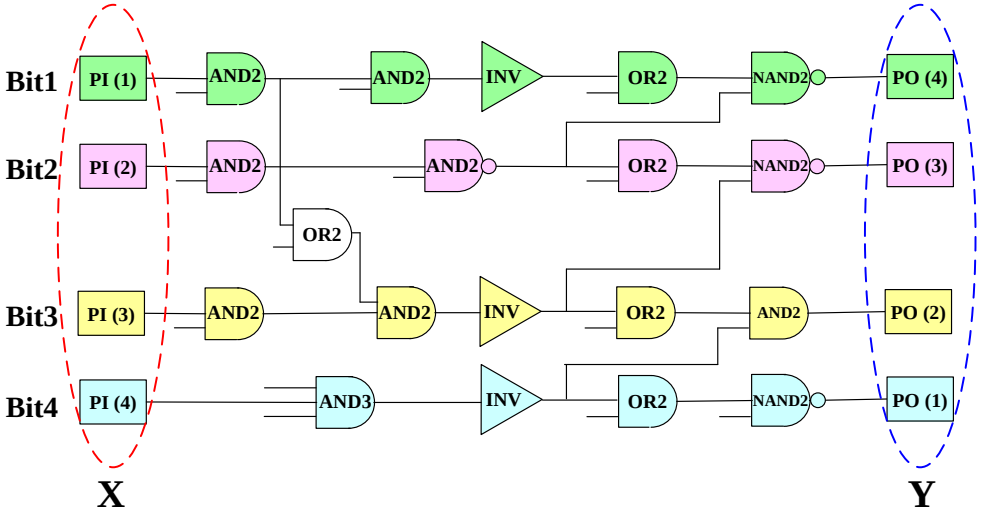
Network Flow Based Datapath Bit Slicing

Hua Xiang Minsik Cho Haoxing Ren
Matthew Ziegler Ruchir Puri

03/27/2013

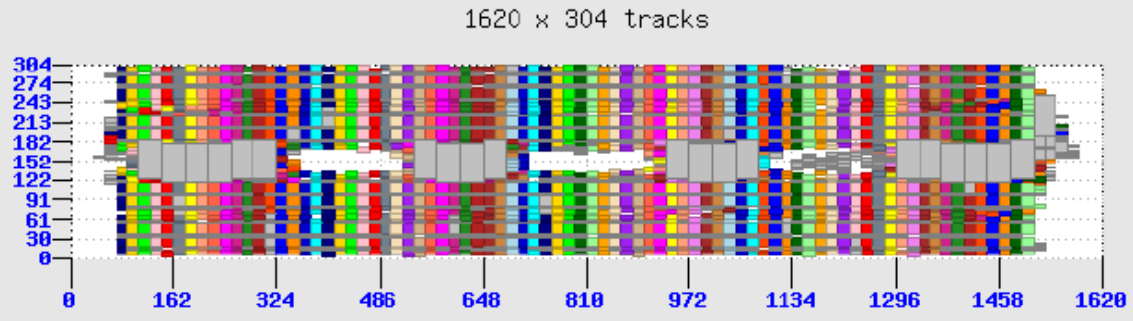
Introduction

- **Datapaths are composed of bit slices**
- **What are bit slices?**
  - For ideal datapath, each bit should have the **same** structure with no or very few connections to other bits
  - In real design, bit slices have **similar** structures
    - Different bits can be implemented differently,
      - e.g., NAND or AND+INV
    - Different bits have connections
      - e.g., Carry bit



Applications for datapath bit slices

- **The bit line alignment imposed on placement/floorplan help to create high density high performance design**



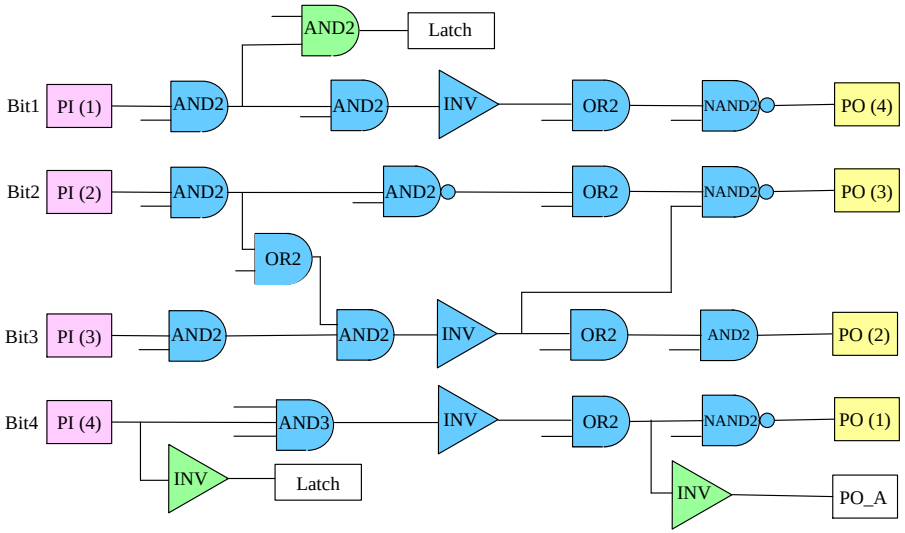
- **Automatic datapath-aware latch bank planning**
  - Designer's hand-crafted manual latch placement
    - Good quality
    - Timing-consuming
    - Understanding design 100%
  - Automatic structured latch placement
    - Datapath bit slicing provides guidance for latch bank placement
      - X location is determined by bit slice alignment
      - Y location draws on the bit height of each bit
    - Provide an early starting point for datapath macros
    - Sweep through many configurations overnight

Bit Slicing Approaches in Literature

- **Maintain datapath structures from VHDL**
  - Limit datapath optimization
  - Impose hard constraints on design
- **Regularity extraction**
  - Template based
    - Templates are either provided or auto generated
    - Exact match with templates
    - Some even assume the bit lines is repeated infinitely
    - ➢ Hard for similar match
    - ➢ A few bits in the datapath might be quite different from the rest
      - E.g., the last bit is very likely to be different
  - Location/Name based
    - Draw on item locations or names for matching
    - ➢ Physical information is not available
    - ➢ Naming is not trustable, especially after optimization
      - Gates/nets may be added or deleted

Datapath Extraction

- **Identify all gates related to the given datapath**
  - For a datapath gate, it must have paths to the input vector and the output vector.
- **Method: Two-way search extraction**
  - First search: mark all gates in the input fan-out cone
  - Second search: mark all gates in the output fan-in cone
  - Only gates marked in both searches are returned
- **All bit line gates are included in the two-way search**
- **But not all gates returned by two-way search are bit line gates**



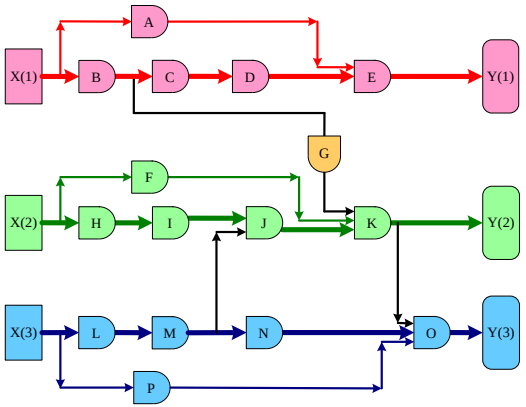
Datapath Bit Matching

- **Datapath extraction identifies the connectivity between two vectors**
- **How to identify each bit slice? → Datapath Bit Matching**
  - Given an input vector X=(x1,…,xn) and an output vector Y=(y1,…,yn)
  - Identify one-to-one matching between X and Y
  - N bit slices can be identified through two-way search algorithm

- **Bit Matching can be done with a bipartite graph? No**
  - The weight of a pair of starting and ending bit cannot be calculated independently
- **Bit Matching is a partition problem? No**
  - Not all gates in the datapath graph belong to bit lines
- **Bit Matching can be done with path tracing? No**
  - One starting bit may have paths connecting to multiple ending bits
- **Bit Matching can by done with enumeration? Long runtime**
  - The searching space is huge

Datapath Main Frame

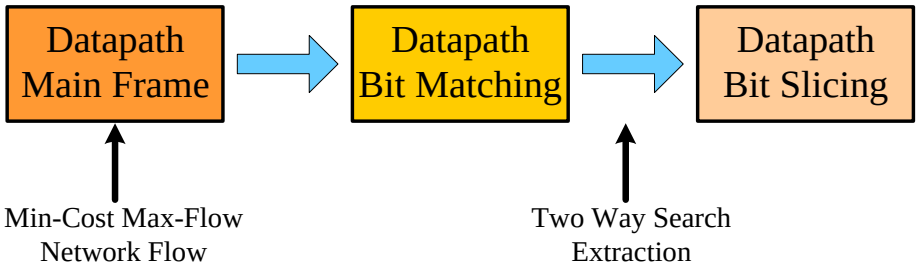
- **Observation:**
  - All bit slices carry similar number of gates
  - The connections among bit slices are limited
  - All bit slices usually have at least one similar path from the input bit to the output bit, and the path is disjoint with the similar paths in other bit lines
  - ➢ Identify the longest similar path?



- **Datapath Main Frame**

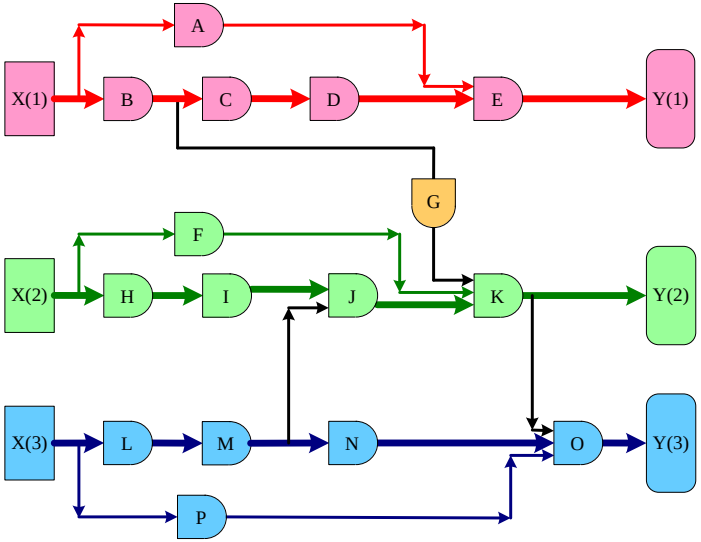
  Given a datapath input vector X=(x1, …, xn), and an output vector Y=(y1, …, yn), identify n disjoint paths from X to Y such that the n paths cover the maximum number of datapath gates.

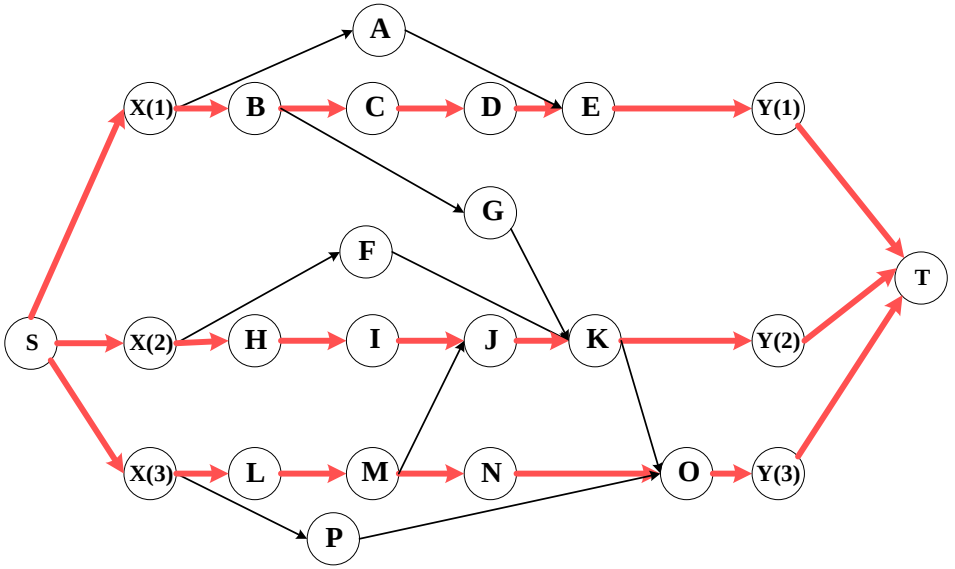
- **Datapath bit slicing flow**



Flow-based Datapath Main Frame Algorithm

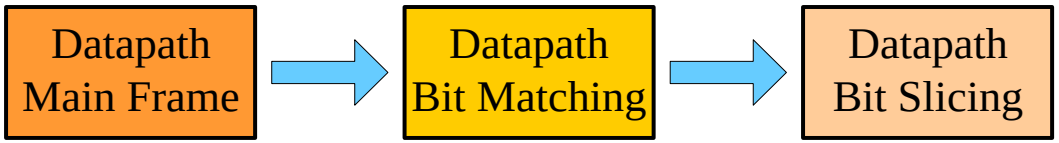
- **The main target is to find n paths which cover as many gates as possible**
- **A flow network is constructed to capture the constraints**
  - To maximize gates on the extraction graph
    - Assign a large negative cost for each gate
  - To minimize crossing between bit lines
    - Assign a small positive cost for each net
  - Apply the min-cost max-flow algorithm to identify bit slices
- **The min cost solution corresponds the max number of gates**



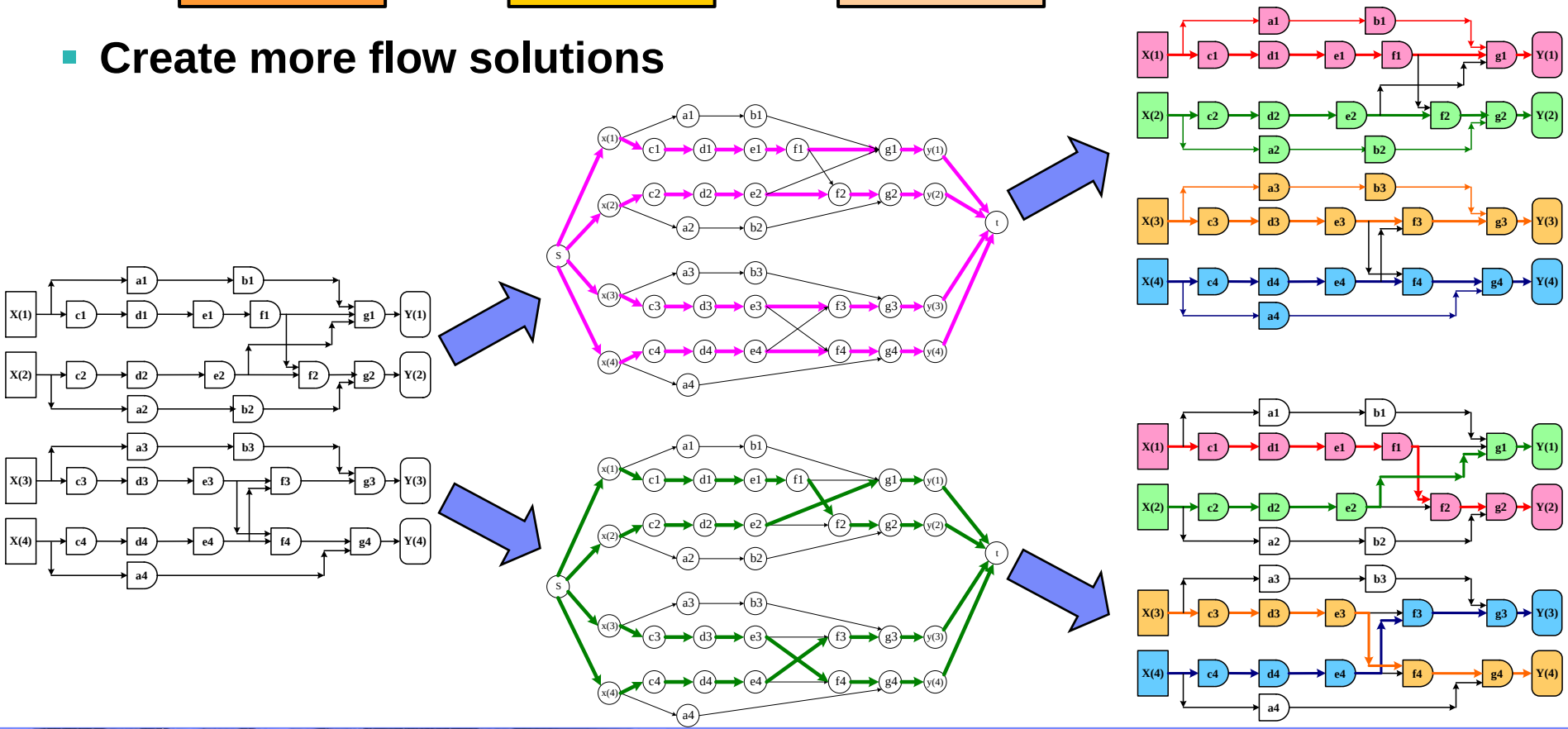


Iterative Enhancement

- **Min-cost max-flow algorithm only returns one optimal solution**
- **There might be multiple optimal flow solutions**



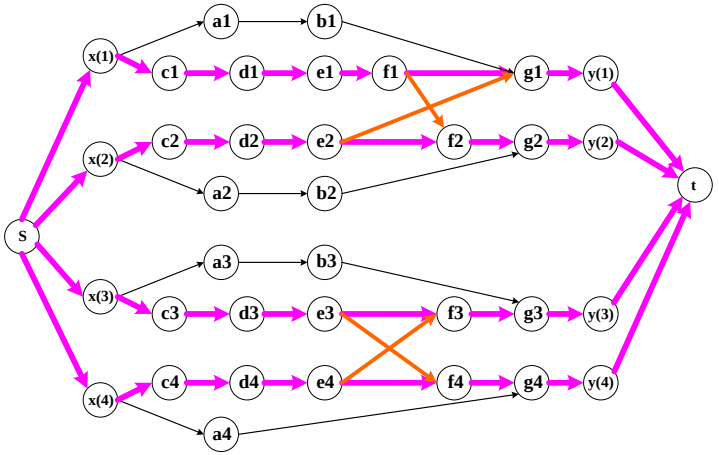
- **Create more flow solutions**



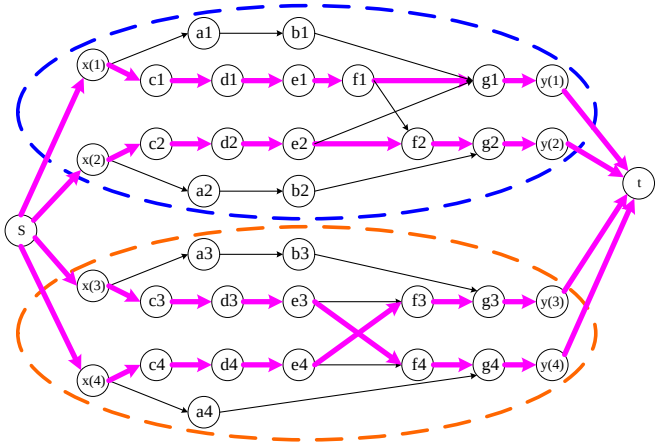
Create More Flow Solutions

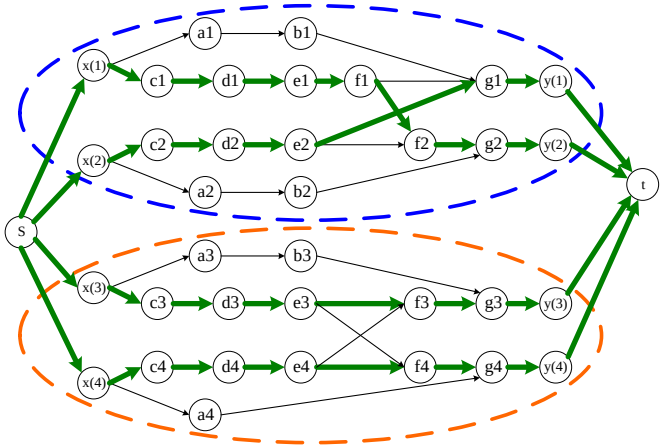
- **Any two optimal solutions include the same number of gates**
  - Very likely they cover the same set of gates
- **Any two optimal solutions include the same number of nets**
  - The two sets of nets must be different
- **Adjust edge weights to generate different flow solutions**

Algorithm Adjust_Flow_Network_Cost (G_f, G_d, $flow$)
1. for each edge e in the $flow$
2. if e corresponds to a net in G_d
3. then $C_f(e) += \delta$

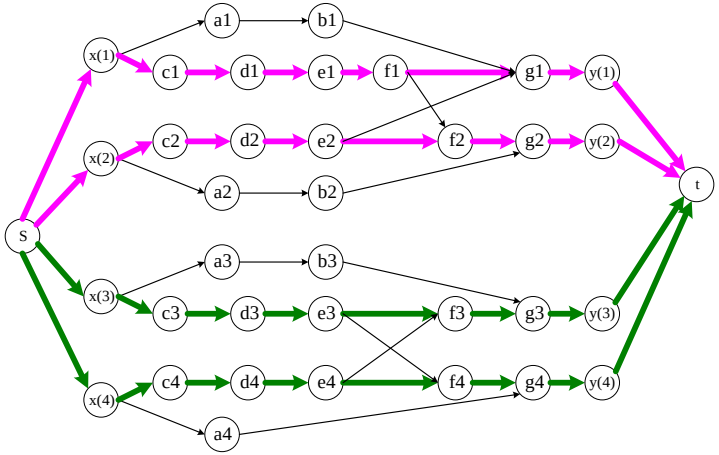


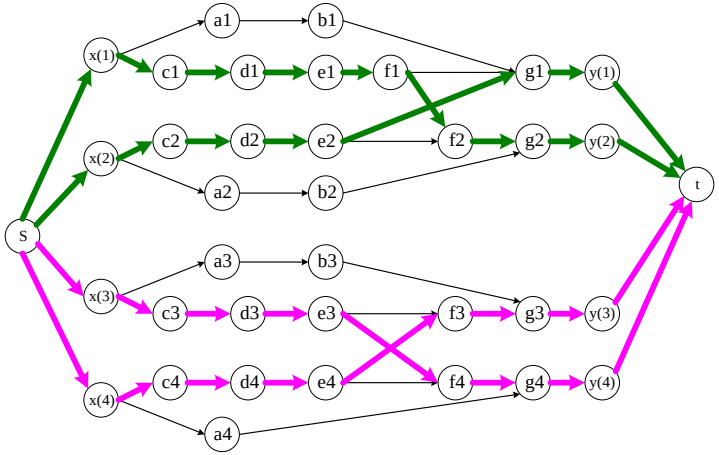
Group-Piece based Flow Creation





Partition flow solutions into groups

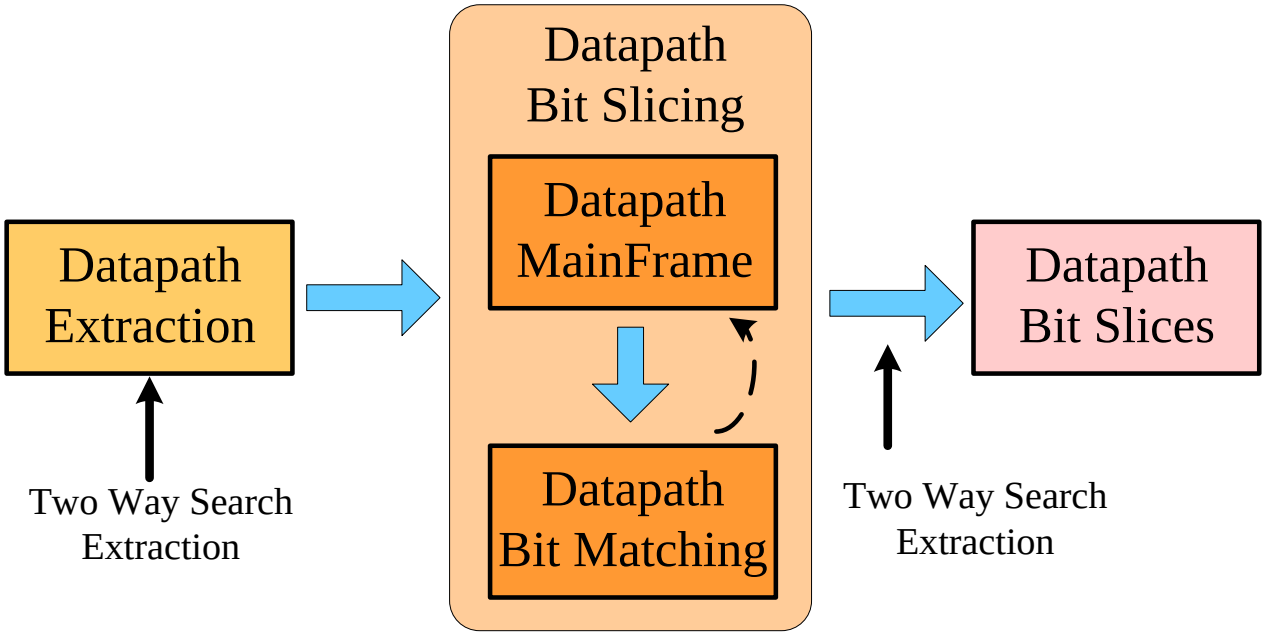




Piece groups from different solutions to create new flow solutions

Datapat Bit Slicing Algorithm

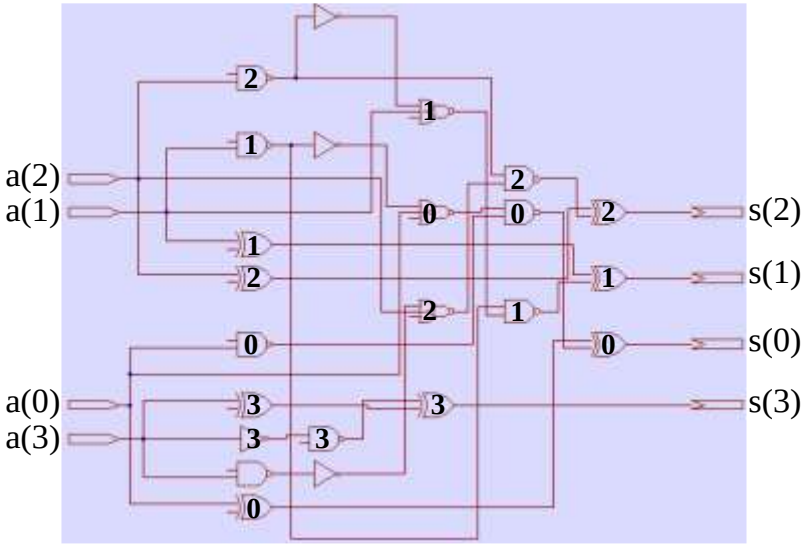
Algorithm Datapath_Bit_Slicing(S, T, $Iters$)
1. G_d = Two_Way_Search_Extraction(S, T);
2. ($InitFlow$, $InitMatch$) = MFI_by_Flow(S, T, G_d);
3. $Slices$=Iter_Improve($Iters$, S, T, G_f, G_d, $InitFlow$, $InitMatch$)
4. return $Slices$;



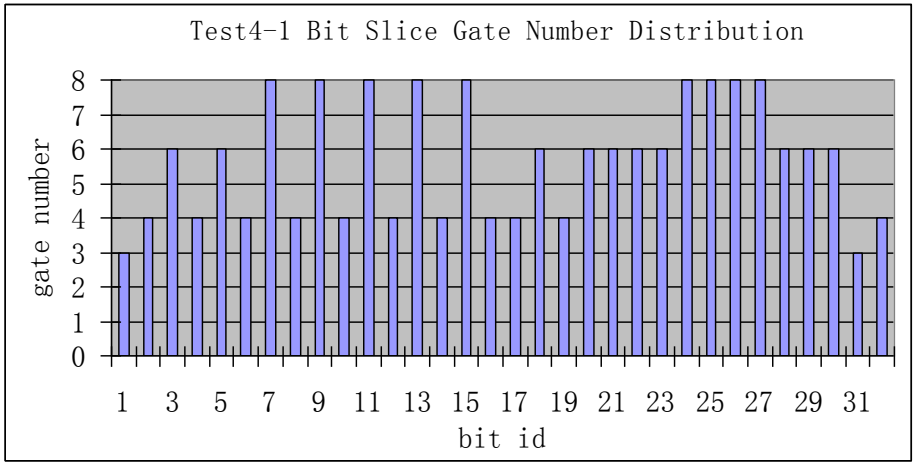
Experimental Results (I)

- **5 designs are created for testing**
- **All tests get perfect bit slicing**

Datapath	Width	Slices	MinSize	MaxSize	runtime
Test1-1	4	4	4	5	0.08s
Test2-1	4	4	4	5	0.09s
Test3-1	7	7	1	5	0.36s
Test4-1	32	32	3	8	6.5s
Test4-2	32	32	2	3	6.9s
Test4-3	32	32	2	3	8.6s



Test1-1



Experimental Results (II)

- **Seven testcases are derived from industrial designs**
- **Tested on a linux workstation (2.8GHz)**

Datapath	Width	Slices	MinSize	MaxSize	runtime
Test5-3	8	8	10	12	0.55s
Test5-4	64	64	8	10	2.58s
Test5-1	10	10	31	37	0.68s
Test5-2	8	8	6	14	0.57s
Test5-5	16	16	13	17	1.45s
Test6-1	64	64	9	13	0.28s
Test6-2	32	32	4	8	0.17s
Test6-3	64	64	5	7	0.22s
Test6-3	64	64	5	5	0.24s
Test6-4	56	56	6	6	0.19s
Test6-4	56	56	4	6	0.20s
Test7-1	56	56	3	7	0.18s

Conclusion

- **By converting datapath bit slicing problem to datapath main frame problem, the request for “similarity” definition is avoided.**
- **A flow network approach is proposed to optimally solve the datapath main frame problems.**
- **An iterative method is presented to create more optimal datapath main frame solutions to improve bit slicing solutions.**
- **An efficient two way search approach is developed to derive the full bit slices.**
- **Experiments on datapath macros give good bit slicing results.**
- **The datapath bit slicing results can be applied to datapath placement and latch bank planning.**