VLSI Design Automation
High-Level Synthesis

Instructor : Haidar M. Harmanani

Fall 2020

haidar@lau.edu.lb

Binding and Sharing Problem

- Given: scheduled sequencing graph
  - Operation concurrency well defined
- Consider operation types independently
  - Problem decomposition (natural)
  - Perform analysis for each resource type
- Operation compatibility
  - Same type
  - Non-concurrent
- Conflicting operations
  - Concurrent, different types
  - Dual to compatibility

Allocation (Binding)

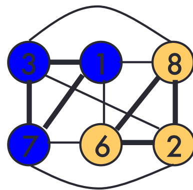
- Allocation = resource binding
  - Spatial mapping between operations and resources
  - Operators can be dedicated or generic (shared)
  - Operators and registers need to be allocated
- Sharing
  - Assignment of a resource to more than one operation
- Constrained resource binding
  - Resource-dominated circuits
  - Fixed number and type of resources available
- NP-complete problem – need heuristics

Binding in Resource-Dominated Circuits

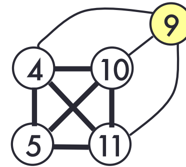
- Resource Compatibility Graph $G_+(V,E)$
  - V represents operations
  - E represents *compatible* operation pairs
- Compatible operations
  - (v_i, v_j) are *compatible* if they are not concurrent and can be implemented by resources of same type
  - Note: concurrency depends on schedule
- Partition the graph into minimum number of cliques in $G_+(V,E)$
  - *Clique* = maximal complete subgraph
  - Partition the graph into minimum number of cliques, or
  - Clique cover number, $\kappa(G_+(V,E))$

Compatibility Graph $G_+(V,E)$

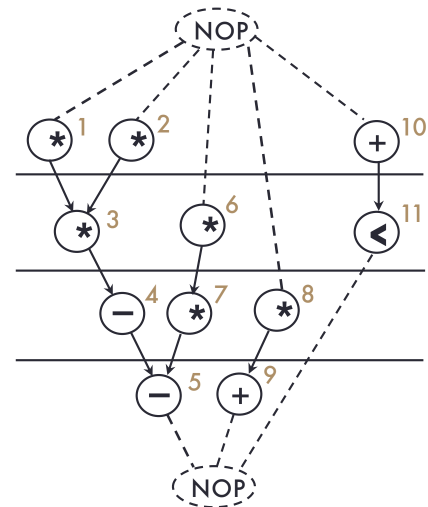
- Minimum Clique covers in $G_+(V,E)$



MULT



ALU



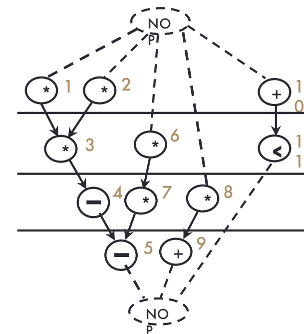
$\kappa(G_+(V,E))$ = 2

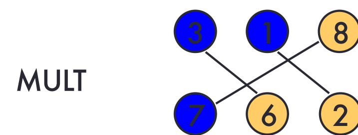
$\kappa(G_+(V,E))$ = 2

LAU Computer Science

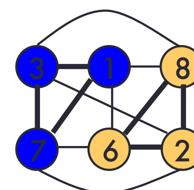
Conflict Graph $G_-(V,E)$

- Resource Conflict Graph $G_-(V,E)$
  - *V* represents operations
  - E represents conflicting operation pairs
- Conflicting operations
  - Two operations are conflicting if they are *not* compatible
- *Complementary* to compatibility graph
- Find independent set of $G_-(V,E)$
  - A set of mutually compatible operations
  - Coloring with minimum number of colors
  - Chromatic number $\chi(G_-(V,E))$





