Halmstad University, Sweden, June 10, 2009

The Dataflow Interchange Format: An Environment for Integrating and Transforming DSP-Oriented Dataflow Models of Computation

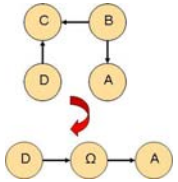
Shuvra S. Bhattacharyya

Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies

University of Maryland, College Park, MD, USA
ssb@umd.edu, http://www.ece.umd.edu/~ssb/

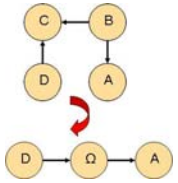
With contributions from Chia-Jui Hsu, Mary Kiemb, William Plishker, Sankalita Saha ,and Nimish Sane

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING



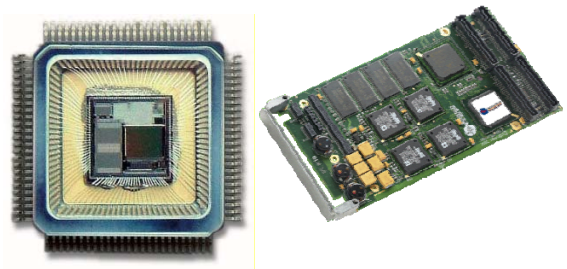
Outline

- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research

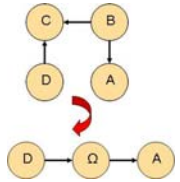


Introduction

- Implementation of digital signal processing applications on SoCs is a multi-faceted problem
- Typical design flow consists of several complex steps
- Implementation constraints and objectives are multidimensional and complex
- Dataflow graph transformations provide an effective class of techniques in the design flow for more effective solutions

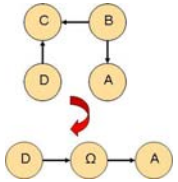


Dataflow-based Design: Related Trends



- Dataflow-based design (in our context) is a specific form of **model-based design**
- Dataflow-based design is complementary to
  - Object-oriented design
  - DSP C compiler technology
  - Synthesis tools for hardware description languages (e.g., Verilog and VHDL)

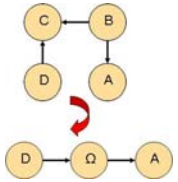
Model-Based Design for Embedded Systems



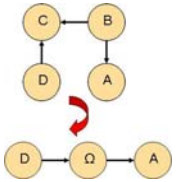
- High level application subsystems are specified in terms of components that interact through formal models of computation
  - C can be used to specify intra-component behavior
  - Object-oriented techniques can be used to maintain libraries of components
- Popular models for embedded systems
  - Dataflow and KPNs (Kahn process networks)
  - Synchronous languages
  - Continuous time
  - Discrete Event
  - FSM and related control formalisms

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

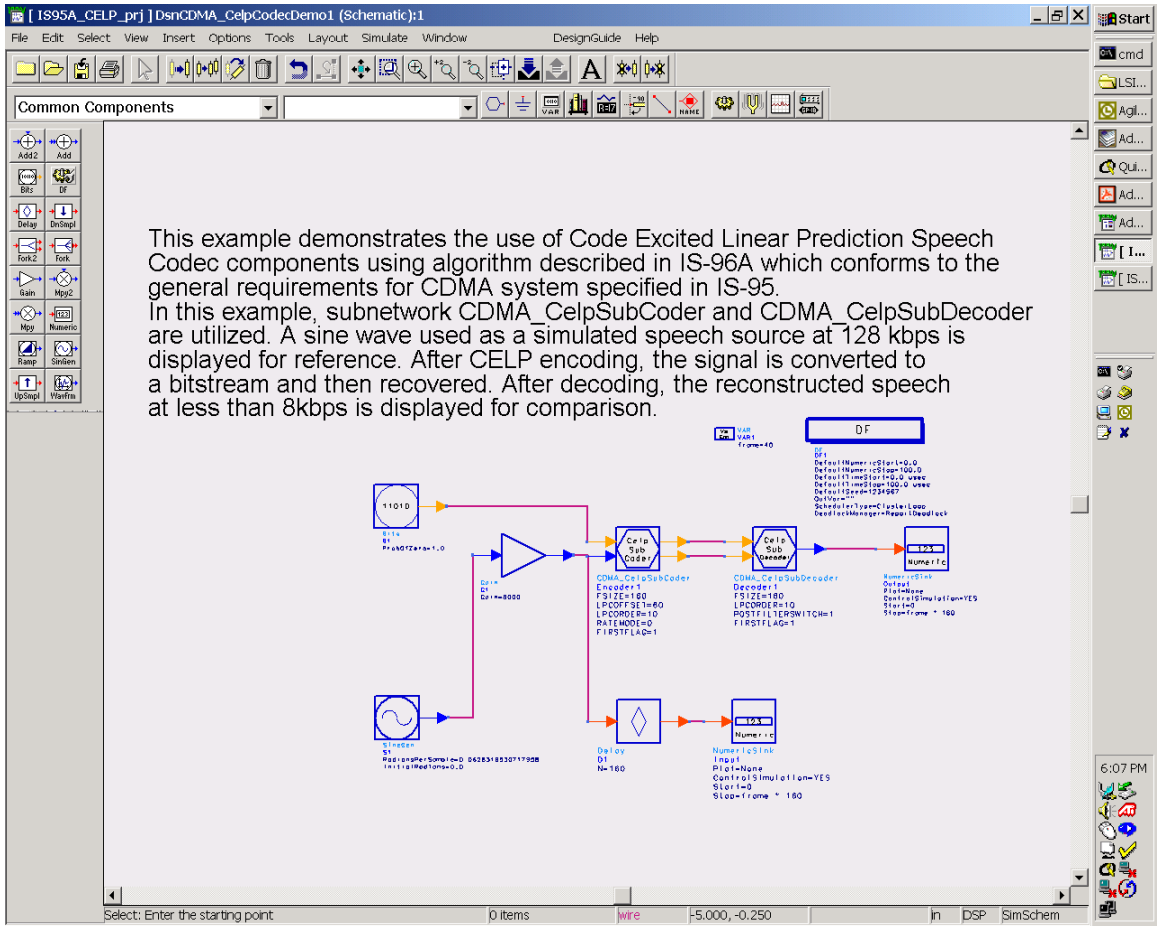
Dataflow Models of Computation



- Used widely in design tools for DSP
- Application is modeled as a directed graph
  - Nodes (actors) represent functions
  - Edges represent communication channels between functions
  - Nodes produce and consume data from edges
  - Edges buffer data in a FIFO (first-in, first-out) fashion
- Data-driven execution model
  - An actor can execute whenever it has sufficient data on its input edges
  - The order in which actors execute is not part of the specification
  - The order is typically determined by the compiler, the hardware, or both
- Iterative execution
  - Body of loop to be iterated a large or infinite number of times



