ESE532: System-on-a-Chip Architecture

Day 5: January 30, 2017
Dataflow Process Model

Today

Dataflow Process Model

- Motivation
- Issues
- Abstraction
- Recommended Approach

Message

- Parallelism can be natural
- Discipline to avoid common pitfalls
  - Maintain determinism
    - …as much as possible
- Identify rich potential parallelism
- Abstract out implementation details
  - Admit to many implementations

Process

- Abstraction of a processor
- Looks like each process is running on a separate processor
- Has own state, including
  - Program Counter (PC)
  - Memory
  - Input/output
- May not actually run on processor
  - Could be specialized hardware block

Thread

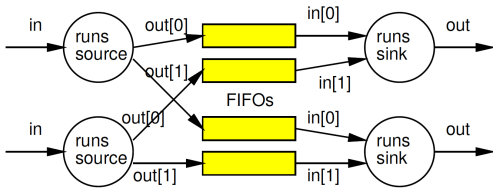
- Has a separate locus of control (PC)
- May share memory
  - Run in common address space with other threads
- For today – no shared memory
  - (technically no threads)

FIFO

- First In First Out
- Delivers inputs to outputs in order
- Data presence
  - Consumer knows when data available
- Back Pressure
  - Producer knows when at capacity
    - Typically stalls
- Decouples producer and consumer
  - Hardware: maybe even different clocks

Preclass 1

- Value of separate processes?



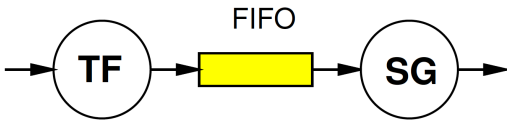
Preclass 2

- Compute Data Parallel GCD
- Why challenge for SIMD implementation?
- What benefit get from multiple processes (processors) running Data Parallel GCD?

- while(a!=b)
  - a=max(a,b)-min(a,b)
  - a=min(a,b)
- return(a);

Preclass 3

- How long to process each input?
- Correlation in delays?
- What benefit from FIFO and processes?



Process

- Processes allow expression of independent control
- Convenient for things that advance independently
- Performance optimization resource utilization

Issues

- **Communication** – how move data between processes?
- **Synchronization** – how define how they advance relative to each other?
- **Determinism** – for the same inputs, do we get the same outputs?

Today's Stand

- Communication – FIFO-like channels
- Synchronization – dataflow with FIFOs
- Determinism – how to achieve
  - …until you must give it up.

Dataflow Process Model

Operation/Operator

- **Operation** – logic computation to be performed
  - A process that communicates through dataflow inputs and outputs
- **Operator** – physical block that performs an Operation

Dataflow / Control Flow

Dataflow

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

Control flow (e.g. C)

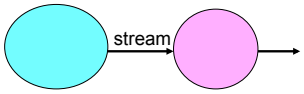
- Program is a sequence of operations
- Operation reads inputs and writes outputs into common store
- One operation runs at a time
  - defines successor

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

Stream

- Logical abstraction of a persistent point-to-point communication link between operators
  - Has a (single) source and sink
  - Carries data presence / flow control
  - Provides in-order (FIFO) delivery of data from source to sink

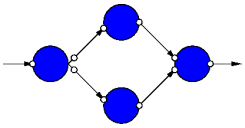


Streams

- Captures communications structure
  - Explicit producer→consumer link up
- Abstract communications
  - Physical resources or implementation
  - Delay from source to sink

Dataflow Process Network

- Collection of Operators
- Connected by Streams
- Communicating with Data Tokens



Dataflow Abstracts Timing

- Doesn't say
  - on which cycle calculation occurs
- Does say
  - What order operations occur in
  - How data interacts
    - i.e. which inputs get mixed together
- Permits
  - Scheduling on different # and types of resources
  - Operators with variable delay [examples?]
  - Variable delay in interconnect [examples?]

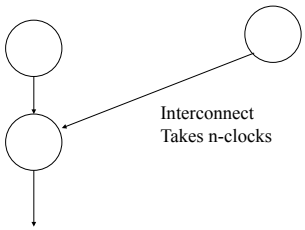
Operations

- Can be implemented on different operators with different characteristics
  - Small or large processor
  - Hardware unit
  - Different levels of internal
    - Data-level parallelism
    - Instruction-level parallelism
- May itself be described as
  - Dataflow process network, sequential, hardware register transfer language

Examples

- Operators with Variable Delay
  - Cached memory or computation
  - Shift-and-add multiply
  - Iterative divide or square-root
- Variable delay interconnect
  - Shared bus
  - Distance changes
    - Wireless, longer/shorter cables
  - Computation placed on different cores

