware Relocation

- Hardware designs are fixed in location in an FPGA after it is implemented and bitstream generated.
- How to locate the hardware in a different location from the one where it was implemented?
- Hardware relocation architectures and tools provide solutions to do this!
  - Swappable Logic Unit [10]
  - Dynamic Hardware Plugins [11]
  - REPLICA (Relocation per online Configuration Alteration) [4]
  - PARBIT [3]

Swappable Logic Unit

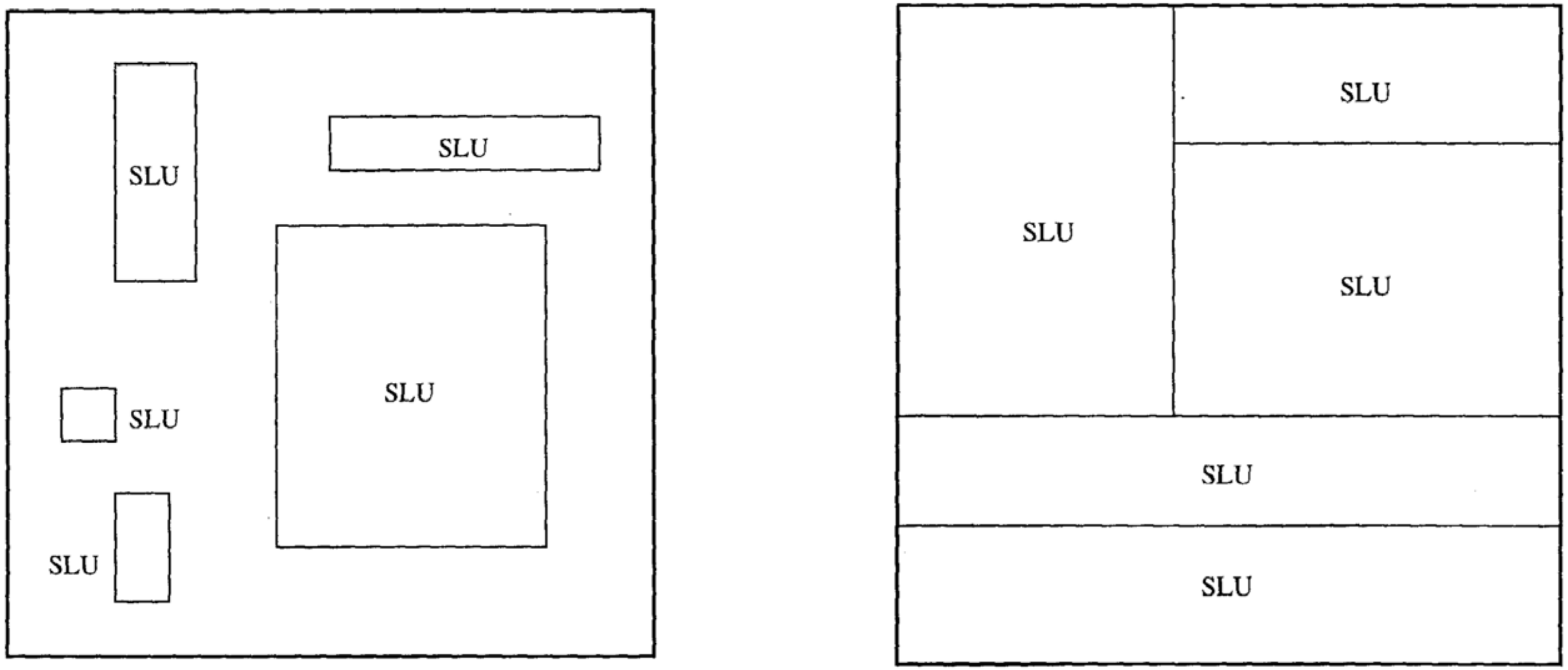
- Virtual hardware analogue of page or segment in virtual memory
- Hardware Features
  - Fixed area in FPGA
  - Fixed I/O interfaces

Swappable Logic Unit

- Hardware Interface
  - – Signals on SLU perimeter
  - – Bus-accessible registers within SLU (Slave)
  - – Active bus access by SLU (Master)
- Software Interface
  - – Use like a library API
  - – Compiler directly supports SLU programming

Swappable Logic Unit

- Sea of Accelerators [10]

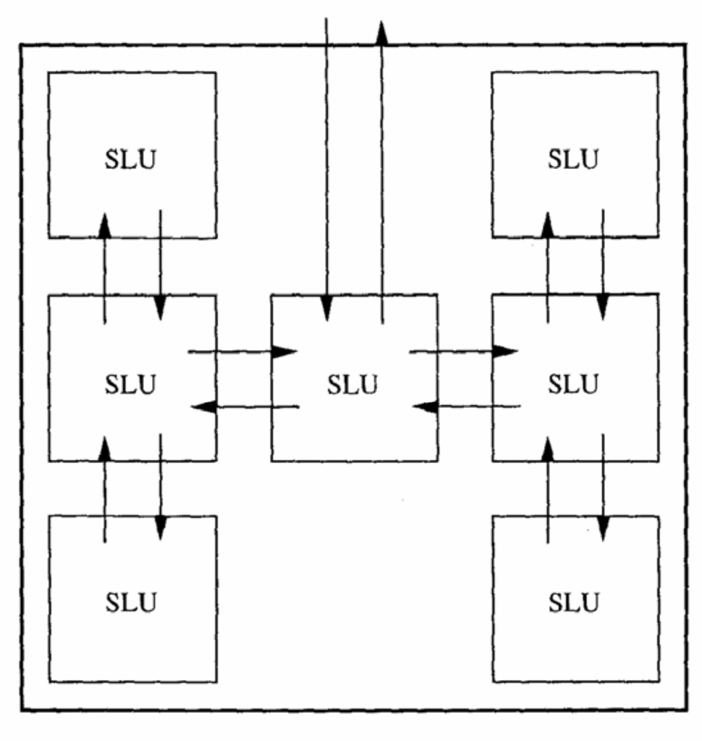


(a) Sea with five accelerators present, placed fairly randomly

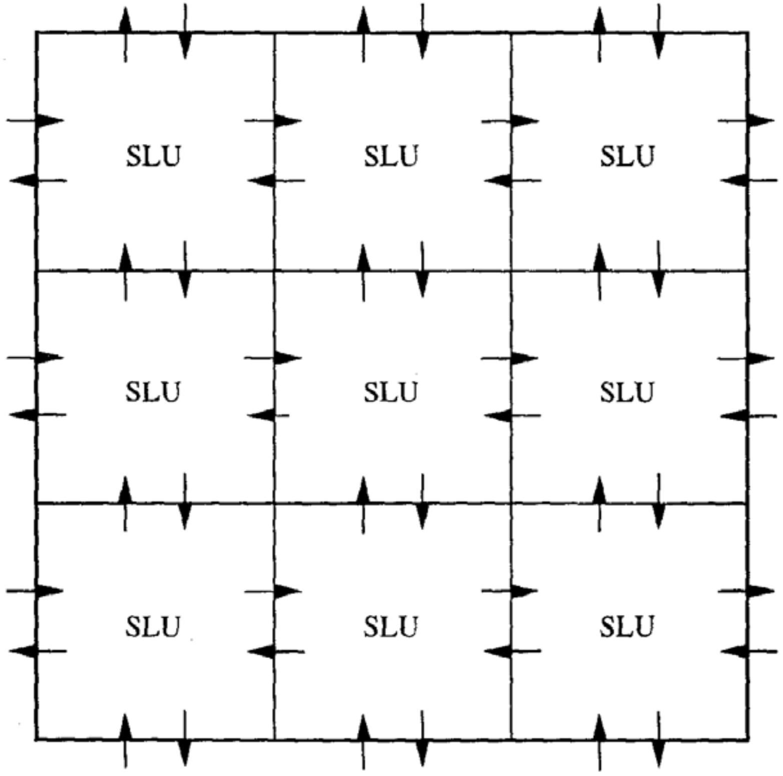
(b) Sea with five accelerators present, densely packed

Swappable Logic Unit

- Parallel Harness [10]



(a) Parallel harness with extra H-tree wiring supplied



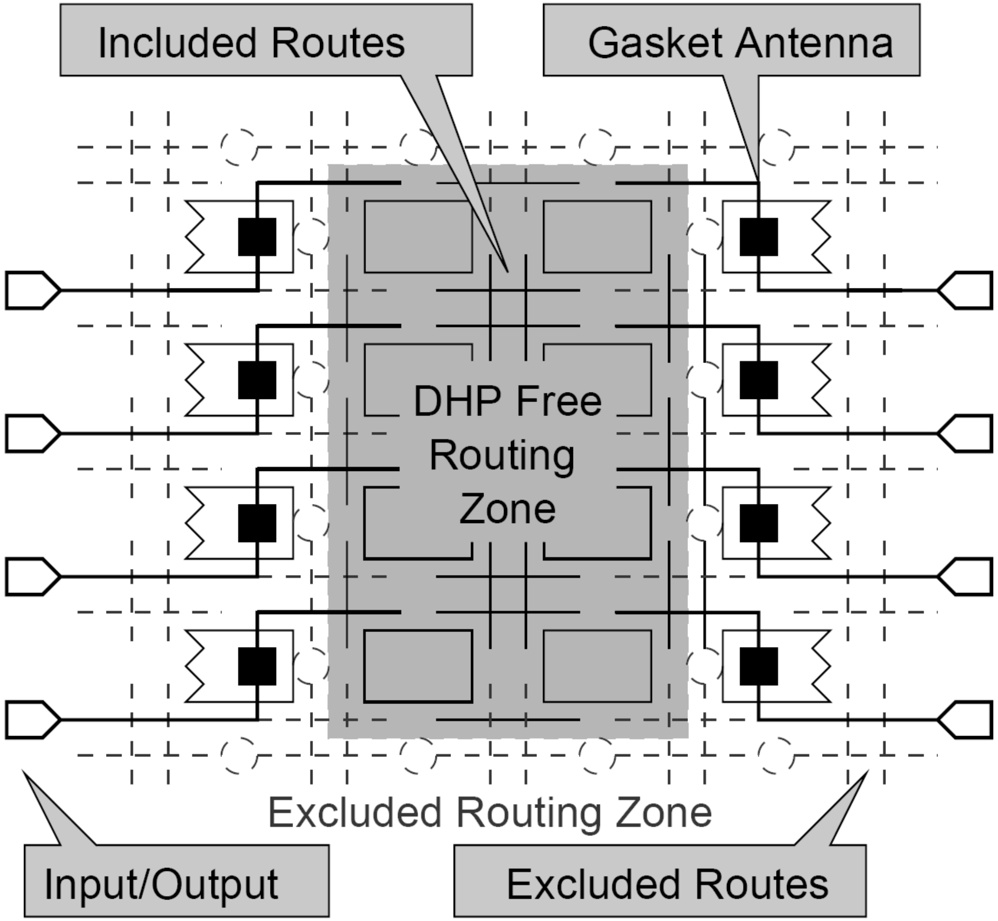
(b) Parallel harness with no extra wiring needed

Dynamic Hardware Plugins

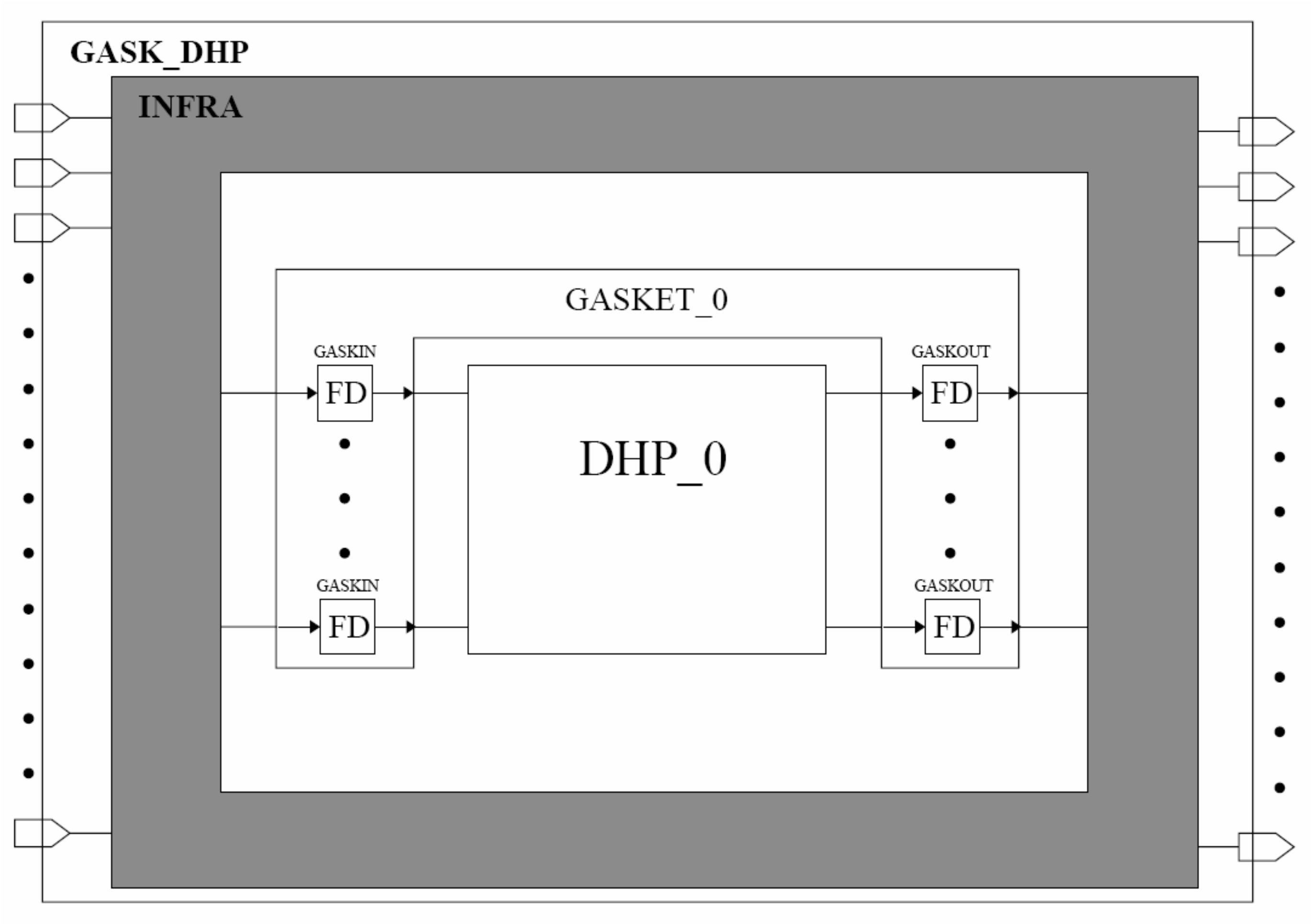
- Similar to Dynamic Linked Libraries (DLL) in software applications
- Useful in packet processing such as firewalls and routers that cannot be suspended
- Platform
  - – Virtex-E XCV2000E
- Tool
  - – PARBIT [3] to restructure bitfiles
- Features
  - – Gasket interface, antennas, DHP, …

Gasket Interface

- Gasket (墊片、墊圈)
  - Between a DHP module and infrastructure
  - Provides antennas (fixed signal interface)
  - Included routes: for DHP only
  - Excluded routes: not for DHP
  - Interface is not changed during configuration



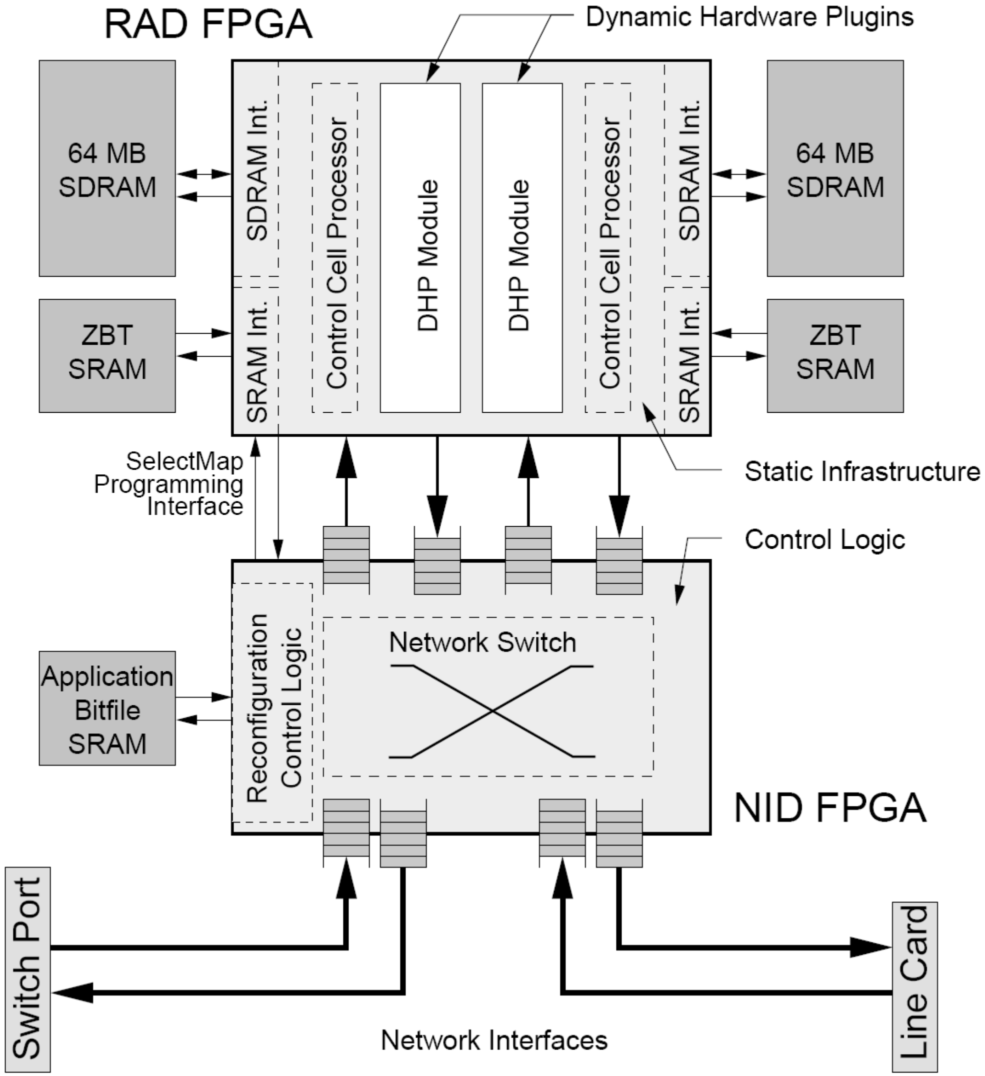
DHP, Gasket, Infrastructure



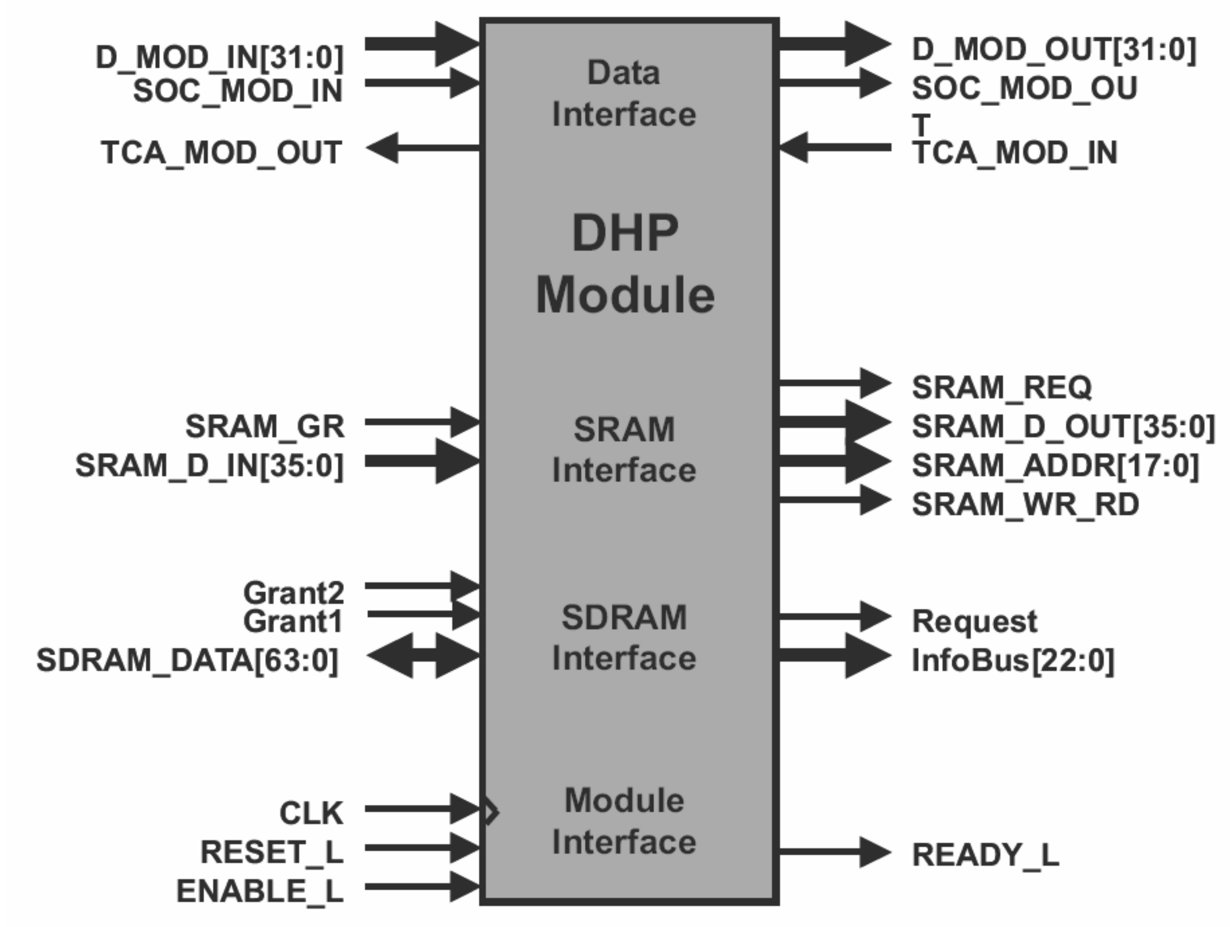
Field-Programmable Port Extender (FPX) System

- Two FPGAs
  - Reprogrammable Application Device (RAD)
    - XCV2000E FPGA
    - All DHP modules and static modules
    - Interface with SRAM, SDRAM
  - Networking Interface Device (NID)
    - XCV600E FPGA
    - Reconfiguration control logic for RAD FPGA
    - Network switch between network interfaces and DHP modules

FPX System


RAD FPGA
Dynamic Hardware Plugins
64 MB
SDRAM
SDRAM Int.
Control Cell Processor
DHP Module
DHP Module
Control Cell Processor
SDRAM Int.
64 MB
SDRAM
ZBT
SRAM
SRAM Int.
SRAM Int.
ZBT
SRAM
SelectMap
Programming
Interface
Static Infrastructure
Control Logic
Application
Bitfile
SRAM
Reconfiguration
Control Logic
Network Switch
NID FPGA
Switch Port
Line Card
Network Interfaces

