A. JAMES CLARK
SCHOOL OF ENGINEERING

Functional DIF

William Plishker, Nimish Sane, Mary Kiemb, Kapil Anand, Shuvra S. Bhattacharyya

University of Maryland, College Park (Dept. of Electrical and Computer Engineering)

Chess Seminar
June 5, 2008



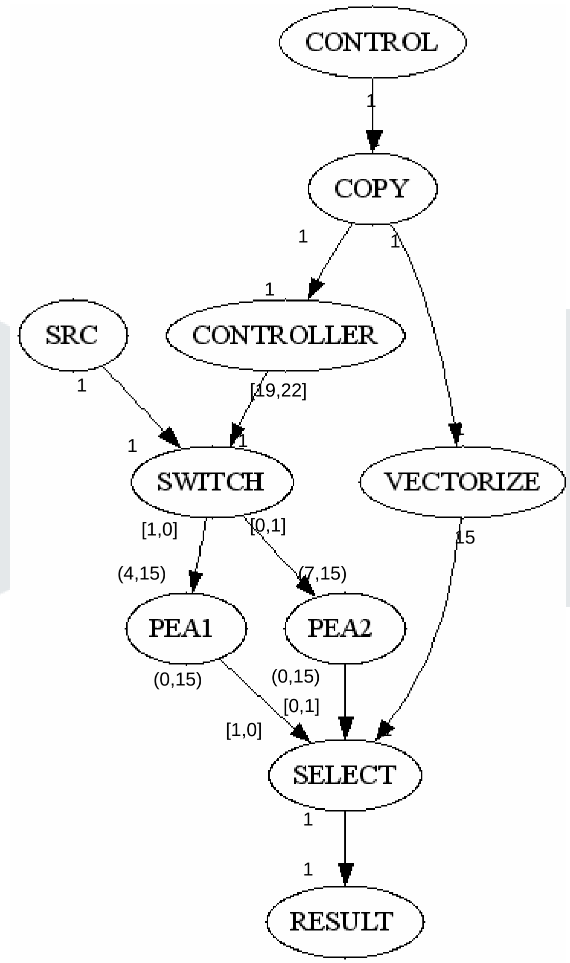

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

Outline

- Introduction
- Overview of the Dataflow Interchange Format
- Functional DIF
- Preliminary Results
- Demo

Introduction

- **Motivation**: dataflow tools can reduce the time to a functional prototype
- **Problem**: going from heterogeneous dataflow to implementation is time consuming and error prone
- **Our Solution**: Extend popular dataflow language with inline functional simulation semantics