Clock Independent Semantics



Semantics

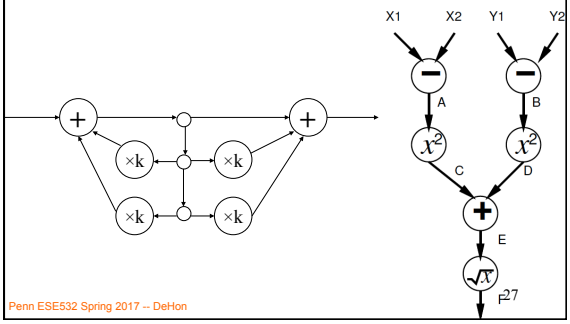
- Need to implement semantics
  - *i.e.* get same result as if computed as indicated
- But can implement any way we want
  - That preserves the semantics
  - Exploit freedom of implementation

Dataflow Variants

Synchronous Dataflow (SDF)

- Particular, restricted form of dataflow
- Each operation
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - When full set of inputs are available
    - Can produce output
  - Can fire any (all) operations with inputs available at any point in time

Synchronous Dataflow

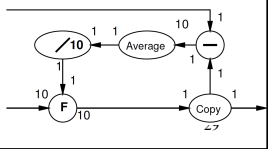


SDF: Execution Semantics

while (true)
  Pick up any operator
  If operation has full set of inputs
    Compute operation
    Produce outputs
    Send outputs to consumers

Multirate Synchronous Dataflow

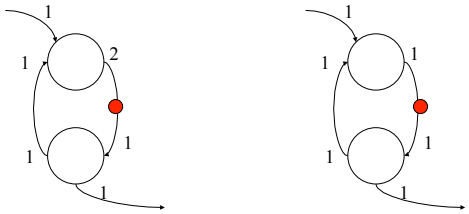
- Rates can be different
  - Allow lower frequency operations
  - Communicates rates to tools
    - Use in scheduling, provisioning
  - Rates must be constant
    - Data independent



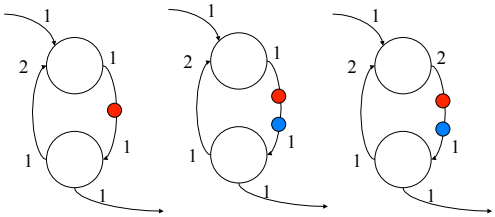
SDF

- Can validate flows to check legal
  - Like KCL → token flow must be conserved
  - No node should
    - be starved of tokens
    - Collect tokens
- Schedule operations onto processing elements
  - Provisioning of operators
- Provide real-time guarantees
- Compute required depth of all buffers
- Model restrictions → analysis power
- Simulink is SDF model

SDF: good/bad graphs



SDF: good/bad graphs

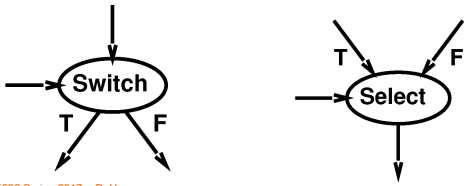


Dynamic Rates?

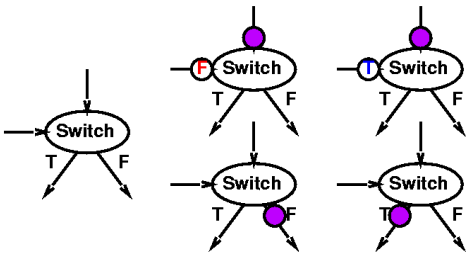
- When might static rates be limiting? (prevent useful optimizations?)
  - Compress/decompress
    - Lossless
    - Even Run-Length-Encoding
  - Filtering
    - Discard all packets from spamRus
  - Anything data dependent

Data Dependence

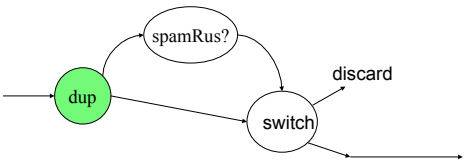
- Add Two Operators
  - Switch
  - Select