DHP Module in FPX System



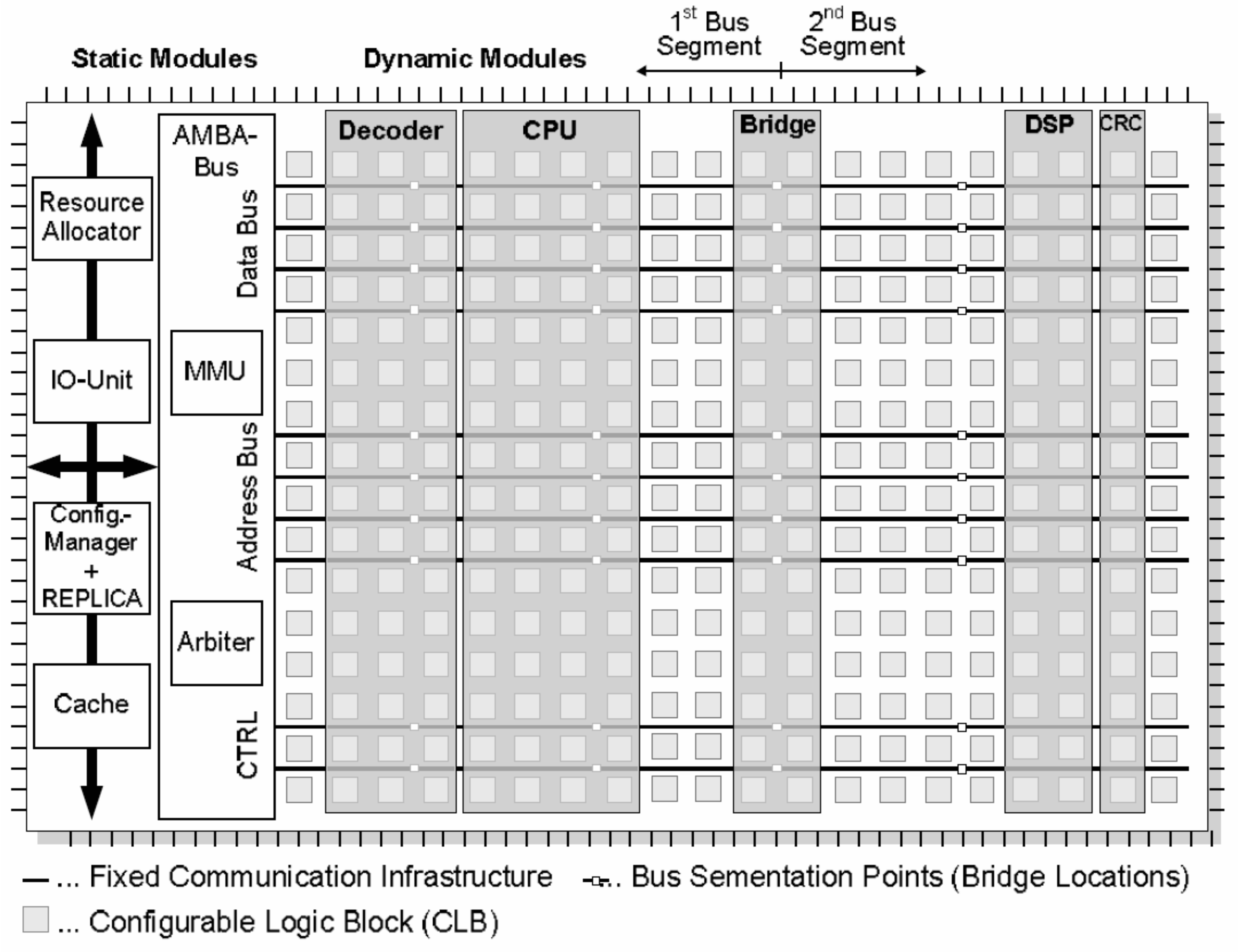
DHP Implementation on FPX

- Two DHP module slots
- Two bitfiles
  - – Infrastructure of RAD
  - – DHP module
    - PARBIT used for partial bitfile generation
      - – 187 KB instead of 1,270 KB for XCV2000E full chip
    - Can be located in either of the two slots using PARBIT

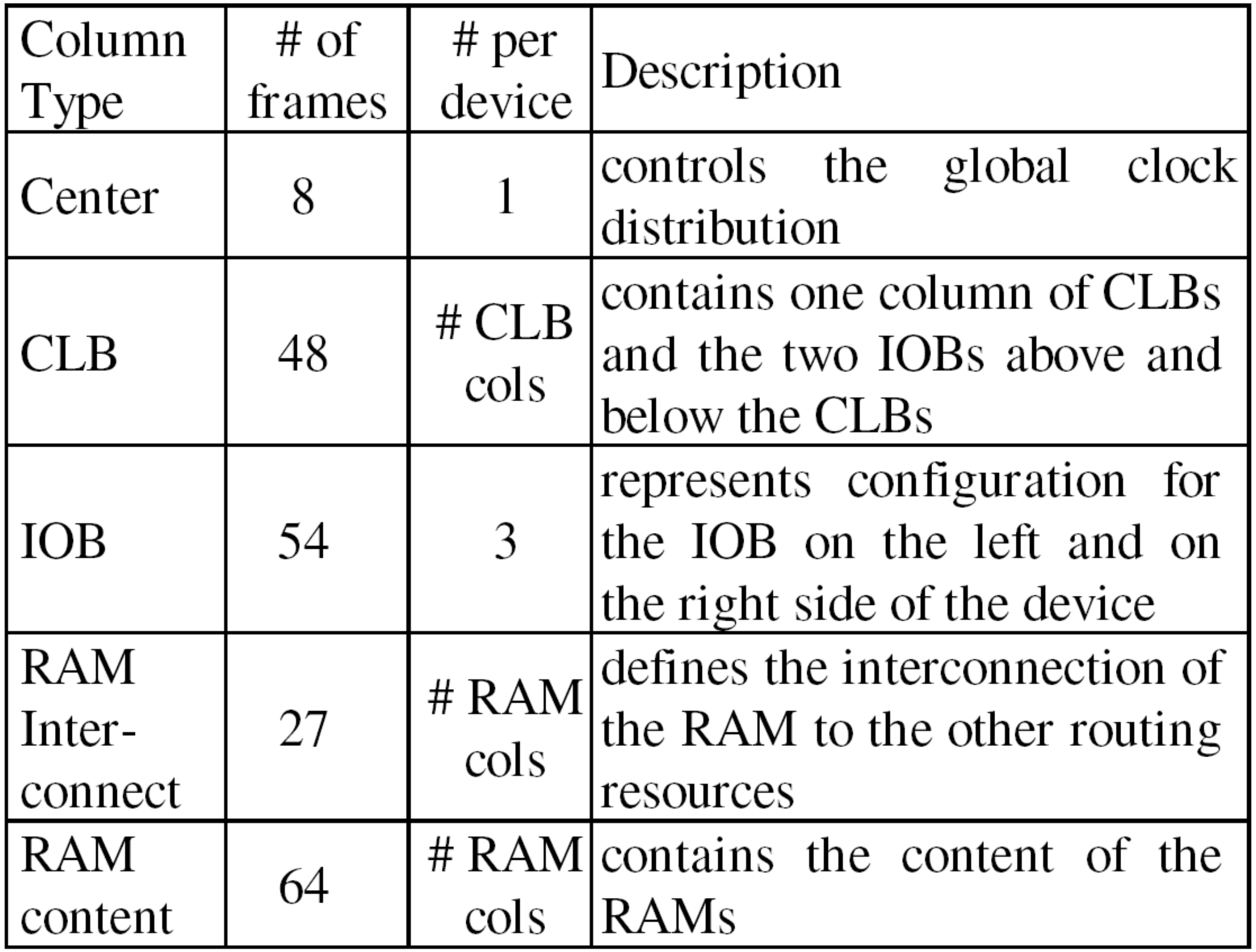
REPLICA

- A bitstream manipulation filter for module relocation
- Module Design
  - Modules occupy full column height
  - Can be relocated along chip width
- Bus Design
  - Bus is chip-wide and homogeneous
  - Can be segmented using bridge

Reconfigurable System Architecture



Virtex Configuration Column Types



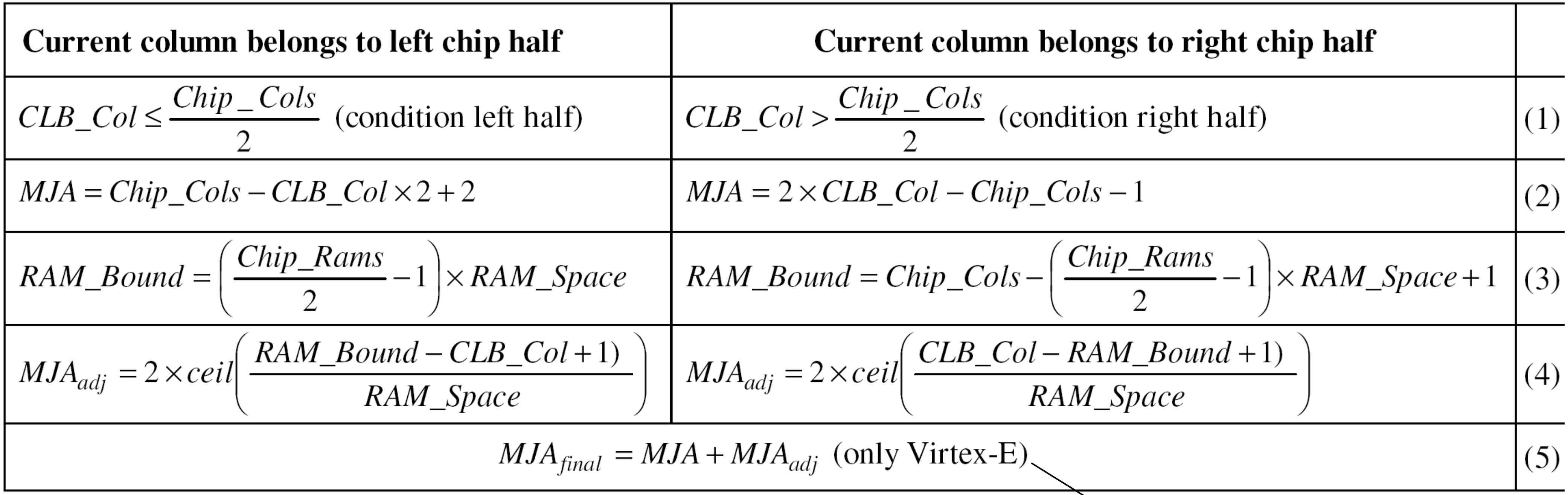
Column Type	# of frames	# per device	Description
Center	8	1	controls the global clock distribution
CLB	48	# CLB cols	contains one column of CLBs and the two IOBs above and below the CLBs
IOB	54	3	represents configuration for the IOB on the left and on the right side of the device
RAM Inter-connect	27	# RAM cols	defines the interconnection of the RAM to the other routing resources
RAM content	64	# RAM cols	contains the content of the RAMs

Virtex Address

- Configuration address space divided into:
  - RAM blocks
    - BlockRAM content
  - CLB blocks
    - All other column types
- Address
  - MJA: Major Address
  - MNA: Minor Address

Virtex Address Calculation

- For Virtex 1 and 2: use only Eqs. (1), (2)



Current column belongs to left chip half	**Current column belongs to right chip half**	
$CLB\_Col \le \frac{Chip\_Cols}{2}$ (condition left half)	$CLB\_Col > \frac{Chip\_Cols}{2}$ (condition right half)	(1)
$MJA = Chip\_Cols - CLB\_Col \times 2 + 2$	$MJA = 2 \times CLB\_Col - Chip\_Cols - 1$	(2)
$RAM\_Bound = \left(\frac{Chip\_Rams}{2} - 1\right) \times RAM\_Space$	$RAM\_Bound = Chip\_Cols - \left(\frac{Chip\_Rams}{2} - 1\right) \times RAM\_Space + 1$	(3)
$MJA_{adj} = 2 \times ceil\left(\frac{RAM\_Bound - CLB\_Col + 1)}{RAM\_Space}\right)$	$MJA_{adj} = 2 \times ceil\left(\frac{CLB\_Col - RAM\_Bound + 1)}{RAM\_Space}\right)$	(4)
$MJA_{final} = MJA + MJA_{adj}$ (only Virtex-E)		(5)

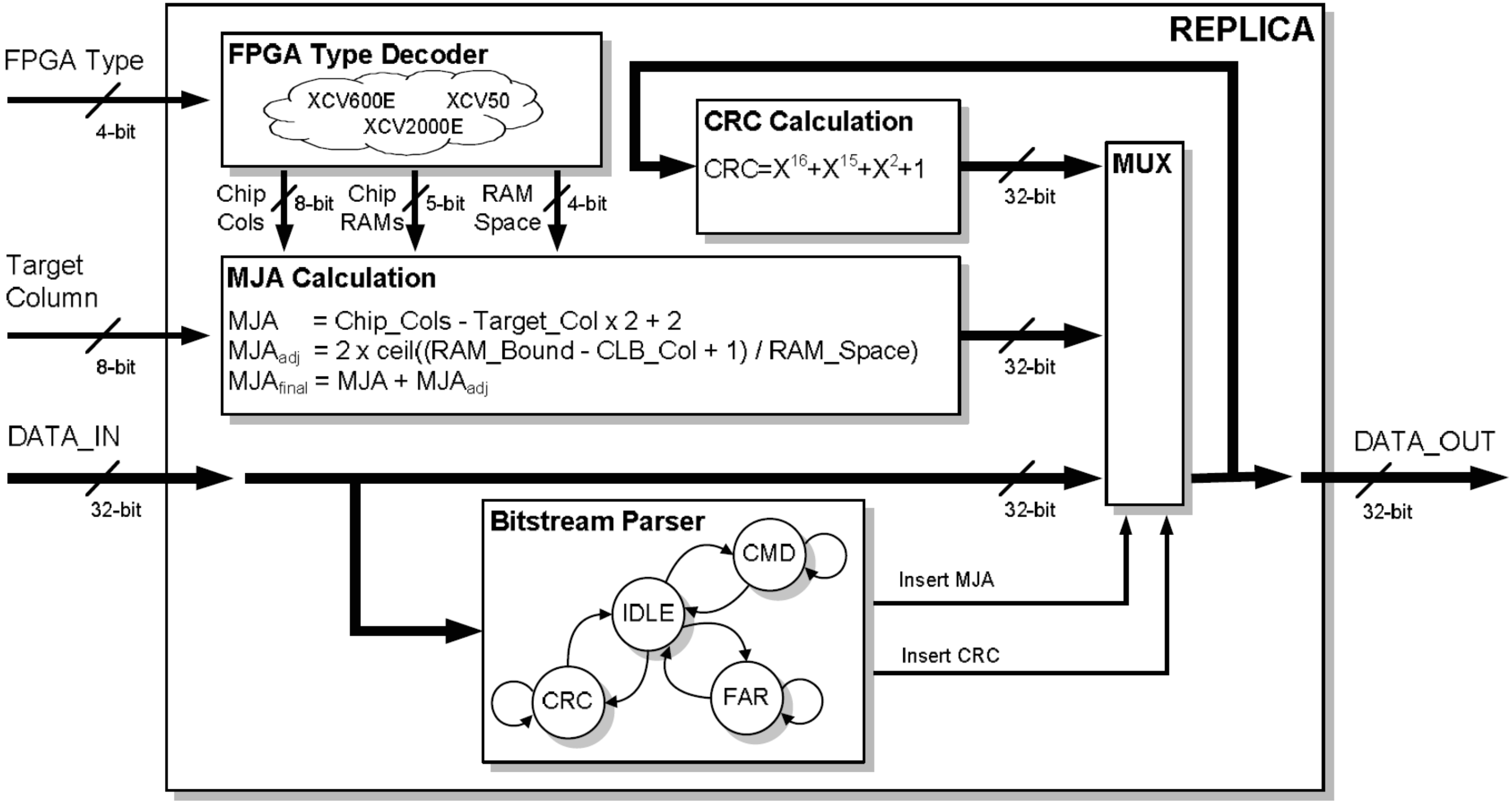
- What about Virtex 4 and 5?

RAM columns in the middle of the chip

REPLICA Filter

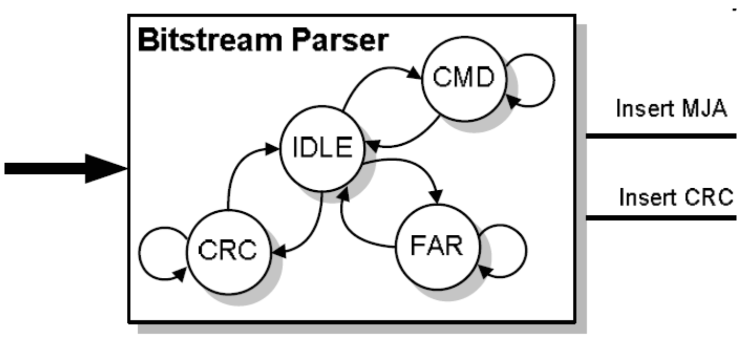
- Four blocks and a data multiplexer
  - – Bitstream Parser
  - – FPGA Type Decoder
  - – CRC Calculation
  - – MJA Calculation

REPLICA Filter



Bitstream Parser

- To replace some words so that a module is relocated to another CLB column
  - One MJA entry per CLB column
  - Two CRC checksums per bitstream
- Distinguish between data / command words
- Search for write commands for
  - FAR (Frame Address Register)
  - CRC



To disable REPLICA:

Set target column to 0

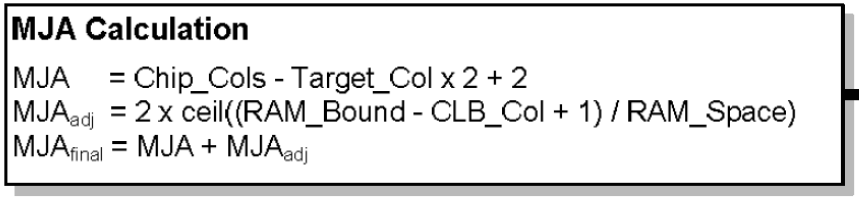
FPGA Type Decoder

- FPGA type can be determined from frame length
- Issue:
  - The first manipulated MJA occurs before frame length
- Possible Solutions
  - Huge shift register delaying output data
    - Resource wasted
  - User given
    - More efficient
  - Hard coded
    - For final implementation

CRC Calculation

- Select data words to be included in calculation and calculate CRC checksum
- CRC polynomial
  – CRC = $X^{16} + X^{15} + X^{2} + 2$
- Parallel implementation
  – To provide new checksum within one clock cycle

MJA Calculation



MJA Calculation

MJA = Chip_Cols - Target_Col x 2 + 2

MJA_{adj} = 2 x ceil((RAM_Bound - CLB_Col + 1) / RAM_Space)

MJA_{final} = MJA + MJA_{adj}

- Calculate MJA
  - Using equations (1) ~ (5),
  - Target column: CLB_Col, and
  - FPGA Parameters: Chip_Cols, Chip_Rams, RAM_Space
- No need of knowing the original column
- 19 clock cycles needed for an MJA calculation
  - Between two MJA entries: 100x~1000x frame data words
  - First MJA entry: 7 data words
    - **7 x 4 = 28 clock cycles > 19**, hence no problem!

(1 word = 32-bit = 4 bytes, 4 cycles on 8-bit SelectMAP port)

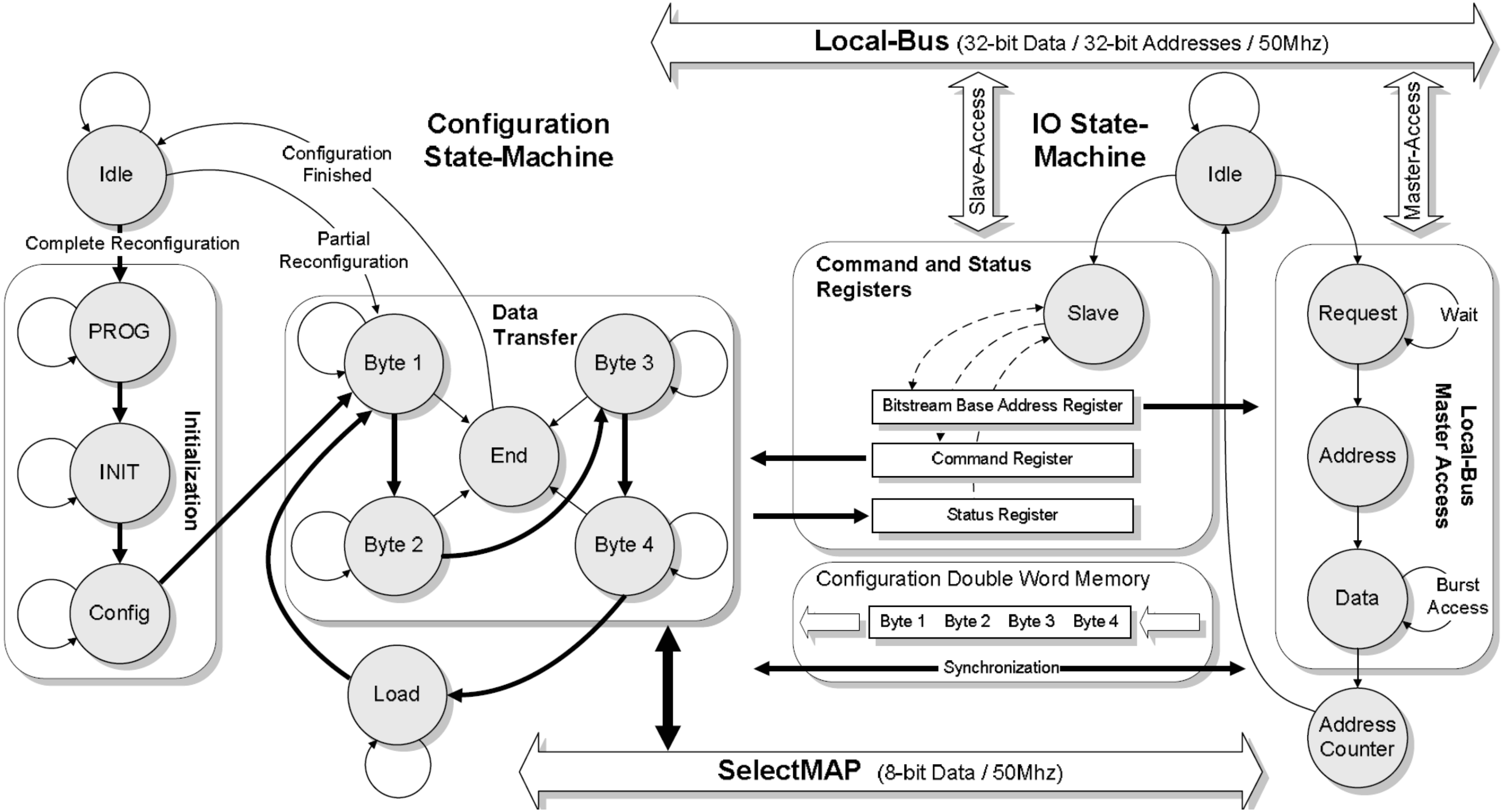
REPLICA Synthesis

- Xilinx ISE 6.1
- 336 slices in Virtex-E device
- 50 MHz clock
  - Same as the maximum allowed by SelectMAP configuration interface
- Critical part
  - Control of data multiplexer

Configuration Manager

- Organizes data transfer of bitstreams from memory to FPGA
- Controls configuration through SelectMAP port
- Part of RAPTOR2000 rapid prototyping platform
- Two FSMs
  - IO State Machine
    - For data transfer
  - Configuration State Machine
    - For configuration control

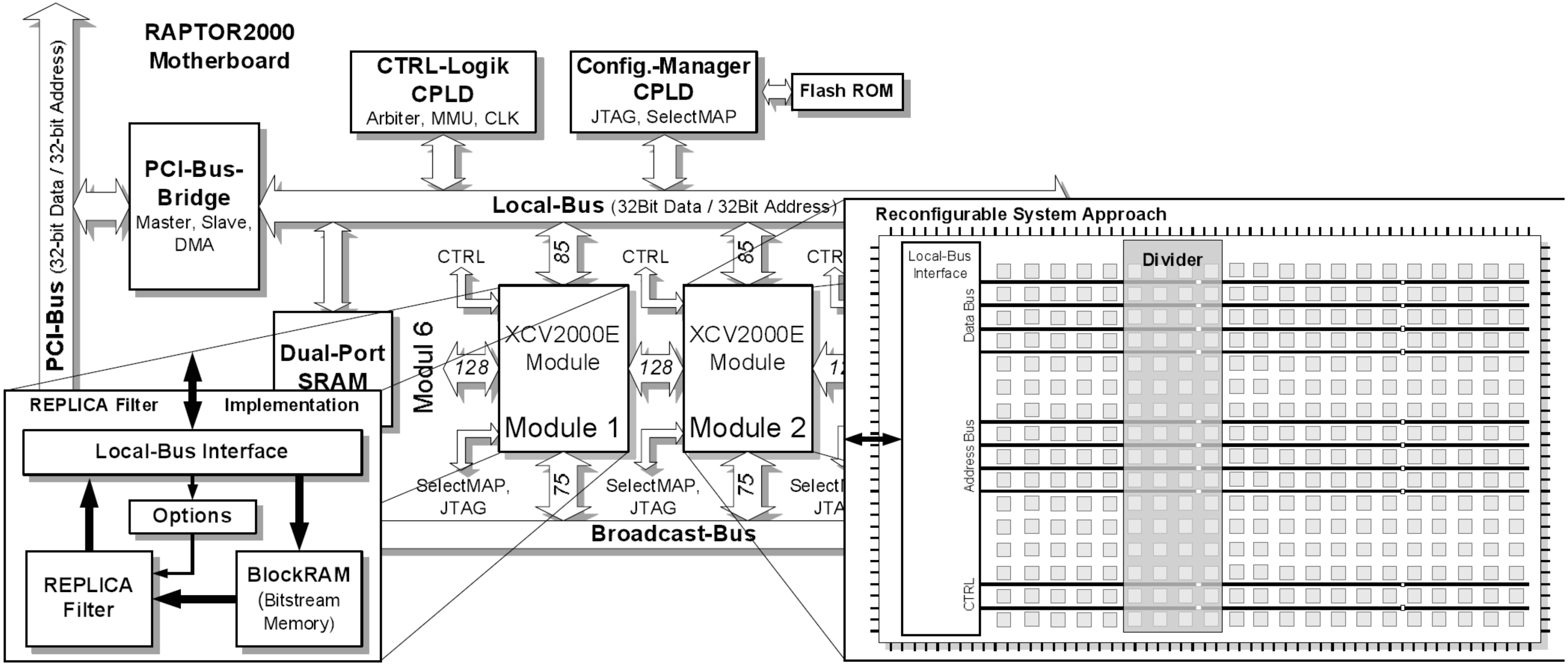
Configuration Manager



Configuration Manager Synthesis

- Two implementations of CM:
  - XC95288XL CPLD on RAPTOR2000
    - 209 out of 288 macro cells (75%)
    - 58 MHz clock (> 50 MHz, more than required)
  - XCV2000E FPGA
    - 151 out of 19200 slices (0.8%)
- REPLICA + CM (on XCV2000E)
  - 490 out of 19200 slices (2.5%)