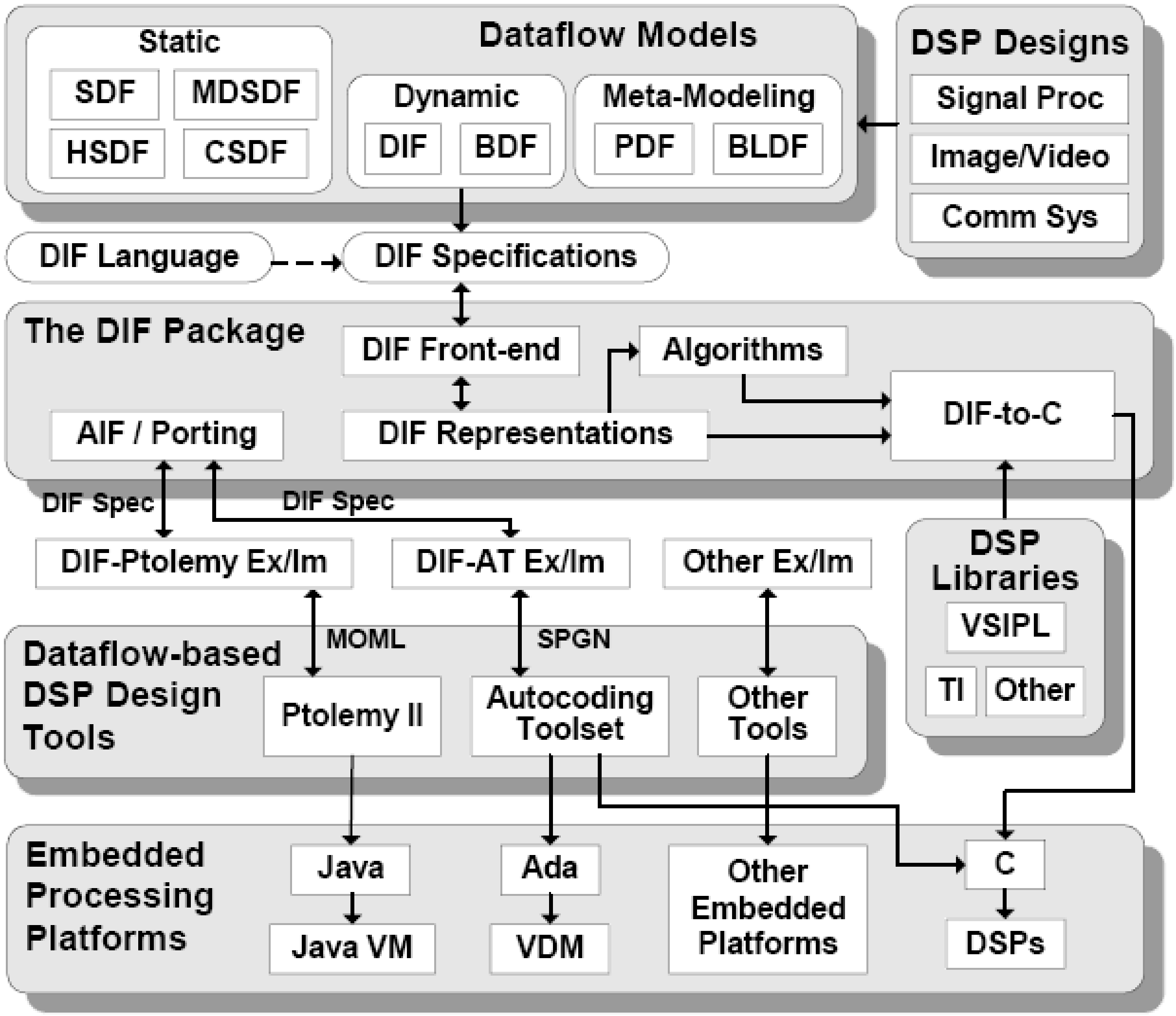


A. JAMES CLARK SCHOOL OF ENGINEERING

UMIACS University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

DIF Package

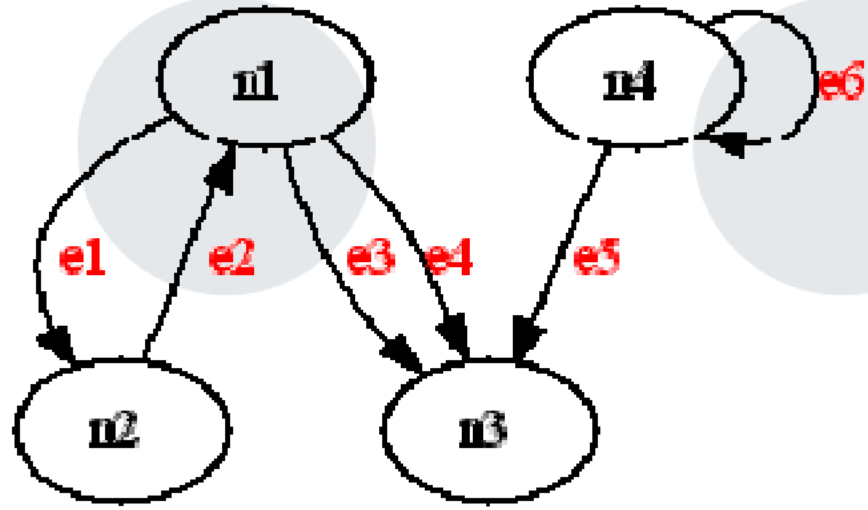

Dataflow Models
Static
SDF
MDSDF
HSDF
CSDF
Dynamic
DIF
BDF
Meta-Modeling
PDF
BLDF
DSP Designs
Signal Proc
Image/Video
Comm Sys
DIF Language
DIF Specifications
The DIF Package
DIF Front-end
Algorithms
DIF-to-C
AIF / Porting
DIF Representations
DIF Spec
DIF Spec
DIF-Ptolemy Ex/Im
DIF-AT Ex/Im
Other Ex/Im
DSP Libraries
VSIPL
TI
Other
MOML
SPGN
Dataflow-based DSP Design Tools
Ptolemy II
Autocoding Toolset
Other Tools
Embedded Processing Platforms
Java
Java VM
Ada
VDM
Other Embedded Platforms
C
DSPs

Dataflow Interchange Format

- Describe DF graphs in text

- Simple DIF file:

```
dif graph1_1 {
    topology {
        nodes = n1, n2, n3, n4;
        edges = e1 (n1, n2),
                e2 (n2, n1),
                e3 (n1, n3),
                e4 (n1, n3),
                e5 (n4, n3),
                e6 (n4, n4);
   }
}
```