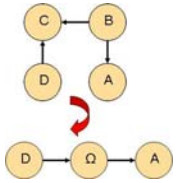
Example: Dataflow-based design for DSP

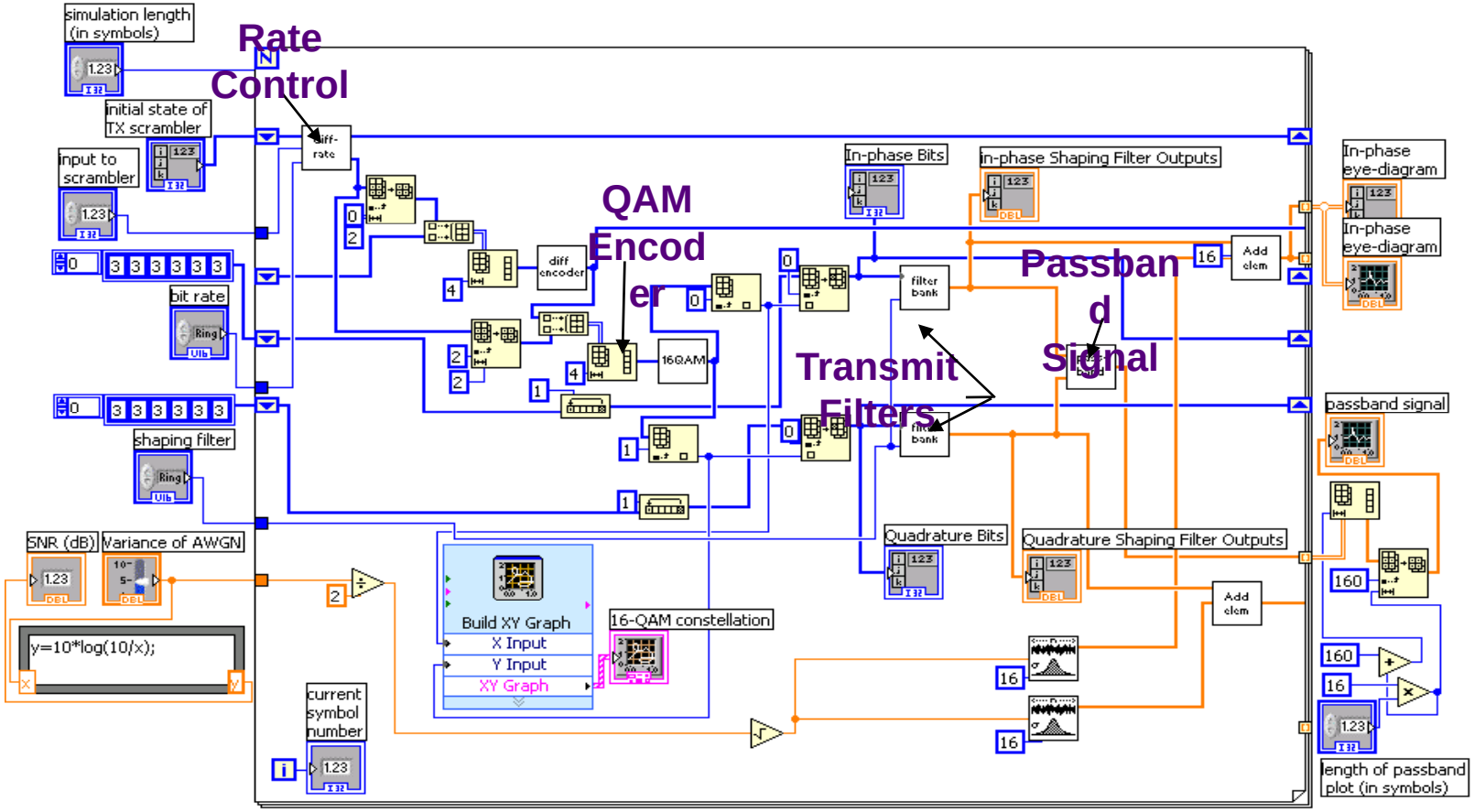


Example from Agilent ADS tool

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

Dataflow Example: QAM Transmitter in National Instruments LabVIEW

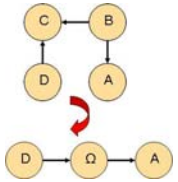




Source: [Evans 2005]

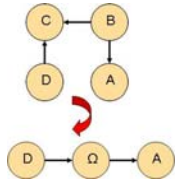
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Evolution of dataflow models of computation for DSP: Representative examples, 1



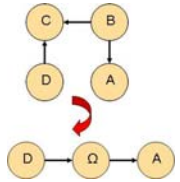
- Computation Graphs and Marked Graphs [Karp 1966, Reiter 1968]
- Synchronous dataflow, [Lee 1987]
  - Static multirate behavior
  - SPW (Cadence) , National Instruments LabVIEW, and others.
- Well behaved stream flow graphs [1992]
  - Schemas for bounded dynamics
- Boolean/integer dataflow [Buck 1994]
  - Turing complete models
- Multidimensional synchronous dataflow [Lee 1992]
  - Image and video processing
- Scalable synchronous dataflow [Ritz 1993]
  - Block processing
  - COSSAP (Synopsys)

Evolution of dataflow models of computation for DSP: Representative examples, 2



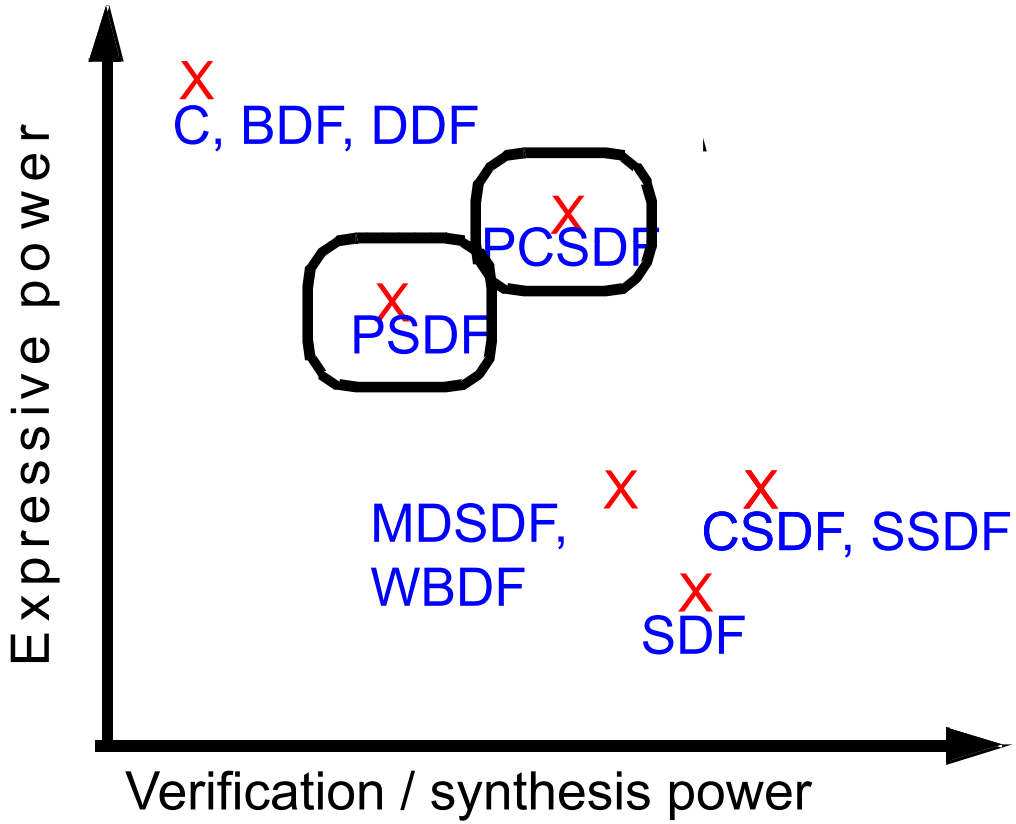
- CAL [Eker 2003]
  - Actor-based dataflow language
- Cyclo-static dataflow [Bilsen 1996]
  - Phased behavior
  - Eonic Virtuoso Synchro, Synopsys El Greco and Cocentric, Angeles System Canvas
- Bounded dynamic dataflow
  - Bounded dynamic data transfer [Pankert 1994]
- The processing graph method [Stevens, 1997]
  - Reconfigurable dynamic dataflow
  - U. S. Naval Research Lab, MCCI Autocoding Toolset
- Stream-based functions [Kienhuis 2001]

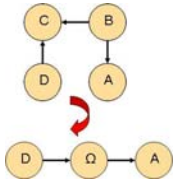
Evolution of dataflow models of computation for DSP: Representative examples, 3



- Parameterized dataflow [Bhattacharya 2001]
  - Reconfigurable static dataflow
  - Meta-modeling for more general dataflow graph reconfiguration
- Reactive process networks [Geilen 2004]
- Blocked dataflow [Ko 2005]
  - Image and video through parameterized processing
- Windowed synchronous dataflow [Keinert 2006]
- Parameterized stream-based functions [Nikolov 2008]
- Enable-invoke dataflow [Plishker 2008]
- Variable rate dataflow [Wiggers 2008]