RAPTOR2000 Platform



Comparisons of Relocation Architectures

Feature	SLU 1996	DHP 2001	REPLICA 2005
Module	2D area	1D slot	1D Column
I/O interface	signal / bus	gasket	AMBA bus
Relocation	static	static	dynamic
Configuration	OS4RS	NID FPGA	HW CM
Platform	Concept (XC6200)	FPX (Virtex-E)	RAPTOR (Virtex-E)

Outline

- Reconfigurable vs. Conventional Hardware
- Hardware Preemption and Relocation
- Area-Time Tradeoff Techniques
- Communication Architectures

Area-Time Tradeoff Techniques

- In a fixed amount of reconfigurable resources, tradeoffs can be made between
  - Area: amount of resources used
    - Sequential
    - Pipelined
    - Spatially parallel
  - Time: data throughput of design
    - Several cycles per data (sequential)
    - One cycle per data (pipelined)
    - Several data per cycle (parallel)

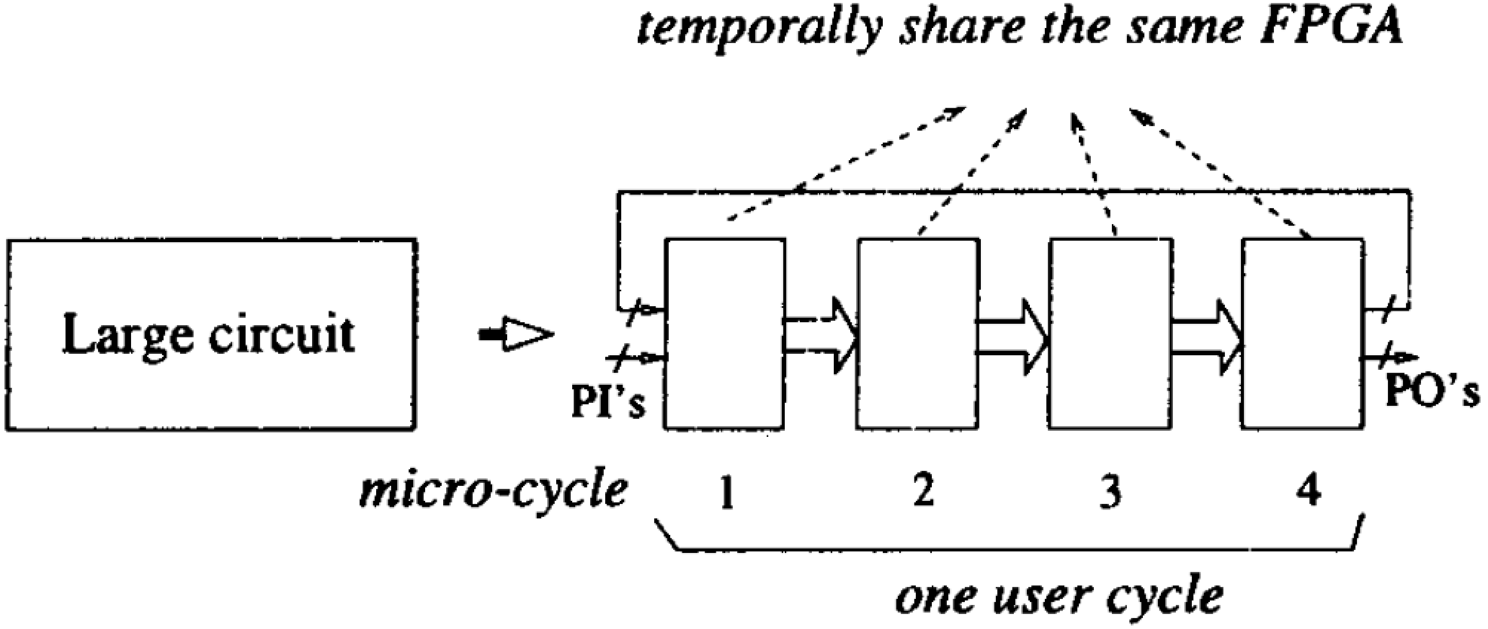
Area-Time Tradeoff Techniques

- Coarse-Grained Time Multiplexing
  - Temporal Partitioning
- Spatial Parallelism
- Pipelining
- Template Specialization

Coarse-Grained Time Multiplexing

- Also called “Temporal Partitioning”
- A large circuit might not fit into a reconfigurable device
  - A large circuit is broken down into several smaller circuits such that
    - each can be executed by a fixed amount of reconfigurable resources and
    - together they function just like the large circuit

Temporal Partitioning



Benefits of Temporal Partitioning

- Makes the execution of an oversized circuit possible using limited resources
- Increases functional density
  - D = 1/(AxT), A: area, T: time, D: density
  - Useful in partial reconfiguration
- Makes multi-context FPGAs more efficient
  - Switch in a single cycle
- Study of minimum granularity to achieve performance far exceeding processors

Temporal Partitioning

- Two types
  - – Gate level
    - Assumes small reconfiguration overhead
  - – High level or operational level
    - Assumes large reconfiguration overhead

Gate Level Temporal Partitioning

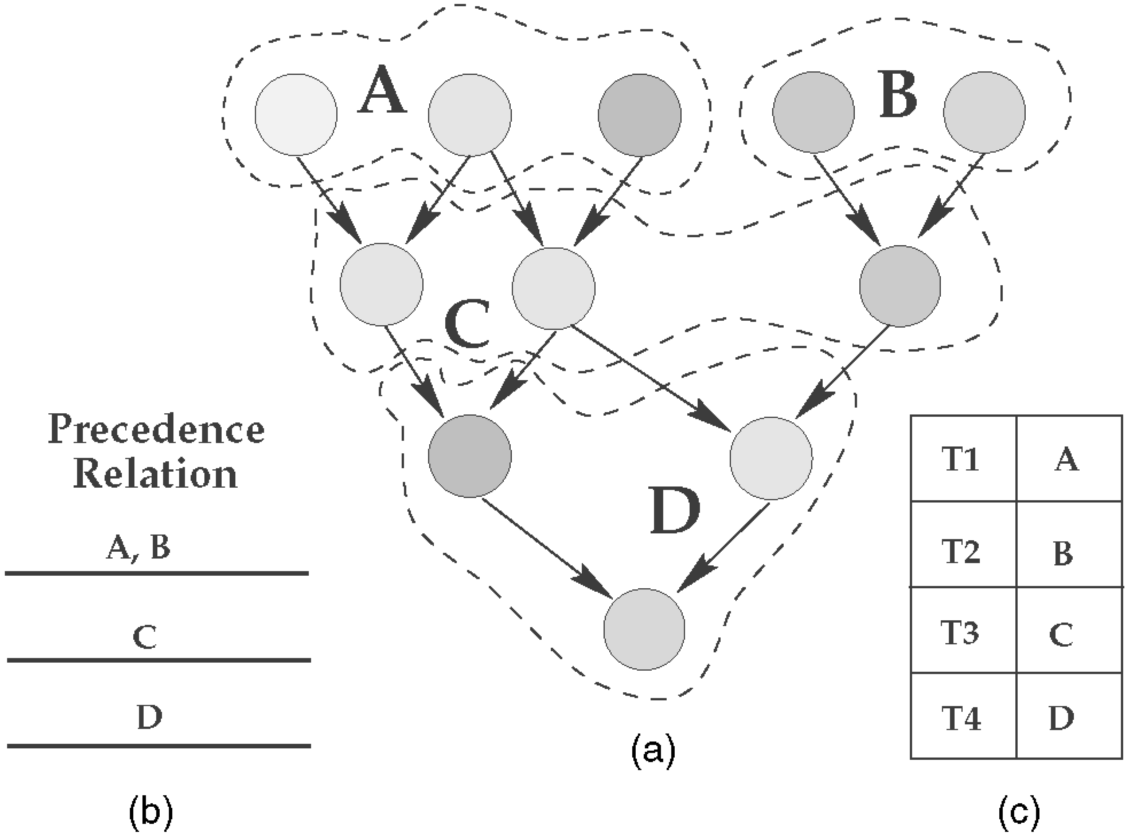
- Mainly scheduling or optimization based
  - List scheduling [12]
  - Enhanced force-directed scheduling [13]
  - Network flow optimization [14]
  - Weighted graph partitioning [15]
- Model
  - Mealy state machine [12]
  - Time-multiplexed communicating logic (TMCL) [13,14]
  - Directed hypergraph [15]
- Takes care of combinational and sequential nets

High-Level Temporal Partitioning

- Methods
  - Integer Linear Programming [16, 17]
  - Data-Flow Graph Partitioning [18]
    - Level based
    - Clustering based
- Models
  - ILP problem
  - DFG

DFG Partitioning and Scheduling

- Example



(a)

(b)

T1	A
T2	B
T3	C
T4	D

(c)

REF: [18]

DFG Partitioning and Scheduling

- Inputs
  - – A data-flow graph (directed acyclic graph)
    - $G = (V, E, W, D)$
      - – V: a node represents a function implementation
      - – E: an edge $<v_i, v_j>$ means v_j needs data output of v_i
      - – W: a weight represents the size of logic
      - – D: a delay represents the function execution time
  - – A configurable unit of size S_{RPU}

DFG Partitioning and Scheduling

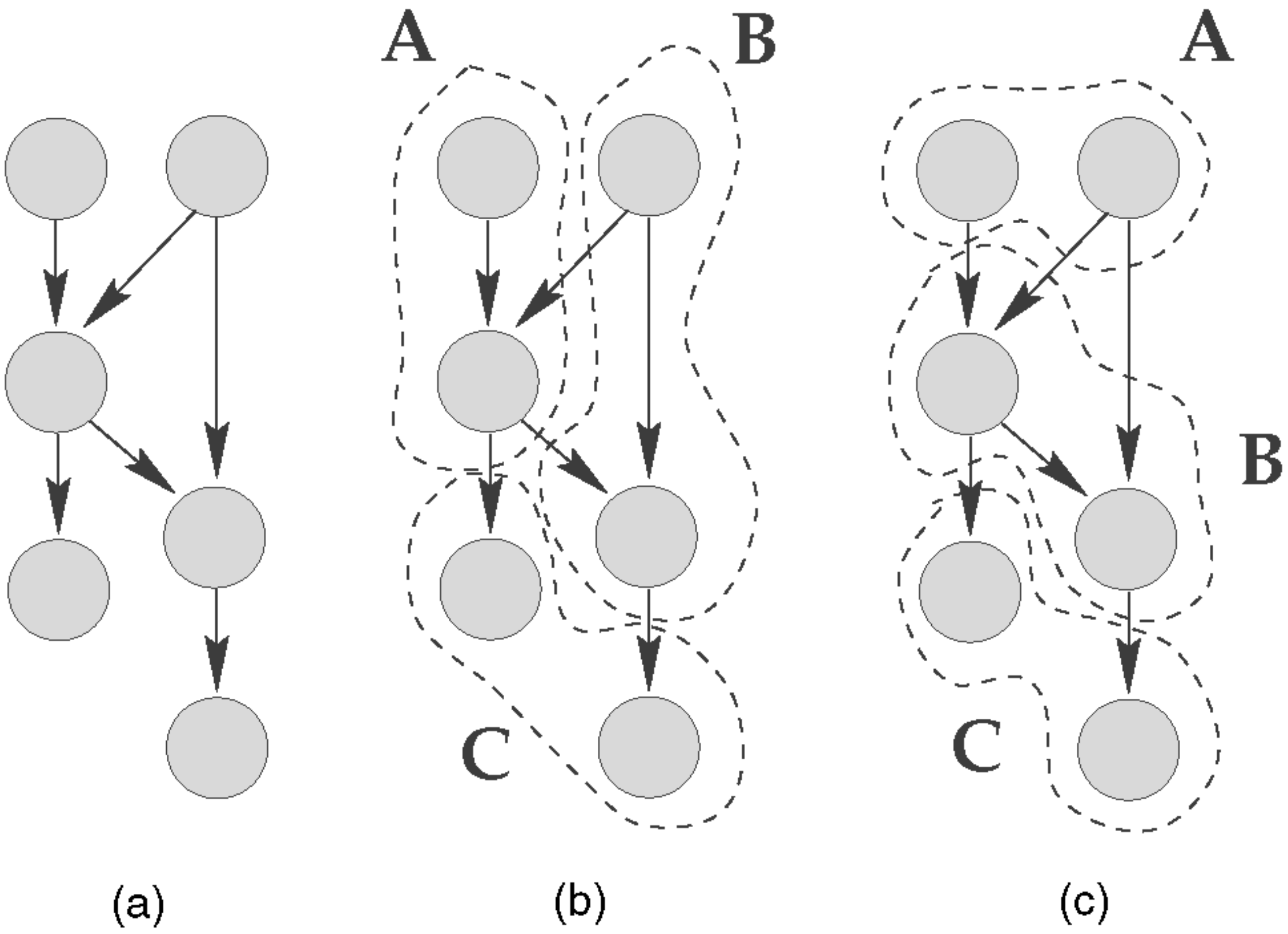
- Objective
  - Divide *G* into *k* segments such that
    - Size of each segment is $\leq S_{RPU}$
    - There exists an acyclic precedence relation for all *k* segments

Acyclic Precedence Relation

(a) Original graph

(b) Cycle exists

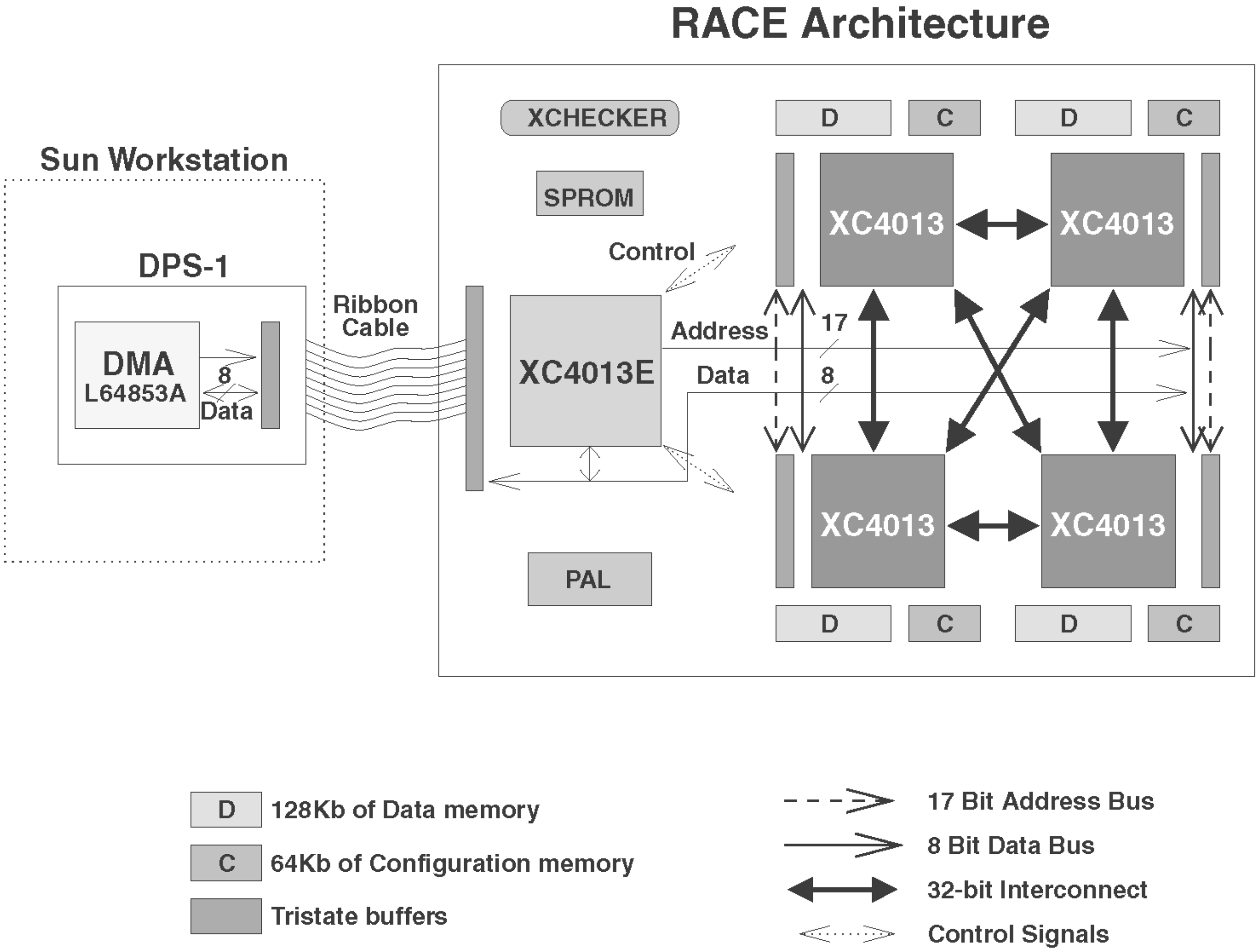
(c) Acyclic



RACE Environment

- Four Xilinx XC4013 FPGAs
  - Interconnected in a complete graph
  - Each FPGA has 128 KB data memory and 64 KB of configuration memory
- One controller XC4013 FPGA for programming
  - FPGA
  - DMA transfers to host system
- SUN SparcStation
  - Connected to FPGAs using SBUS interface

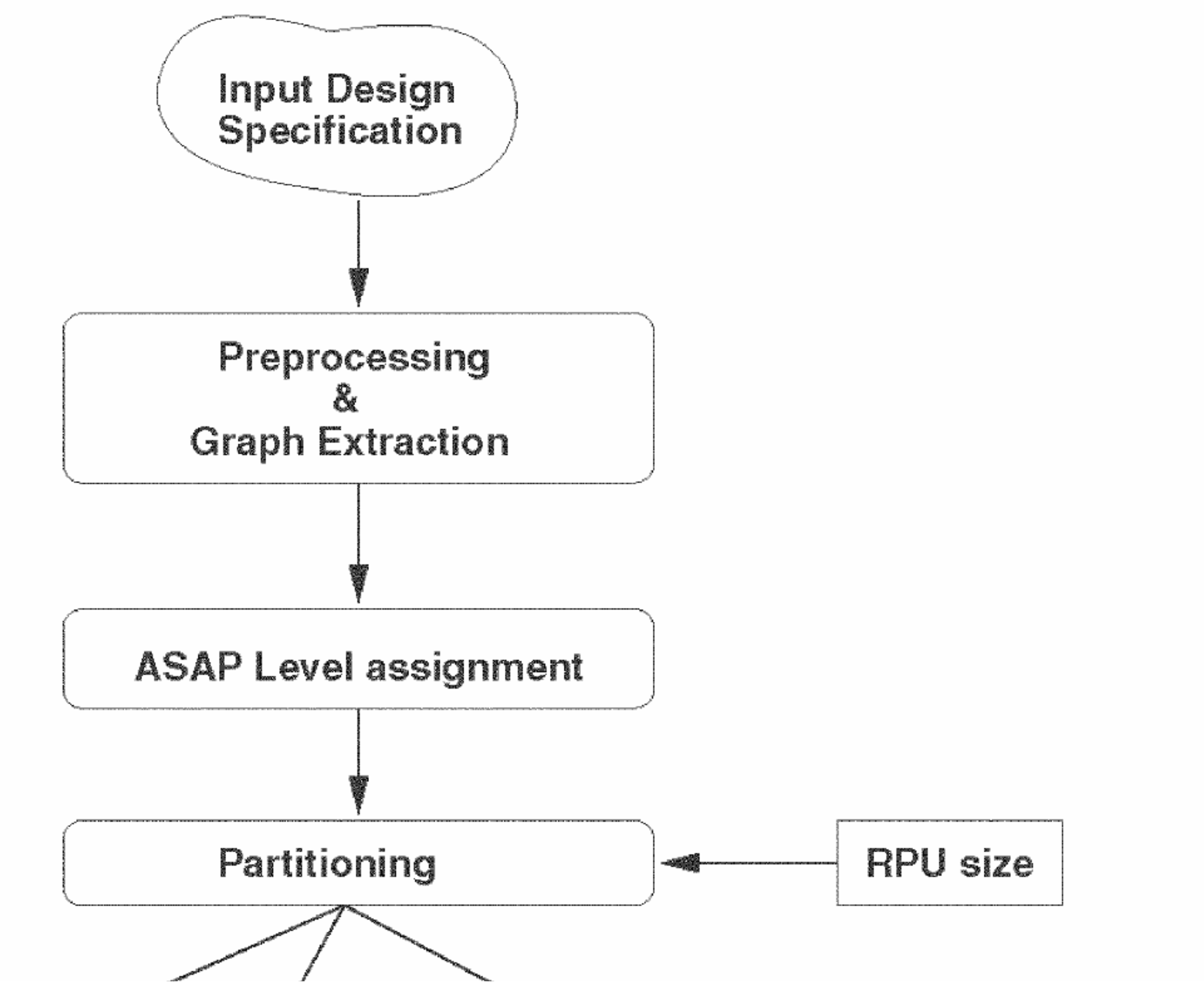
RACE Environment



DFG Partitioning and Scheduling

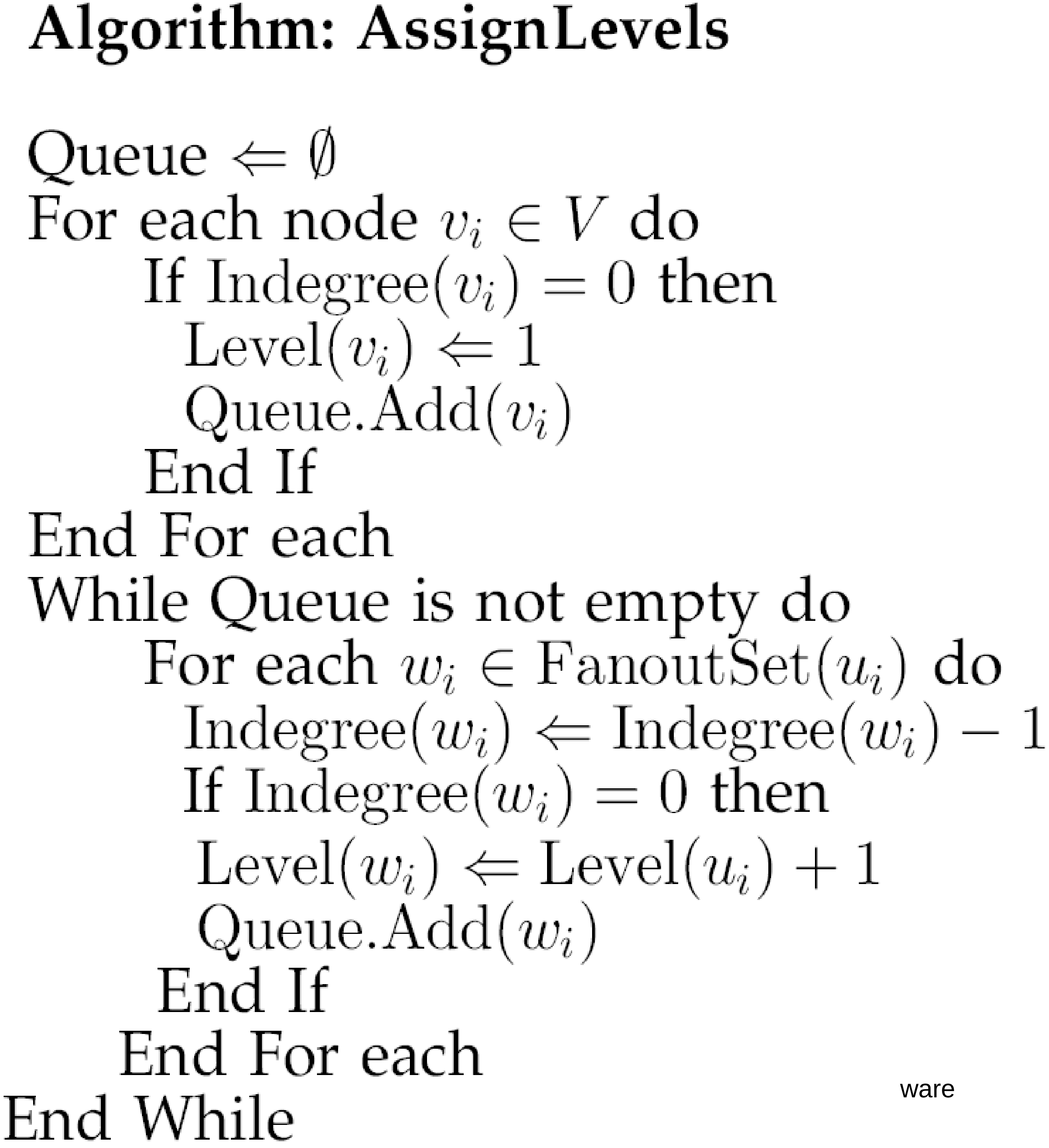
- ASAP Level Assignment
- Partitioning Algorithms
  - Level Based Partitioning
  - Clustering Based Partitioning
- Data Controller Synthesis
- DFG Scheduling