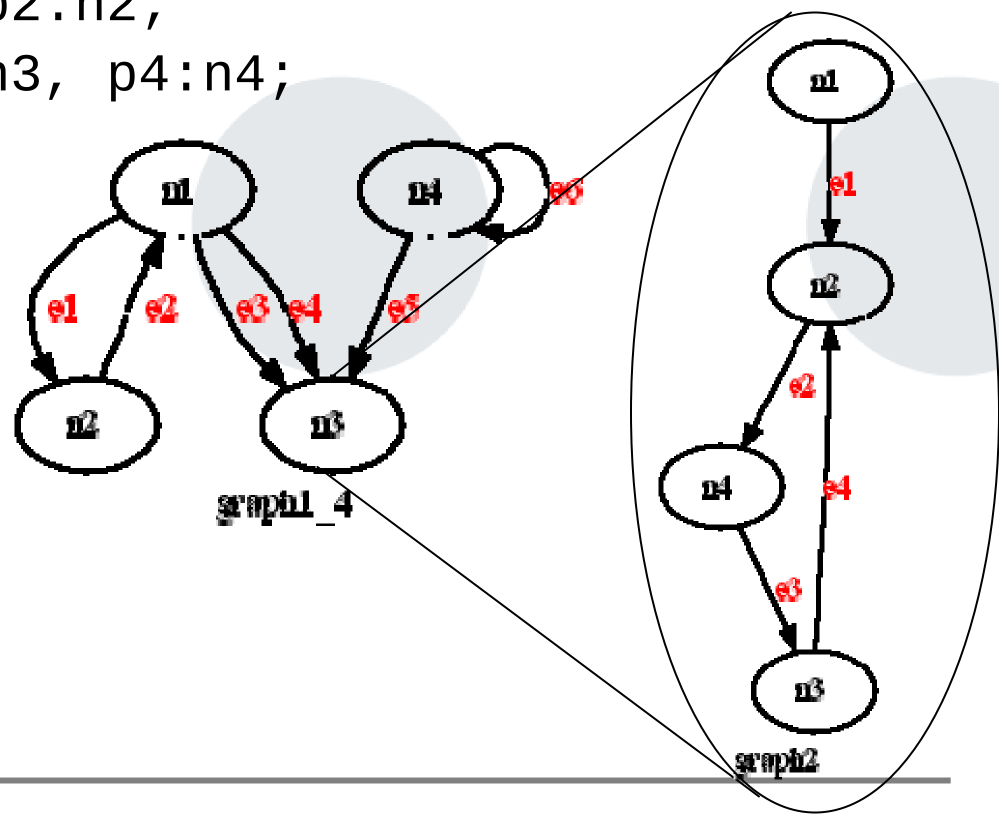
More features of DIF

- Ports

```
interface {
    inputs = p1, p2:n2;
    outputs = p3:n3, p4:n4;
}
```

- Hierarchy

```
refinement {
    graph2 = n3;
    p1 : e3;
    p2 : e4;
    p3 : e5;
    p4 : p3;
}
```



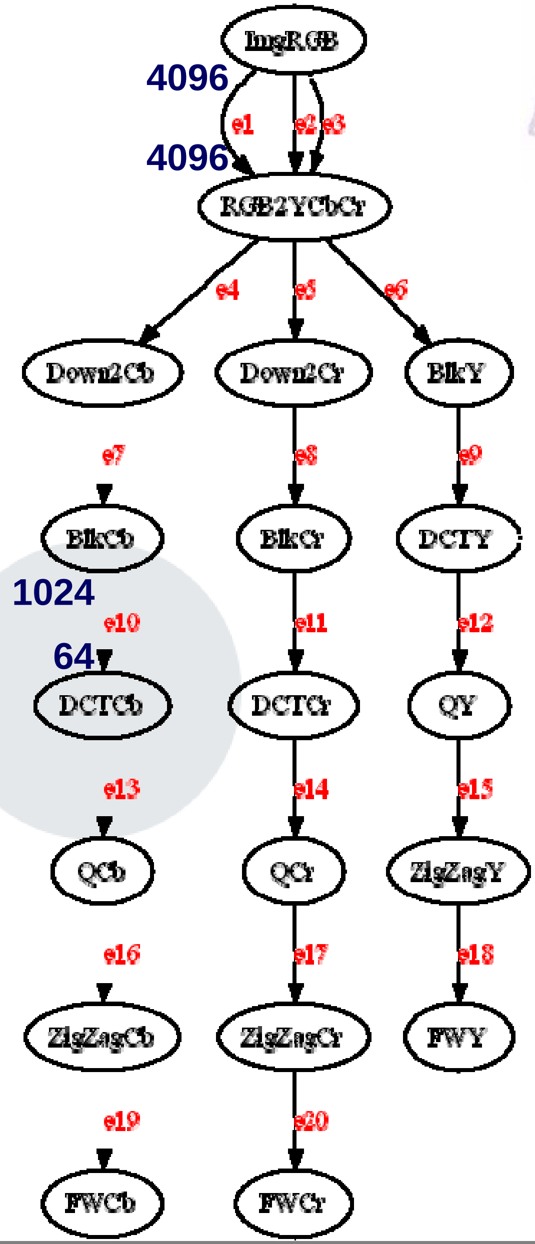
A. JAMES CLARK
SCHOOL OF ENGINEERING

More features of DIF

- Production and consumption

```
production {
      e1 = 4096;
      e10 = 1024;
      ...
}
consumption {
      e1 = 4096;
      e10 = 64;
       ...
}
```