Modeling Design Space

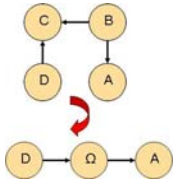




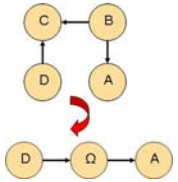
Outline

- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research

Transformations in DSP System Implementation



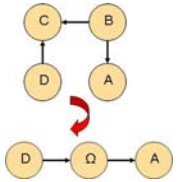
- High-level transformations have been studied in various contexts
  - High-level synthesis
  - Synthesis of DSP software
  - Fault detection in parallel processing systems
  - Transformation of high-level models of DSP applications such as dataflow graph transformations
    - Primarily explored in the context of synchronous dataflow and its important special classes



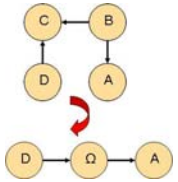
High Level Dataflow Transformations

- A well designed dataflow representation exposes opportunities for high level algorithm and architecture transformations.
- High level of abstraction → high implementation impact
- Dataflow representation is suitable both for behavior-level modeling, structural modeling, and mixed behavior-structure modeling
  - Transformations can be applied to all three types of representations to focus subsequent steps of the design flow on more favorable solutions
- Complementary to advances in
  - C compiler technology (intra-actor functionality)
  - Object oriented methods (library management, application service management)
  - HDL synthesis (intra-actor functionality)

Representative Dataflow Analyses and Optimizations

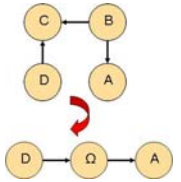


- Bounded memory and deadlock detection: consistency
- Buffer minimization: minimize communication cost
- Multirate loop scheduling: optimize code/data trade-off
- Parallel scheduling and pipeline configuration
- Heterogeneous task mapping and co-synthesis
- Quasi-static scheduling: minimize run-time overhead
- Probabilistic design: adapt system resources and exploit slack
- Data partitioning: exploit parallel data memories
- Vectorization: improve context switching, pipelining
- Synchronization optimization: self-timed implementation
- Clustering of actors into atomic scheduling units



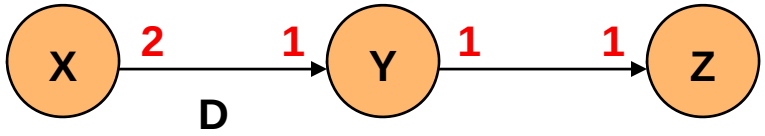
Outline

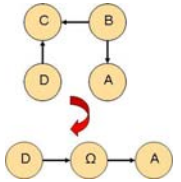
- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research



Background: Dataflow graphs

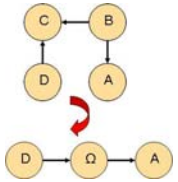
- Vertices (actors) represent computation
- Edges represent FIFO buffers
- Edges may have delays, implemented as initial tokens
- Synchronous dataflow (SDF)
  - The number of data values (tokens) produced and consumed by each actor is fixed





Parameterized Dataflow

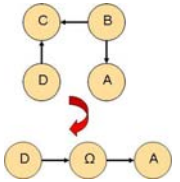
- A hierarchical meta-modeling technique based on a structured, hierarchical concept of reconfiguration
  - Application parameters
  - Parameter domains
  - Parameter configurations
  - Unspecified parameter values (configured at run-time)
  - Can be applied to architectural and behavioral attributes
- Exposes precise *locality* conditions and constraints for preserving useful subsystem properties throughout well-defined windows of time

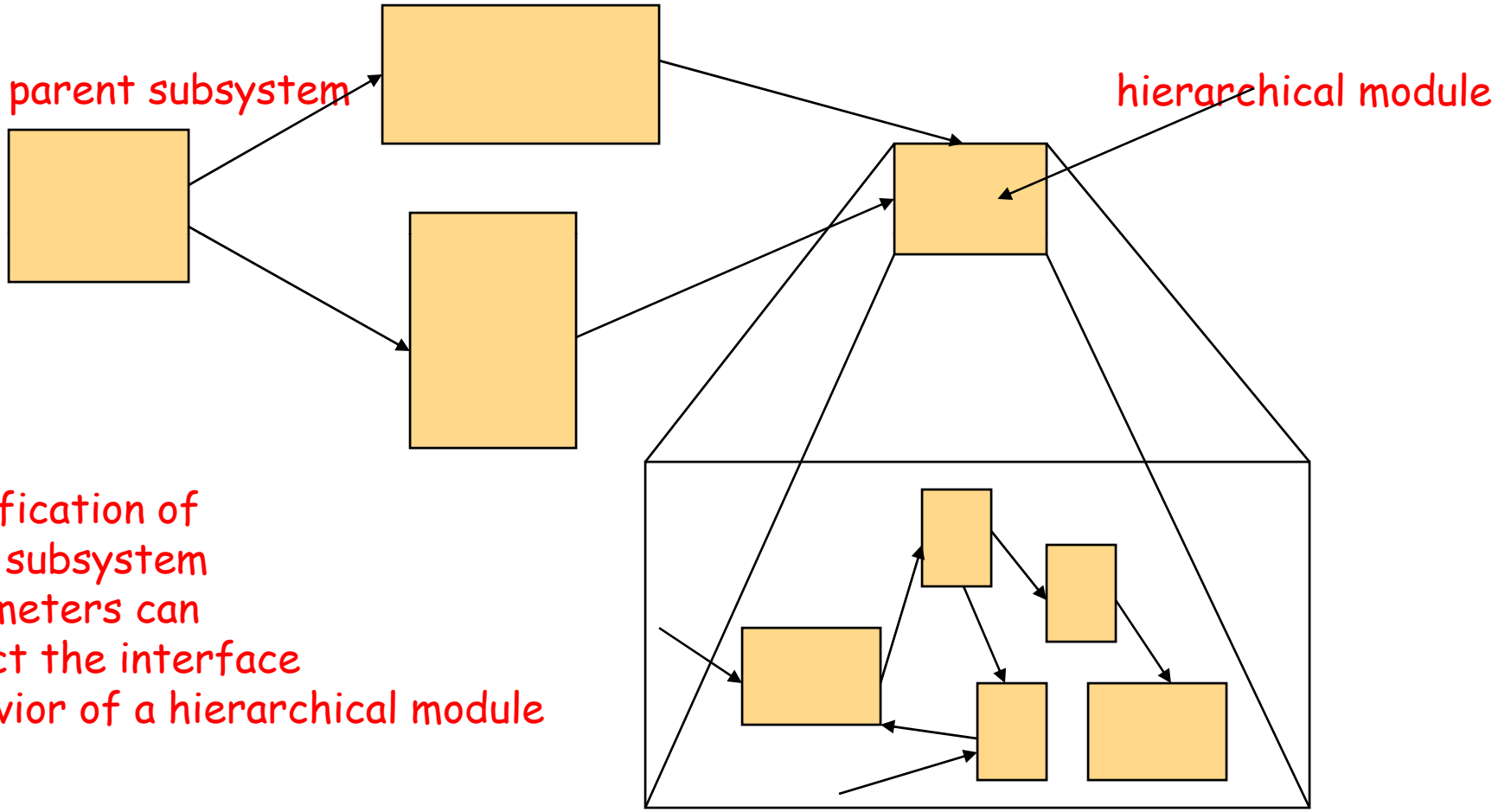


Parameterized Dataflow Terminology

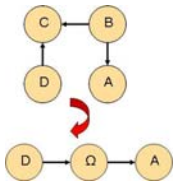
- Actors, edges, and subsystems can be parameterized, each with any number of parameters.
- Each parameter has an associated parameter *domain*, which can be used to understand the extent to which different aspects of the system can vary.
- When all actor, edge, and subsystem parameters are assigned specific values from their respective domains, this is called a configuration of the overall dataflow graph.
  - This can be viewed as an *instance* of the underlying parameterized dataflow graph.
- Parameterized dataflow graphs are reconfigurable in that their parameters can change dynamically.
  - This gives rise to a *sequence of dataflow graph instances* (configurations) in the execution of a given parameterized dataflow graph.