Conflict graph $G_-(V,E)$

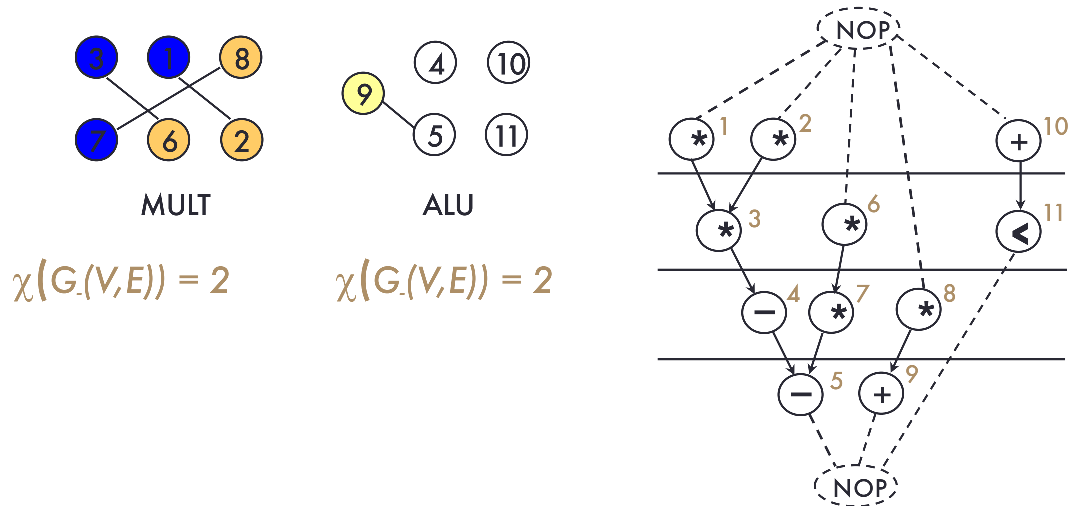


Compatibility graph $G_+(V,E)$

LAU Computer Science

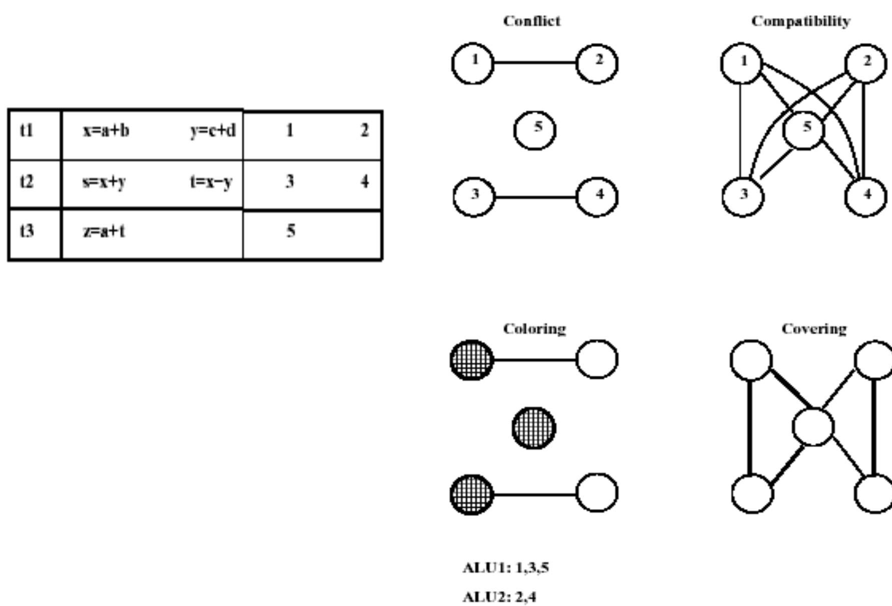
Conflict Graph $G_{-}(V,E)$ - Example

- Chromatic numbers in $G_{-}(V,E)$



LAU Computer Science

Clique vs Coloring - Example



t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x−y	3	4
t3	z=a+t		5	

LAU Computer Science

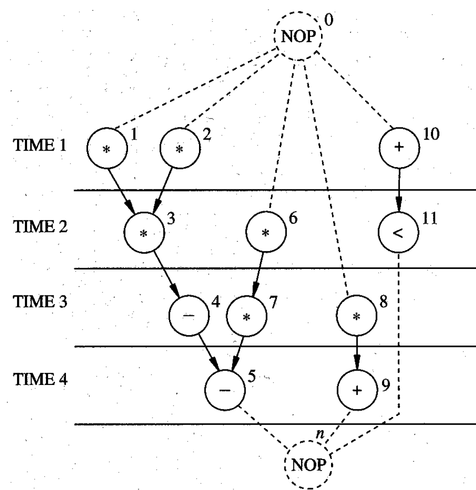
Special Graphs

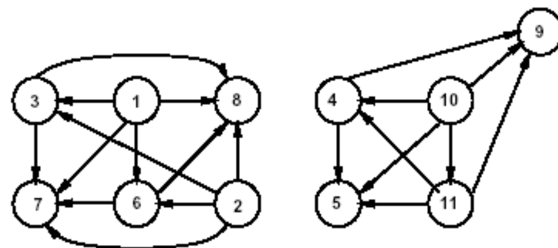
- *Comparability graph*
  - Graph G(V,E) has an orientation (G(V,F) with transitive property:
    $(v_i, v_j) \in F$ and $(v_j, v_k) \in F \Rightarrow (v_i, v_k) \in F$
- *Interval graph*
  - Vertices correspond to *intervals*
  - Edges correspond to interval *intersections*
  - Subset of *chordal* graphs
    - Every loop with more than 3 edges has a chord
- The compatibility/conflict graphs have special properties
  - *Compatibility* $\Rightarrow$ comparability graph
  - *Conflict* $\Rightarrow$ interval graph

Comparability Graph

Representation of compatible relations

- Note: sequencing graph is assumed to be scheduled



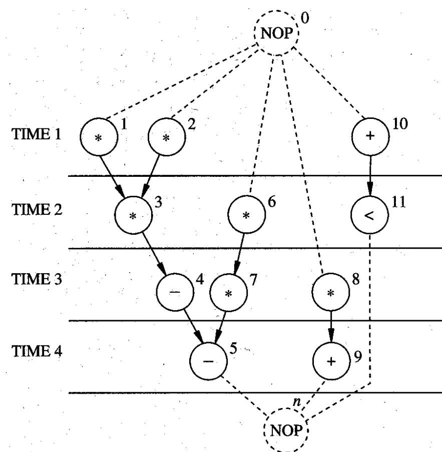


Note the orientation of edges, compare to compatibility graph.

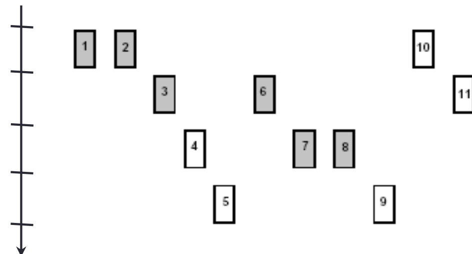
Interval Graph Representation

Interval representation of conflicting relation

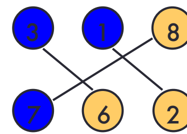
- Note: sequencing graph is assumed to be scheduled

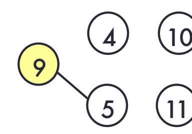


Intervals with “*Left*” and “*Right*” coordinates



Compare with conflict graphs:

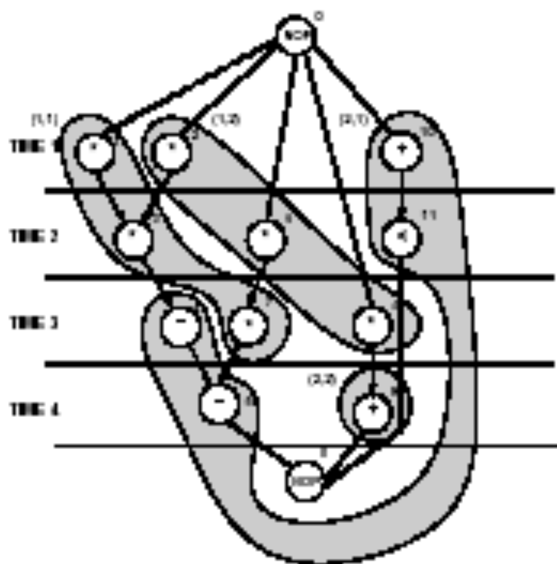


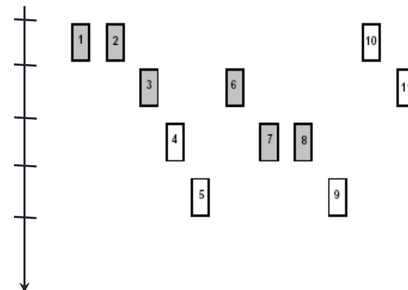


MULT ALU

LAU Computer Science

Operation Binding - Solution





Mult1: { 1, 3, 7 }, Mult2: { 2, 6, 8 }
ALU1: { 10,11, 4,5 }, ALU2: { 9 }

LAU Computer Science

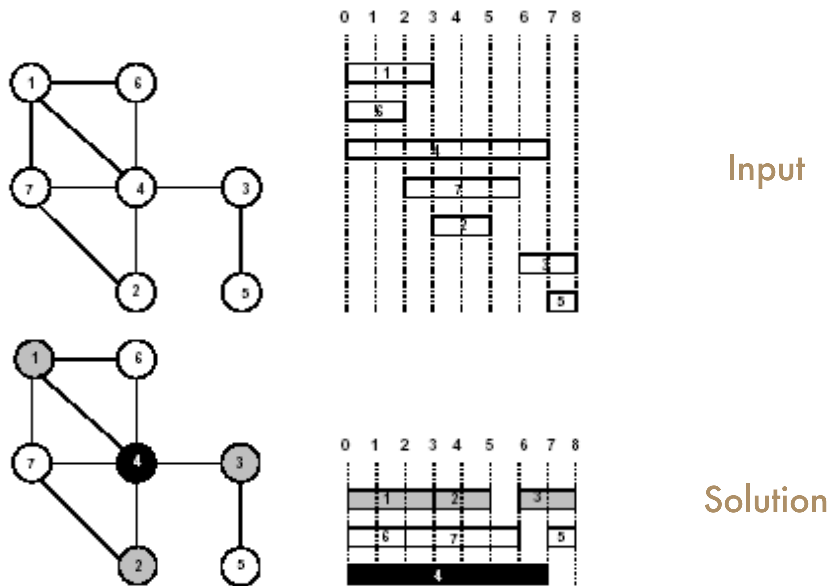
Left-Edge Algorithm

- Input
  - Set of intervals sorted with left and right edge coordinates
- Algorithm
  - Sort intervals by their left edge coordinates
  - Assign non-overlapping intervals to first track (color) using the sorted list
  - When possible intervals are exhausted, increase track (color) counter and repeat.
- Efficiency
  - Simple, polynomial time algorithm

Left-Edge Algorithm

```
LEFT_EDGE(I) {
    Sort elements of I in a list L in ascending order of l_i;
    c = 0;
    while (some interval has not been colored ) do {
        S = ∅;
        r = 0;
        while  ( ∃s ∈ L such that l_s > r) do{
            s = First element in the list L with l_s > r;
            S = S ∪ {s};
            r = r_s;
            Delete s from L;
        }
        c = c + 1;
        Label elements of S with color c;
    }
}
```

Left-Edge Algorithm - Example



ILP Formulation of Operation Binding

- Boolean variables b_{ir}

  $b_{ir} = \begin{cases} 1 & \text{if operator } i \text{ is bound to resource } r \\ 0 & \textit{otherwise} \end{cases}$

- Boolean variables x_{il}

  $x_{il} = 1$ if operation i is scheduled to start at step l

- Each operation is bound to *one resource*

$$\sum_{r=1}^{a} b_{ir} = 1 \quad \forall i$$

(*a* = limit on resource *r*)

- At each step l, *at most one operation* can be executing for a given resource (horizontal constraint)

$$\sum_{i=1}^{n_{ops}} b_{ir} \sum_{m=l-d_i+1}^{l} x_{im} \leq 1 \quad \forall l \quad \forall r$$

Operation Binding - Solution

- Equations for two multipliers:

  $b_{i1} + b_{i2} = 1,\ i=\{1,2,3,6,7,8\}$

MULT 1:

$\sum_{i=\{1,2,3,6,7,8\}} b_{i1}x_{il} \leq 1,\ l=1,2,\ldots,5$

MULT 2:

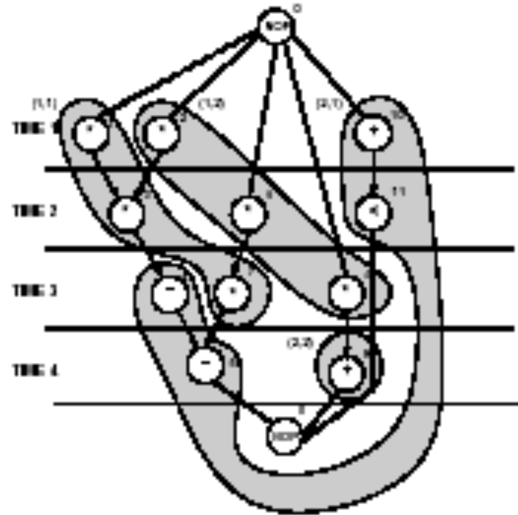
$\sum_{i=\{1,2,3,6,7,8\}} b_{i2}x_{il} \leq 1,\ l=1,2,\ldots,5$

- Solution:

  $b_{11} = b_{31} = b_{71} = 1$

  $b_{22} = b_{62} = b_{82} = 1$

  all other $b_{ij} = 0$



LAU Computer Science

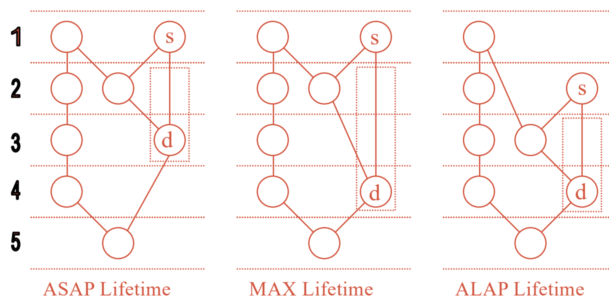
Register Binding Problem

- Registers are *storage resources*, holding variable values across control steps
- Given a schedule, generate:
  - *Lifetime intervals* for variables
  - *Lifetime overlaps*
- Construct a *conflict graph (interval graph)*
  - Vertices *V : variables (operations)*
  - Edges *E: overlaps*
  - Build an *interval graph*
- *Compatibility graph (comparability graph)*
  - Complement of *conflict graph*

LAU Computer Science

Minimization of Register Costs

- Given a scheduled sequencing graph
  - Minimum set of registers required is given by the largest number of data arcs crossing a C-step boundary
- Create *storage operations*, at output of any operation that transfers a value to a destination in a later C-step
- Generate *Storage DG* for these "operations"
- Length of storage operation depends on final schedule

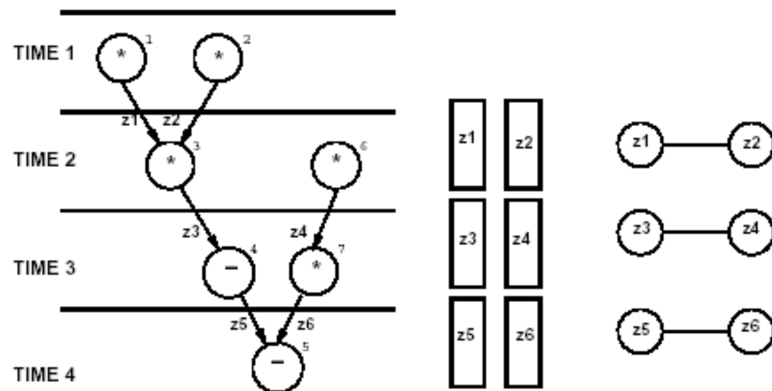


Register Binding Problem

- Given
  - *Variable lifetime conflict graph*
- Find
  - Minimum number of *registers* storing *all variables*
- Simple case
  - Non-iterative designs: *Interval graph*
    - Solve using *left-edge algorithm* (polynomial time)

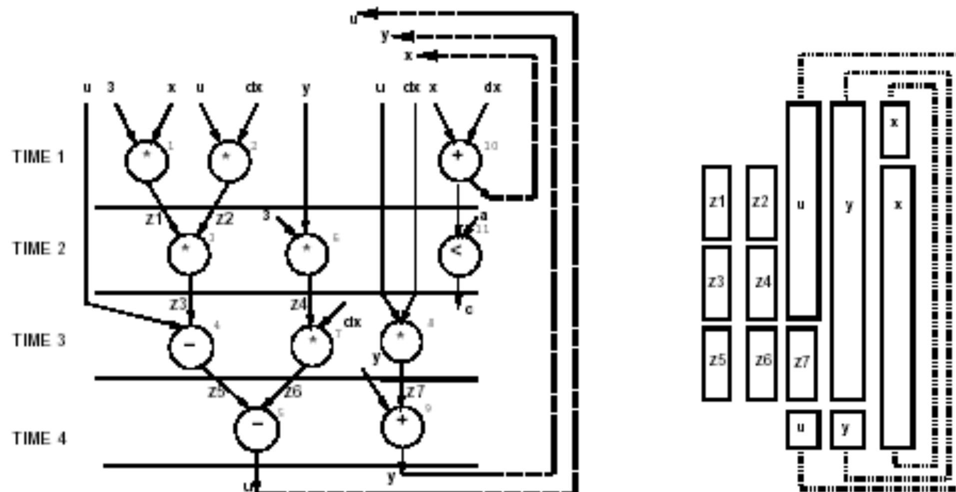
Register Binding Problem – Example 1

- Non-iterative designs
  - Create variable *compatibility* graph or *conflict* graph
  - Use left-edge algorithm to minimize the number of registers



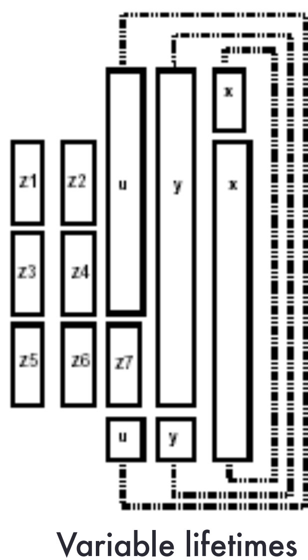
Register Binding – Example 2

- Iterative designs
  - Sequencing graph and variable lifetimes

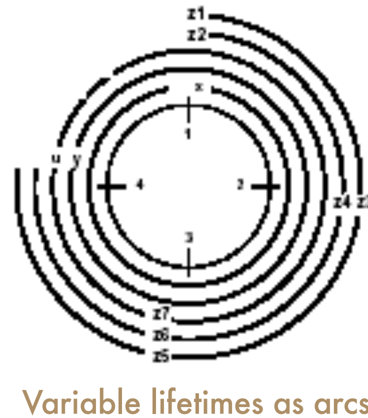


Circular Arc Conflict Graph

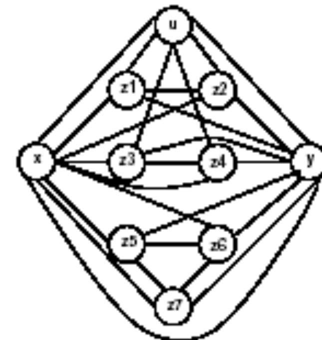
- Overlapping lifetimes of variables represent conflicts



Variable lifetimes



Variable lifetimes as arcs



Circular-arc conflict graph

Register Sharing – General Case

- Iterative constructs
  - Preserve values across iterations
  - *Circular-arc* conflict graph (not simple intervals)
    - Coloring is intractable
- Hierarchical graphs:
  - General conflict graphs
    - Coloring is intractable

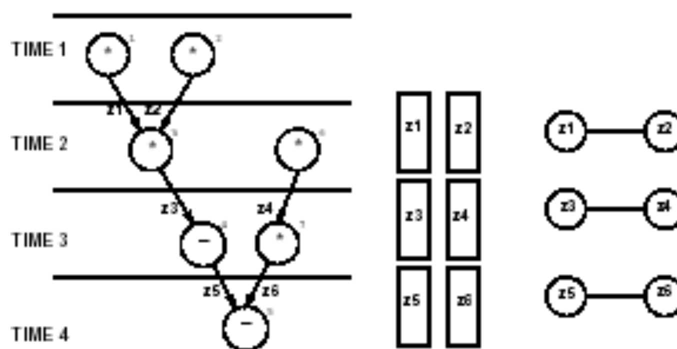
- Heuristic algorithms required

Bus Sharing and Binding

- Buses act as transfer resources
  - See architecture produced by GAUT
- Find the minimum number of buses to accommodate all data transfer
- Find the maximum number of data transfers for a fixed number of buses
- Similar to memory binding problem
- Possible solutions
  - ILP formulation
  - Heuristic algorithms

Bus Sharing and Binding - Example

- One bus
  - 3 variables
- Two buses
  - All variables can be transferred



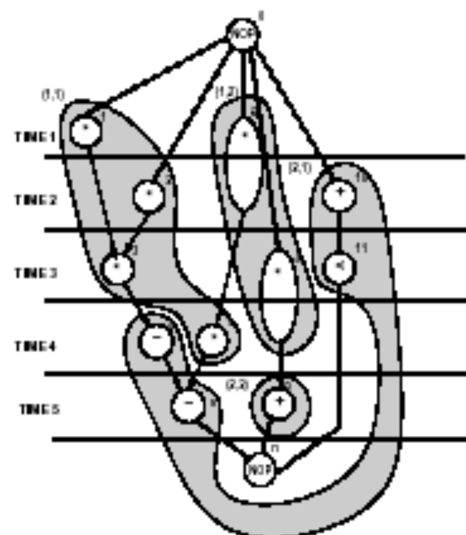
Module Selection Problem

- Resource-type (module) selection problem
  - Generalization of the binding problem
- Library of resources:
  - More than one resource per type
- Example:
  - Ripple-carry adder vs. carry look-ahead adder
- Resource modeling
  - Resource subtypes with *(area, delay)* parameters
- Solution
  - ILP formulation:
    - Decision variables: select resource subtype,
      determine *(area, delay)*
  - Heuristic algorithms:
    - Determine minimum latency with fastest resource subtypes
    - Recover area by using slower resources on non-critical paths

Module Selection - Example 1

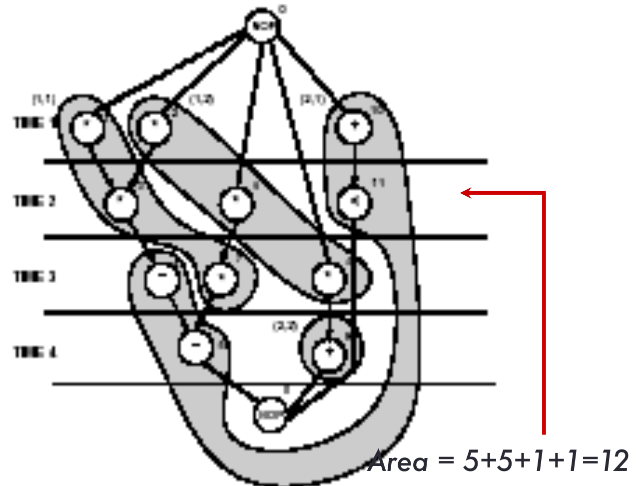
- Latency bound of 5
- Two multipliers available:
  - MULT1 with (*area, delay*) = (5,1)
  - MULT2 with (*area, delay*) = (2,2)
- Two ALUs available:
  - ALU with (*area, delay*) = (1,1) each

Area = 5+2+1+1 = 9



Module Selection Example 2

- Latency bound of 4
  - Fast multipliers for $\{v_1, v_2, v_3\}$
  - Slower multipliers can be used elsewhere
    - less sharing
- *Minimum latency* design
  - used *fast* multipliers only.
- Area recovery
  - On non-critical paths replace fast (large) multipliers by slow (small) ones



High-Level Synthesis for Testability

Haidar M. Harmanani

haidar@lau.edu.lb

Background

- Two approaches for *test synthesis* at the structural level
  - *Design for Test approach* – insertion of test structures is used to improve testability
    - Logic Level and RTL
  - *Testable Synthesis approach* – designs are synthesized with testability properties
    - Testable High-Level Synthesis

Background: High-Level Synthesis

- A transformation from behavior to structure
- High-Level Synthesis raises the abstraction level
  - Determines the macroscopic structure of a circuit by creating a Data Path and Control Unit
  - Optimize area/delay/power of the implementation
- Two main steps in HLS
  - Scheduling
  - Allocation

Background: BIST Methodology

- Goal is to generate test patterns and verify them on chip
- Pseudorandom BIST
  - Patterns generated using TPGRs
  - Test responses evaluated using MISRs
- Determine fault coverage using logic level fault simulation
- Select the test length so as to achieve an acceptable level of fault coverage.

Motivation

- Goal
  - Develop a test synthesis tool based on the BIST methodology that operates concurrently with datapath allocation
  - Generate controller concurrently with least area and power
- Key Elements
  - Test analysis of designs at the RT level based on metrics
  - Test considerations at an early stage
  - Test points selection

Problem Description

- Given a behavioral level description of a circuit represented in the form of a scheduled DFG, a technology library and a set of constraints, generate a self-testable RTL data path such that:
  - The data path conforms to all user constraints
  - The overhead of test registers in the data path is minimized
  - Power consumption is minimized during test and normal mode

Approach

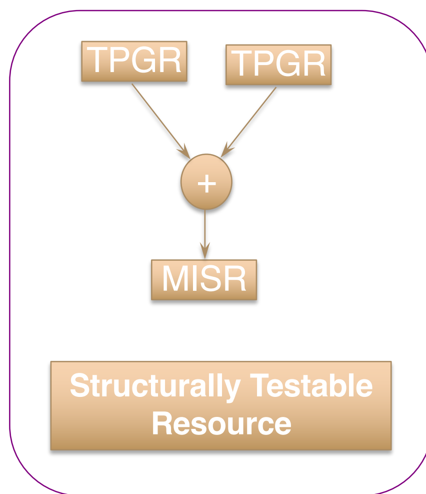
- The above tasks are implemented by:
  - A model for the testable synthesis of RTL datapath structures. This is done through the synthesis of designs with structural properties proven to be good for testing.
  - A test point selection scheme that concurrently explores, during the synthesis process, designs with low test and design cost.
  - Concurrent allocation of BIST registers and Functional Units
  - Minimizing the cost of test registers using functional test metrics
  - Generate a distributed controller, one per test kernel

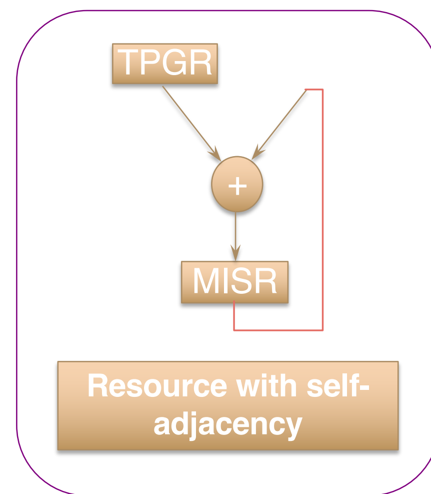
Structural Testability

- A structurally testable resource under the BIST methodology is a resource such that:
  - Test patterns can be applied at the modules input port
  - Signature can be observed at the output port
- A datapath is structurally testable if it consists of structurally testable resources
- Structural Testability
  - Testability as related to the macroscopic circuit structure in terms of registers and combinational blocks
  - Problems arise when a register has to generate patterns and compress a signature during the same test session
    - Self adjacency problem

LAU Computer Science

Structural Testability





LAU Computer Science

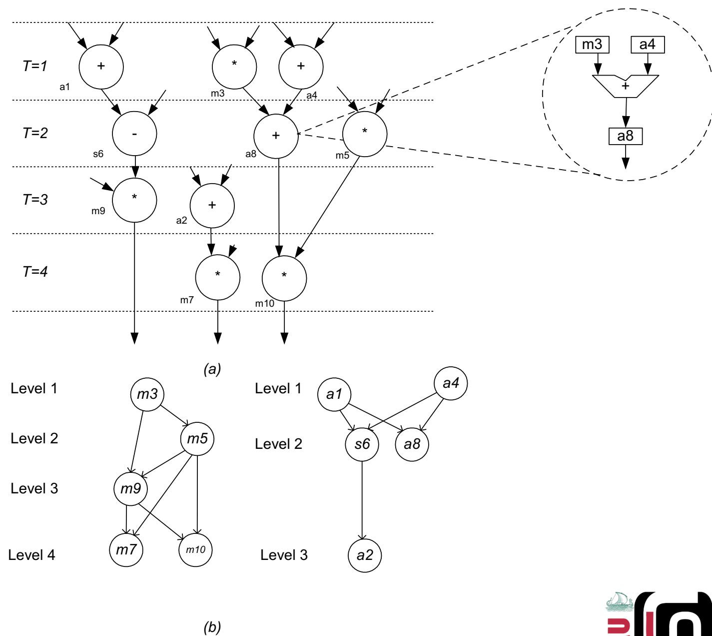
Testable Functional Block

- A Testable Functional Block (TFB) is a test kernel that has:
  - An ALU
  - At least two registers at the input port that can be configured as TPGRs during testing
  - One register at the output port that can be configured as MISR during testing
- A datapath that consists of TFBs is structurally testable
  - Incrementally create a datapath of TFBs
  - A TFB cannot have a self-loop

Testable Allocation

- Generate a compatibility graph for the DFG that indicates which nodes are compatible
- Map DFG nodes to TFBs
- Select TFBs one level at a time and merge the TFBs in one level and the TFBs in the second level
- Select the test points so as to minimize the cost
- Construct the testable datapath incrementally

Example



Test Scheduling

- Objective
  - Minimize number of test sessions by maximizing the number of ALUs tested in the same test session
- Two conditions must be observed
  - If two ALUs have the same MISR, then they cannot be tested in the same session
  - If an ALUs MISR is another ALUs TPGR then these two ALUs cannot be tested at the same time (maybe possible with a CBILBO)

Test Scheduling

- Two main steps
  - ALUs that have the same MISR are assigned to different test sessions with each ALU assigned a weight that is equal to the number of times its MISR is used by other ALUs
  - Every ALUs TPGR is compared to every other ALUs MISR in the same test session. If they use the same register, then the one of the two ALUs that has the minimal weight is moved to another test session
- Running time of the algorithm is $O(a^2)$

Test Points Selection

- In order to minimize the number of test points, two conditions must be satisfied
  - The TPGRs at the input ports of an ALU can not be the same due to correlation problem
  - A TPGR cannot be an MISR for the same ALU in the same test session in order to avoid self-adjacency problem
- An initial test point selection based on BILBO and characterized by a high fault coverage *can* selected by assigning:
  - All registers at the primary input to be TPGR
  - Primary Output MISR
  - Registers that are in between are BILBOs
  - Test points are next relaxed through concurrent selection

Concurrent Test: Registers

- Registers in the data path can be:
  - Controllable – TPGR
  - Observable – MISR
  - Pseudo-controllable – A normal register through which random patterns are sensitized
  - Pseudo-observable – A normal register that is transparent enough to pass faults unaltered to a real MISR

Test Merger Algorithm

- Four Steps
  - TFBs pseudo merger
  - Select input test registers
  - Select output test registers
  - Breaking functional self-loop

TFBs Pseudo Merger

- Based on the merging algorithm and a cost function, select the TFBs whose merger will have the least cost and then combine them into one
- Link test plans into a preliminary "merged test plan."

Select Input Registers

- Need to select a TPGR for every port in the datapath after the TFBs pseudo-merger
- Selection priority is sorted in least cost
  - Normal, MISR, TPGR, and BILBO
- If current TFB is transparent then set the TFB output register to be a pseudo-TPGR

Select Output Registers

- Next, there are three cases that may result from the merger of two TFBs:
  - ➊Both TFBs have a BIST register at the output
    - If one is MISR and one is TPGR, assign register to be a BILBO
    - If both are TPGR and pseudo-MISR, then resulting register is a TPGR and a pseudo-MISR
    - If both are MISR, then attribute is MISR and pseudo-TPGR
  - Remove the attribute (TPGR or MISR) if test time is not exceeded

Select Output Registers

- There are three cases that may result from the merger of two TFBs:
  - ➋Both TFBS have normal registers at the output
    - Resulting register is normal unless
      - Module is not random enough and test time is exceeded in which case we set the register attribute to TPGR
      - Patterns originate from a non-transparent module, then we set the register to MISR

Select Output Registers

- There are three cases that may result from the merger of two TFBs:
  - ❸One TFB has a BIST register and one has a Normal register
    - If the BIST is TPGR then it can be pseudo-TPGR if test time can be increased without increasing the maximum time
    - Otherwise, it remains a TPGR

Self-Loops Breaking

- Resulting datapath cannot have self-loops due to merger and compatibility rules
  - Structural testability property of the TFB
- However, we may have functional loops after test points removals in the previous step

Self-Loops Breaking

- Traverse the datapath starting at every module
  - Generate a list of all children and store in a queue
  - For every node, check for a cycle.
    - If yes, then add to the node an implementation attribute (TPGR or MISR)

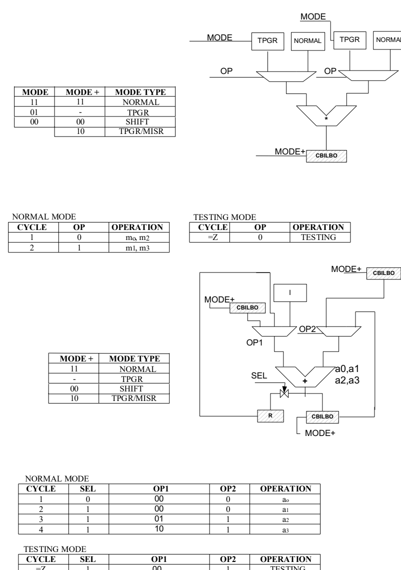
Test Control Design

- Controller design must support both normal and test modes
- Two styles were implemented
  - Central Mode
  - Distributed
- Central control is implemented as an FSM where the total number of states corresponds to the schedules clock cycles
  - Implemented for comparison purposes
- Distributed control implemented concurrently with the synthesis process
  - Associate a controller with each *test kernel* (TFB) active during the corresponding clock cycles
  - Control the distributed controllers with a relatively small central controller

Distributed Control Design

- First, test kernels are formed based on test kernels and the underlying test registers
  - It is possible that registers may belong to two test kernels
  - External I/O are handled by the main controller
  - Construct a decoding table for each sub-controller
- The algorithm adds the cost of the distributed controller to the TFB thus optimizing datapath and control in the same time
  - Also minimizes power consumption through the RTL gating of distributed controllers

Distributed Control: Kernels



MODE	MODE +	MODE TYPE
11	11	NORMAL
01	-	TPGR
00	00	SHIFT
	10	TPGR/MISR

NORMAL MODE

CYCLE	OP	OPERATION
1	0	m_0, m_2
2	1	m_1, m_3

TESTING MODE

CYCLE	OP	OPERATION
=Z	0	TESTING

MODE +	MODE TYPE
11	NORMAL
-	TPGR
00	SHIFT
10	TPGR/MISR

NORMAL MODE

CYCLE	SEL	OP1	OP2	OPERATION
1	0	00	0	a_0
2	1	00	0	a_1
3	1	01	1	a_2
4	1	10	1	a_3

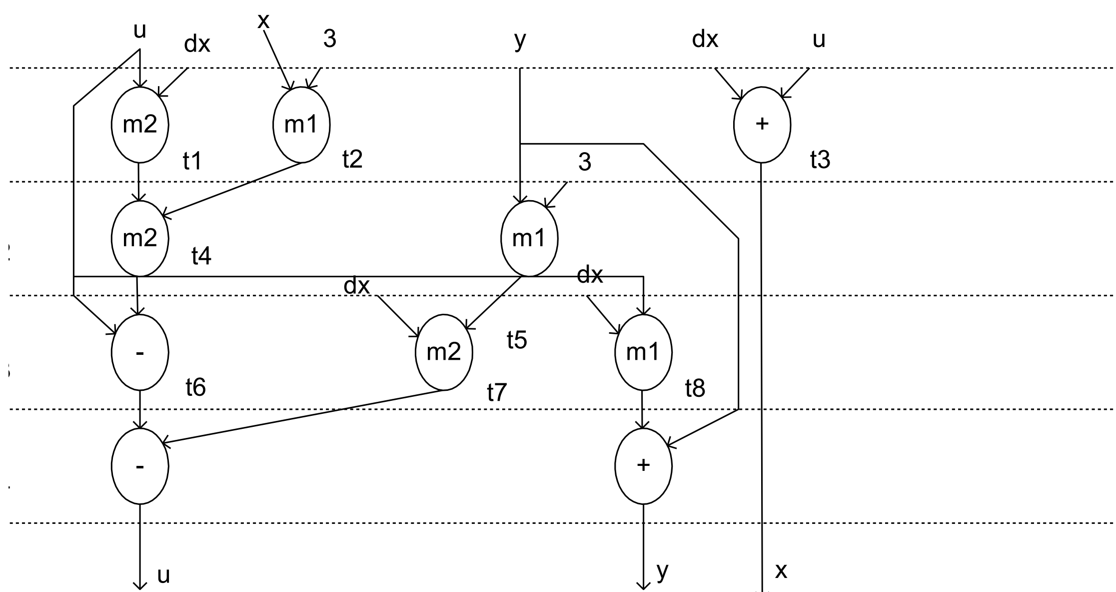
TESTING MODE

CYCLE	SEL	OP1	OP2	OPERATION
=Z	1	00	1	TESTING

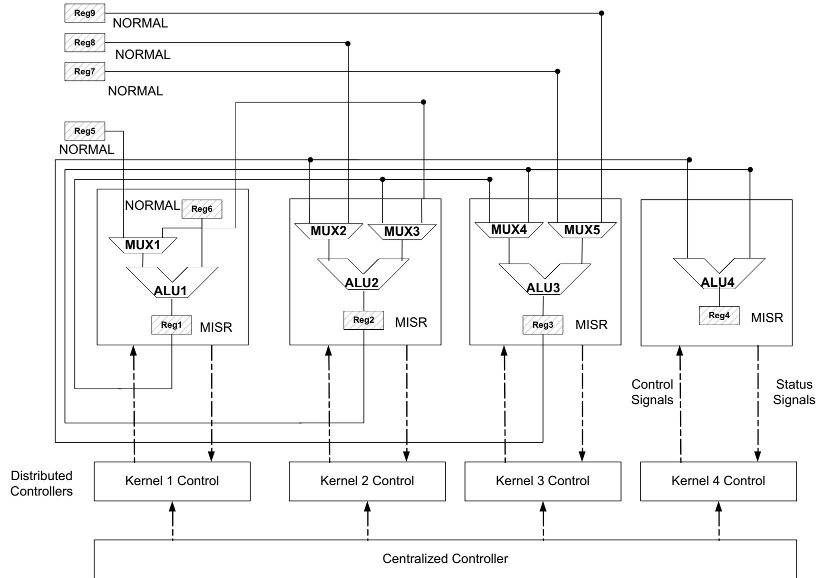
Experimental Procedure

- Use benchmark circuits in the literature
  - Synthesize using high-level synthesis tool
  - Generate various test styles based on the tradeoff mechansim
  - Generate Test Controller
  - Compare results based on the Synopsys CAD Tools
  - Generate fault simulation for one example for validation (DCT)

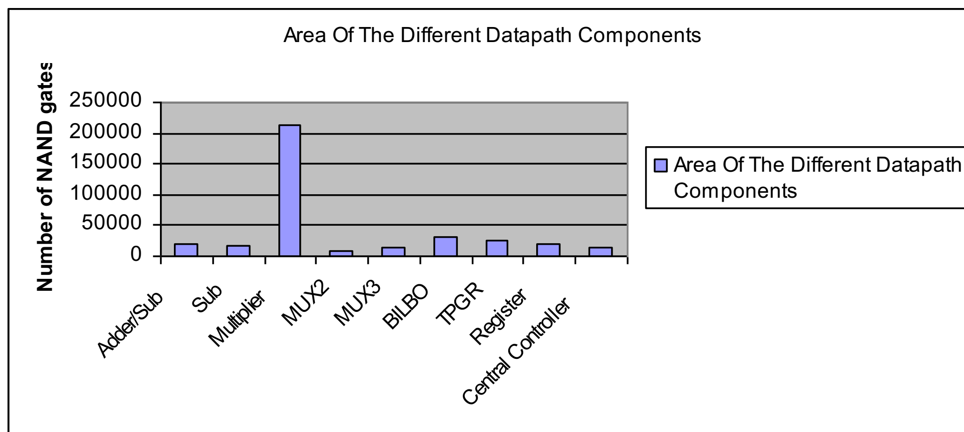
DFG Example: DIFFEQ



Distributed Control: Diffeq



Diffeq Results – Central Control Style



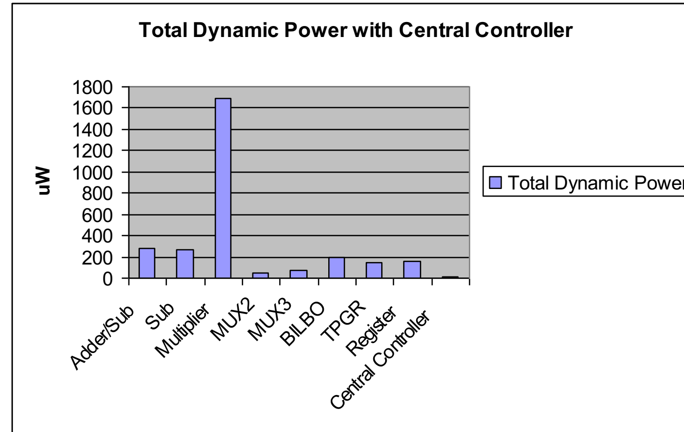
Diffeq Results – Central Control Style

Component	Area (units)
NAND4	2
OR2	2
NAND8	7
Multiplier	211,734
Adder/Subtractor	19,413
Sub	17,415
2-to-1 Mux	14,796
3-to-1 Mux	14,796
BILBO Register	30,771
TPG Register	25,344
Normal Register	20,898
Controller (Central)	13,266
Overall Design	**1,108,017**

Central Control Style Analytic Power Estimation (V=1.8v)

Component	Internal Power (uw)	Switching Power (uw)	Dynamic Power (u)	Leakage Power (nw)
Multipler	899.6268	793,4720	1.6931 (mw)	138.5580
Sub	92.9174	168.9458	261.8632	14.0698
Adder/Sub	106.1745	175.5958	281.7703	15.7282
2-to-1 Mux	40.4850	11.3344	51.8194	7.0472
3-to-1 Mux	53.5307	19.4851	73.0158	12.0218
BILBO	91.0881	103.6256	194.7137	26.0170
TPGR	75.0382	74.9551	149.9933	23.0786
Register	76.9240	82.4911	159.4151	18.9777
Central Control	9.1441	5.9083	15.0525	9.9557
Overall Design	**1.0088 (mw)**	**10.8958 (mw)**	**11.9046 (mw)**	**819.7382**

Central Control Style – Dynamic Power estimation



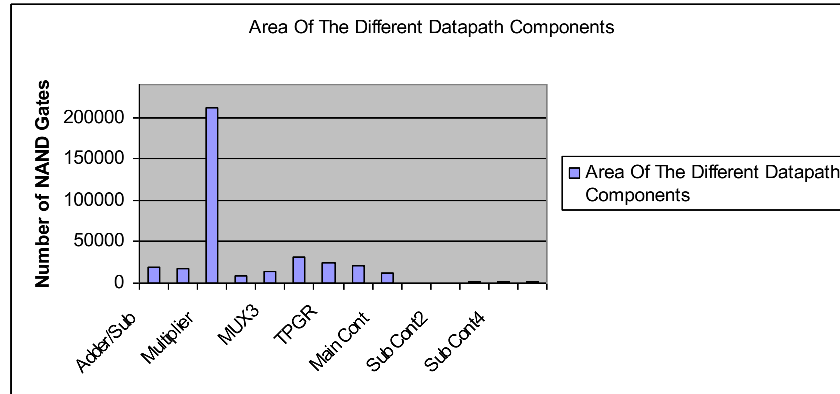
LAU Computer Science

Distributed Control Style – Analytical Power Estimation (V=1.8v)

Component	**Cell Area**	**Dynamic Power (uw)**	**Leakage Power**
Central Control	12,060	14.9929	9.0107
Sub Control 1	810	3.1426	709.5832
Sub Control 2	810	3.1417	709.5851
Sub Control 3	1,341	4.8498	1.1556
Sub Control 4	1,458	6.2736	1.2673
Sub Control 5	1,467	6.3050	1.2918
Overall Design	**1,113,237**	**12.4411 (mw)**	**827.6843**

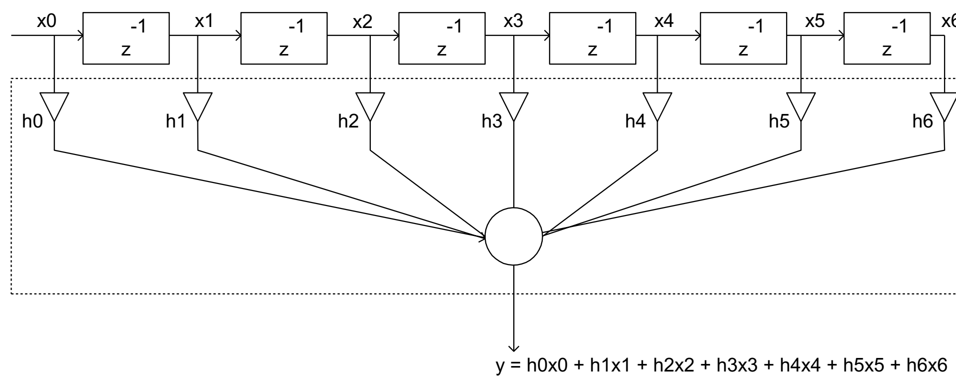
LAU Computer Science

Datapath Components Area



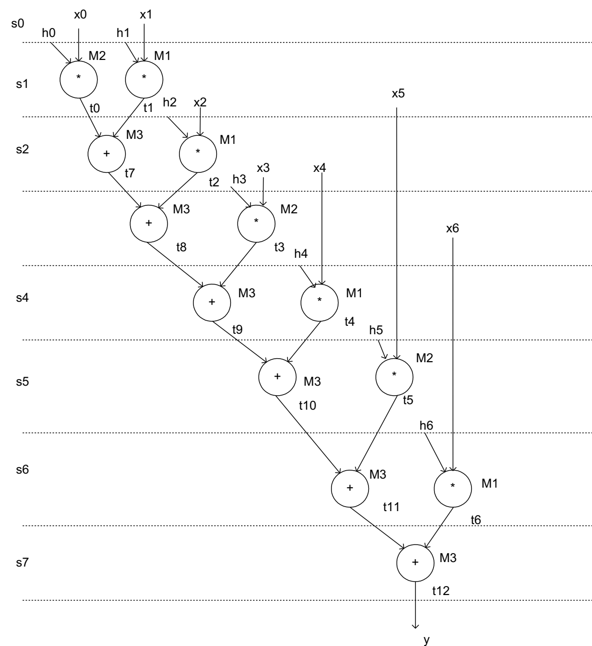
LAU Computer Science

6th Order FIR Filter

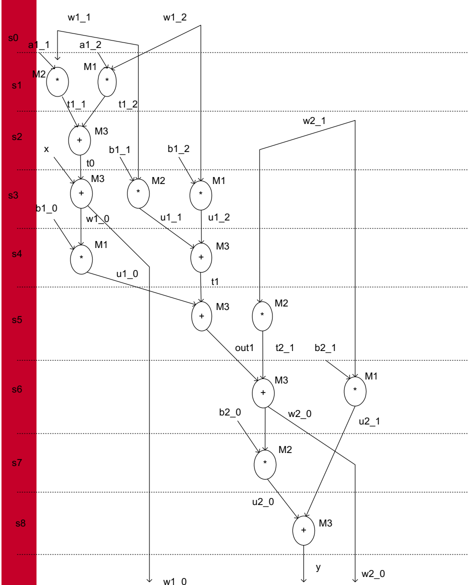


LAU Computer Science

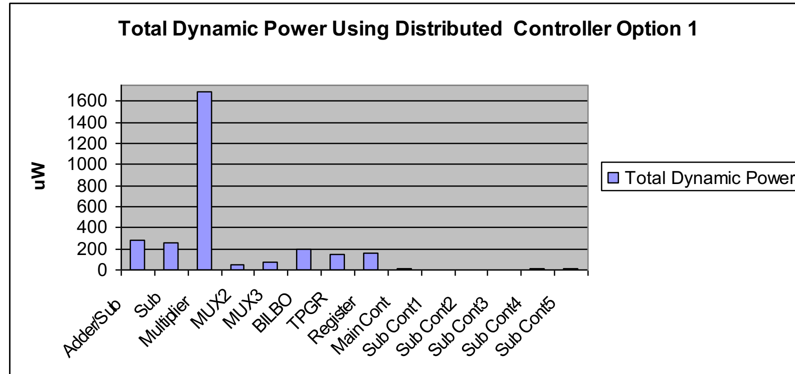
6th Order FIR Filter DFG



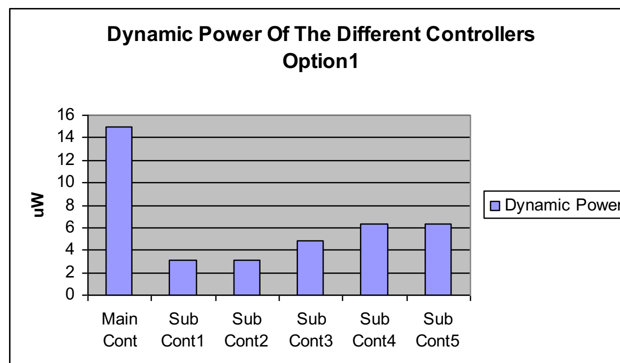
3rd Order IIR Filter DFG



Total Dynamic Power: FIR



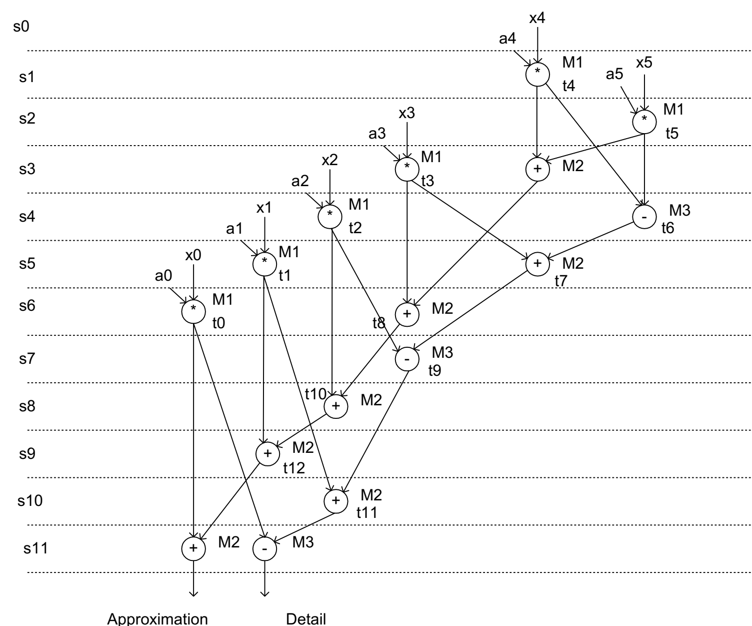
Sub-Controller Power Estimation: FIR



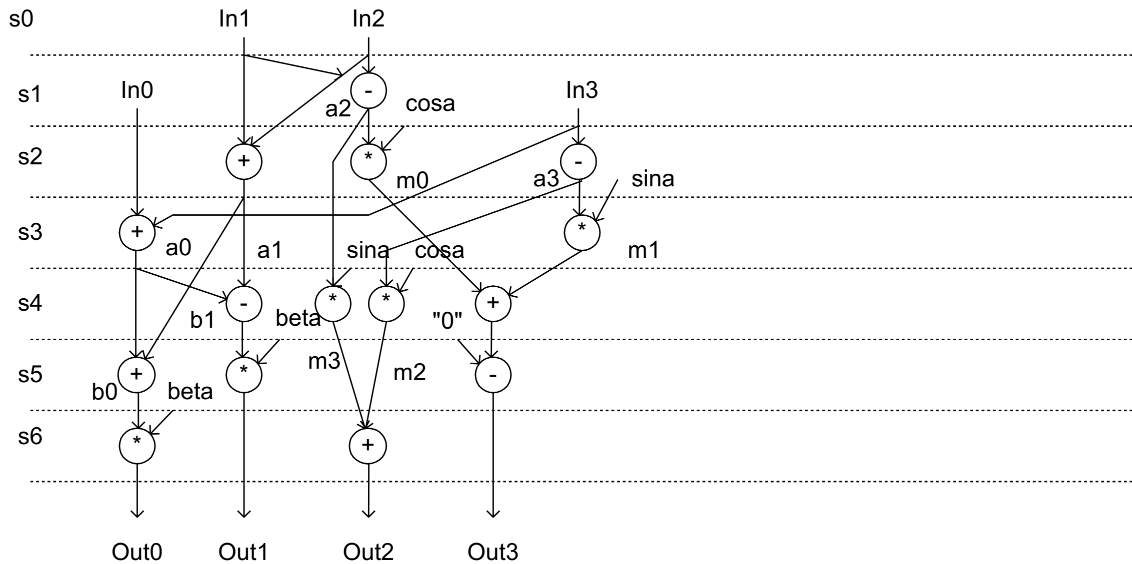
Power Simulation Results

Control Mode	**Mode**	**Dynamic Power (mw)**	**Cell Leakage (nw)**
Distributed	Normal Mode	1.4454	825.0923
	Test Mode	1.1184	827.9899
Central	Normal Mode	1.5172	878.4852
	Test Mode	1.1690	876.544

6-tap Wavelet Filter



4-Point Discrete Cosine Transfer (DCT) DFG



Other Results Comparison

Ckt	Type	R	TPGR	MISR	BILBO	M	Area	OH%
Tseng	1	1248	0	0	0	728	1976	
	2	416	0	1520	388	400	2724	27.46
	3	0	1024	0	1552	400	2976	33.60
Diffeq (2)	1	1872	0	0	0	672	2544	
	2	208	1024	608	0	960	2800	9.14
	3	0	1280	0	1552	1056	3888	34.56
Diffeq (1)	1	2288	0	0	0	800	3088	
	2	624	1536	608	388	576	3732	17.26
	3	0	1536	0	2328	576	4440	30.45
DCT4	1	2704	0	0	0	1148	3852	
	2	1664	1536	304	0	1248	4752	18.94
	3	0	2560	0	1940	1088	5588	31.07

Results Comparison (Cont.)

Ckt	Type	R	TPGR	MISR	BILBO	M	Area	OH%
Wavelet6	1	2912	0	0	0	1312	4224	
	2	2496	1024	304	0	1112	4936	14.42
	3	0	2816	0	1940	1328	6084	30.57
IIR3	1	3120	0	0	0	1020	4140	
	2	1664	1280	608	0	992	4544	8.89
	3	0	2816	0	1552	864	5232	20.87
Fir6	1	2912	0	0	0	768	3680	
	2	1040	2048	608	0	752	4448	17.27
	3	0	2816	0	1552	752	5120	28.13