- Computation keyword
- User defined attributes

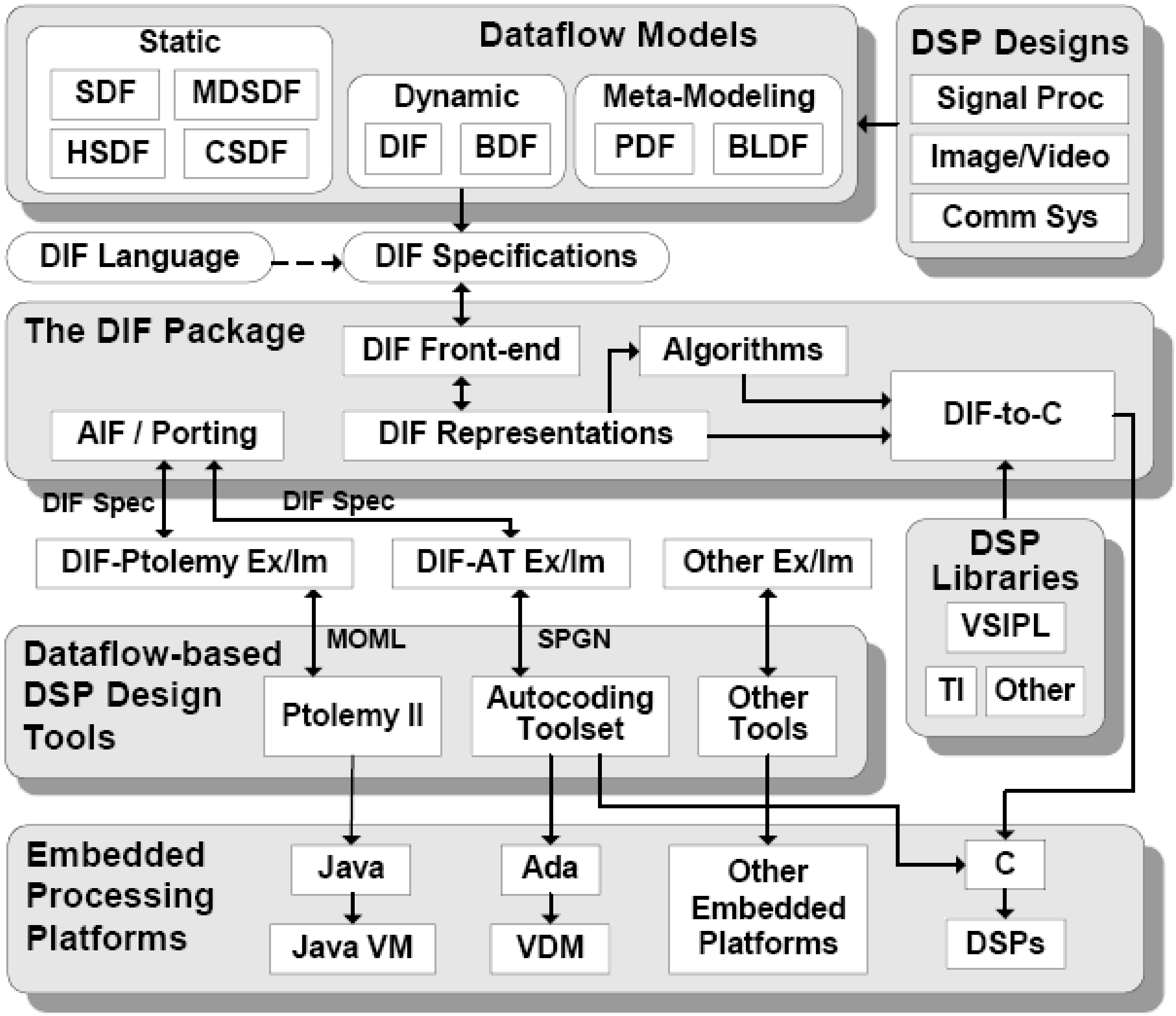


UMIACS
University of Maryland Institute for Advanced Computer Studies
DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

The DIF Language Syntax

```
dataflowModel graphID {
  basedon { graphID; }
  topology {
    nodes = nodeID, ...;
    edges = edgeID (srcNodeID,
      snkNodeID), ...; }
  interface {
    inputs = portID [:nodeID], ...;
    outputs = portID [:nodeID], ...; }
  parameter {
    paramID [:dataType];
    paramID [:dataType] = value;
    paramID [:dataType] : range; }
  refinement {
    subgraphID = supernodeID;
    subPortID : edgeID;
    subParamID = paramID; }
  builtInAttr {
    [elementID] = value;
    [elementID] = id;
    [elementID] = id1, id2, ...; }
  attribute usrDefAttr{
    [elementID] = value;
    [elementID] = id;
    [elementID] = id1, id2, ...; }
  actor nodeID {
    computation = stringValue;
    attrID [:attrType] [:dataType] = value;
    attrID [:attrType] [:dataType] = id;
    attrID [:attrType] [:dataType] = id1, ...; }
}
```

Need: Functional Sim in DIF Package:



Functional DIF

- Natural actor description
- Semantic foundation for simulating deterministic dataflow applications
- Scheduler/simulator for heterogeneous designs

- All with a focus to get to a correct implementation faster

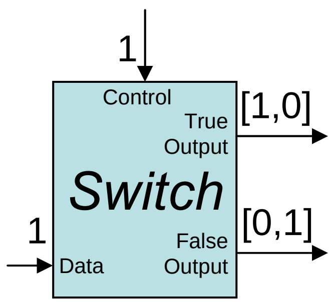
Related Work

- Actor Description Languages
  - Xilinx’s CAL, Mathwork’s S-Functions
- Heterogeneous Semantic Formalisms
  - Stream Based Functions (SBF)
  - Metropolis
  - Ptolemy
  - Transactors
- Dataflow Design Environments
  - Autocoding Toolset from MCCI
  - CoCentric System Studio from Synopsis
  - Compaan from Leiden University
  - Gedae from Gedae Inc.
  - Grape from K. U. Leuven
  - LabVIEW from National Instruments
  - PeaCE from Seoul National University
  - Ptolemy II from U. C. Berkeley
  - StreamIt from MIT.