Reconfiguration of Subsystems in Parameterized Dataflow



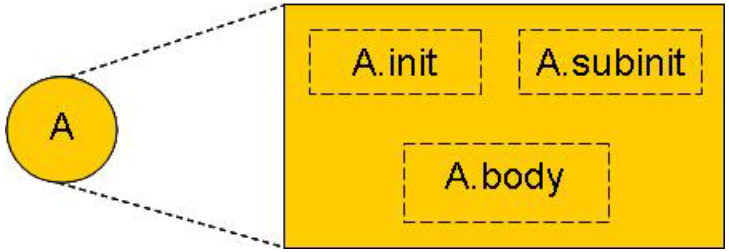


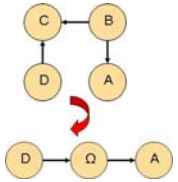
Modification of child subsystem parameters can affect the interface behavior of a hierarchical module



Parameterized SDF (PSDF)

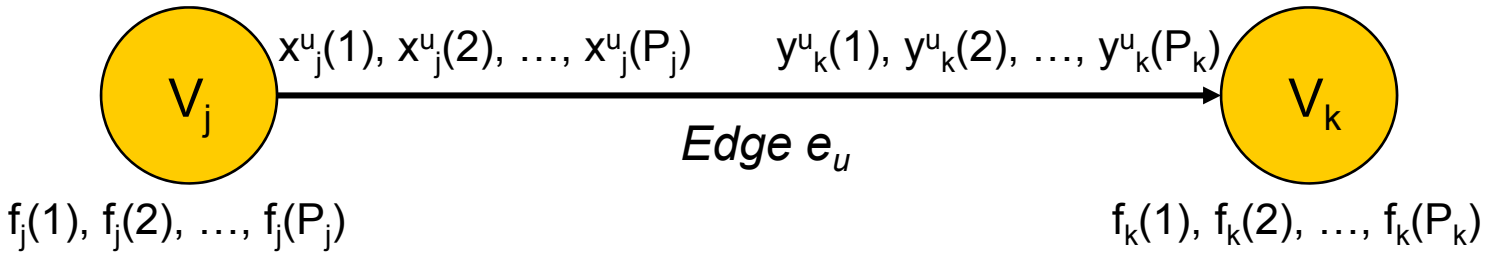
- A PSDF graph is obtained by parameterizing an SDF graph
- A PSDF graph is composed of PSDF actors and PSDF edges
- A PSDF actor is characterized by a set of parameters that can control the actor's functionality
- A PSDF subsystem consists of three distinct graphs
  - body graph: models the main functional behavior
  - init graph: for configuration of the graph with respect to iterations of the parent subsystem [more flexible / less frequent configuration]
  - subinit graph: for configuration of the body graph between subsequent iterations [less flexible / more frequent configuration]



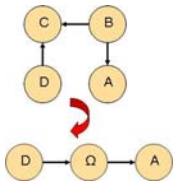


Cyclo-static Dataflow (CSDF)

- CSDF was introduced at K. U. Leuven [Bilsen 1996]
- CSDF is an important generalization of SDF (synchronous dataflow)
- Every actor v_j has an execution sequence $[f_j(1), f_j(2), ..., f_j(P_j)]$ of length P_j
- Production and consumption rates are also sequences in CSDF
- The production rate of vertex v_j on edge e_u, is represented as a sequence of constant integers $[x^u_j(1), x^u_j(2), ..., x^u_j(P_j)]$
- The consumption rate of vertex v_j on edge e_u, is represented as a sequence of constant integers $[y^u_k(1), y^u_k(2), ..., y^u_k(P_k)]$

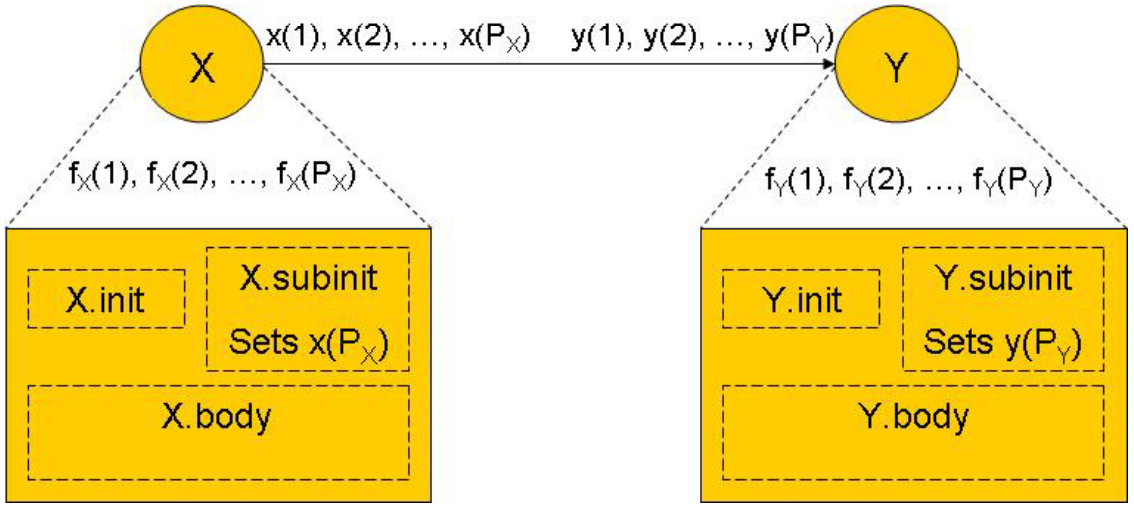


DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

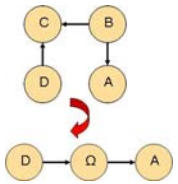


Parameterized CSDF (PCSDF)

- A PCSDF graph is obtained by parameterizing CSDF graphs
- The syntax and semantics are similar to those of PSDF
- There are two fundamental kinds of dataflow properties that can be parameterized
  - the period of the cycle of phases
  - the data rates associated with each phase

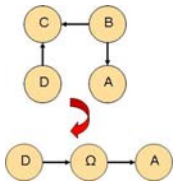


DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING



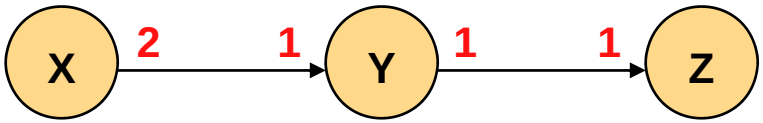
Dataflow Graphs: Scheduling

- A schedule may be static, dynamic or a combination of both
- Valid (periodic) schedule for an SDF graph G = (V,E)
  - Deadlock-free
  - Balance equations: $\forall e \in E$, prd(e) × $\boldsymbol{q}$[src(e)] = cns(e) × $\boldsymbol{q}$[snk(e)]
  - Repetitions vector $\boldsymbol{q}_G$: minimum positive integer solution
    - Repetition count $\boldsymbol{q}_G$[v]
  - Minimal periodic schedule
    - Each actor v is fired $\boldsymbol{q}_G$[v] times
- Static schedules are generally employed through infinite iteration of periodic schedules
  - Low overhead
  - High predictability
  - Large potential for optimization



Static Looped Schedules

- Iteration in DSP-oriented dataflow graphs is *implicit*
- Looped Schedule S = $L_1 L_2 \ldots L_N$
  - loop L = ($n\ T_1 T_2 \ldots T_m$)
  - Execute n times in succession the sequence $T_1 T_2 \ldots T_m$
  - T_i : an actor firing or a nested schedule loop

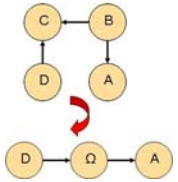


2 x **q**[X] = 1 x **q**[Y]
1 x **q**[Y] = 1 x **q**[Z]

q[X] =1, **q**[Y] = 2, **q**[Z] = 2

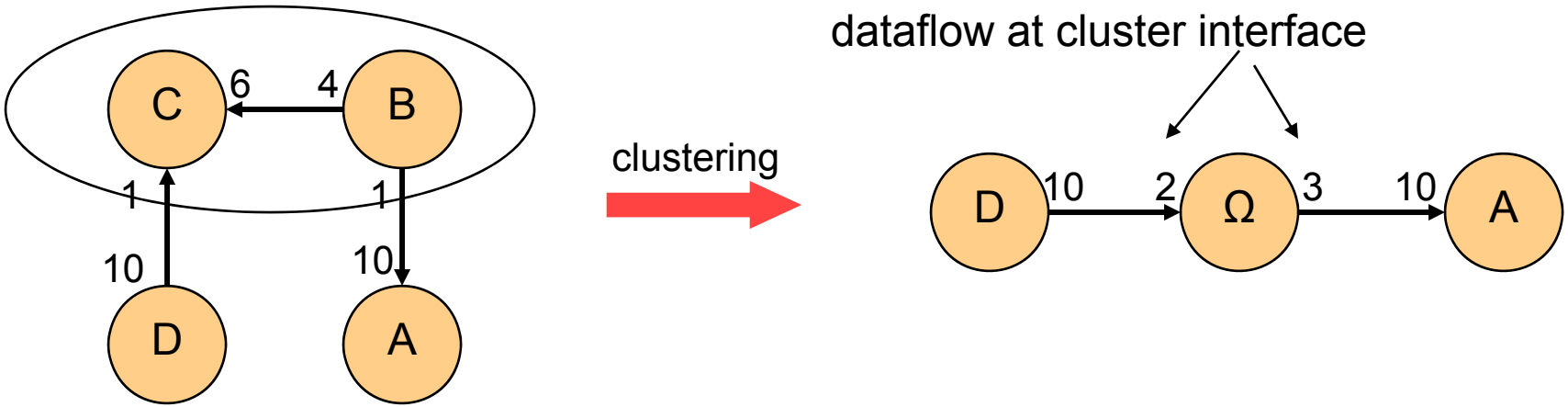
Looped schedule

Static periodic schedules → YXZYZ, XYZYZ, X(2 YZ)

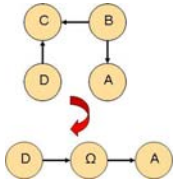


Dataflow *Clustering* Transformations

- Dataflow transformations may be applied before creating a schedule, as well as during the process of constructing a schedule
- A clustering transformation for a dataflow graph is one that replaces a set of multiple actors (actor phases) in the graph with a single actor (actor phase) such that there are no deadlocks



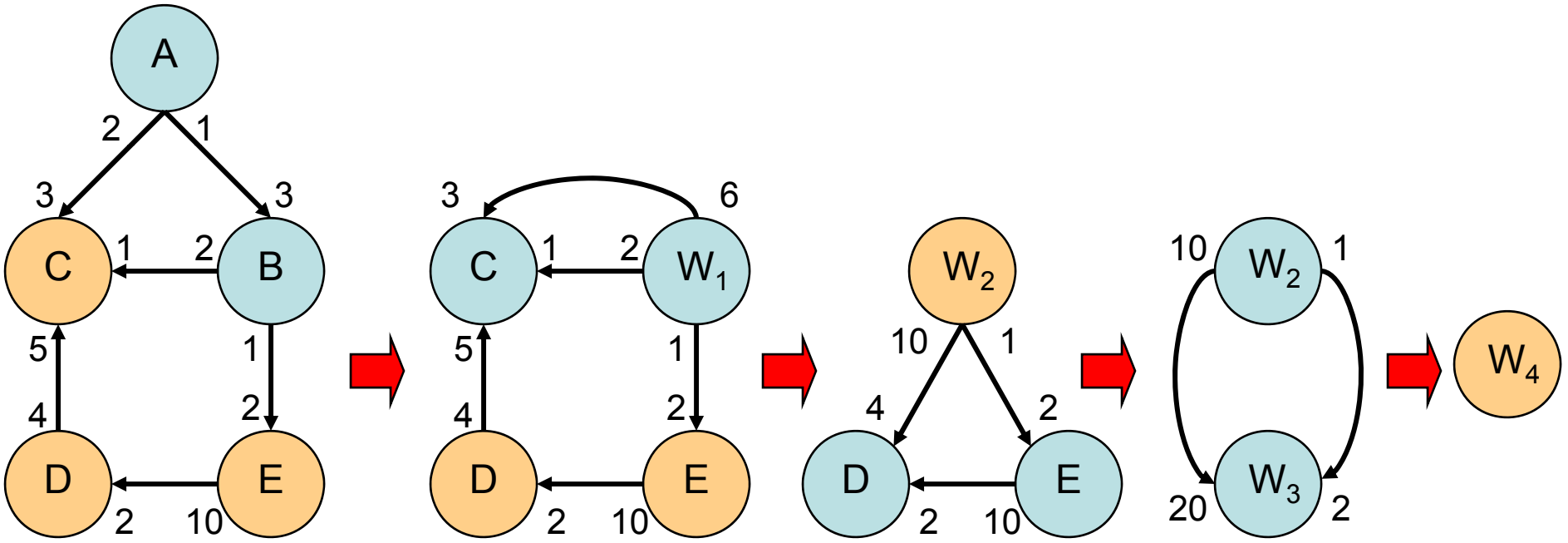
SDF clustering example



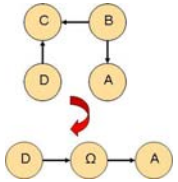
Clustering SDF Graphs using APGAN

- Single appearance schedules
  - A class of schedules for SDF graphs that permits inlined implementations of SDF graphs with minimum code size
  - Each actor is restricted to appear only once in the schedule
  - Thus, repetition of actors is achieved entirely through use of looping constructs
  - The number of possible single appearance schedules in general grows combinatorially in the size of the SDF graph
- Acyclic Pairwise Grouping of Adjacent Nodes (APGAN)
  - Develops a loop hierarchy by iteratively clustering two pairs of adjacent actor that maximize a clustering priority function at each step
  - After constructing the cluster hierarchy, APGAN retraces the clustering steps to translate the cluster hierarchy into a corresponding hierarchy of nested loops.

Example of clustering using APGAN



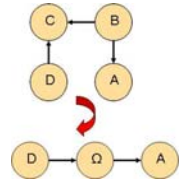
$(2((3A)B)(2C))\ (E(5D))$ ← $(2(W_1(2C)))\ (E(5D))$ ← $(2W_2)\ (E(5D))$ ← $(2W_2)\ W_3$ ← W_4



Optimality of APGAN

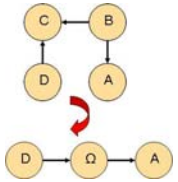
- The *buffer memory lower bound* (BMLB) for an SDF edge is a tight lower bound on the memory requirements for the edge over all valid single appearance schedules
- A *BMLB schedule* for an acyclic SDF graph is a single appearance schedule whose buffering cost is equal to the sum of the BMLB costs for the individual edges
- For a broad class of SDF graphs, the APGAN algorithm always finds a BMLB schedule whenever one exists
  - These schedules minimize data memory requirements over all minimum code size schedules.
- Many practical signal processing systems fall in the class for which APGAN finds optimal schedules

APGAN/PGAN as a general scheduling framework



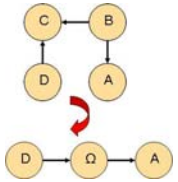
- The core principle of the (A)PGAN is the iterative construction of cluster hierarchies two (adjacent) actors at a time.
- These hierarchies can then be decomposed into corresponding hierarchies of nested loops to schedule the original dataflow graph
- The priority function used to select the next adjacent pair for clustering can be adapted based on the most relevant cost function
- This provides a general, priority-based scheduling framework that can be adapted to many different kinds of scenarios
- The methodology is analogous to the general framework of *list scheduling* for task graphs in high level synthesis, and multiprocessor scheduling

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING



Parameterized Looped Schedules

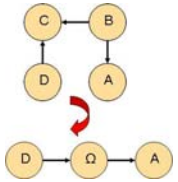
- For SDF, loop scheduling can be performed by using only loops that have constant and statically-known iteration counts.
- Loop scheduling for PSDF is challenging because the loop bounds generally involve symbolic expressions in terms of actor and subsystem parameters.
- PSDF loop scheduling can be performed efficiently by clustering the graph based on an adaptation of APGAN called parameterized APGAN (P-APGAN).
- This kind of scheduling targets a more general class of looped schedules called p**arameterized loop schedules**
  - Loop iteration counts can be parameterized expressions, where the associated parameter values can change dynamically as the schedule executes --- e.g., **(p_1 A) (3 B ($2p_2$ C))**
- This is an example of the adaptability of the APGAN framework in providing clustering techniques for a variety of models and objectives.



Outline

- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research

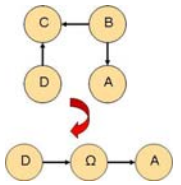
Dataflow Interchange Format (DIF): Objectives



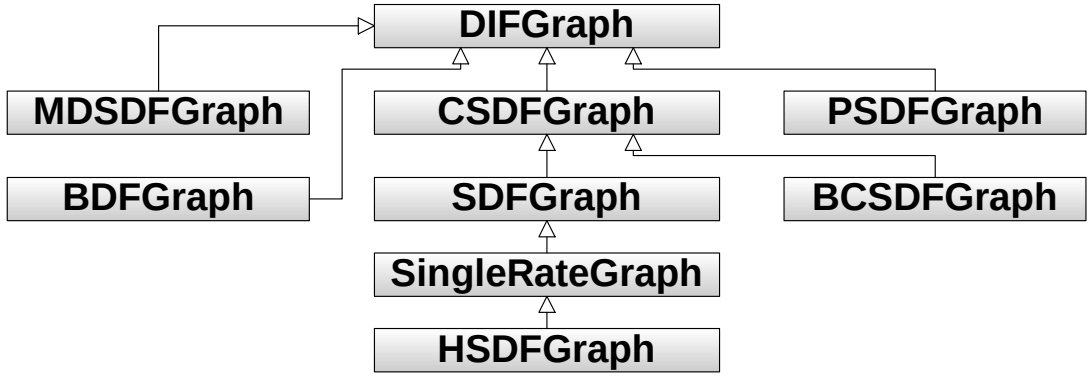
1. Design a standard language for specifying dataflow semantics.
   - Dataflow Interchange Format
   - Keywords associated with specific dataflow MoCs
2. Develop a software package for working with and developing dataflow models, and dataflow-oriented transformations
   - The DIF package
   - Functional DIF
3. Facilitate transferring DSP applications across dataflow-based design tools.
   - The DIF porting mechanism
4. Automate hardware/software implementation of DSP system designs from dataflow modeling specifications.
   - The DIF-to-C software synthesis framework
   - DIF-to-VSIPL synthesis capability

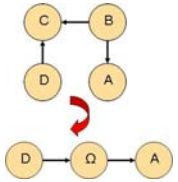
DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

The DIF Package



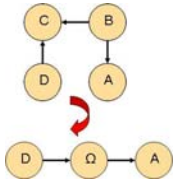
- DIF representation
  - Internal structures (Java classes) for representing and manipulating dataflow graphs.
- Algorithm implementation
  - Dataflow-based analysis, scheduling, and optimization.
- DIF front-end (language parser)
  - Translates between DIF specifications and DIF representations.
- Infrastructure
  - Porting
  - Software synthesis





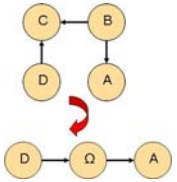
Outline

- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research



Functional DIF Overview

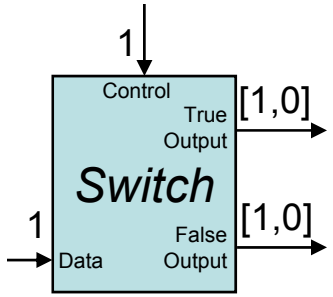
- Motivation
  - The DIF language has formal semantics for various forms of dataflow
  - The DIF intermediate representation is coupled with analyzers, a software synthesizer, and porting mechanisms to/from other design environments
  - Function DIF provides the DIF environment with inline functional simulation, and facilities rapid prototyping, and experimentation with modeling/transformation co-design
- Functional DIF extends DIF with
  - Natural actor description
  - Efficient interface for prototyping static, quasi-static, and dynamic schedulers, and heterogeneous schedulers for different specialized dataflow models
  - Semantic foundation for simulating non-deterministic and deterministic dataflow applications



Writing actors in Functional DIF

- Divide actors into sets of *modes*
  - Each mode has a fixed consumption and production behavior
- Write the enabling conditions for each mode
- Write the computation associated with each mode
  - Including next mode to enable and then invoke
- For example, consider a standard Switch:

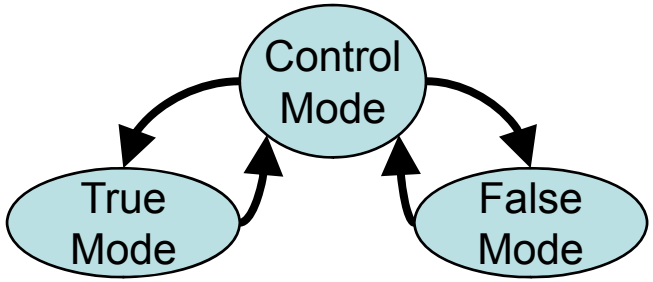
Switch Actor



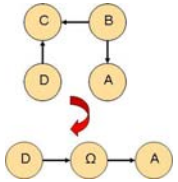
Production & consumption behavior of switch modes

mode	consumes		produces	
	Control	Data	True	False
Control	1	0	0	0
True	0	1	1	0
False	0	1	0	1

Mode transition diagram between switch modes



DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING



Semantic Foundation

- Enable-Invoke Dataflow (EIDF)
  - Enabling Function, ε, for an actor, a: $\varepsilon_a : (T_a \times M_a) \rightarrow B$,
    - T_a, the number of input tokens on each edge
    - M_a, the set of modes associated with actor a
    - B is the Boolean set of *{true, false}*
  - Invoking function, κ (Non-Deterministic)

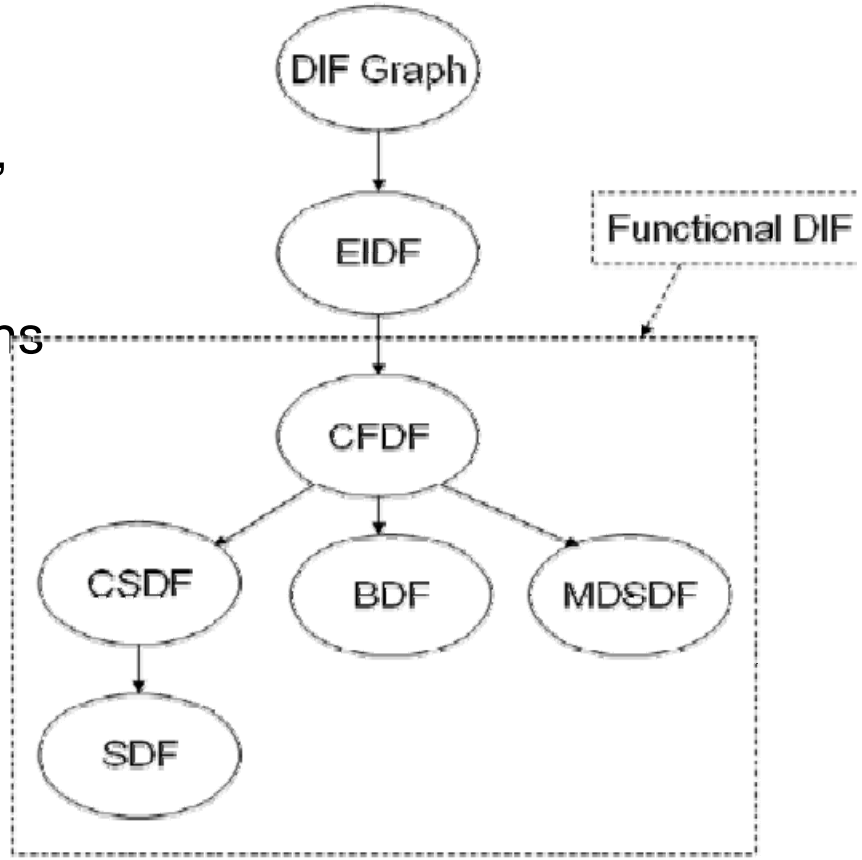
$$\kappa_a : (I_a \times M_a) \rightarrow (O_a \times Pow(M_a)),$$

    - I_a,O_a, input and output tokens consumed by this mode
    - $Pow(M_a)$, set of valid next modes
- Core Functional Dataflow (CFDF)
  - Modify invoke to be deterministic by ensuring only one unique next mode:

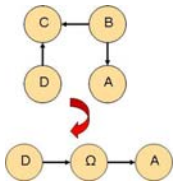
$$\kappa_a^* : (I_a \times M_a) \rightarrow (O_a \times M_a)$$

Semantic Hierarchy

- EIDF is a refinement of DIF Graphs that captures basic dataflow features (nodes, edges, tokens, etc.)
- CFDF restricts EIDF to deterministic dataflow applications
- Many popular forms of dataflow are **directly** supported by CFDF
  - SDF needs only one mode
  - CSDF phases correspond to different modes
- *Functional DIF integrates CFDF into the DIF package*

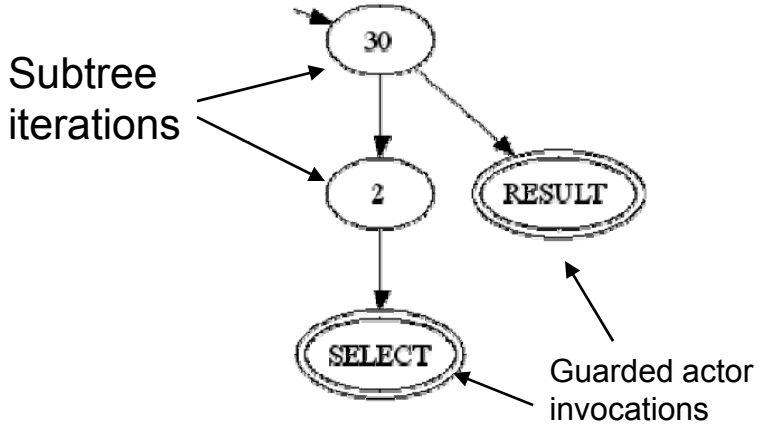


Generalized Schedule Trees



- Represents a schedule as a tree with internal nodes representing iteration counts
- Leaves represent actor invocations
- Execution may be guarded using the enabling function of CFDF

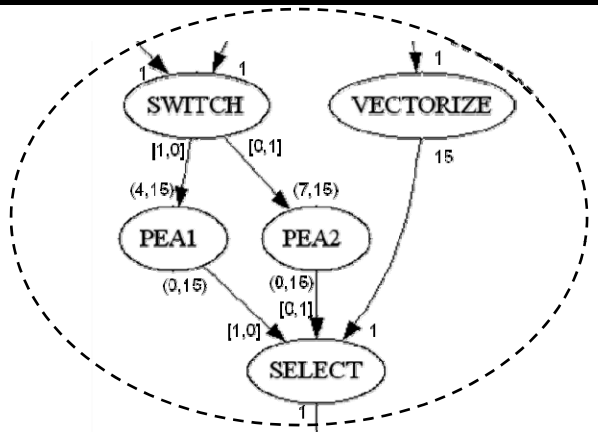
Example GST of a CFDF application



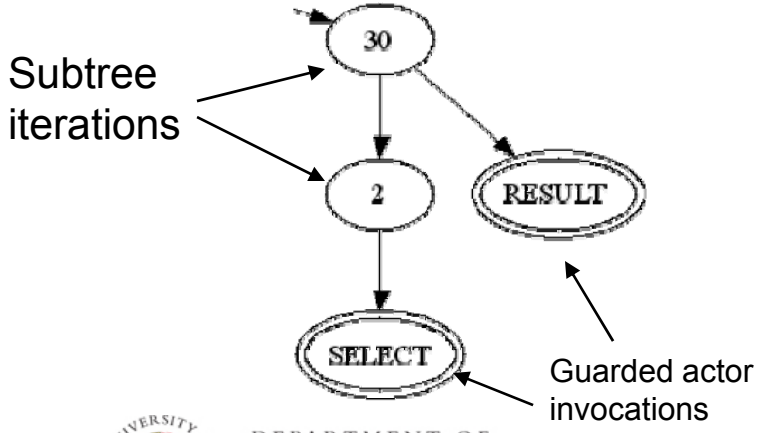
Functional DIF Application Design Features

- Supports Heterogeneous Composition
- No need for blocking reads
- Generalized Schedule Tree
  - Represents a schedule as a tree with internal nodes representing iteration counts
  - Leaves represent an actor invocation
  - Execution may be guarded using the enabling function of CFDF
  - Canonical, functionally correct schedules can be built automatically regardless of dataflow model mixes within CFDF
- Heterogeneous Simulator
  - Simulating is as simple as walking the schedule tree

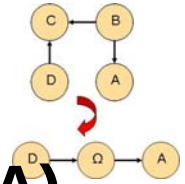
Example design using CSDF and BDF



Example GST of a CFDF application



Results: Polynomial Evaluation Accelerator (PEA)

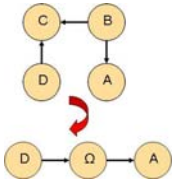


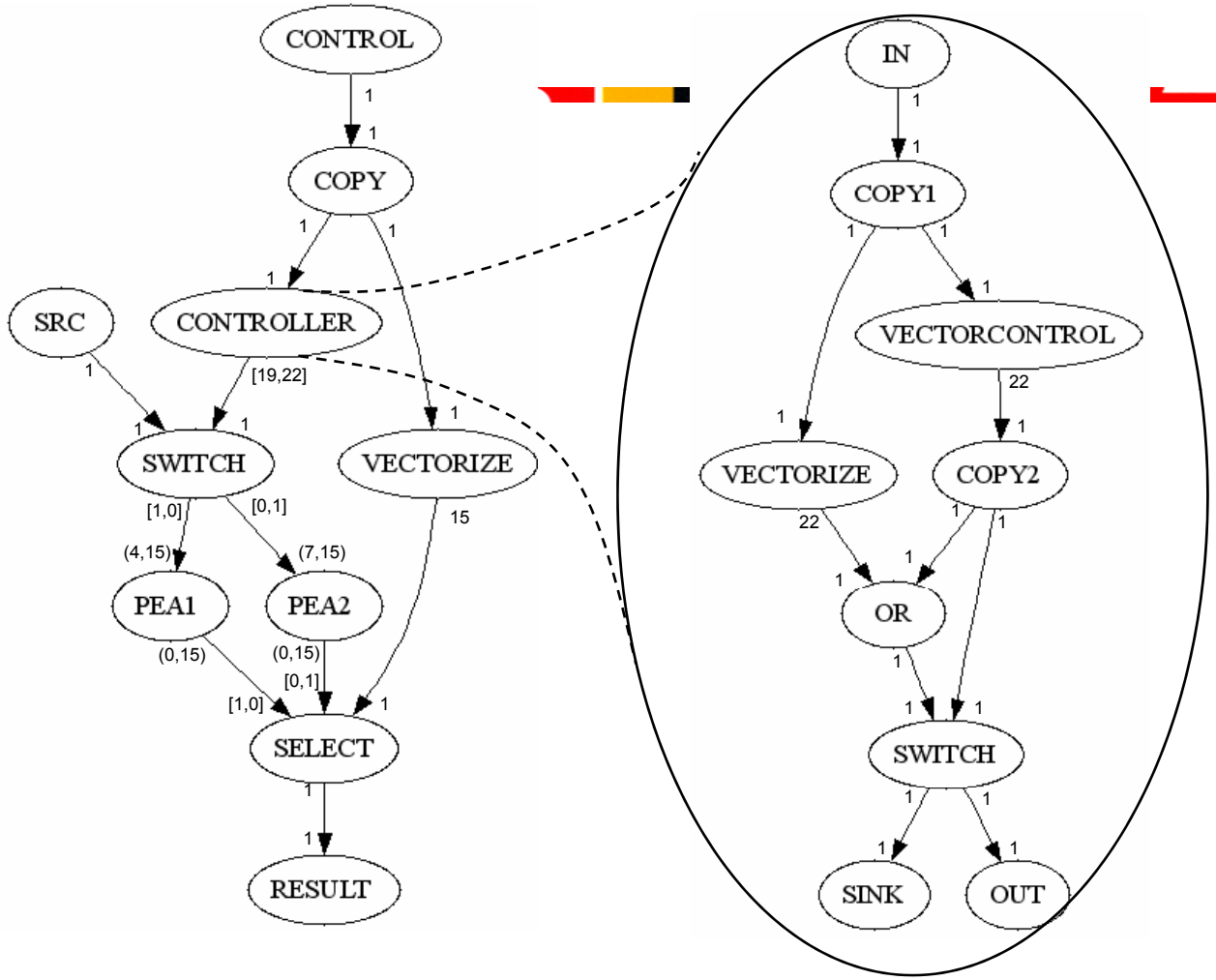
- Polynomial evaluation is a commonly used primitive in the digital communication domain.

$$P_i(x) = \sum_{k=0}^{n_i} C_k \times x^k$$

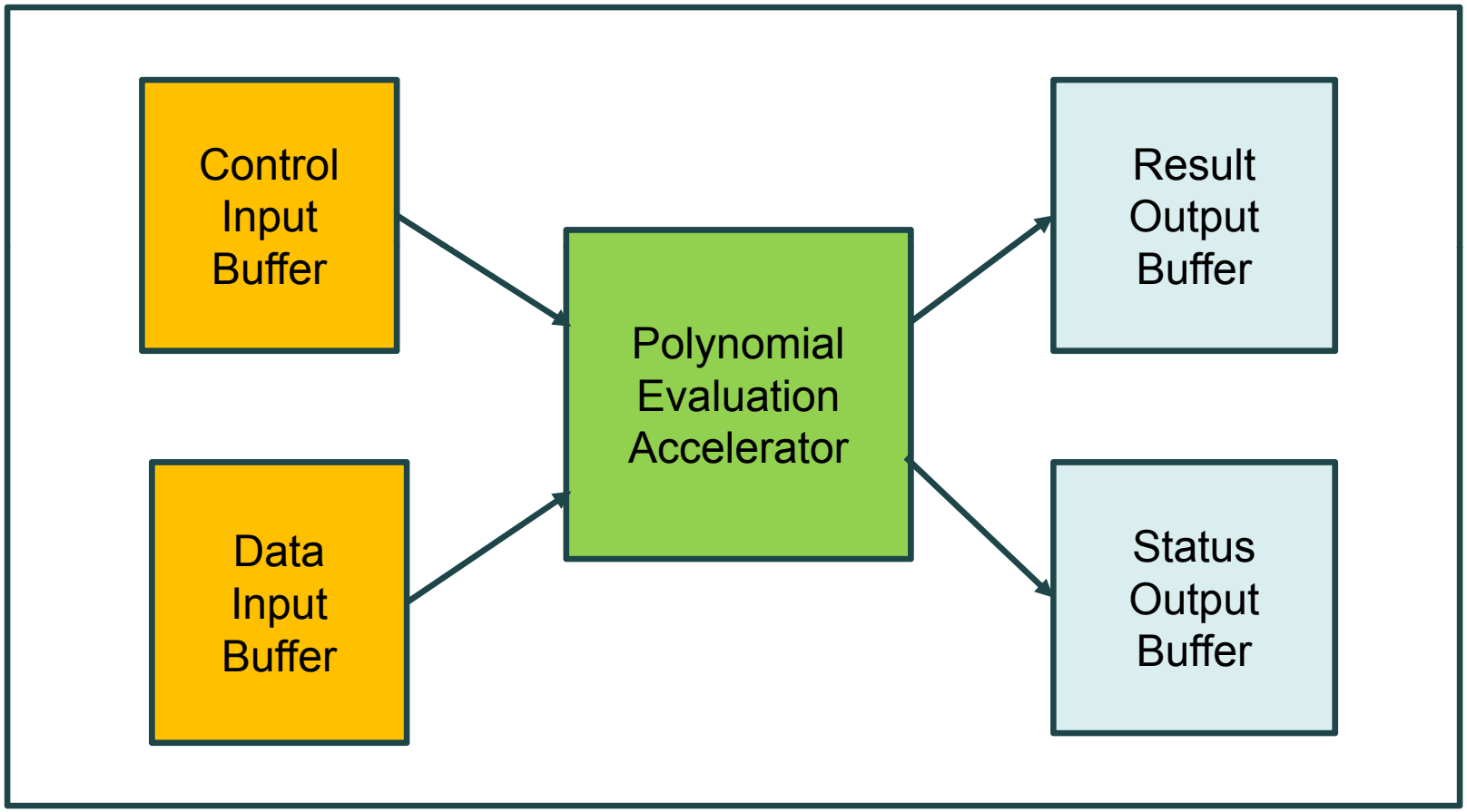
- The degree of P and the coefficients can change at run time.
- There are four types of instructions.
  - Reset (RST), Store Polynomial (STP), Evaluate Polynomial (EVP), Evaluate Block (EVB)

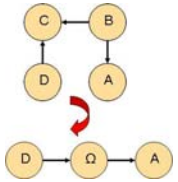
EXAMPLE: A Reconfigurable Accelerator for Polynomial Evaluation





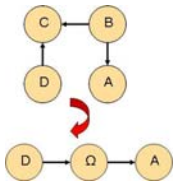
Dataflow Modeling of PEA Testbench





The modes in PEA

Mode	behavior	Consumption		Production	
		Control	Data	Result	Status
Normal	Wait for an instruction	1	0	0	0
RST	Reset all of the coefficients	0	0	0	0
STP	Store coefficients	0	1	0	1
EVP	Evaluate the value of P	0	1	1	1
EVB	Evaluate block	0	1	1	1

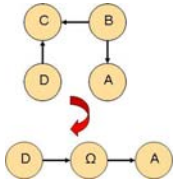


PEA Results

Simulation times of Verilog and Functional DIF for two different sets of instructions

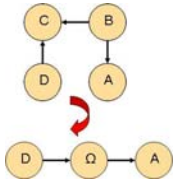
Instruction set	Verilog Simulation Time (ms)	Functional DIF Simulation Time (ms)	Speedup
Case 1	250	55	4.6x
Case 2	170	33	5.1x
Average	210	44	4.9x

Provides for efficient functional debugging/validation of dataflow modeling architecture before migration to hardware



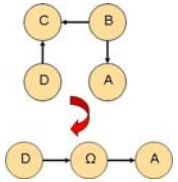
Functional DIF Summary

- Extends DIF with functional simulation by
  - Actor design considerations
  - Semantic foundation for execution
  - Supporting simulation and scheduling in the DIF package
- Simulation speeds better than Verilog
- Future Work
  - More heterogeneous applications
  - Parameterization of CFDF



Outline

- Motivation for DSP-oriented dataflow models, and co-design of models and transformations
- Overview and examples of high level dataflow transformations
- Reconfigurable dataflow graphs, and parameterized dataflow
- The *dataflow interchange format* (DIF)
- Functional DIF
- Conclusions and ongoing research



Conclusions and Ongoing Work

- Conclusions
  - High-level transformation techniques provide a power framework for design and implementation of signal processing systems
  - Because of their high level of abstraction, they involve relatively low computational complexity, but have high impact
  - Challenge: the underlying models must match well with key application characteristics → the models and transformations must be considered jointly
- Ongoing directions of research in the DIF project
  - Reconfigurable dataflow graphs
  - Rapid prototyping and optimization implementation methodologies based on functional DIF
  - Modeling and scheduling for wireless communications and software-defined radio