DFG Partitioning and Scheduling



ASAP Level Assignment

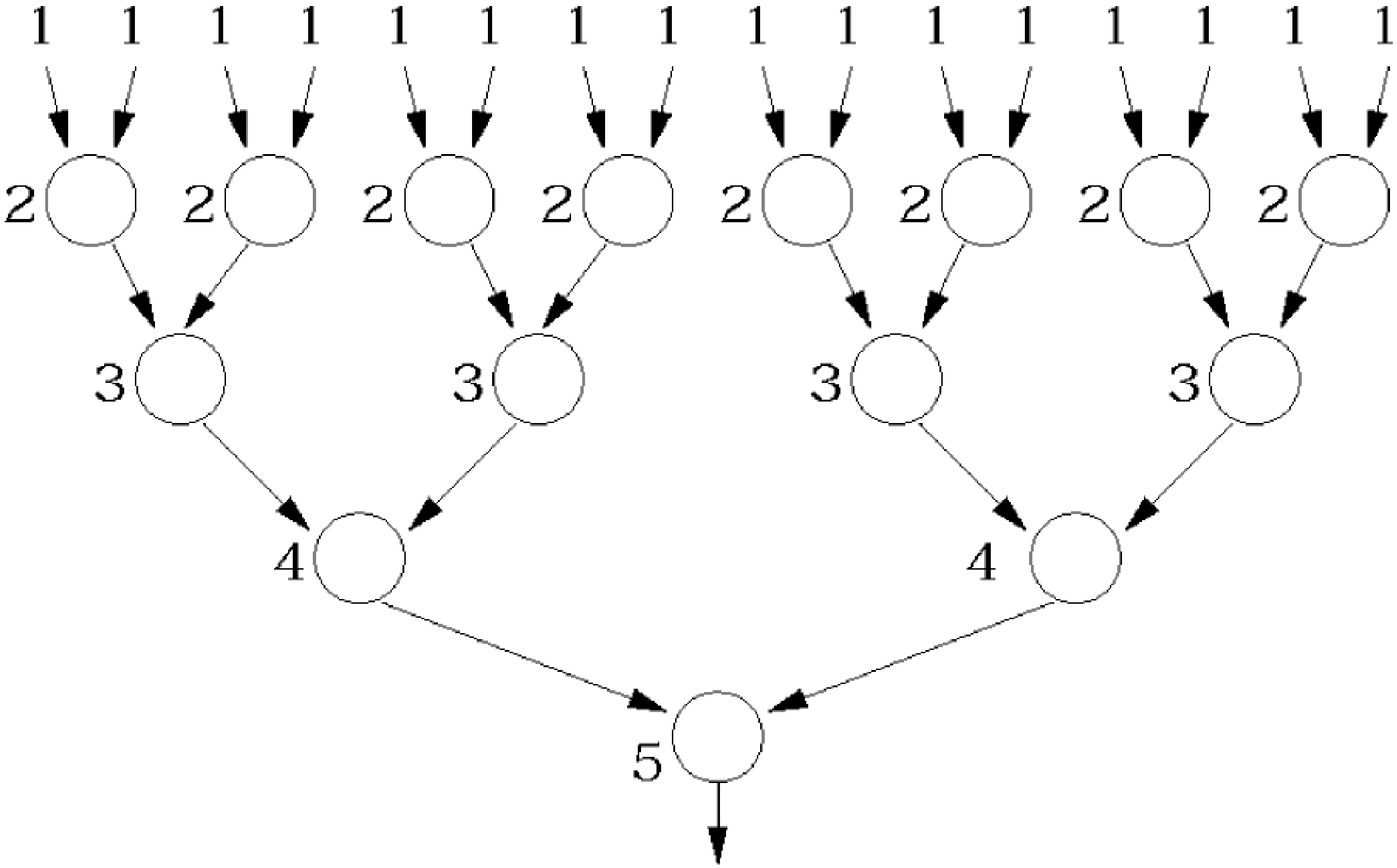
- To assign each node v_i a *Level*(v_i)
- Indegree(v_i)
  - # incoming edges with Level = ∞
- FanoutSet(v_i)
  - Set of fanout nodes of v_i
- FaninSet(v_i)
  - Set of fanin nodes of v_i



Algorithm: AssignLevels

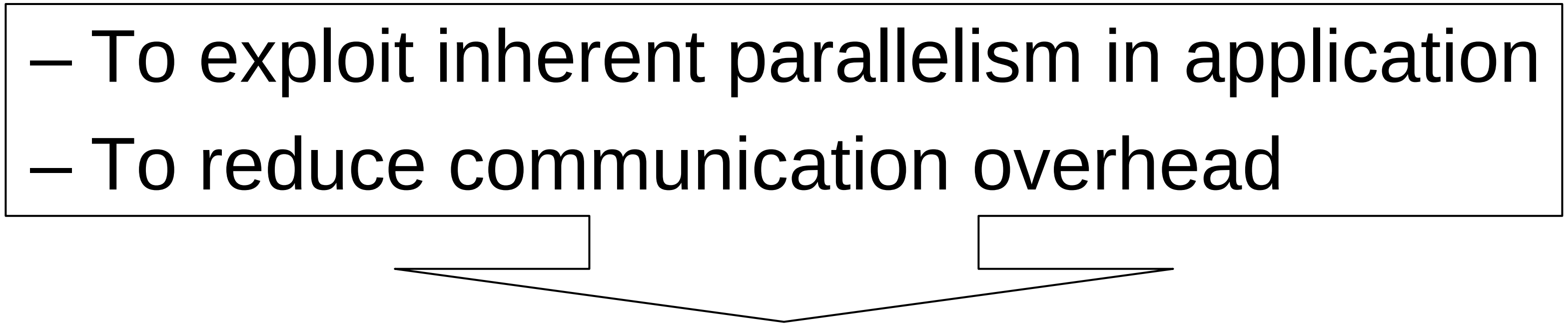
```
Queue $\Leftarrow \emptyset$
For each node $v_i \in V$ do
    If Indegree$(v_i) = 0$ then
      Level$(v_i) \Leftarrow 1$
      Queue.Add$(v_i)$
    End If
End For each
While Queue is not empty do
    For each $w_i \in$ FanoutSet$(u_i)$ do
      Indegree$(w_i) \Leftarrow$ Indegree$(w_i) - 1$
      If Indegree$(w_i) = 0$ then
       Level$(w_i) \Leftarrow$ Level$(u_i) + 1$
       Queue.Add$(w_i)$
     End If
   End For each
End While
```

ASAP Level Assignment Example



Partitioning Algorithms

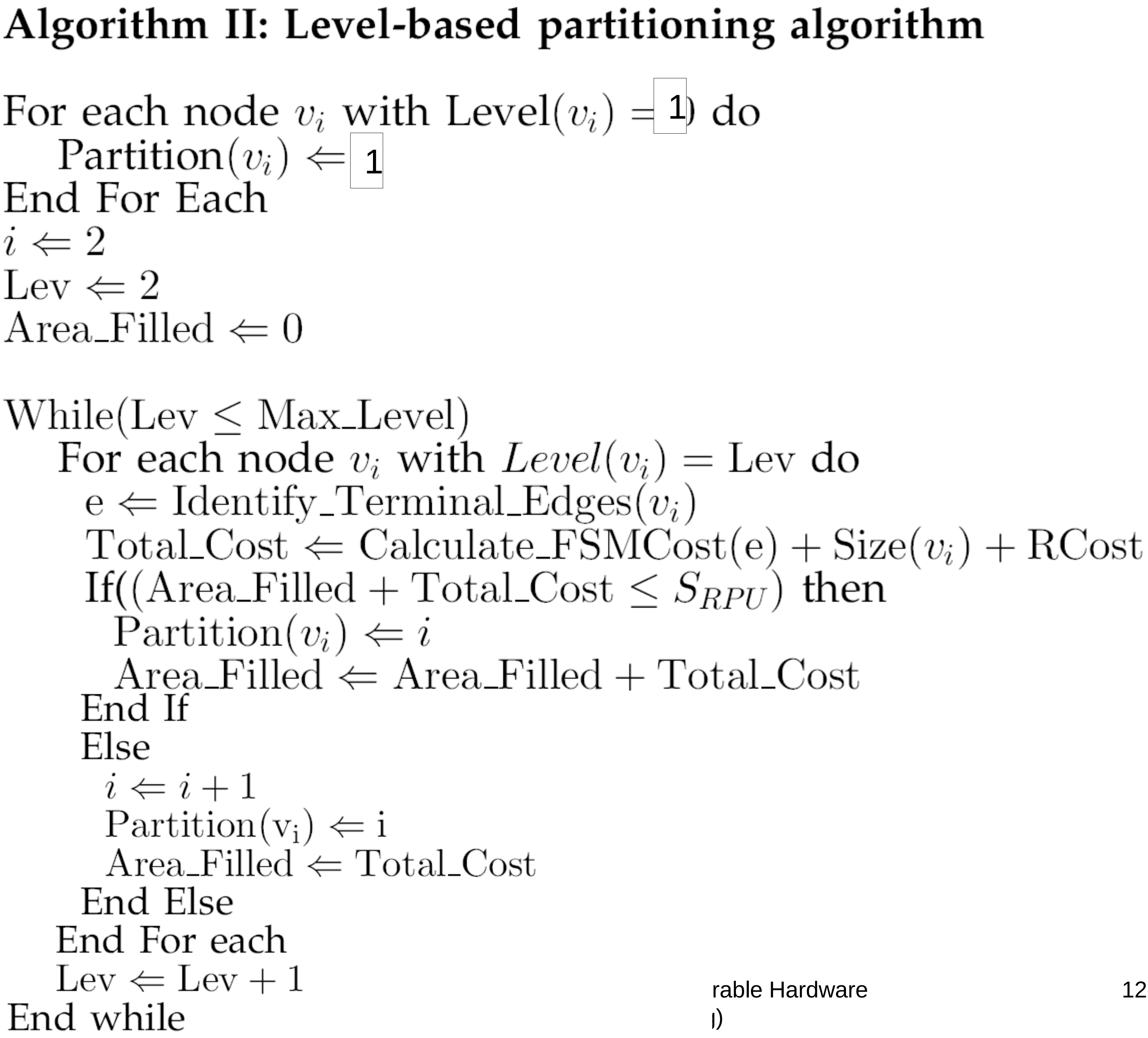
- Objectives
  - To satisfy area constraint S_{RPU}



  - To exploit inherent parallelism in application
  - To reduce communication overhead
  - Tradeoff between parallelism and communication overhead
  - Level based vs. Clustering based

Level Based Partitioning

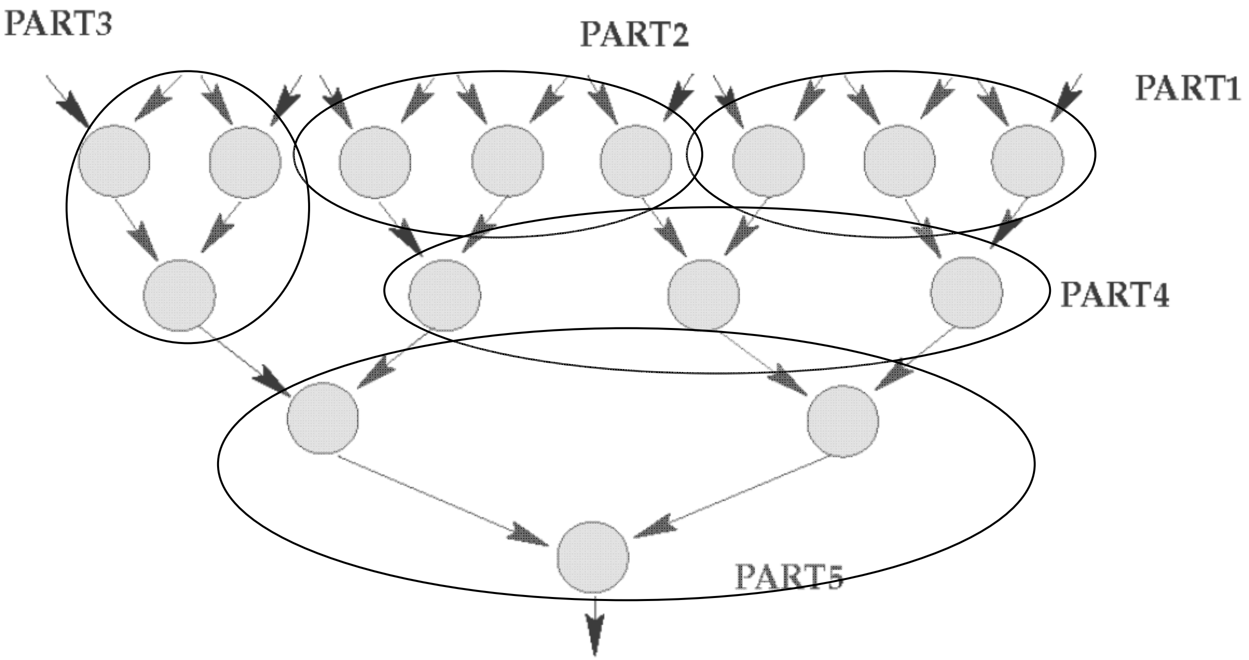
- Exploits parallelism in application
- All nodes at the same level can be considered for parallel execution
- Overheads
  - Limited routing resources (*RCost*)
  - Need a data controller (*FSMCost*)
  - $Available\_Area = S_{RPU} – FSMCost – RCost$



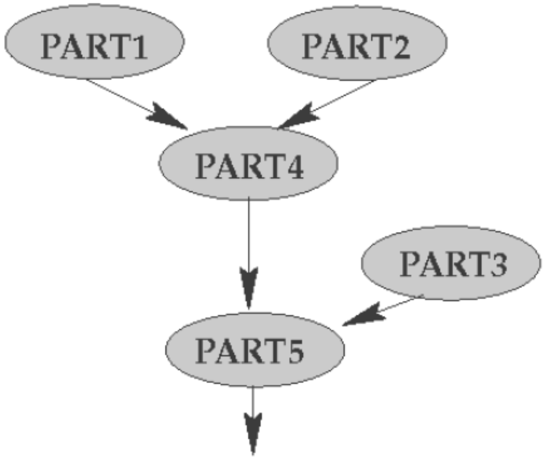
Algorithm II: Level-based partitioning algorithm

```
For each node $v_i$ with Level($v_i$) = 1) do
    Partition($v_i$) $\Leftarrow$ 1
End For Each
$i \Leftarrow 2$
Lev $\Leftarrow$ 2
Area_Filled $\Leftarrow$ 0

While(Lev $\leq$ Max_Level)
    For each node $v_i$ with $Level(v_i)$ = Lev do
      e $\Leftarrow$ Identify_Terminal_Edges($v_i$)
      Total_Cost $\Leftarrow$ Calculate_FSMCost(e) + Size($v_i$) + RCost
      If((Area_Filled + Total_Cost $\leq S_{RPU}$) then
        Partition($v_i$) $\Leftarrow i$
        Area_Filled $\Leftarrow$ Area_Filled + Total_Cost
      End If
      Else
        $i \Leftarrow i + 1$
        Partition($v_i$) $\Leftarrow$ i
        Area_Filled $\Leftarrow$ Total_Cost
      End Else
    End For each
    Lev $\Leftarrow$ Lev + 1
End while
```

Example of Level Based Partitioning



(a)



(b)

PART	IN	OUT
1	6	3
2	6	3
3	4	1
4	6	3
5	4	1

(c)

Level Based Partitioning

- The algorithm complexity is O(|*V*| + |*E*|)

Clustering Based Partitioning

- To decrease communication overhead
  - Reduce number of terminal edges of a partition (incoming + outgoing edges)
- Clustering tends to reduce the number of terminal edges, hence communication overhead
- An alternative way is proposed in [19], based on local minima of memory usages

Algorithm III: Clustering based partitioning

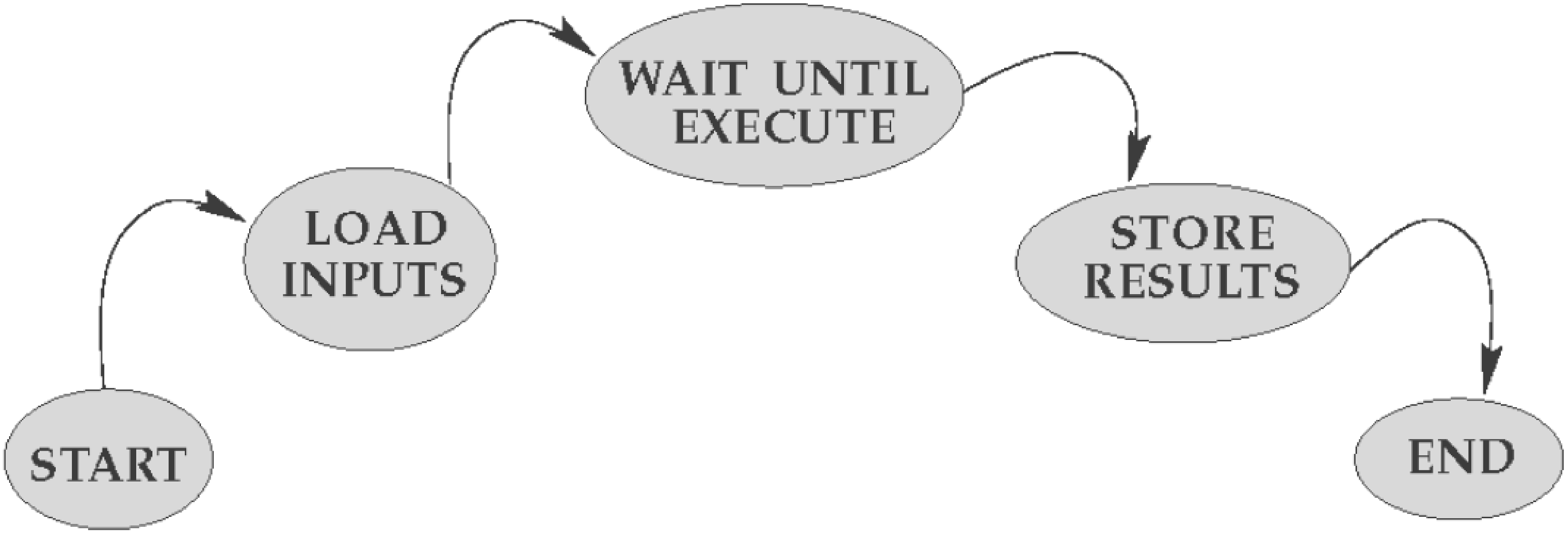
```
ReadyList $\Leftarrow \emptyset$
For each node $v_i \in V$
    Calculate_successor_id$(v_i)$
End For each
For each node $v_i \in V$ with Level$(v_i) = 1$
    Partition$(v_i) \Leftarrow 1$
    ReadyList.update$(v_i)$
End For each

$i \Leftarrow 2$
While ReadyList is not empty do
    $u_i \Leftarrow$ ReadyList.front()
    e $\Leftarrow$ Identify_Terminal_Edges$(u_i)$
    Total_Cost $\Leftarrow$ Calculate_FSMCost(e) + Size($u_i$) + RCost
    If((Area_Filled + Total_Cost $\leq S_{RPU}$) then
        Partition($u_i$) $\Leftarrow i$
        Area_Filled $\Leftarrow$ Area_Filled + Total_Cost
    End If
    Else
        $i \Leftarrow i + 1$
        Partition$(u_i) \Leftarrow i$
        Area_Filled $\Leftarrow$ Total_Cost
    End Else
    ReadyList.update$(u_i)$
End While
```

Clustering Based Partitioning

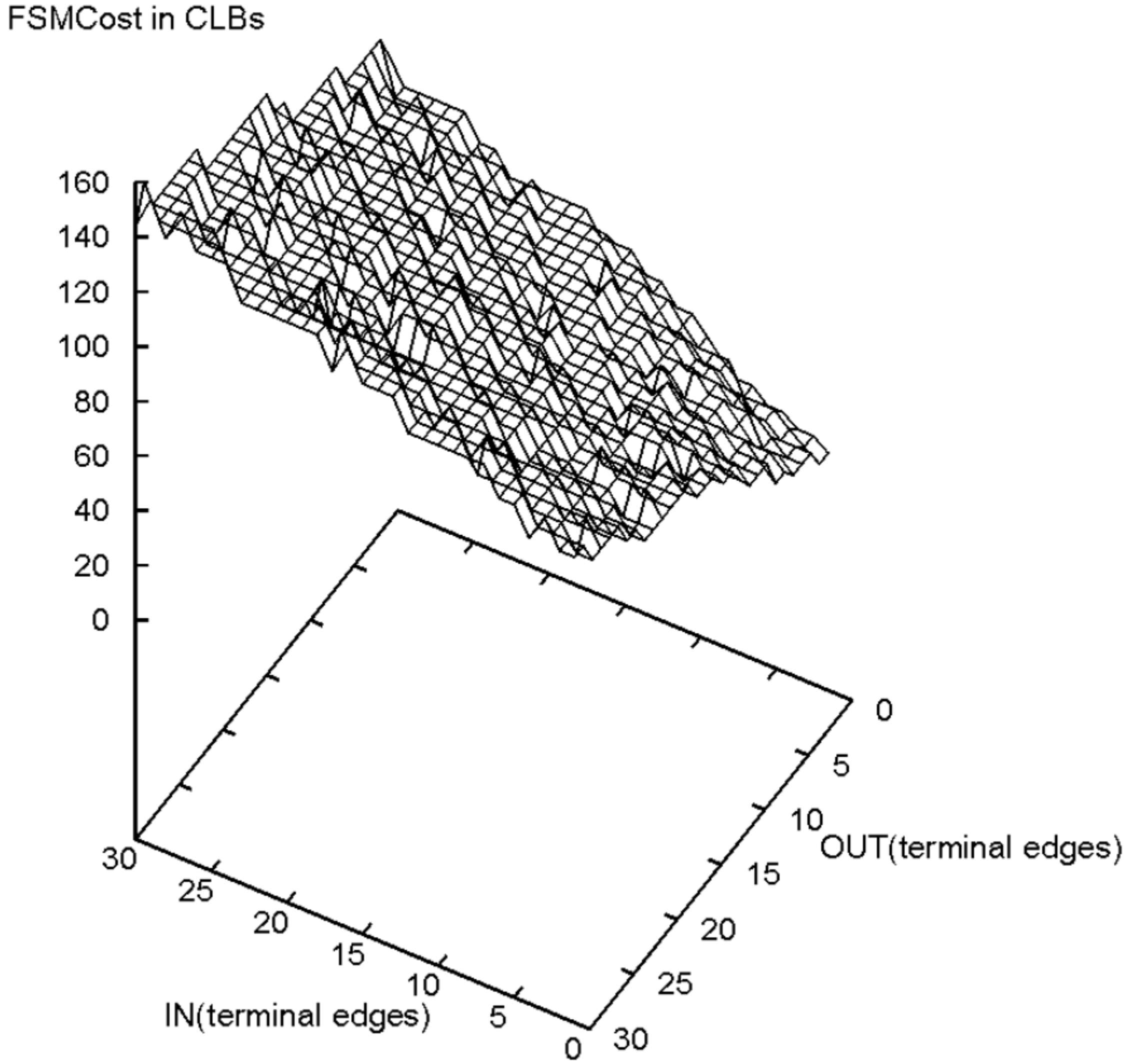
- ReadyList: nodes ready to be executed
- ReadyList.update(): adds new nodes that are ready to execute in front of ReadyList
  - Tends to decrease terminal edges
- Algorithm complexity is O(|*V*| + |*E*|)

Data Controller Synthesis

- FSM of data controller

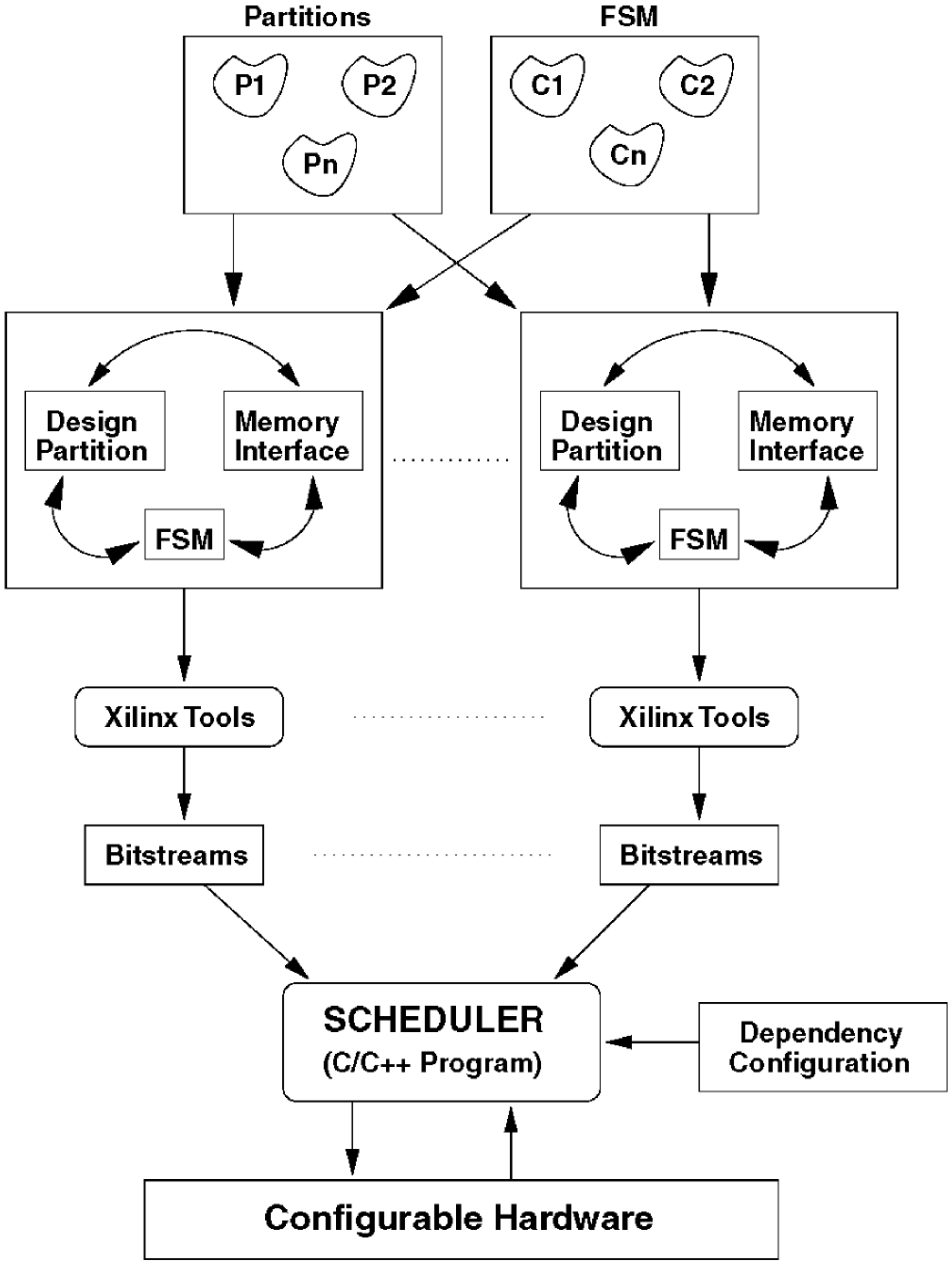


FSMCost Calculation



DFG Scheduling

- Scheduling of temporal partitions must satisfy
  - Precedence relation between partitions
  - Data dependencies among partitions



Partitions

P1

P2

Pn

FSM

C1

C2

Cn

Design Partition

Memory Interface

FSM

Design Partition

Memory Interface

FSM

Xilinx Tools

Xilinx Tools

Bitstreams

Bitstreams

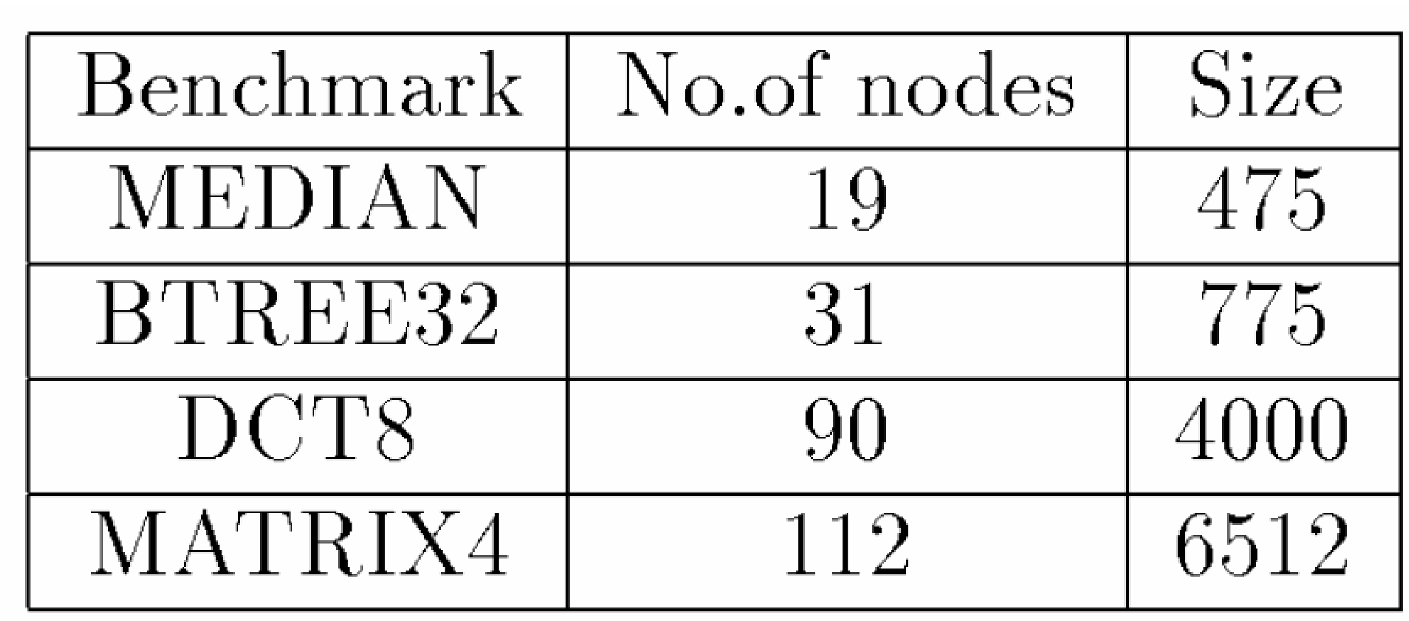
SCHEDULER (C/C++ Program)

Dependency Configuration

Configurable Hardware

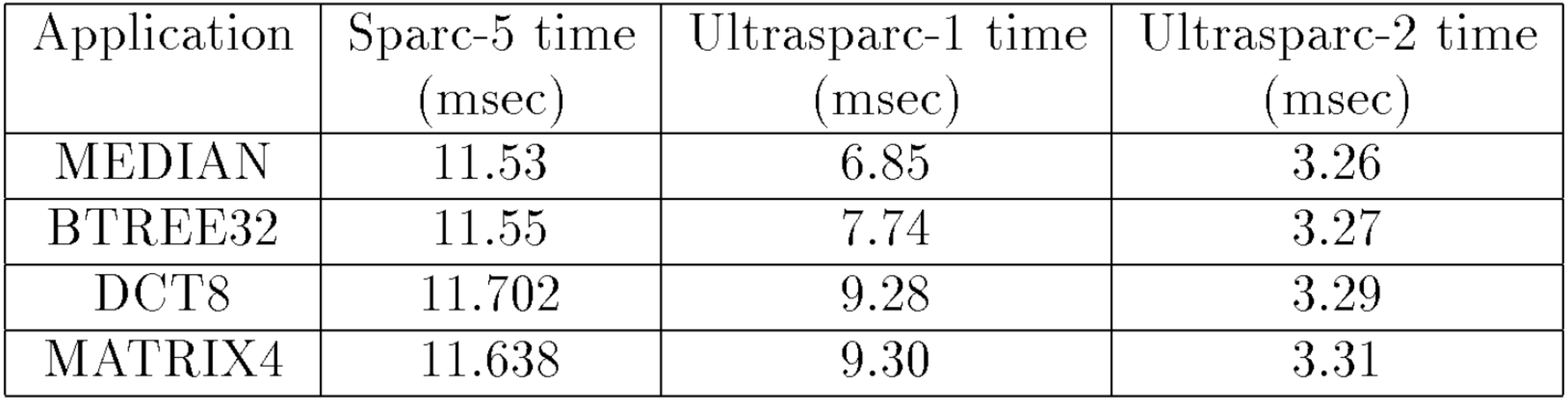
Application Examples

- Size of circuits in XC4000 CLBs



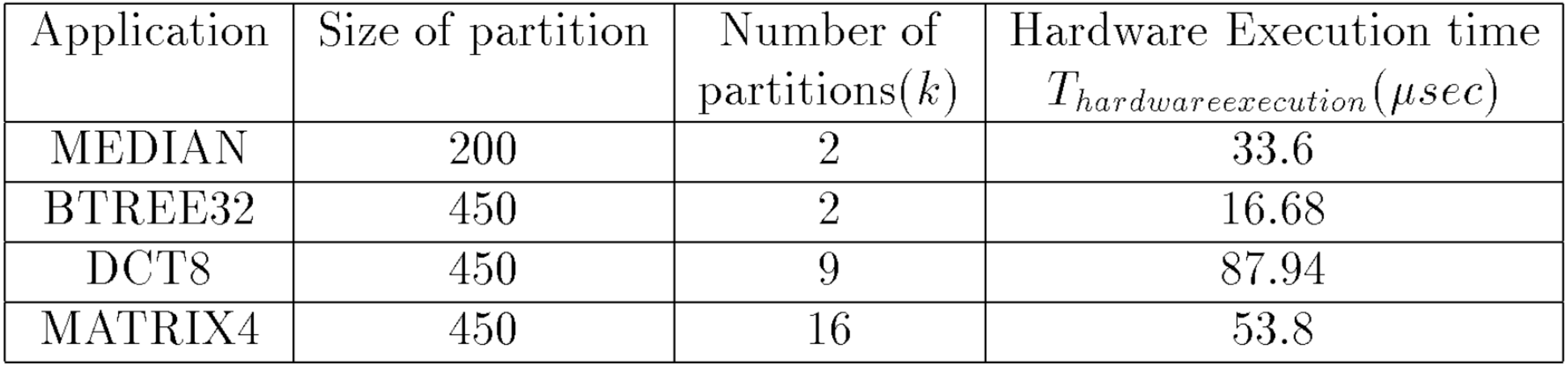
Benchmark	No.of nodes	Size
MEDIAN	19	475
BTREE32	31	775
DCT8	90	4000
MATRIX4	112	6512

SW Execution Times



Application	Sparc-5 time (msec)	Ultrasparc-1 time (msec)	Ultrasparc-2 time (msec)
MEDIAN	11.53	6.85	3.26
BTREE32	11.55	7.74	3.27
DCT8	11.702	9.28	3.29
MATRIX4	11.638	9.30	3.31

Reconfigurable HW Execution Times

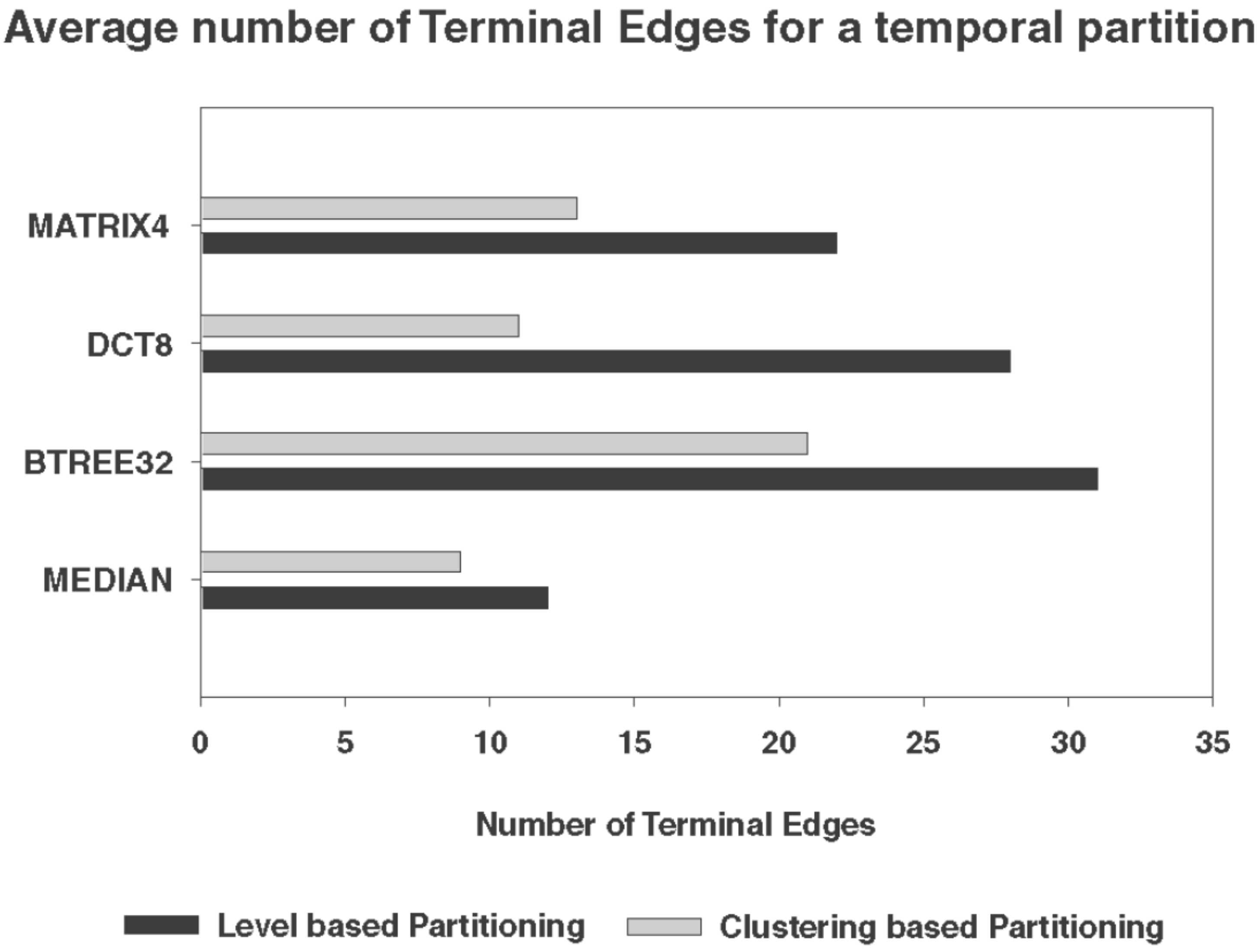


Application	Size of partition	Number of partitions(k)	Hardware Execution time $T_{hardwareexecution}(\mu sec)$
MEDIAN	200	2	33.6
BTREE32	450	2	16.68
DCT8	450	9	87.94
MATRIX4	450	16	53.8

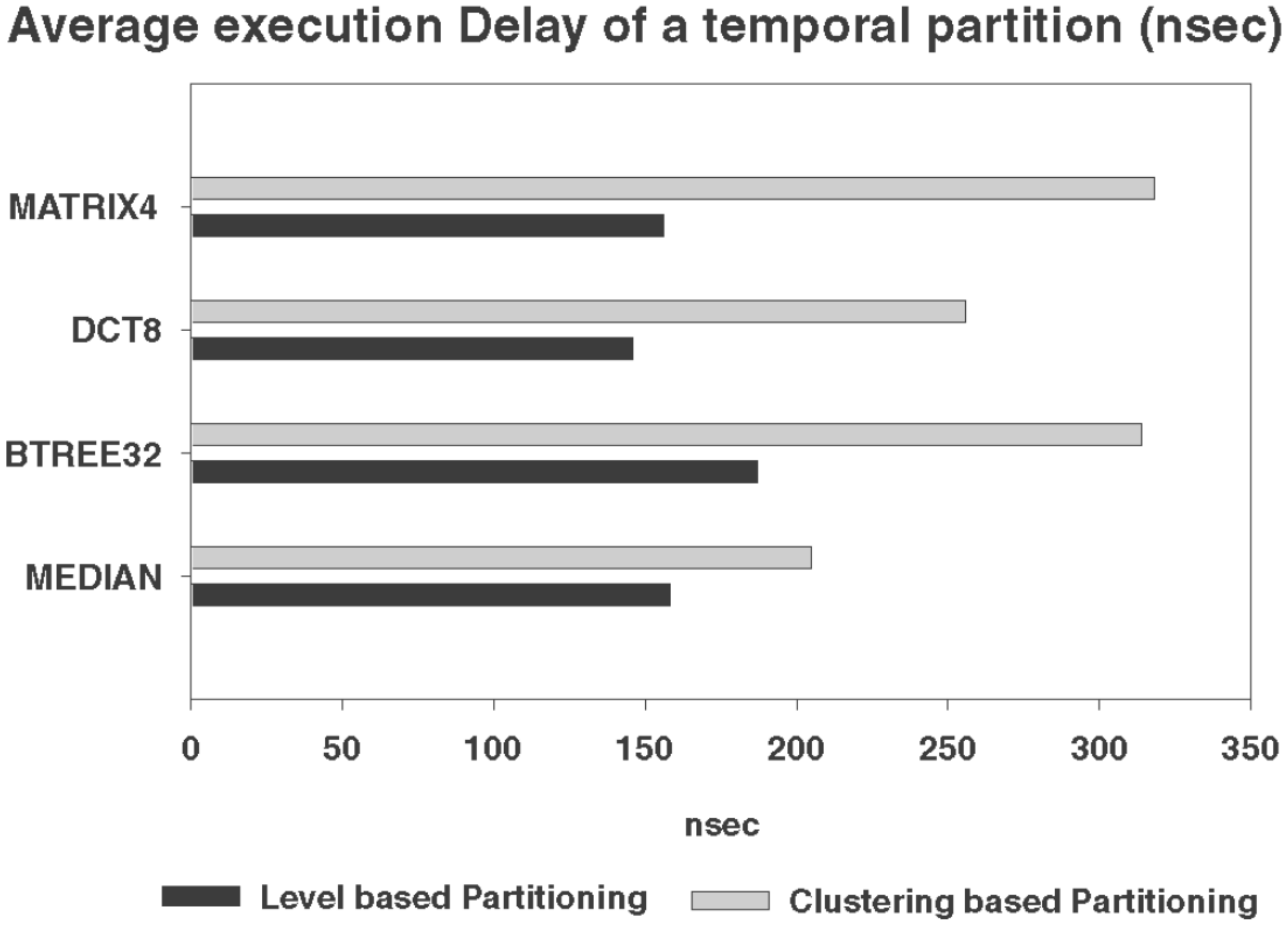
- Segment reconfiguration time: 242 ms

$$T_{exec} = (k * 242msec) + T_{hardwareexecution}$$

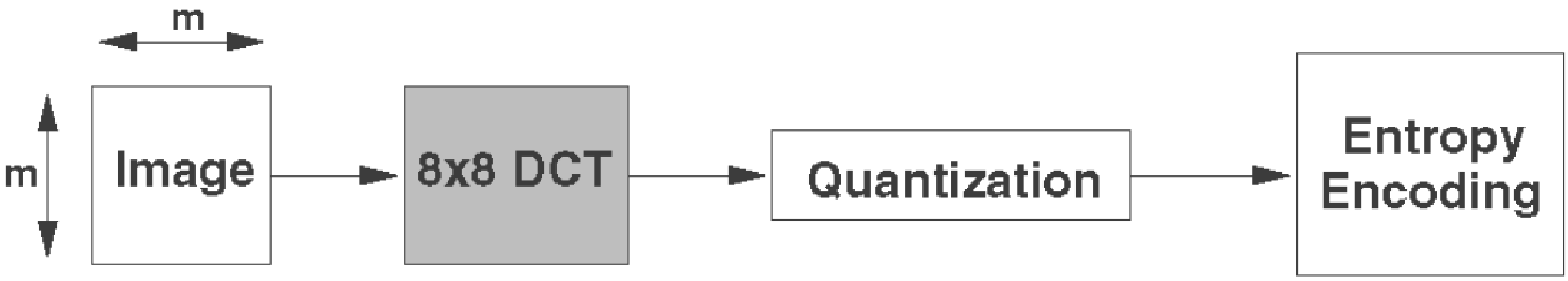
Comparison of Partitioning Algorithms



Comparison of Partitioning Algorithms



Discrete Cosine Transform



$$T_{soft-DCT} = \left\lceil \frac{m}{8} \right\rceil \times \left\lceil \frac{m}{8} \right\rceil \times T_{soft-DCT8},$$

$$T_{hard-DCT} = \sum_{i=1}^{k} \left\lceil \frac{m}{8} \right\rceil \times \left\lceil \frac{m}{8} \right\rceil \times T^{i}_{hard-DCT8} + k \cdot T_{reconfig},$$

Discrete Cosine Transform

- For *m* as small as 128,
  - Hardware outperforms software
  - Reconfiguration overheads completely absorbed
- For large images
  - Temporally partitioned hardware performance far exceeds software performance

Spatial Parallelism

- Replicating hardware modules so as to accelerate execution (data processing)
- Example
  - Two adders/multipliers instead of one
  - Two encoders/decoders instead of one
- Needs
  - Data distribution before parallel execution
  - Data integration after parallel execution

Pipelining

- Replicating modules, however due to data dependence, the modules are pipelined
- Increases data throughput
  - No pipelining: One data per iteration
  - With pipelining: One data per pipeline cycle
- Needs careful functional and timing designs

Template Specialization

- Instead of two variable operands, often one input operand is a constant
  - No need of full function implementation
  - Can use table lookup, shifter register, etc.
  - Example
    - Multiply by 4 or by 7
- Can decrease both resource usage and computation time

Outline

- Reconfigurable vs. Conventional Hardware
- Hardware Preemption and Relocation
- Area-Time Tradeoff Techniques
- Communication Architectures

Reconfigurable Communication Architectures

- Three types of communication architectures
  - Fixed module communication
    - A bus or NoC connecting all fixed modules
  - Dynamic module communication
    - A bus or NoC connecting all dynamic modules
    - Needs bus or slice macros (from Xilinx)
  - Fixed-Dynamic module communication
    - A bridge connecting fixed part and dynamic part
      - Example: OPB Dock
      - Needs bus or slice macros (from Xilinx)

Fixed Module Communication

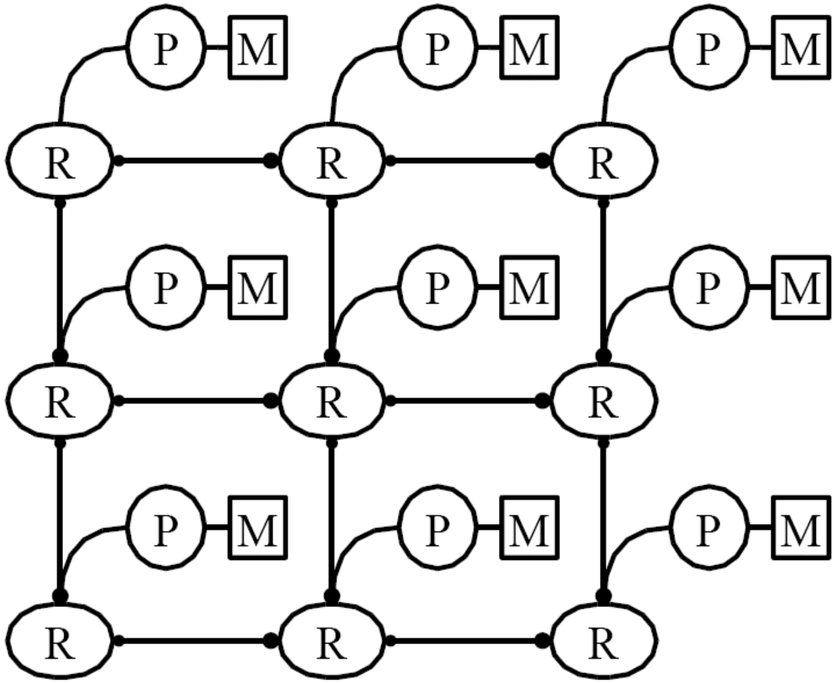
- A standard bus
  - – ARM AMBA
  - – IBM CoreConnect
- A standard NoC
  - – Topology: Mesh, torus, ring, fat tree, …
  - – Link: circuit switched, packet switched
  - – Switch mode: store-and-forward, virtual cut-through, wormhole

Dynamic Module Communication

- Static NoC [20], [22]
  - Fixed network
  - Modules can be placed only in slots connected to the fixed network
  - Examples: 2D Torus, Artemis (2D Mesh)
- Dynamic NoC [21], [26]
  - Dynamically changing network
    - Parts of the network can be included inside a module when it is placed [21]
    - QoS parameters, error detection/correction, fault isolation [26]

2D Mesh

- An array of routers interconnecting an array of processors [20]

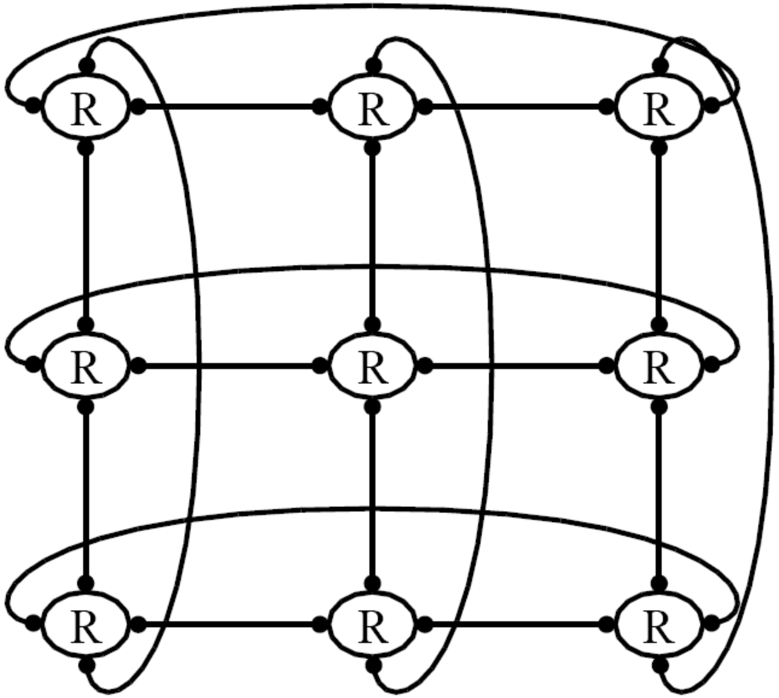


Routing:

N, E, W, S

2D Torus

- Static NoC
- Row and columns connected in rings



Routing:

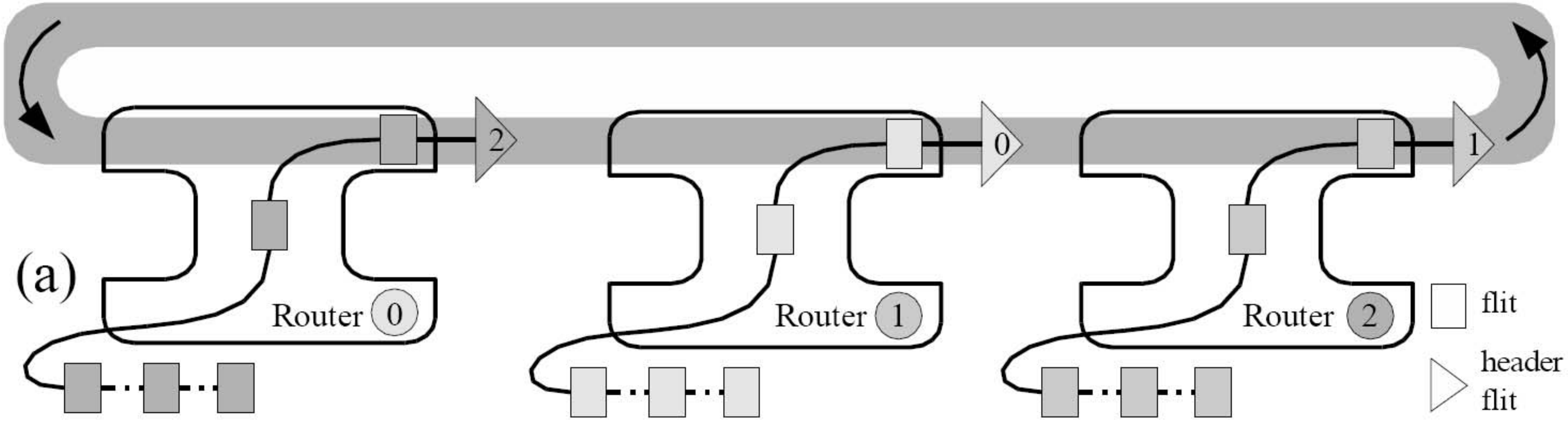
E, S

Wormhole Routing Algorithm

- Blocking, hop-based, deterministic routing algorithm
- Pipeline through the network
- Message is broken into flits (flow control units)
  - Flits: routing information and data message
    - First : X direction (X header flit)
    - Second : Y direction (Y header flit)

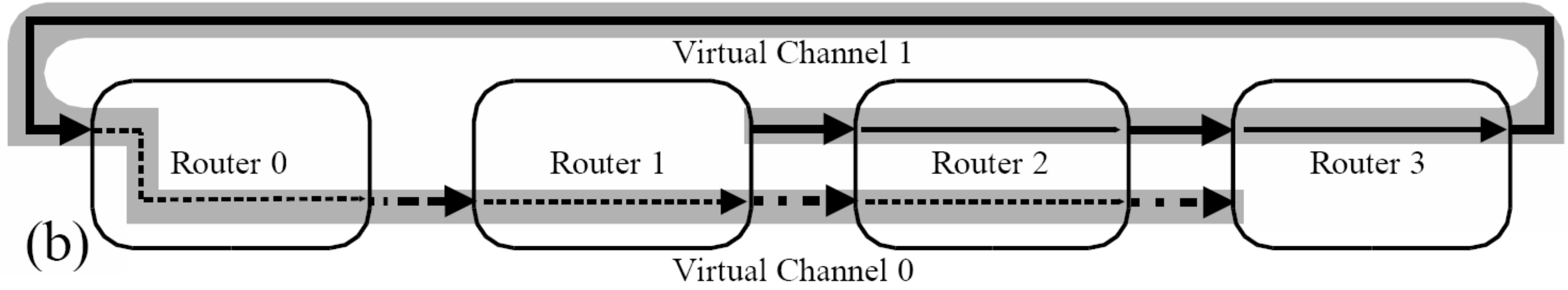
Wormhole Routing Algorithm

- Deadlock
  - All queues are full
  - No message can advance toward



Wormhole Routing Algorithm

- Avoid Deadlock
  - Two Virtual Channels (VC)
  - Router0 only sends messages on VC_0
  - The others may initiate messages only on VC_1



2D Torus Implementations

- Platform
  - Virtex XCV800, Virtex XC2V6000
  - Compaq iPaq PDA: SA-1100 (206 MHz), RT-Linux
- Routing
  - Wormhole
- Transfer rate between routers
  - 77.6 MB/s at 40 MHz (38.8 MB/s per VC)

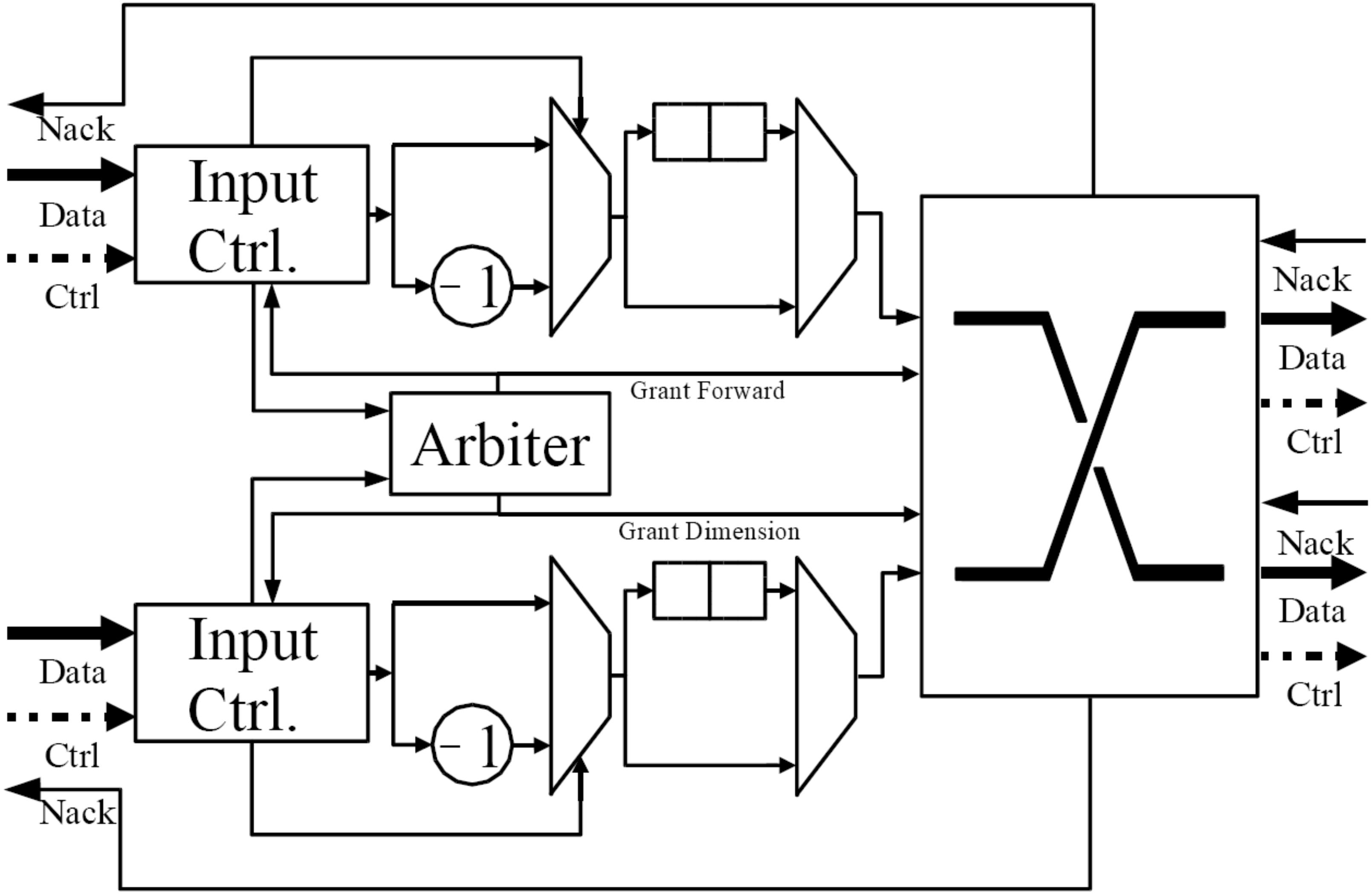
2D Torus Implementations

- Power Overhead
  - +15% power consumption for 4x4 folded torus compared to mesh
- Area Overhead
  - Virtex XCV800: 35%
  - Virtex XC2V6000: 9.5%
- Fully pipelined, 2 cycles to transmit one 16-bit flit on a given virtual channel

1D Router for 2D Torus

- 2 I/O channels
  - – 16-bit data path
  - – 3-bit control signals
    - 2 bits: indicate message
    - 1 bit: back-pressure (nack): to block message entry into a busy router or interface
  - – Time-multiplexed (interleaved one cycle each)
    - 2 cycles for a flit to leave a router

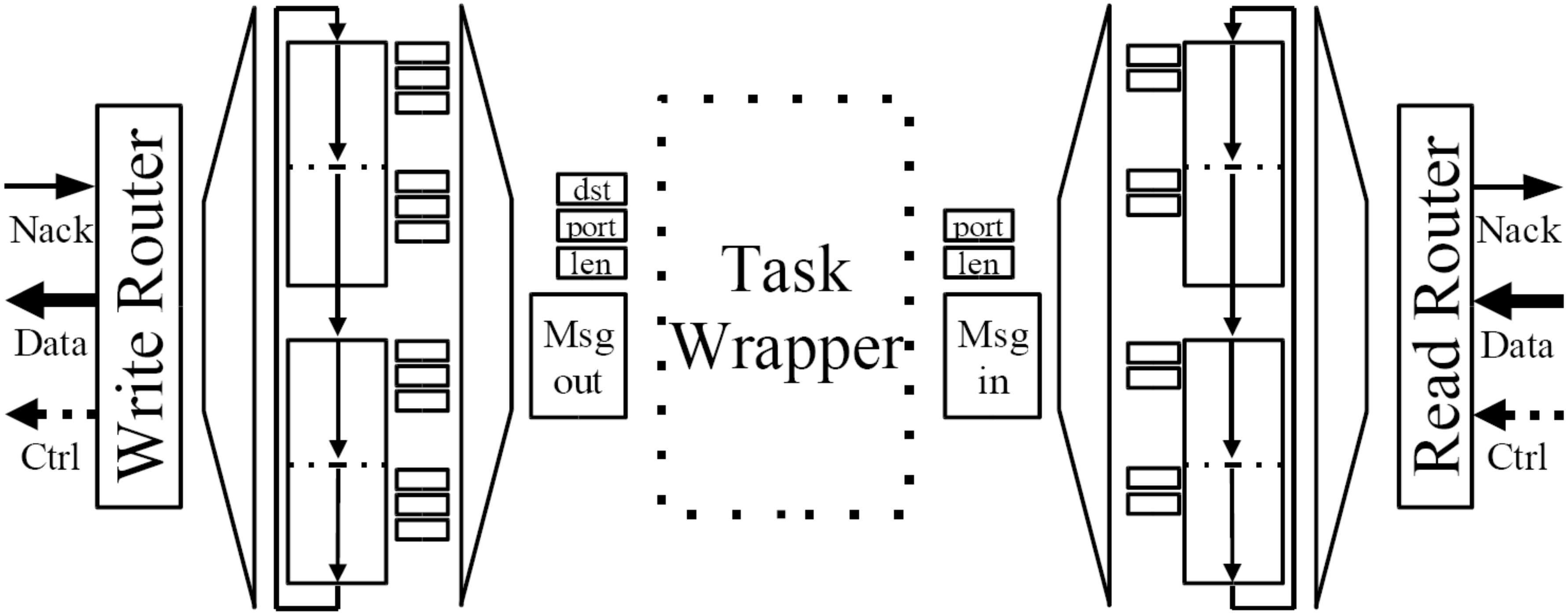
1D Router for 2D Torus



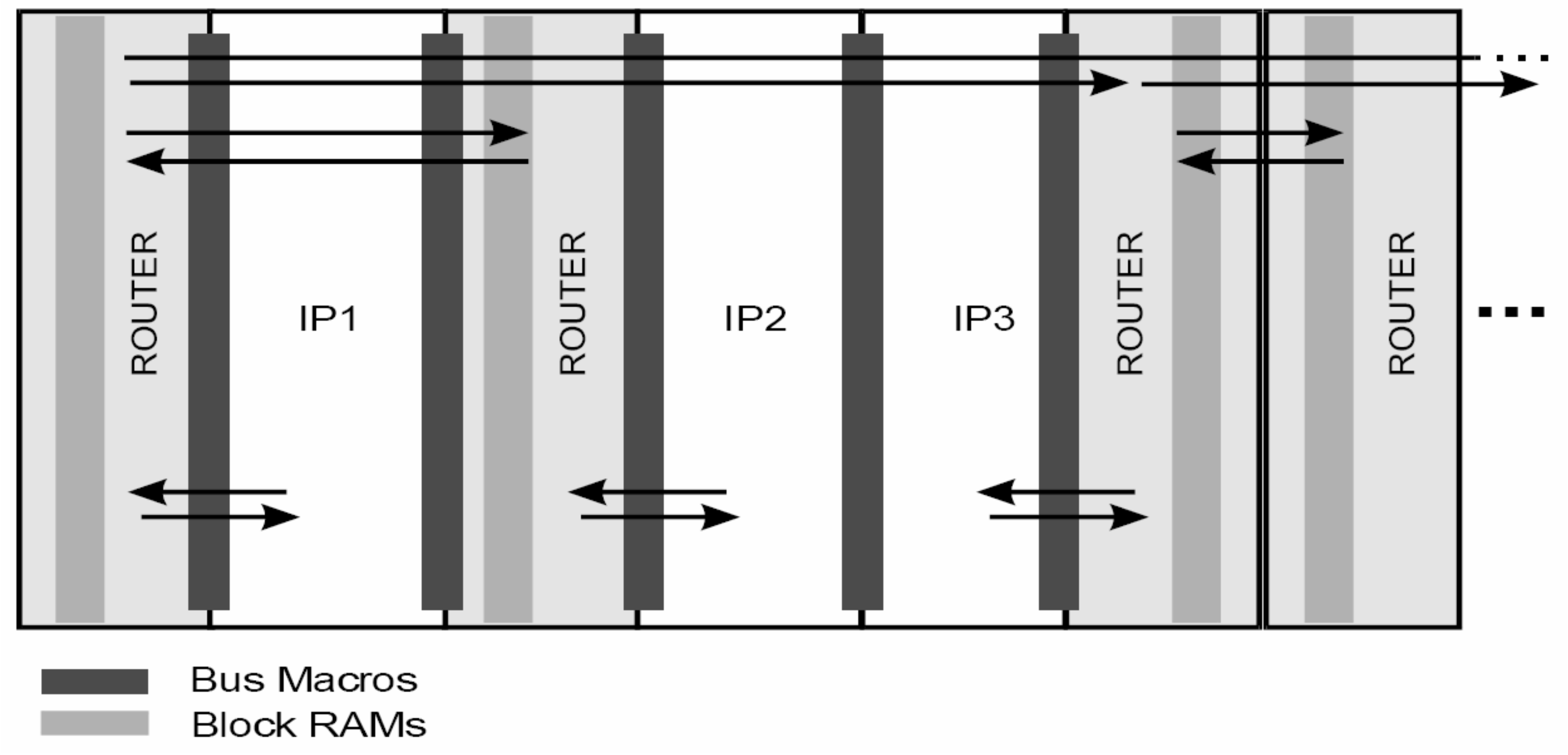
Interface between Task and Router

- Dual-port RAMs are used as message buffers
- Net-cell
  - High-level communication by routing tables
  - Logical address + port number
  - Routing tables transform a destination logical address into number of X and Y hops
    - Updated by RT-OS to match position of IP on network
  - Buffers 2 input and 2 output messages on Virtex I and 8 input and 8 output messages on Virtex II
    - Circular linked-list
- Same fixed interface for all tasks
  - Message-in, message-out, BRAMs, Multipliers, …

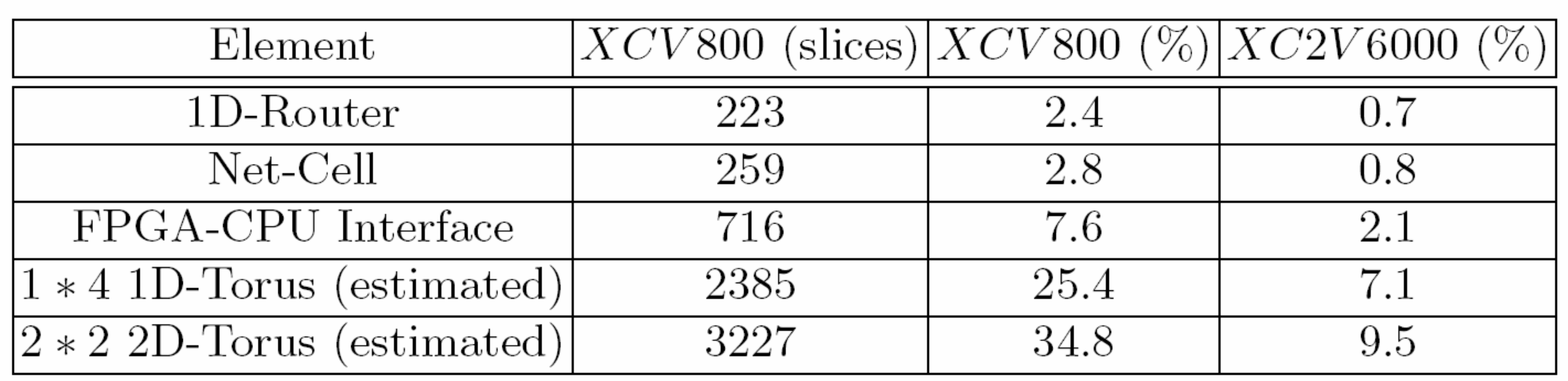
Task-Router Interface



Placement of Routers and IPs



Hardware Overheads

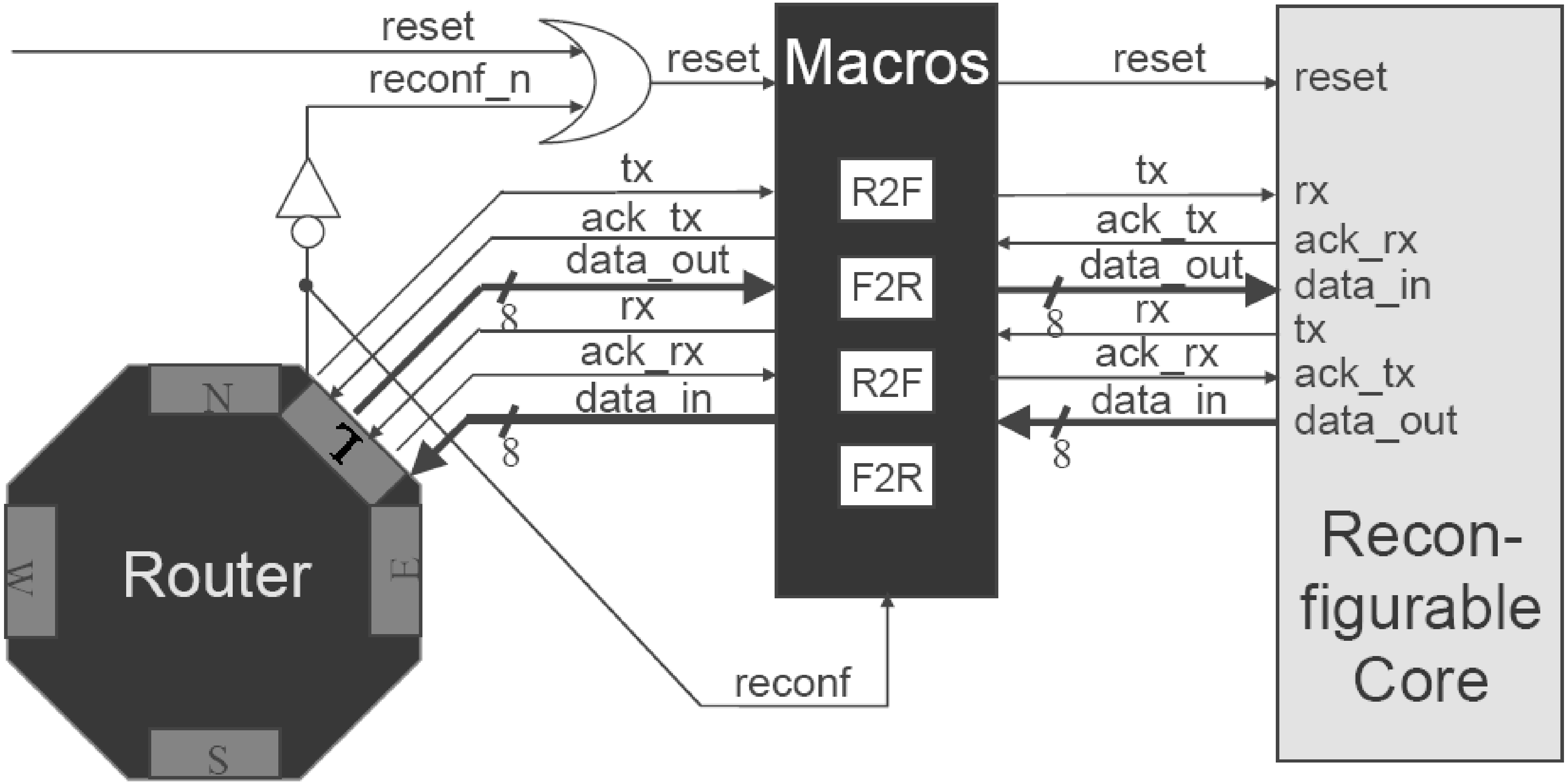


Element	$XCV800$ (slices)	$XCV800$ (%)	$XC2V6000$ (%)
1D-Router	223	2.4	0.7
Net-Cell	259	2.8	0.8
FPGA-CPU Interface	716	7.6	2.1
$1 * 4$ 1D-Torus (estimated)	2385	25.4	7.1
$2 * 2$ 2D-Torus (estimated)	3227	34.8	9.5

Artemis

- Static NoC [22]
- 2D Mesh
  - Based on the HERMES [23]
- Special macros for core-router interface
  - R2F: Reconfigurable to Fixed
  - F2R: Fixed to Reconfigurable

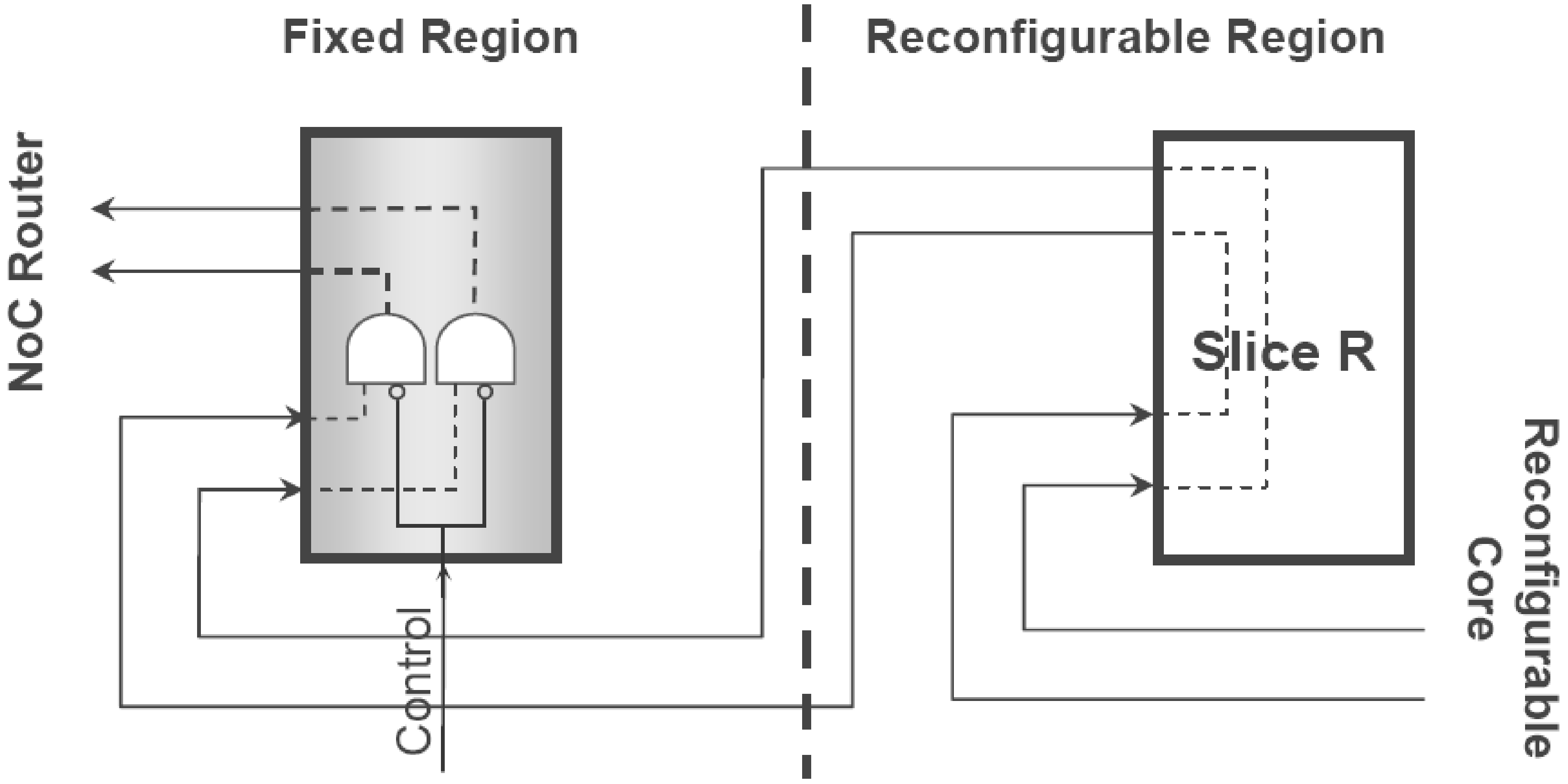
Artemis Core-Router Interface



R2F Macro

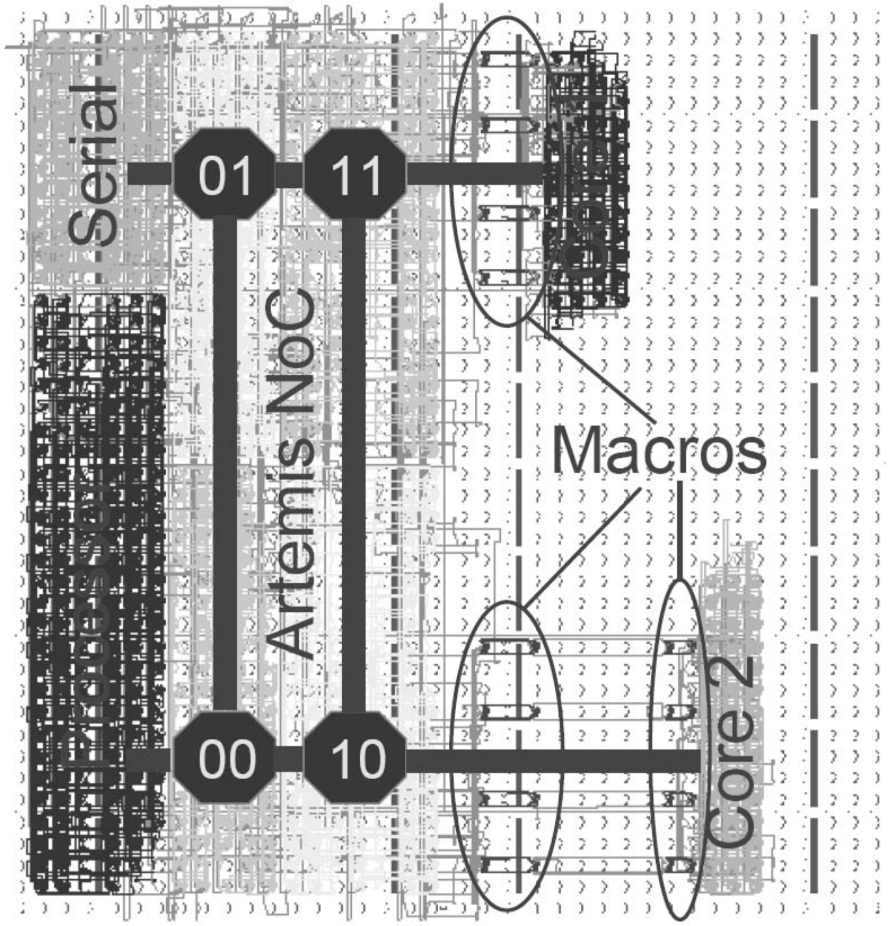
- No need of Xilinx bus macros
- Designed with FPGA slices
  - Reduces number of routing problems
  - Wider interfaces between regions
- 8 bits: one Virtex-II Pro CLB (4 slices)
- Right side: feed through
- Left side: 2 AND gates

R2F Macro

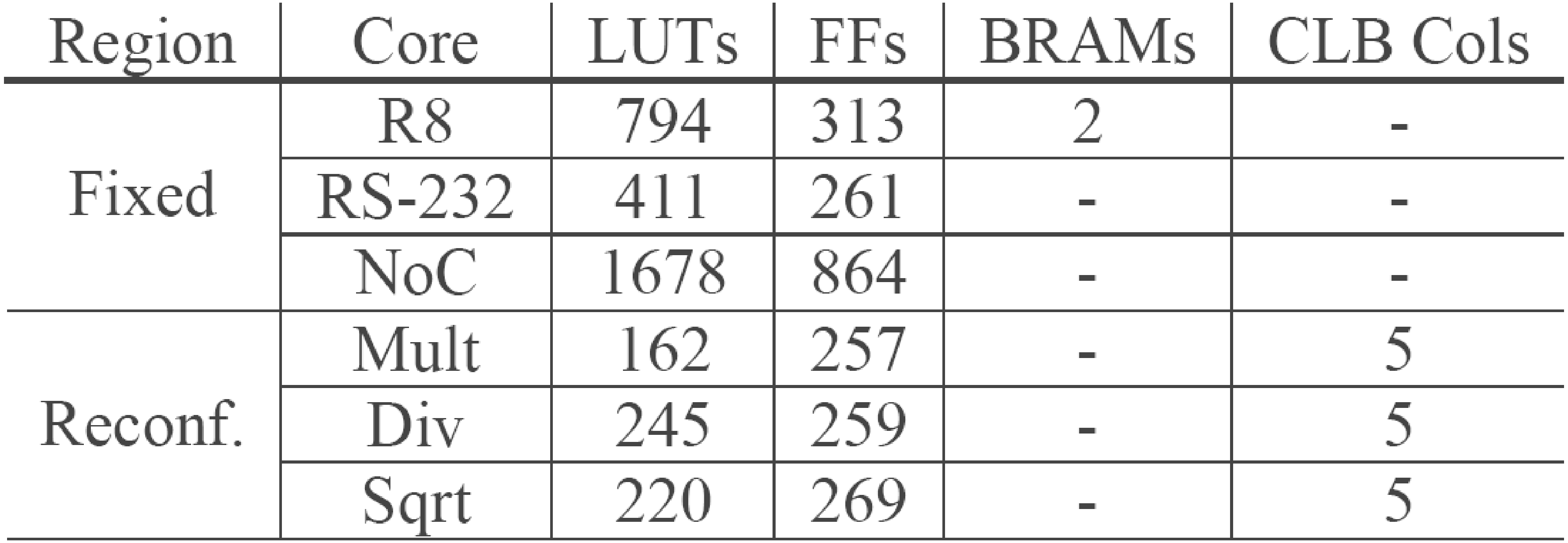

Fixed Region
Reconfigurable Region
NoC Router
Slice R
Control
Reconfigurable Core

Artemis NoC

- 4 cores
  - – R8 processor
  - – RS-232
  - – 2 reconfigurable cores
    - Mult, div, sqrt
    - 9.17% of full bitstream



Cores in Artemis Case Study



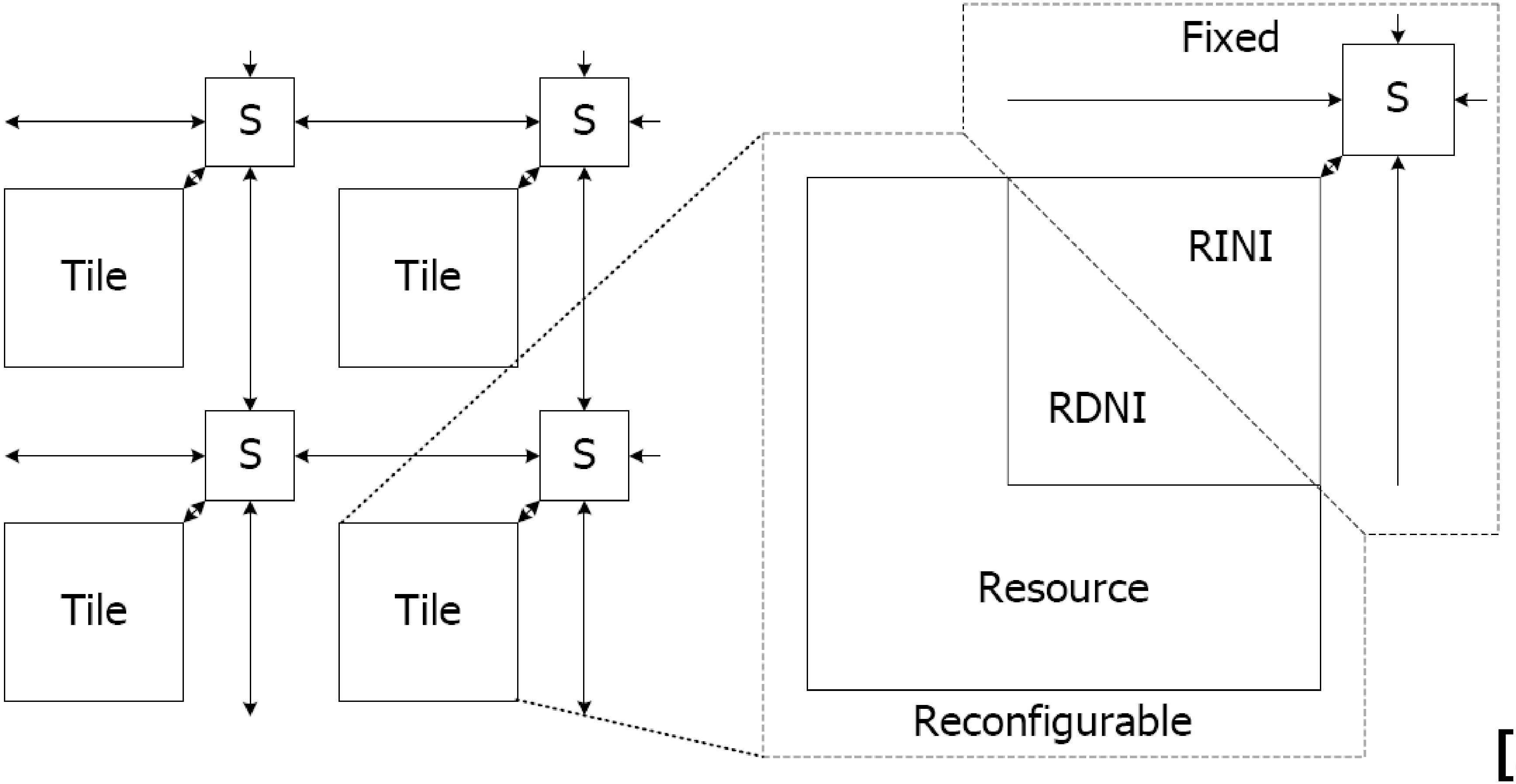
Region	Core	LUTs	FFs	BRAMs	CLB Cols
Fixed	R8	794	313	2	-
	RS-232	411	261	-	-
	NoC	1678	864	-	-
Reconf.	Mult	162	257	-	5
	Div	245	259	-	5
	Sqrt	220	269	-	5

NoC and Protocol Stack

- Tile [25]
  - A dynamically reconfigurable homogeneous and distinct part of logic
  - Connects to switch fabric via Reconfigurable Network Interface (RNI) with two parts:
    - Fixed: Resource Independent Network Interface (RINI)
    - Reconfigurable: Resource Dependent Network Interface (RDNI)

RNI = RINI + RDNI

Reconfigurable Network Interface



[25]

Protocol Stack

[25]

Similar to ISO/OSI model and TCP/IP stack

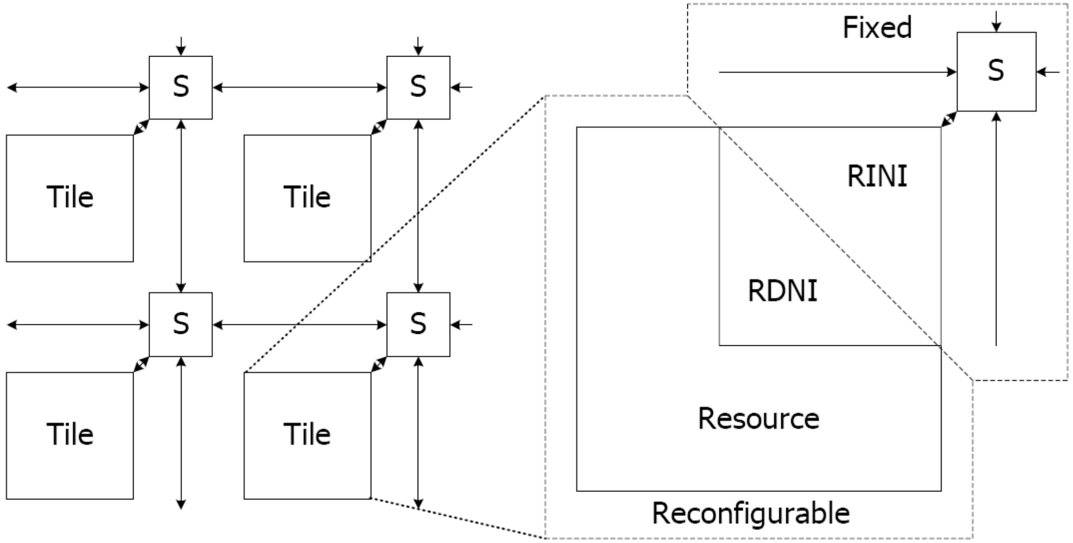
Layer	Protocols	Implementation
Appl. Layer	User Application Prot.	Resource
Transp. Layer	User Transport Prot.	RDNI

Reconfigurable

Fixed

Network Layer	NOC_NL	RINI
Datalink Layer	NOC_DL	Switch
Physical Layer	NOC_PH	Wires

NoC and Protocol Stack



Layer	Protocols	Implementation
Appl. Layer	User Application Prot.	Resource
Transp. Layer	User Transport Prot.	RDNI
Reconfigurable		
Fixed		
Network Layer	NOC_NL	RINI
Datalink Layer	NOC_DL	Switch
Physical Layer	NOC_PH	Wires

[25]

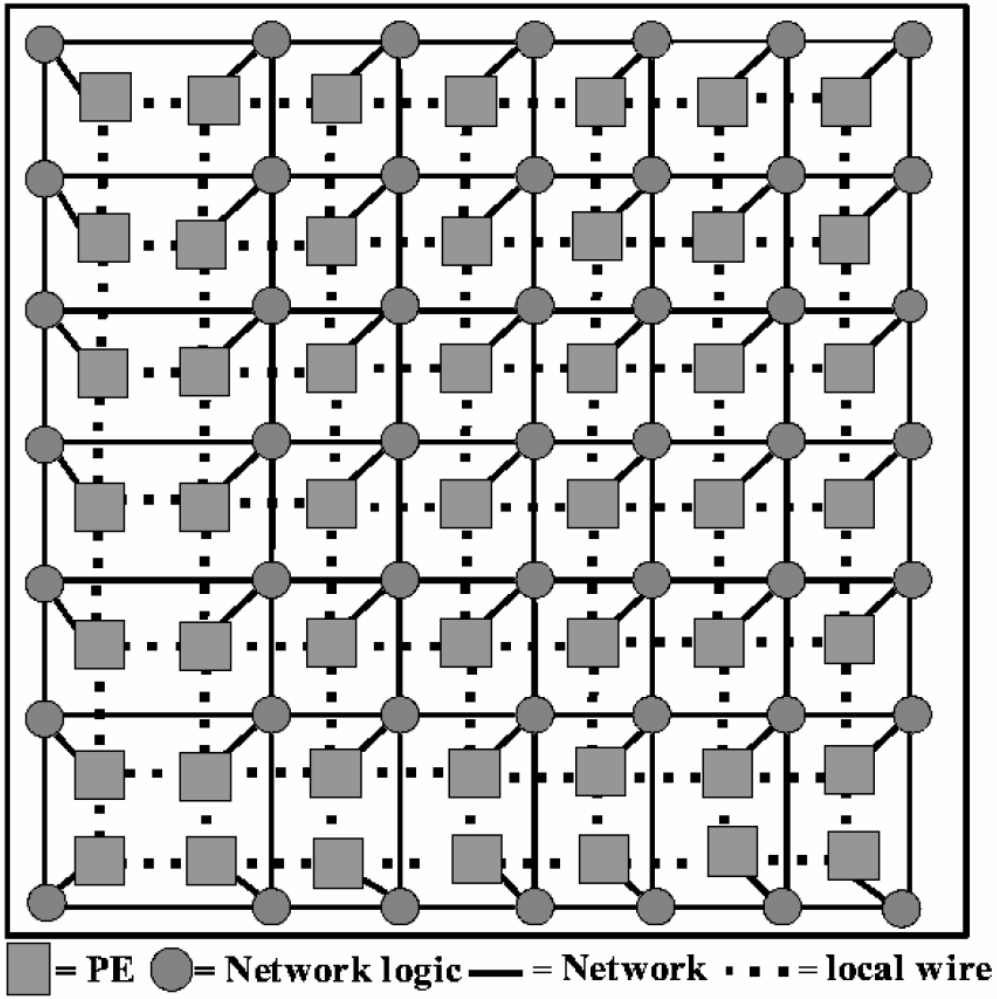
Dynamic NoC

- Two kinds of dynamic NoC
  - Dynamically Changing Structure [21]
    - Parts of an NoC are changed dynamically, used for PE logic
  - Dynamically Changing Behavior [26]
    - QoS parameters such routing, switching, packet size are dynamically changed
    - Error detection and correction
    - Faulty node isolation

NoC with Dynamically Changing Structure

- Task
  - Rectangular box encapsulating a circuit implemented with resources in a given area
- Network access
  - Using one network element on boundary
    - Assume network element attached to upper right PE of component
    - Module address = upper right network element

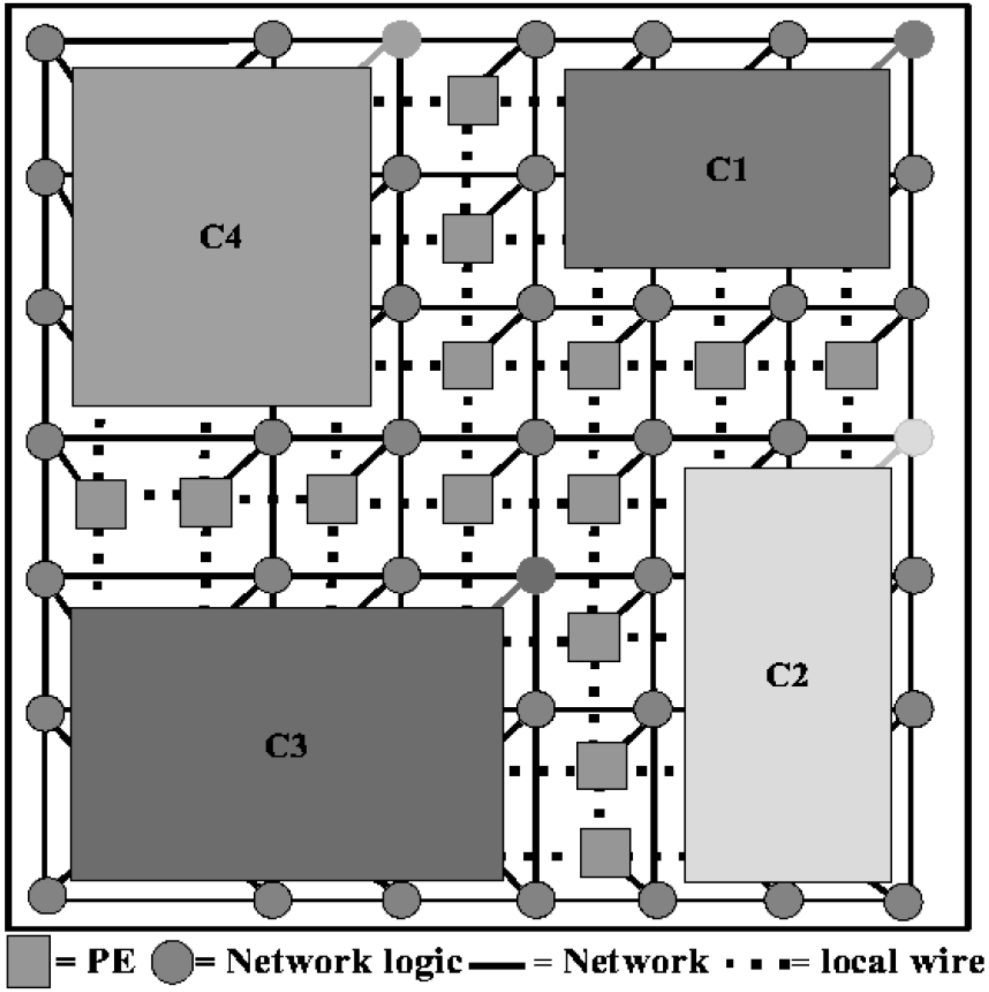
NoC with Dynamically Changing Structure



NoC with Dynamically Changing Structure

- Placement
  - – When placed on device, components hide part of the network, which is restored when they complete execution
  - – Hence, it is called a dynamic NoC [21]!
  - – Must maintain a strongly connected network
    - A path exists between each pair of network elements

Temporal Placement on Dynamic NoC



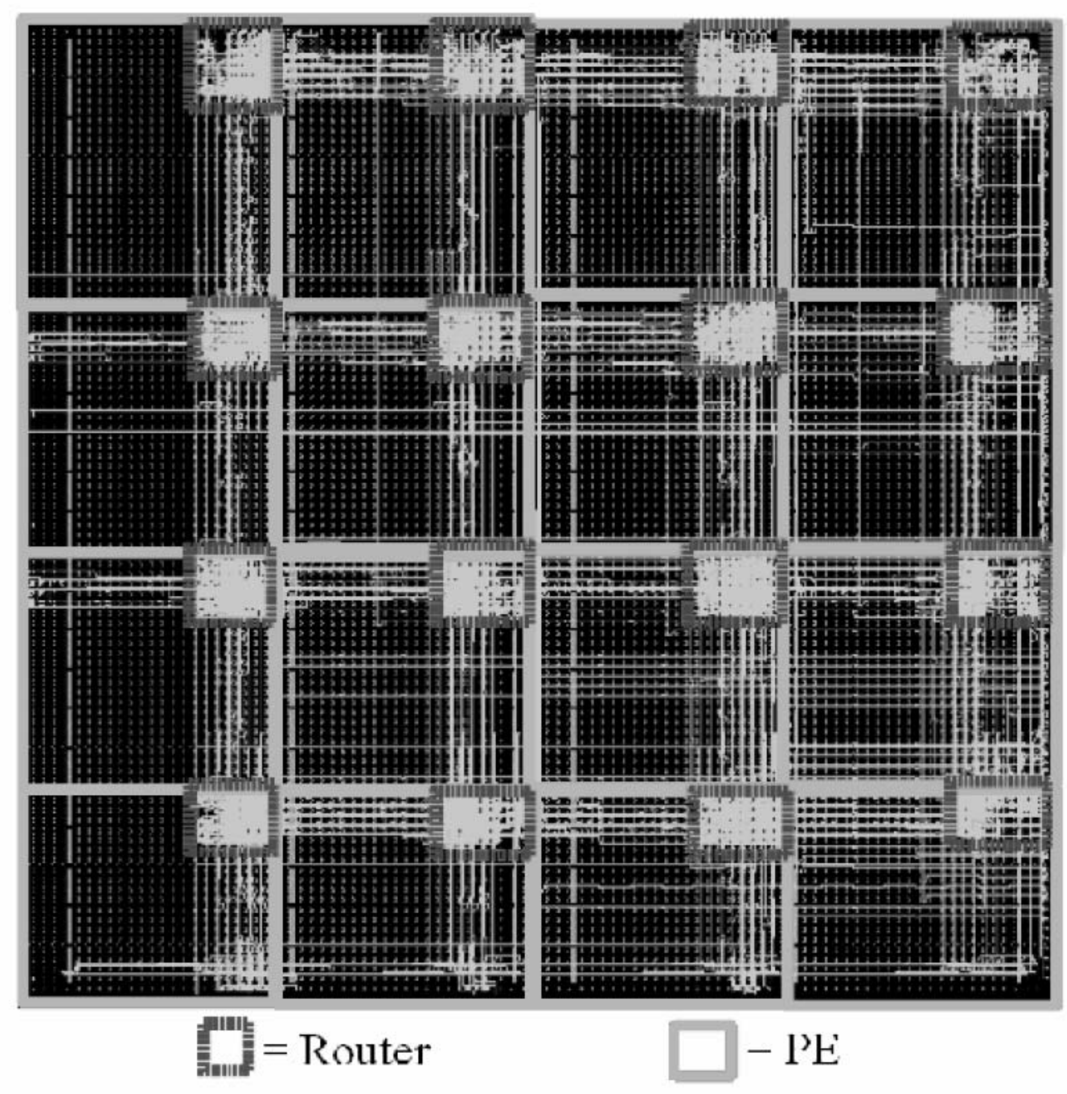
NoC with Dynamically Changing Structure

- Routing
  - Due to placement of tasks that cover the routers in its area, the routers are deactivated
    - Deactivation: by setting control signals
    - Reactivation: by resetting control signals
  - Before sending packet in a direction, a router must check if the router in that direction is activated
    - Deactivated: route in perpendicular direction

NoC with Dynamically Changing Structure

- Implementation
  - **FPGA**: Virtex II 6000
  - **NoC Topology**: Mesh network
  - **Network Size**: 4 x 4 = 16 routers
  - **Router connection**:
    - 32-bit bus,
    - 4 control lines,
    - six 32-bit FIFO buffers with depth 4

NoC with Dynamically Changing Structure



NoC with Dynamically Changing Structure

- Synthesis result: 7% device area
  - **Each router**
    - 0.5% device area
    - 2.553 ns latency with 391 MHz frequency
  - **Path Latency**
    - Max 6 routers on a path ➔
      50 MHz components can communicate without delay

WHY???

NoC with Dynamically Changing Behavior

- Network parameters such as routing paths, switching mode, packet size are usually fixed statically during network design
  - Cannot cope with network data bursts
  - Require large buffers for storing packets
  - Under/over utilization of network bandwidth by different processing elements
  - Wastage of resources

NoC with Dynamically Changing Behavior

- Dynamic NoC [26]
  - Reconfigures itself according to communication demand
    - Routing path, switching mode, packet size determined dynamically
    - Faulty nodes can be isolated dynamically
    - Data errors can be detected and corrected or retransmitted

Smart Network Stack

- Makes decision about packet size, switching and routing for data
- Writes this information in packet header
- Uses
  - For handling data bursts
  - For handling faulty nodes

Smart Network Stack

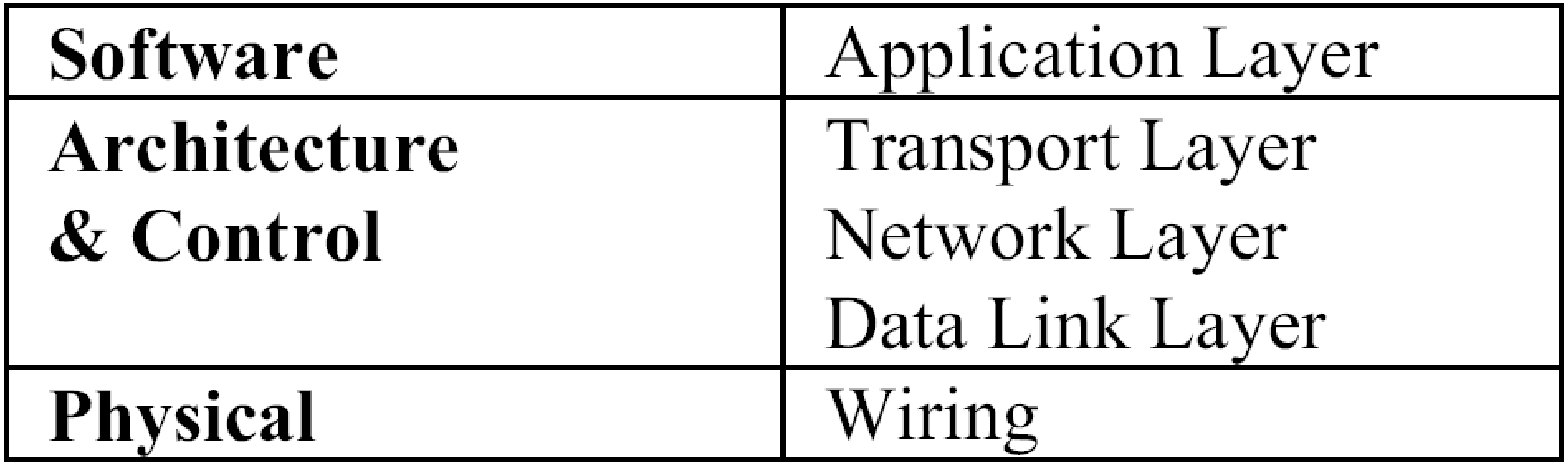
- PEs with high bandwidth
  - – **SNS Actions**
    - Increase packet size
    - Change packet switching to circuit switching
  - – **Routing Results**
    - Increased data throughput
    - Decreased switching power
    - Decreased timing delays

Smart Network Stack

- Faulty node
  - **Detection**
    - No response from the node
  - **Action**
    - Deactivate the router that leads to a local faulty node

Smart Network Stack

- Five Layers



Software	Application Layer
Architecture & Control	Transport Layer Network Layer Data Link Layer
Physical	Wiring

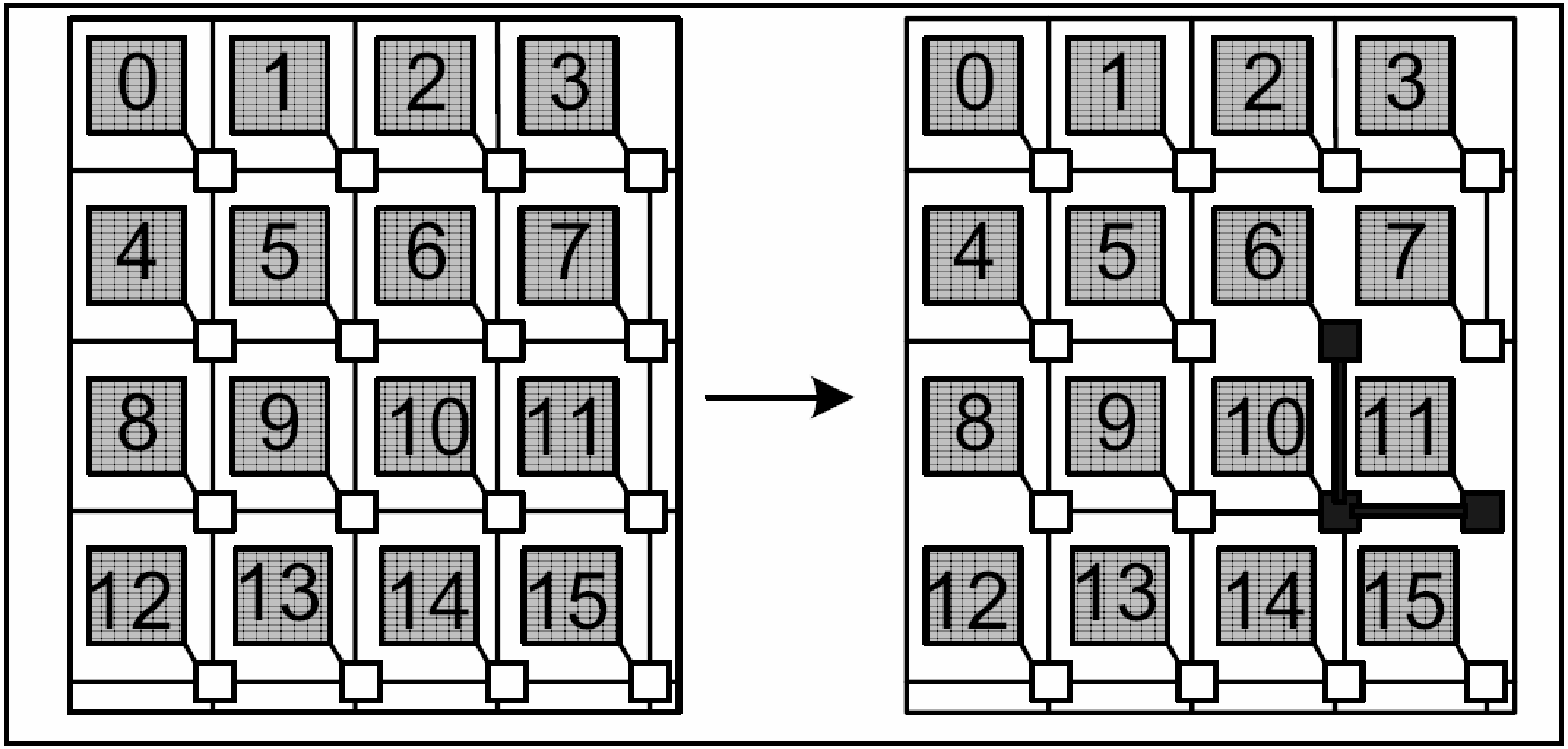
Smart Network Stack

- Application Layer
  - user interface to communication system, hides details
- Transport Layer
  - **Packetization of data**
    - Packet Header
    - Data Payload
  - **Packet Size**
    - Normal, moderate, heavy data transfer
    - Can be dynamically changed by transport layer of SNS

Smart Network Stack

- Network Layer
  - Deals with switching and routing of packetized data
  - Types of Switching
    - Packet Switching: for data sizes ≤ threshold
      - Wormhole routing
    - Circuit Switching: for data sizes > threshold
  - Features
    - Both types of switching exist concurrently in the same network
    - Circuit switched path is excluded from packet switching routes

Circuit Switching by SNS



Smart Network Stack

- Data Link Layer
  - Hides transmission errors in physical layer
    - Increases reliability upto a minimum level
  - 2 types of error detection and correction
    - Error detection with retransmission
    - Error detection with correction using information in packet
      - Requires encoder/decoder at channel's end
  - SNS uses error detection with retransmission
    - To keep silicon cost low
    - Uses checksum calculation and checking

Smart Network Stack

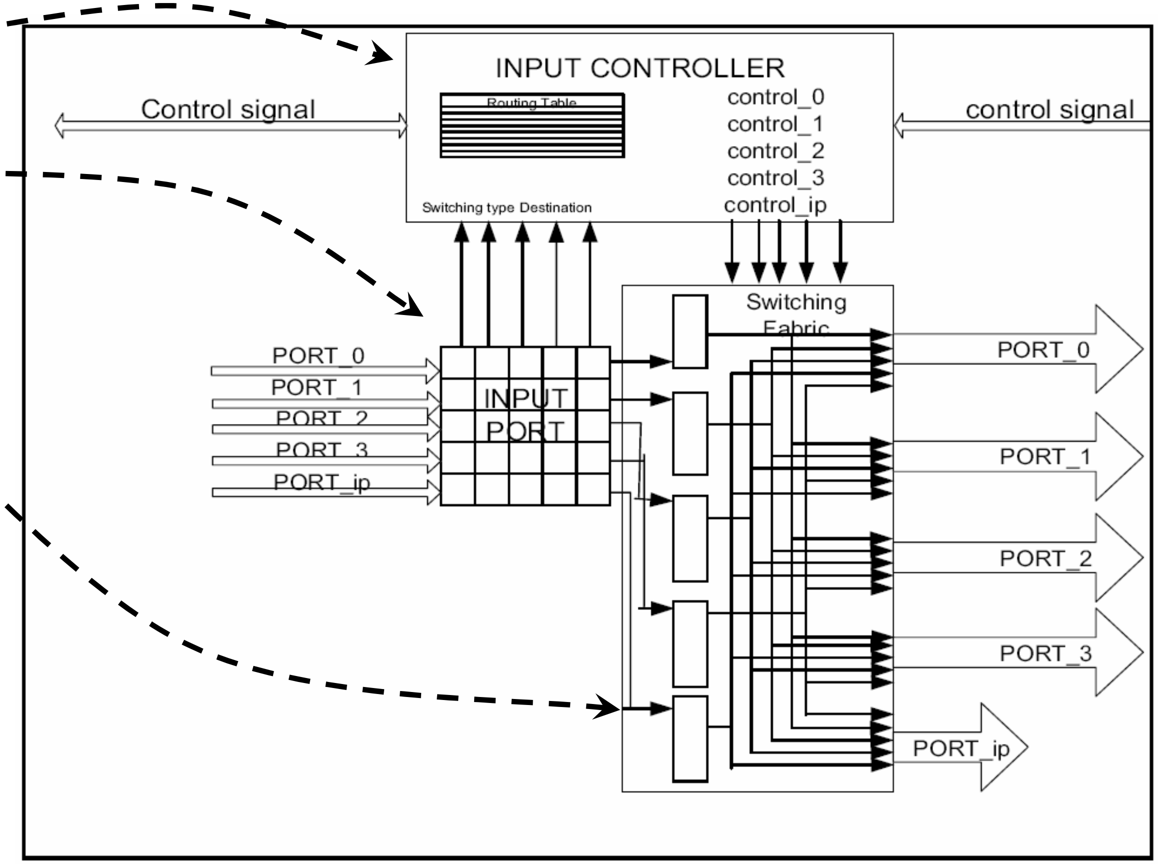
- Physical Layer
  - – FPGA has abundant wiring around each tile
  - – Can be used for the physical wiring of network-on-chip
  - – Use separate wires for
    - Data wires
    - Control wires

Router Design

- Objectives and Solutions
  - Low silicon cost
  - Keep internal buffers as small as possible
  - Prevent data queuing up in router buffers
    - Use control signals to update routing tables for adaptive routing
    - Packets know in advance of coming to routers what the congestion is like, and are thus routed to alternative paths to avoid having to queue up

Router Design

- 3 components in router
  - Input Controller
  - Input Port
  - Switching Logic



Router Design: Input Controller

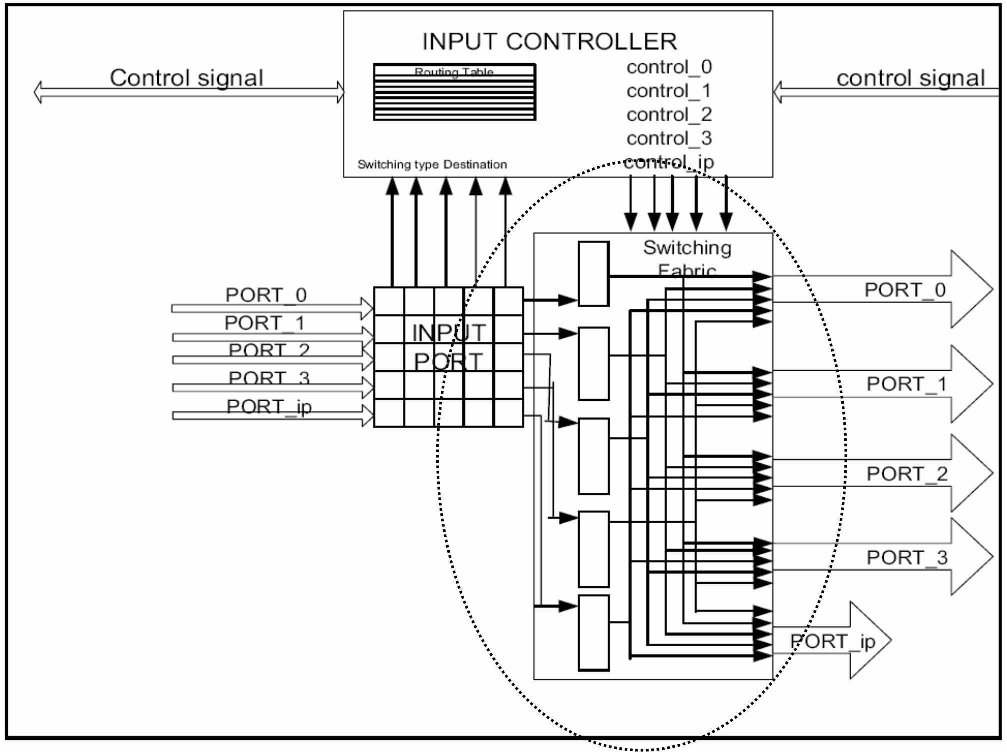
- Manages routing tables
- Inspects header and determines fate of arrived packets
- Neighboring input controllers are all connected
  - To update routing tables
- Clock speed: 5 times that of router
- Output port checking: round robin
- Node failure: Will be excluded from routing tables after no response is detected from a node

Router Design: Input Port

- Point of entry of incoming packets
- Buffer to store one packet for header inspection
- Information extraction from packet
  - Destination address
  - Type of switching

Router Design: Switching Logic

- Connects input ports to output ports depending on the instructions from input controller



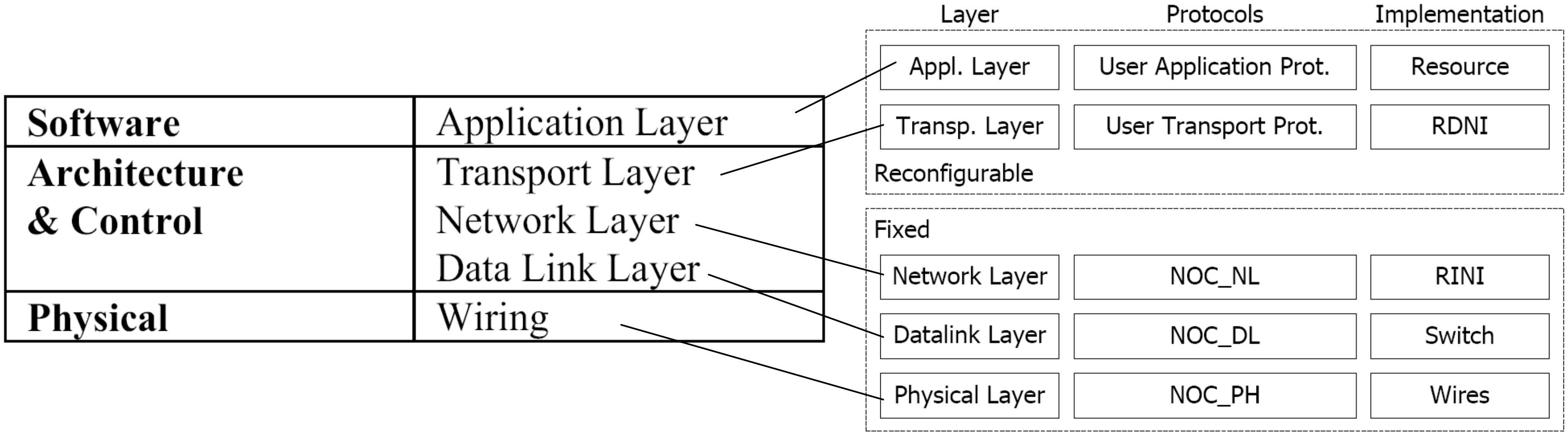
Router Design: Implementation

- Implemented in Verilog
- 32-bit link for each input port
- 27% lesser area overhead than [27]
  - Only one input controller instead of one for each port as in a normal NoC.

Summary (NoC Comparison)

Features	**2D-Torus [20]**	**Artemis [22]**	**DyNoC [21]**	**AdNoC [26]**
Structure	Static	Static	Dynamic	Static
Behavior	Static	Static	Dynamic	Dynamic
Topology	2D-Torus	Mesh	Mesh	2D-Torus
Routing	Wormhole	Packet	Packet	Wormhole
Switching	Packet	Packet	Packet	Circuit/Packet
Packet Size	Fixed	Fixed	Fixed	Changeable
Fault Tolerance	No	No	Yes	Yes
Error Handling	No	No	No	Yes
Size	4x4	2x2	4x4	4x4
Validation	Virtex XCV800, XC2V6000	Virtex II Pro	Virtex II 6000	NS-2 Simulation

Protocol Stack Comparison



Software	Application Layer
Architecture & Control	Transport Layer Network Layer Data Link Layer
Physical	Wiring

Layer	Protocols	Implementation
Appl. Layer	User Application Prot.	Resource
Transp. Layer	User Transport Prot.	RDNI
Reconfigurable		
Fixed		
Network Layer	NOC_NL	RINI
Datalink Layer	NOC_DL	Switch
Physical Layer	NOC_PH	Wires

Protocol Stack Comparison

Layer	RNI [25]	SNS [26]
Application	Hardware (R)	Software (R)
Transport	RDNI (R)	Data packetization (R)
Network	RINI (F)	Switching and routing (F)
Data Link	Switch (F)	Error correction (F)
Physical	Wire (F)	Wire (F)

RNI: Reconfigurable Network Interface

SNS: Smart Network Stack

Fixed-Dynamic Module Communication

- Needs special modules
  - Bus macros: tri-state buffers
  - Slice macros: specially designed logic blocks
- For example: from Xilinx
- Used for interface reconfigurable and static areas
- Can be constructed into a bridge such as OPB Dock, Gasket, Wrapper, Bus, …

References

1. H. Simmler, L. Levinson, and R. Manner, “Multitasking on FPGA Coprocessors,” 10th FPL, 2000.
2. H. Kalte and M. Porrmann, “Context Saving and Restoring for Multitasking in Reconfigurable Systems,” 15th FPL, 2005.
3. E. Horta and J. W. Lockwood, “PARBIT: A tool to transform Bitfiles to Implement Partial Reconfiguration of FPGAs,” Tech Report WUCS-10-13, Washington Univ., July 2001.
4. H. Kalte, G. Lee, M. Porrrmann, U. Ruckert, “REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems,” 12th RAW, April 2005.
5. M. Ullmann, M. Hubner, B. Grimm, J. Becker, “An FPGA Run-Time System for Dynamical On-Demand Reconfiguration,” 18th IPDPS, 2004 (also in IJES Vol. 1, No. 3/4, pp. 193-204, 2005)

References

6. C.-H. Huang, K.-J. Shih, C.-S. Lin, S.-S. Chang, and P.-A. Hsiung, “Dynamically Swappable Hardware Design in Partially Reconfigurable Systems,” ISCAS, May 2007.
7. J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, R. Lauwereins, “Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip,” DATE’2003.
8. T. Streichert, D. Koch, C. Haubelt, and J. Teich, “Modeling and Design of Fault-Tolerant and Self-Adaptive Reconfigurable Networked Embedded Systems,” EURASIP Journal of Embedded Systems, Vol. 2006 Article ID 42168, pp. 1-15. 2006.
9. S. Guccione, D. Levi, P. Sundararajan, “JBits: A Java-based Interface for Reconfigurable Computing,” 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD).
10. G. Brebner, “The Swappable Logic Unit: a Paradigm for Virtual Hardware,” IEEE Symposium on FPGAs for Custom Computing Machines, FCCM, 1997.

References

11. E. L. Horta, J. W. Lockwood, D. Parlour, “Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration,” DAC, pp. 343-348, 2002.
12. S. Trimberger, “Scheduling Designs into a Time-Multiplexed FPGA,” Procs. of the International Symposium on FPGAs, 1998.
13. D. Chang and M. Marek-Sadowska, “Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs,” Procs. of the International Symposium on FPGAs, 1998.
14. H. Liu and D. F. Wong, Circuit Partitioning for Dynamically Reconfigurable FPGAs, Procs. of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays, pp. 187-194, ACM Press, 1999.
15. E. Canto, J.M. Moreno, J. Cabestany, I. Lacadena, J.M. Insenser, “A Temporal Bipartitioning Algorithm for Dynamically Reconfigurable FPGAs,” IEEE Trans. On VLSI Systems, Vol. 9, No. 1, pp. 210-218, Feb. 2001.

References

16. M. Kaul, R. Vemuri, S. Govindarajan, I. Ouaiss, “An Automated Temporal Partitioning and Loop Fission Approach for FPGA based Reconfigurable Synthesis of DSP Applications,” DAC, pp. 616-622, 1999.
17. M. Kaul, R. Vemuri, “Temporal Partitioning Combined with Design Space Exploration for Latency Minimization of Run-Time Reconfigured Designs,” DATE, pp. 202-209, March 1999.
18. K.M. Gajjala Purna, D. Bhatia, “Temporal Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computers,” IEEE Trans. On Computers, Vol. 48, No. 6, pp. 579-590, June 1999.
19. P. Brunet, C. Tanougast, Y. Berviller, S. Weber, “Hardware Partitioning Software for Dynamically Reconfigurable SoC Design,” 3rd IEEE Intl Workshop on SoC for Real-Time Applications, 2003.
20. T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, “Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs,” FPL, September 2002.

References

21. C. Bobda, M. Majer, D. Koch, A. Ahmadinia, J. Teich, “A Dynamic NoC Approach for Communication in Reconfigurable Devices,” FPL, LNCS 3202, pp. 1032-1036, 2004.
22. L. Moller, I. Grehs, N. Calazans, F. Moraes, “Reconfigurable Systems Enabled by a Network-on-Chip,” FPL’2006.
23. F. Moraes, N. Calazans, A. Mello, L. Moller, L. Ost, “HERMES: an Infrastructure for Low Area Overhead Packet-Switched Networks on Chip,” Integration the VLSI Journal, Vol. 38, No. 1, pp. 69-93, October 2004.
24. D. Ching and P. Schaumont, “Integrated Modelling and Generation of a Reconfigurable Network-on-Chip,” Int. J. of Embedded Systems, Vol. 1, Nos. 3/4, 2005.
25. S. Kubisch, R. Hecht, D. Timmermann, “Adaptive Hardware in Autonomous and Evolvable Embedded Systems,” Embedded World, 2005.

References

26. B. Ahmad, A. T. Erdogan, S. Khawam, “Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC,” 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 405-411, IEEE CS Press, June 2006.
27. J. Hu and R. Marculescu, “DyAD – Smart Routing for Networks-on-Chip,” IEEE/ACM International Design Automation Conference (DAC), June 2004.