Writing actors in Functional DIF

- Divide actors into a set 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:

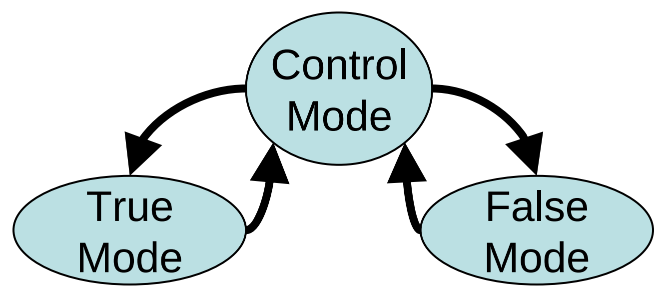
Switch Actor



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



A. JAMES CLARK
SCHOOL OF 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 making one unique next mode:

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

UMIACS
University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

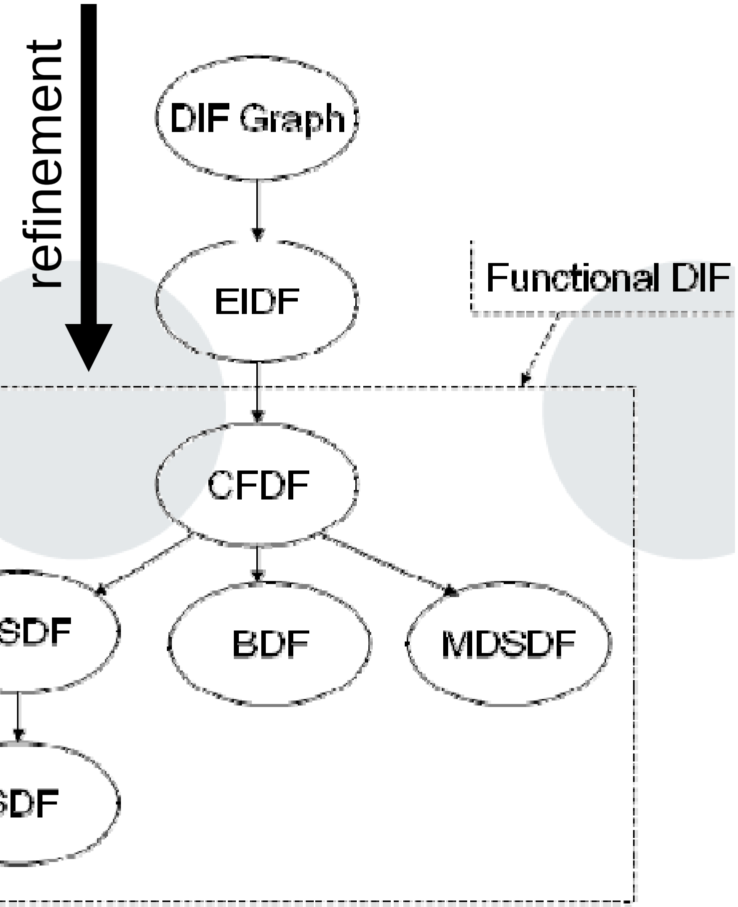
A. JAMES CLARK
SCHOOL OF ENGINEERING

Semantic Hierarchy

- DIF Graph -capture basic dataflow features (nodes, edges, tokens, etc)
- EIDF – Functional, nondeterministic dataflow
- CFDF – Deterministic dataflow

- Many popular forms of dataflow are **directly** supported by CFDF
  - SDF needs only one mode
  - CSDF phases correspond to modes

- *Functional DIF integrates CFDF into the DIF package*



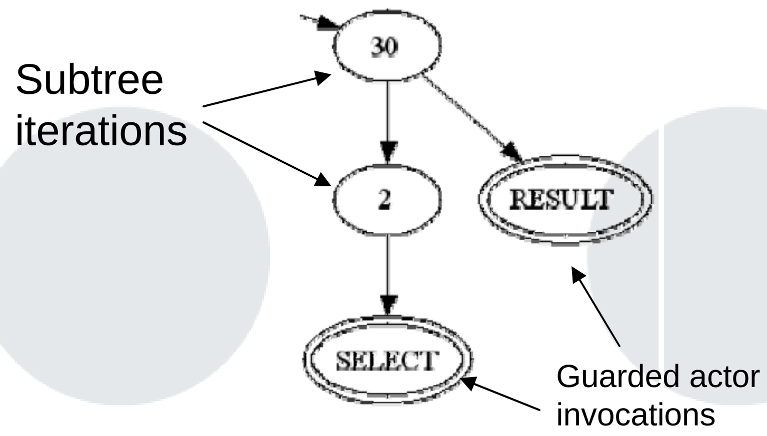
UMIACS
University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