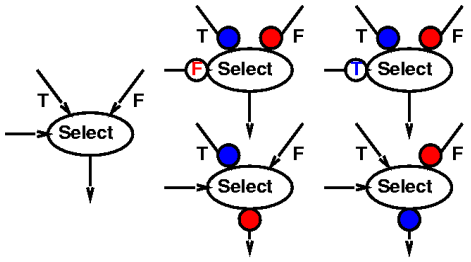
Switch



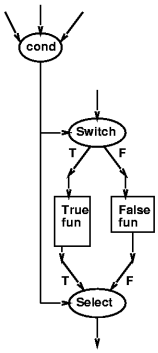
Filtering Example



Select



Constructing If-Then-Else

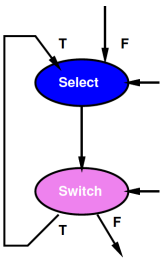


In-Order Merge

- **Task:** Merge to ordered streams in order onto a single output stream
  - Key step in merge sort
- Use to illustrate switch/select

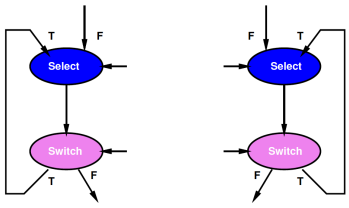
Idiom to Selectively Consume Input

- Hold onto current head on loop
  - Shown left here
  - With T-side control



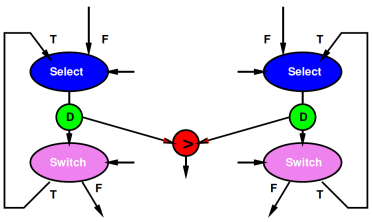
In-Order Merge

- Use one for each of the two input streams



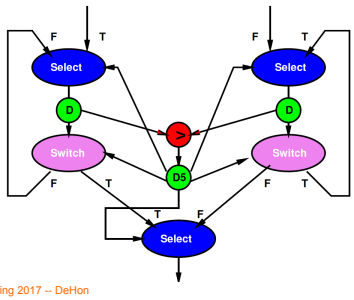
In-Order Merge

- Perform Comparison



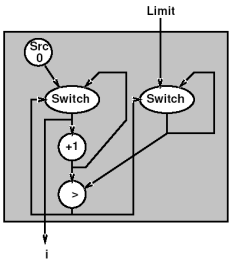
In-Order Merge

- Act on result of comparison



Looping

- for (i=0;i<Limit;i++)



Universal

- Once we add switch and select, the dataflow model is as powerful as any other
  - E.g. can do anything we could do in C
  - "Turing Complete" in formal CS terms

Dynamic Challenges

- In general, cannot say
  - If a graph is well formed
    - Will not deadlock
  - How many tokens may have to buffer in stream
    - Will not *bufferlock*
      - deadlock on finite buffer size
      - When not deadlock on unbounded buffer
  - Right proportion of operators for computation
- More powerful model → weaker analysis
  - Larger burden to guarantee correctness

Deterministic

- As described so far,
  - Timing-independent
- Always gets same answer
  - Regardless of operator and stream delays
- Execution *time* can be data and implementation dependent

No Peaking

- Key to determinism: behavior doesn't depend on timing
  - Cannot ask if a token is present
- If (not_empty(in))
  - Out.put(3);
- Else
  - Out.put(2);

When peaking necessary?

- What are cases where we need the ability to ask if a data item is present?
  - Preclass 1
  - User Input
  - Server

When Peak Optimization?

- What are cases where asking about data presence might allow performance optimization?
  - Process data as soon as arrives

Peaking

- Removed model restriction (no peaking)
- Gained expressive power
  - Can grab data as shows up
- Weaken our guarantees
  - Possible to get non-deterministic behavior

Data-Dependent Parallelism

- Sequential code
  - With malloc(), new()
  - Size of data determines memory footprint, size of structures
  - Amount of computation performed
- Here process graph may want to match data structure
  - Add a process spawn

Data-Dependent Parallelism

- Where might benefit having data-dependent parallelism?
  - Searching variable number of targets
  - Simulation

Data-Independent Parallelism

- What happens if we cannot spawn?
- Why likely to be less important for a particular SoC target?

Approach

Approach (1)

- Identify natural parallelism
- Convert to streaming flow
  - Initially leave operators software
  - Focus on correctness
- Identify flow rates, computation per operator, parallelism needed
- Refine operators
  - Decompose further parallelism?
  - E.g. SIMD changes making now
  - model potential hardware

Approach (2)

- Refine coordination as necessary for implementation
- Map operators and streams to resources
  - Provision hardware
  - Scheduling: Map operations to operators
  - Memories, interconnect
- Profile and tune

Big Ideas

- Capture gross parallel structure with Process Network
- Use dataflow synchronization for determinism
- Abstract out timing of implementations
  - Give freedom to optimize implementation for performance
- Minimally use non-determinism as necessary

Admin

- Reading for Day 6 on web
- HW3 due Friday
- Expanded feedback incl. HW2