

ESE532: System-on-a-Chip Architecture

Day 4: September 13, 2017
Parallelism Overview



Today

- Types of Parallelism
- Compute Models

Message

- Many useful models for parallelism
 - Help conceptualize
- One-size does not fill all
 - Match to problem

Types of Parallelism

Types of Parallelism

- **Data Level** – Perform same computation on different data items
- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- **Instruction Level** – Within a single sequential thread, perform multiple operations on each cycle.

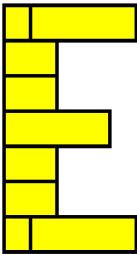
Pipeline Parallelism

- Pipeline – organize computation as a spatial sequence of concurrent operations
 - Can introduce new inputs before finishing
 - Instruction- or thread-level
 - Use for data-level parallelism
 - Can be directed graph

Build 1

Sequential

- Single person build E
- Latency?
- Throughput?



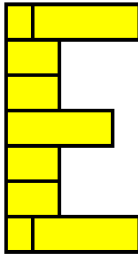
7

Penn ESE532 Fall 2017 -- DeHon

Build 2

Data Parallel

- Everyone in class build own E
- Latency?
- Throughput?
- Ideal speedup?
- Resource Bound?
 - 100 Es, 12 people
- When useful?



8

Penn ESE532 Fall 2017 -- DeHon

Data-Level Parallelism

- **Data Level** – Perform same computation on different data items
- Ideal: $T_{dp} = T_{seq}/P$

9

Penn ESE532 Fall 2017 -- DeHon

Build 3

Thread Parallel

- Each person build indicated letter
- Latency?
- Throughput?
- Speedup over sequential build?

10

Penn ESE532 Fall 2017 -- DeHon

Thread-Level Parallelism

- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- Ideal: $T_{dp} = T_{seq}/P$
- Can produce a diversity of calculations
 - Useful if have limited need for the **same** calculation

11

Penn ESE532 Fall 2017 -- DeHon

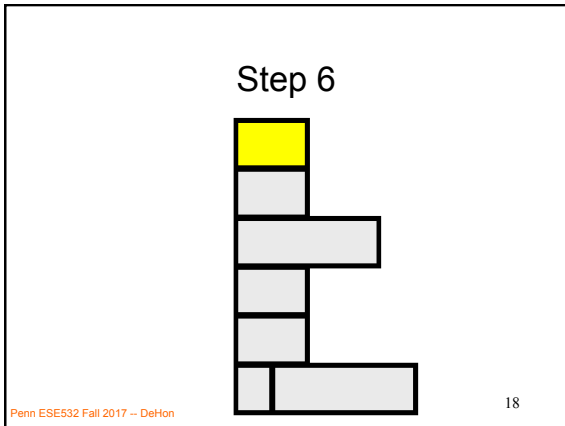
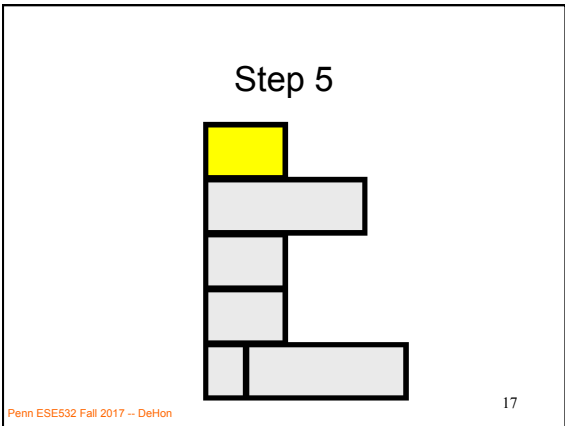
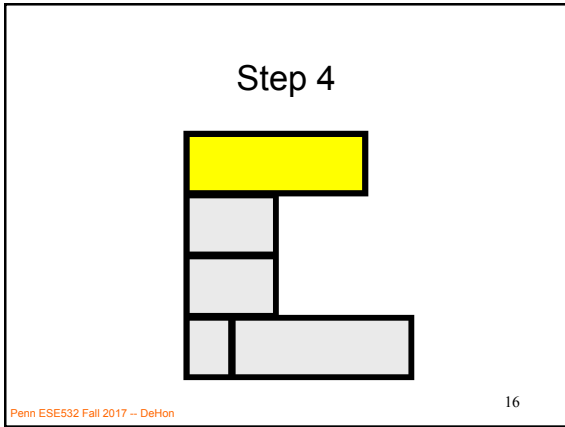
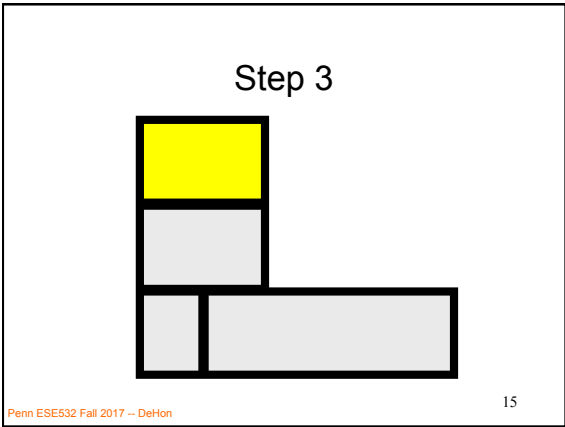
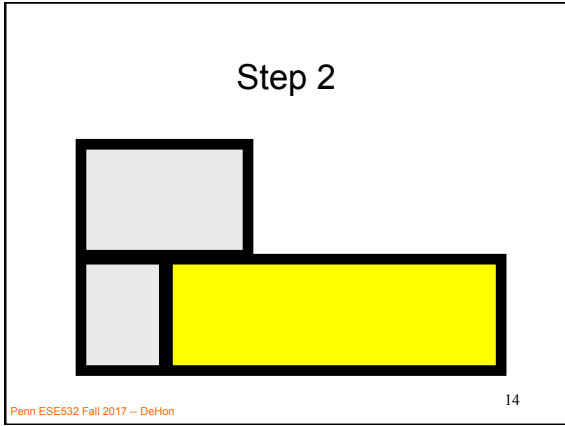
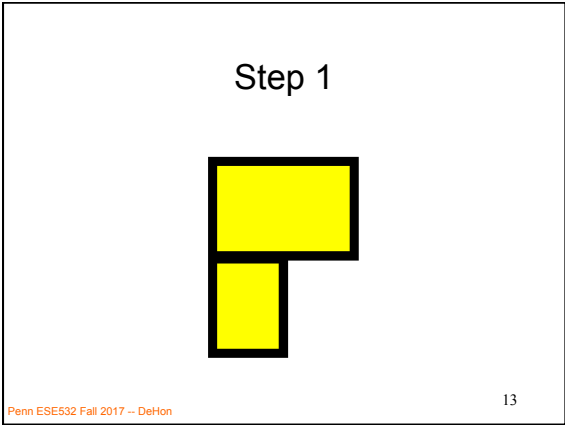
Build 4

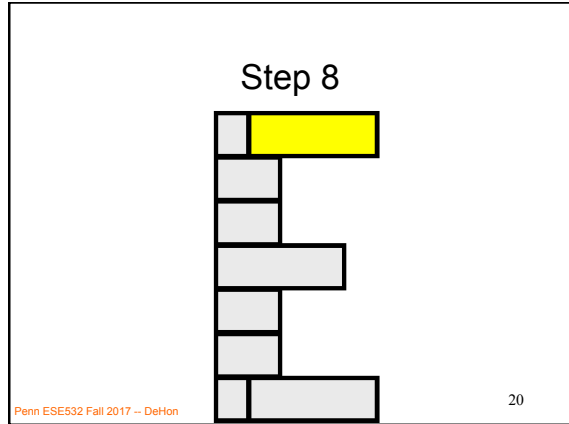
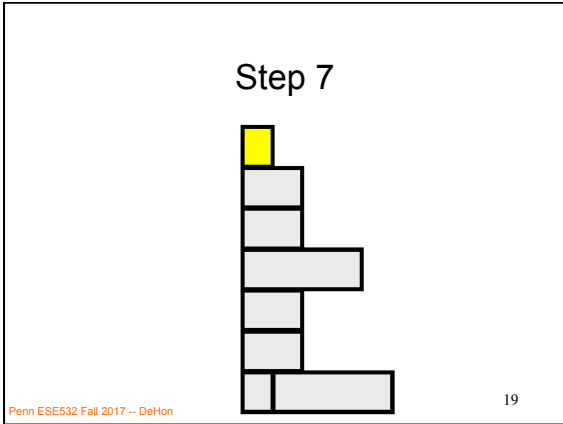
Instruction-Level Data Parallel

- Build E together in lock step
- Instructions from slides
 - To get proper flavor, please stay together
 - (do not build ahead)

12

Penn ESE532 Fall 2017 -- DeHon





Instruction-Level Data Parallel

- Latency?
- Throughput?

21

Penn ESE532 Fall 2017 -- DeHon

Instruction-Level Data Parallel

- **Instruction Level** – Within a single sequential thread, perform multiple operations on each cycle.
- + Data parallel
 - Able to share instructions

22

Penn ESE532 Fall 2017 -- DeHon

Build 5

Instruction-Level Parallelism

- Build word in lock step
- Announce step from slides
 - Don't build ahead
- Consult local instructions on what to do on step

23

Penn ESE532 Fall 2017 -- DeHon

Step 0

24

Penn ESE532 Fall 2017 -- DeHon

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

Step 7

Step 8

Instruction-Level Parallelism

- Latency?
- Throughput?

Instruction-Level Data Parallel

- **Instruction Level** – Within a single sequential thread, perform multiple operations on each cycle.
- Able to perform different calculations concurrently
- Requires local instructions

Build 6

Instruction-Level Parallelism

- Build single letter in lock step
- 4 groups of 3
- Resource Bound for 3 people building 9-brick letter?
- Announce steps from slide
 - Stay in step with slides

Group Communication

- Groups of 3
- Note who was person 1 task
- 2, 3 will need to pass completed substructures

Step	Person 1	Person 2	Person 3
0			
1			
2			
3			

Step 0

Step 1

Step 2

Step 3

Instruction-Level Parallelism

- Latency?
- Throughput?
- Can reduce latency for single letter
- Ideal: $T_{\text{latency}} = T_{\text{seqlatency}}/P$

Build 7

Instruction-Level Pipeline

- Each person adds one brick to build
- Run pipeline once alone
- Latency?
- Then run pipeline with 5 inputs
- Throughput?

Thread Pipeline

- Each person builds letter and adds to work
- Identify where pipeline flows
- Run once
- Latency?
- Throughput?

Thread Graph

- Each person build designated sub-assembly and pass off
- Who gets E, S?
- Who gets 3, 2?
- Who gets E, 5, and ES, 32 sub-assemblies?
- Run once
- Latency?
- Throughput?

Parallel Compute Models

Sequential Control Flow

- Control flow**
- Program is a sequence of operations
 - Operation reads inputs and writes outputs into common store
 - One operation runs at a time
 - defines successor
- Model of correctness is sequential execution
- Examples
C (Java, ...)
FSM / FA

Parallelism can be explicit

- ILP Build example
- Coordinate data parallel operations
- Multiply, add for quadratic equation

cycle	mpy	add
1	B,x	
2	x,x	(Bx)+C
3	A,x ²	
4		Ax ² +(Bx+C)

- Coordinate ILP

Parallelism can be implicit

- Sequential expression
 - Infer data dependencies
- $T1=x*x$
 $T2=A*T1$
 $T3=B*x$
 $T4=T2+T3$
 $Y=C+T4$
- Or
- $Y=A*x*x+B*x+C$

Implicit Parallelism

- $d=(x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)$
- What parallelism exists here?

Parallelism can be implicit

- Sequential expression
 - Infer data dependencies
- ```
for (i=0;i<100;i++)
 y[i]=A*x[i]*x[i]+B*x[i]+C
```
- Why can these operations be performed in parallel?

## Term: Operation

- **Operation** – logic computation to be performed

## Dataflow / Control Flow

### Dataflow

- Program is a graph of operations
- Operation consumes **tokens** and produces tokens
- All operations run concurrently

### Control flow (e.g. C)

- Program is a sequence of operations
- Operation reads inputs and writes outputs into common store
- One operation runs at a time
  - defines successor

## Token

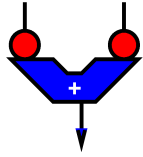
- Data value with presence indication
  - May be conceptual
    - Only exist in high-level model
    - Not kept around at runtime
  - Or may be physically represented
    - One bit represents presence/absence of data

## Token Examples?

- What are familiar cases where data may come with presence tokens?
  - Network packets
  - Memory references from processor
    - Variable latency depending on cache presence
  - Start bit on serial communication

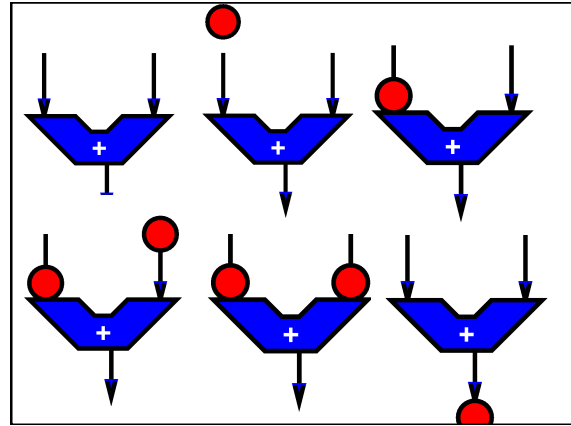
## Operation

- Takes in one or more inputs
- Computes on the inputs
- Produces results
- Logically **self-timed**
  - “Fires” only when input set present
  - Signals availability of output



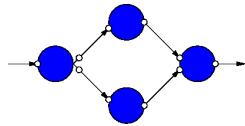
Penn ESE532 Fall 2017 -- DeHon

55



## Dataflow Graph

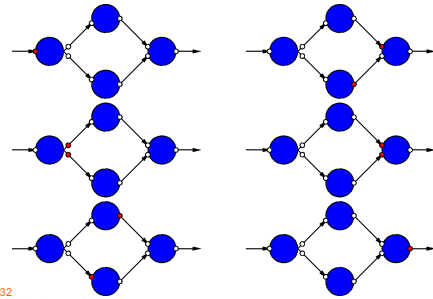
- Represents
  - computation sub-blocks
  - linkage
- Abstractly
  - controlled by data presence



Penn ESE532 Fall 2017 -- DeHon

57

## Dataflow Graph Example



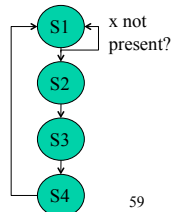
Penn ESE532

58

## Sequential / FSM

- FSM is degenerate dataflow graph where there is exactly one token

| cycle | mpy              | add                     | next               |
|-------|------------------|-------------------------|--------------------|
| S1    | B,x              |                         | x-->S2,<br>else S1 |
| S2    | x,x              | (Bx)+C                  | S3                 |
| S3    | A,x <sup>2</sup> |                         | S4                 |
| S4    |                  | Ax <sup>2</sup> +(Bx+C) | S1                 |



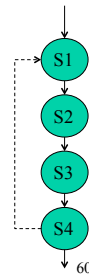
Penn ESE532 Fall 2017 -- DeHon

59

## Sequential / FSM

- FSM is degenerate dataflow graph where there is exactly one token

| cycle | mpy              | add                     | next               |
|-------|------------------|-------------------------|--------------------|
| S1    | B,x              |                         | x-->S2,<br>else S1 |
| S2    | x,x              | (Bx)+C                  | S3                 |
| S3    | A,x <sup>2</sup> |                         | S4                 |
| S4    |                  | Ax <sup>2</sup> +(Bx+C) | S1                 |



Penn ESE532 Fall 2017 -- DeHon

60

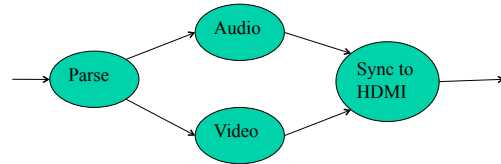
## Communicating Threads

- Computation is a collection of sequential/control-flow “threads”
- Threads may communicate
  - Through dataflow I/O
  - (Through shared variables)
- View as hybrid or generalization
- CSP – Communicating Sequential Processes → canonical model example

Penn ESE532 Fall 2017 -- DeHon

61

## Video Decode

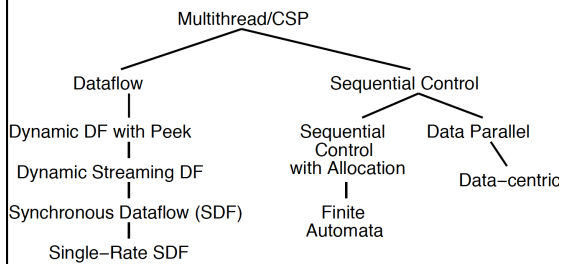


- Why might need to synchronize to send to HDMI?

Penn ESE532 Fall 2017 -- DeHon

62

## Compute Models



Penn ESE532 Fall 2017 -- DeHon

63

## Value of Multiple Models



- When you have a big enough hammer, everything looks like a nail.
- Many stuck on single model
  - Try to make all problems look like their nail
- Value to diversity / heterogeneity
  - One size does not fit all

Penn ESE532 Fall 2017 -- DeHon

64

## Big Ideas

- Many parallel compute models
  - Sequential, Dataflow, CSP
- Find natural parallelism in problem
- Mix-and-match

Penn ESE532 Fall 2017 -- DeHon

65

## Admin

- Reading Day 5 ...
- HW2 due Friday

Penn ESE532 Fall 2017 -- DeHon

66