Generalized Schedule Trees

- 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

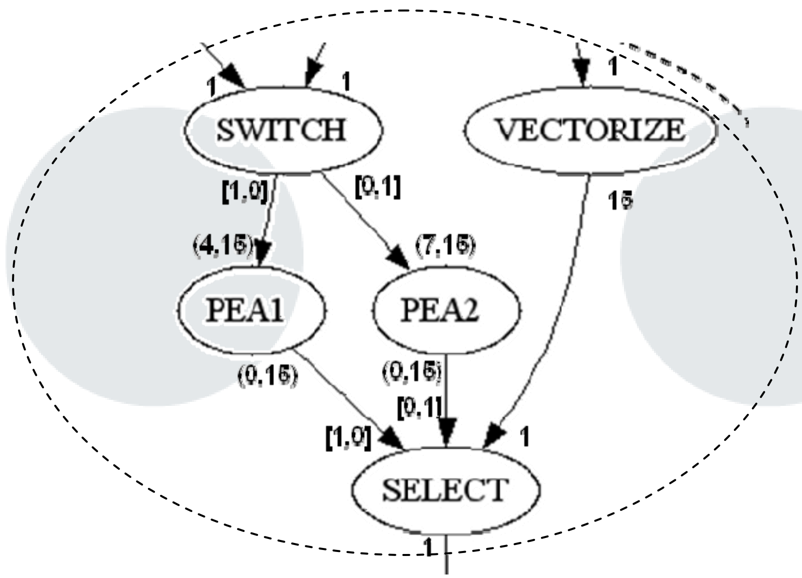
Example GST of a CFDF application



Functional DIF Application Design Features

- Supports Heterogeneous Composition
- Unit testing to determine models of actors
- Heterogeneous Simulator
  - Simulating is as simple as walking the schedule tree

Example design using CSDF and BDF



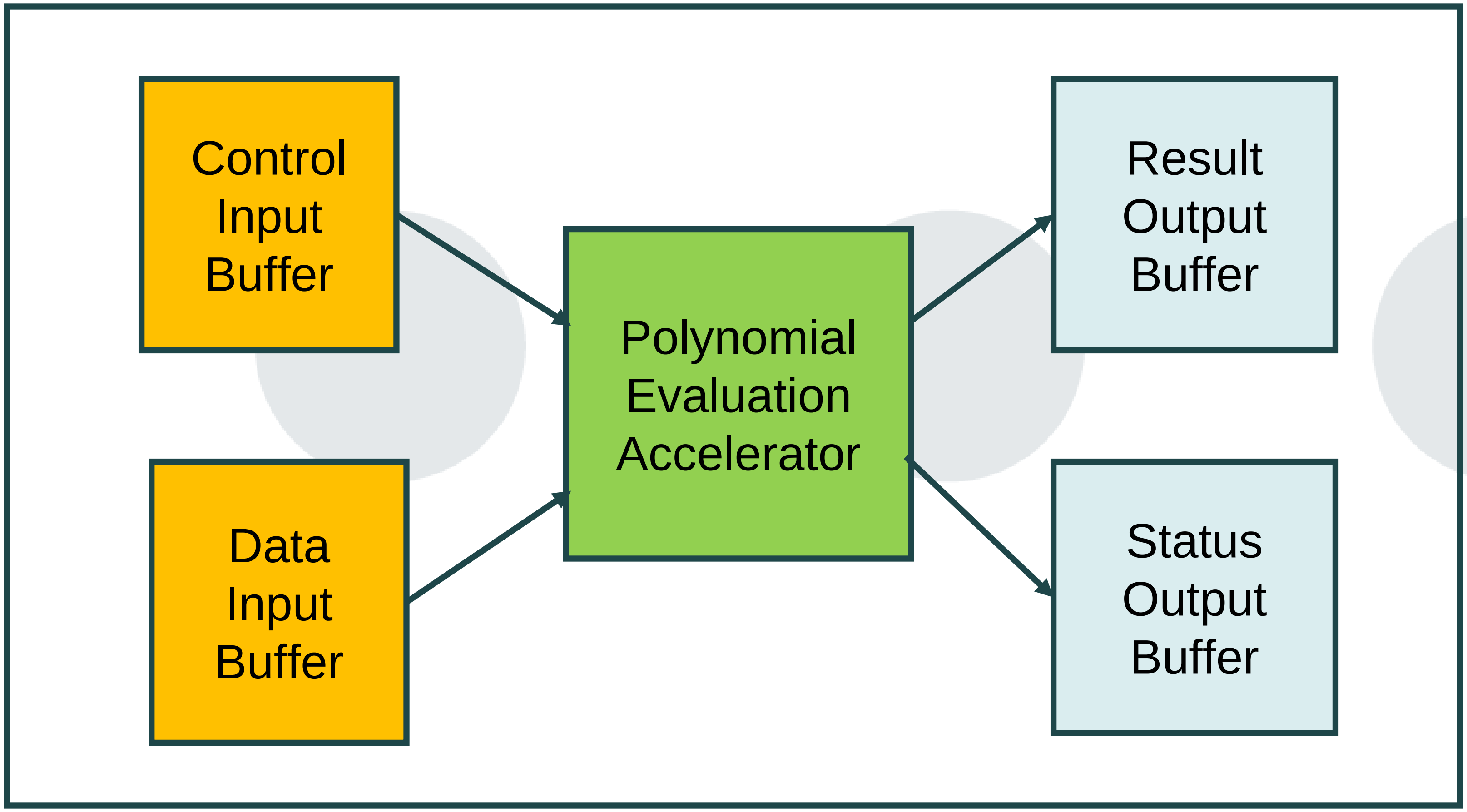
Results: Polynomial Evaluation Accelerator (PEA)

- Polynomial Evaluation is a commonly used primitive in communication area.

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

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

Dataflow Modeling of PEA Testbench



A. JAMES CLARK
SCHOOL OF ENGINEERING

The modes of 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

A. JAMES CLARK
SCHOOL OF ENGINEERING

Enable method pseudocode of PEA

```
Bool A.enable(x1, x2, …, xn, mode){
    if ( mode = Normal ) then
        if( there is 1 token in Control buffer)  then return true;
        else        return false;
    else if ( mode = RST ) then return true;
    else if ( mode = STP ) then
        if ( there is 1 token in Data buffer) then return true;
        else        return false;
    else if ( mode = EVP ) then
        if( there is 1 token in Data buffer ) then return true;
        else        return false;
    else if ( mode = EVB ) then
        if ( there is 1  token in Data buffer) then return true;
        else        return false;
    end if;
}
```

UMIACS
University of Maryland Institute for Advanced Computer Studies

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

Invoke method pseudocode of PEA

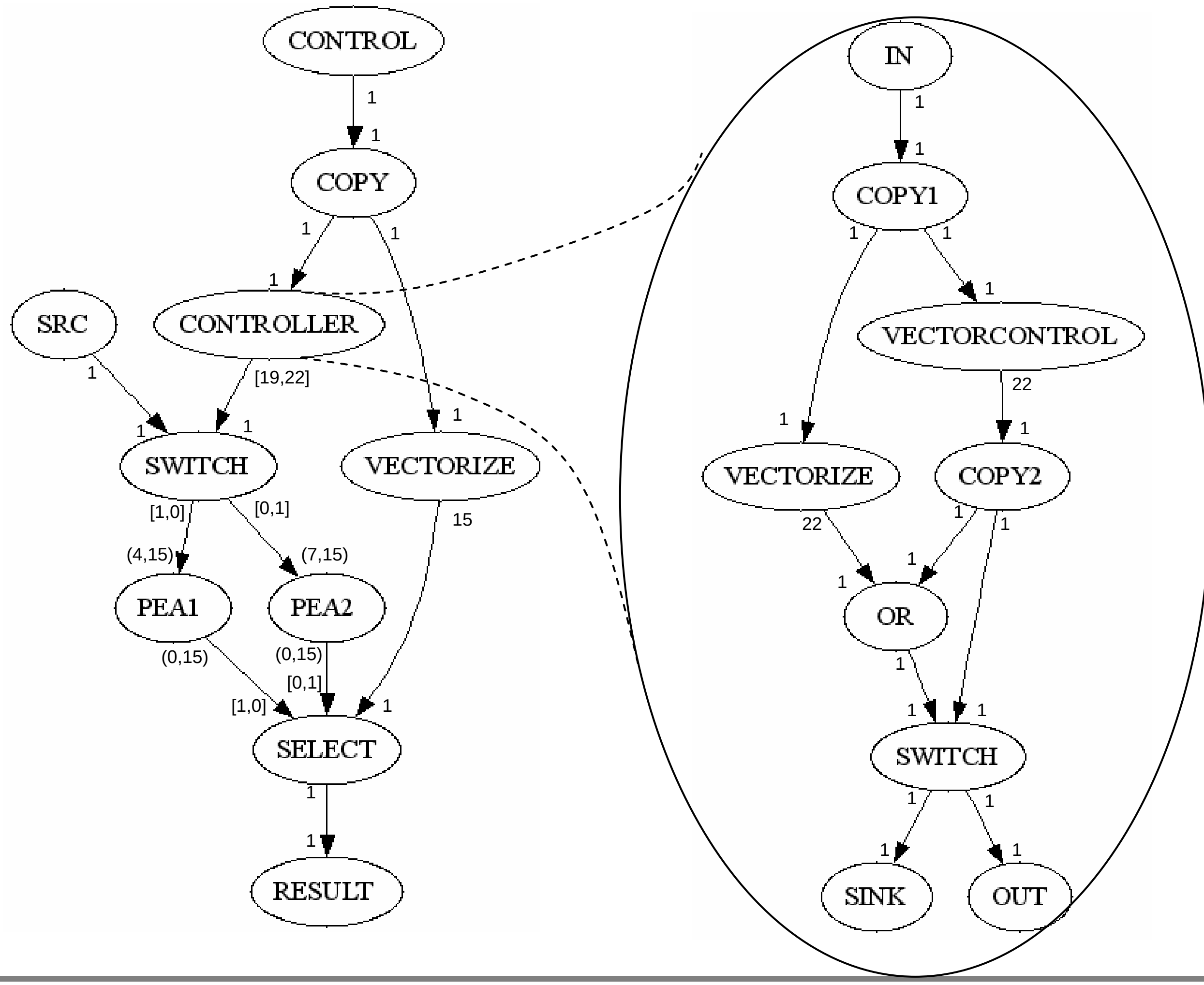
```
nextMode A.invoke(x1, x2, …, xn, mode){
    if ( mode = Normal ) then
            Decode instruction token
            return next mode based on instruction
    else if ( mode = RST ) then
            Reset all of the coefficients;
    else if ( mode = STP ) then
            Store coefficient for P_i(x);
            if( not done storing coefficients ) then return STP mode;
    else if ( mode = EVP ) then
             Evaluate P_i(x);
    else if ( mode = EVB ) then
             Evaluate  P_i(x);
            if( not done with block) then return EVB mode;
    end if;
    return Normal mode;
}
```

PEA Results

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

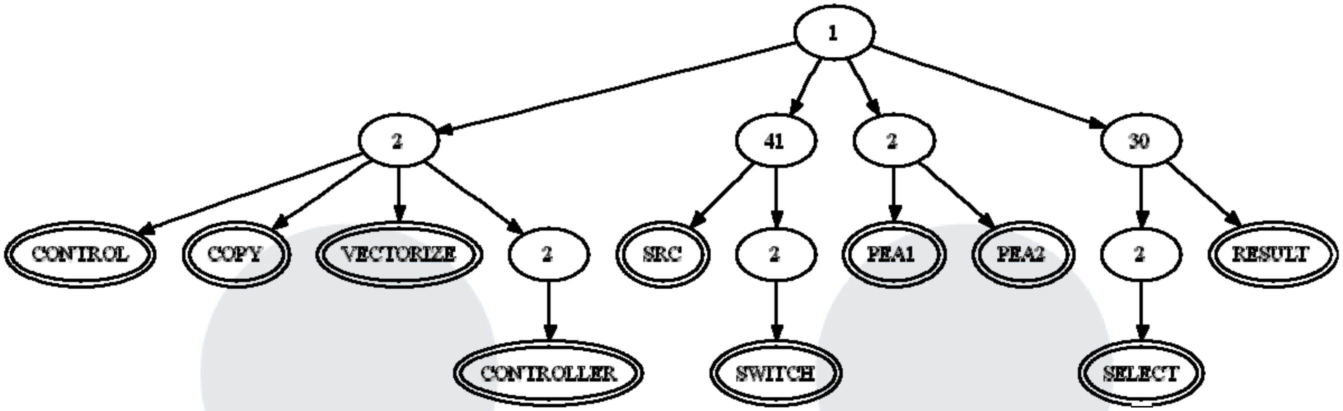
Instruction set	Verilog Simulation Time (ms)	Functional DIF Simulation Time (ms)	Speedup
Case 1	250	55	4.6x
Case 2	170	33	5.1x
Average	210	44	4.9x

Heterogeneous Example: Dual CSDF PEAs

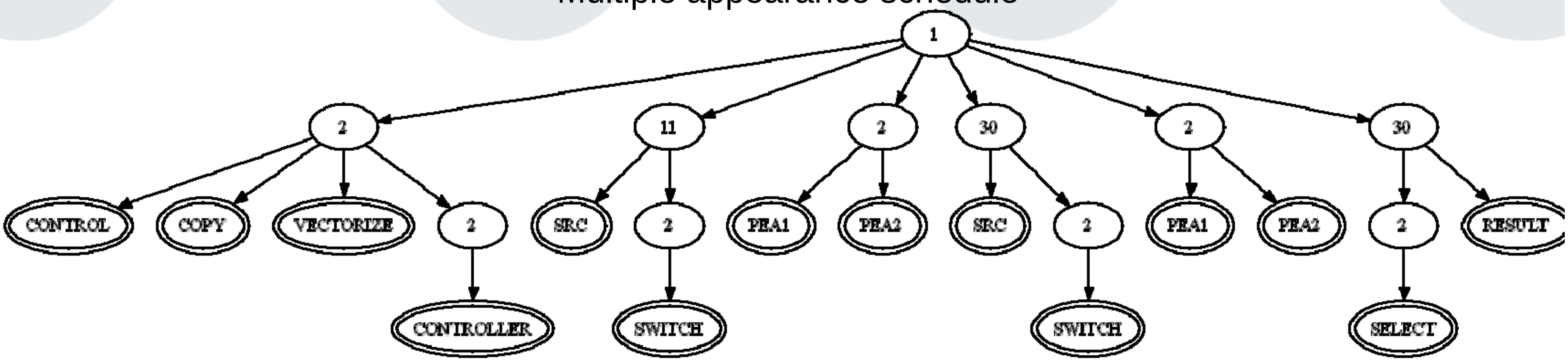


Dual PEA Schedules

Single appearance schedule



Multiple appearance schedule



Dual PEA results

Simulation times and max buffer sizes using different schedules

Application style	Schedule	Simulation Time (s)	Max observed buffer size (tokens)
BDF Strict	Canonical	6.88	2,327,733
BDF Strict	Single appearance	1.72	1,729
BDF Strict	Multiple appearance	1.59	1,722
CFDF	Canonical	3.57	1,018,047
CFDF	Single appearance	0.95	1,791
CFDF	Multiple appearance	0.99	1,800

Demo

Current status: Software Architecture

- DIF packages made up of a set of Jars
  - MoCGraph – graph package for models of computation
  - MAPSS – Core DIF package
  - DIF2C – Software synthesis plug-in
- Unit Testing Infrastructure
- Reliance on Ptolemy
  - Typing package
  - Kernel exceptions
  - Make scripts

Summary

- Extend 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

A. JAMES CLARK
SCHOOL OF ENGINEERING

Thank you!

Special thanks to the members of the DSPCAD group at the University of Maryland. This research was sponsored in part by the U.S. National Science Foundation (Grant number 0720596), and the US Army Research Office (Contract number TCN07108, administered through Battelle-Scientific Services Program).

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING