



# Scheduling and Assignment for DSP

## RASSP Education & Facilitation Program

### Module 22

Version 03.00

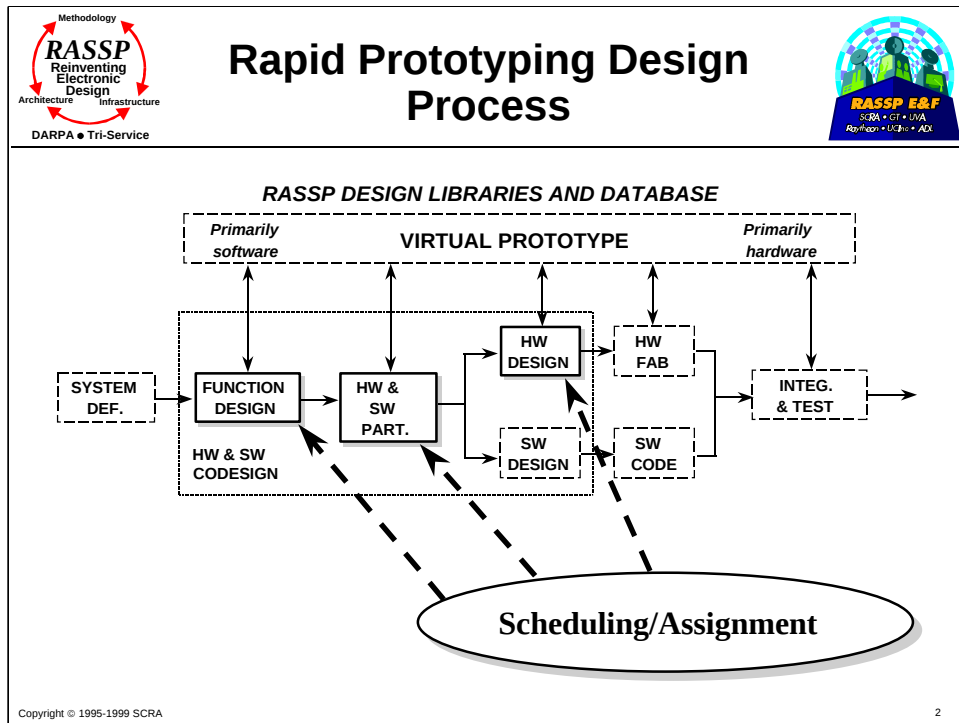
Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds "Unlimited Rights" in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice .

Copyright © 1995-1999 SCRA

1



Architecture design is based on the functional computational and communications requirements of the algorithm or algorithms selected to meet the specification. These trade-offs fall under the functional design and partitioning sections of the RASSP process.



## Module Goals



- | **To present a state of the art review of scheduling and assignment methodologies for DSP**
- | **Examine and study the impact of these methods on RASSP design environments**
- | **Consolidate understanding through examples**

Copyright © 1995-1999 SCRA

3

This module will cover the areas listed above. The purpose of this module is to expose the user of the RASSP process to the area of architecture design.



## Module Outline



- | **Introduction**
- | **FSFG as a Representation of DSP Algorithms**
- | **Measures of Performance**
- | **Retiming**
- | **Iteration Period and Iteration Period Bounds**
- | **Formal Methods for FSFGs**
- | **Perfect-rate FSFGs and Unfolding**
- | **Lookahead**
- | **Scheduling Approaches**
- | **Optimal Scheduling and Assignment Considering communication Delays and Finite Resources**
- | **Summary**

Copyright © 1995-1999 SCRA

4

The module describes in detail the process of scheduling, allocation, and mapping of DSP algorithms onto multiprocessor DSP architectures.



# Module Outline



## | Introduction

- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

5

We first describe how the input to the scheduling environment is specified (in terms of a Fully Specified Flow Graph).



## Introduction



- | **Graphical capture of algorithms via block-oriented environments is equivalent to a fully-specified flow graph representation of the timing**
- | **A formal approach to scheduling allows the transformation of the timing graphs to optimize execution times, processor utilization, and sample periods**
- | **Most commercial and research tools begin with the flow graph methodology that follows but only recently has communications and I/O been included within the cost constraints**

Copyright © 1995-1999 SCRA

6

A unified representation of DSP algorithms (at the level of a block oriented graphical language) is used to describe the computational flow and concurrency in the DSP algorithm. An equivalent language based specification could be used, if available.



# Module Outline



- | Introduction
- | **FSFG as a Representation of DSP Algorithms**
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

7



## Scheduling and Assignment for DSP



- | **Common representation of DSP algorithms**
  1. Language-driven executable specification
  2. Graphics/flow graph-driven specification
- | **Option 2 is preferred by most optimization algorithms**
- | **Goals of optimization**
  - m Provide efficient implementation with respect to sampling rate, clock period, latency, area, power, etc., of a given algorithm
  - m Provide modular and scaleable implementation capable of being upgraded in performance and functionality
  - m Include communications cost and cost of memories and other peripherals

Copyright © 1995-1999 SCRA

8

Both language-driven and graphical-driven front ends have their advantages and disadvantages, though the graphical approach is utilized in this module for purposes of ease of representation of the underlying scheduling, mapping, and allocation technologies.



## Difference Between DSP and General-purpose Schedulers



- | **DSP schedules are optimized over multiple iterations of data (essentially a semi-infinite stream of data); GP schedules are executed only once**
- | **Real-time and memory/buffer constraints exist for DSP**
- | **DSP uses numerically intensive arithmetic with high data bandwidths (most arithmetic is deterministic)**
- | **Library-based synthesis and design procedure is suitable for DSP applications**

Copyright © 1995-1999 SCRA

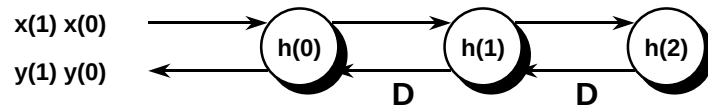
9

Application specificity results from determinism. An ever-present stream of data adds to the efficiency of the DSP scheduling process.

## FSFG Used to Represent DSP Functional/Timing

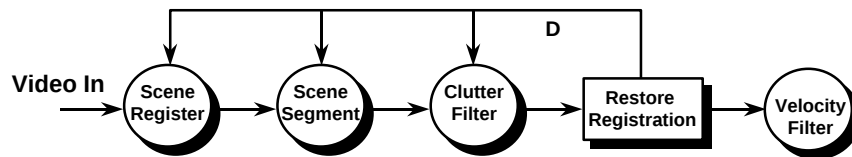
FSFG = Fully Specified Flow Graph (Data/Control Flow Graph)

### | A convolution - A fine-grain depiction



$x(.)$  = data input  $y(.)$  = data output  $h(.)$  = coefficients  $D$  = unit delays

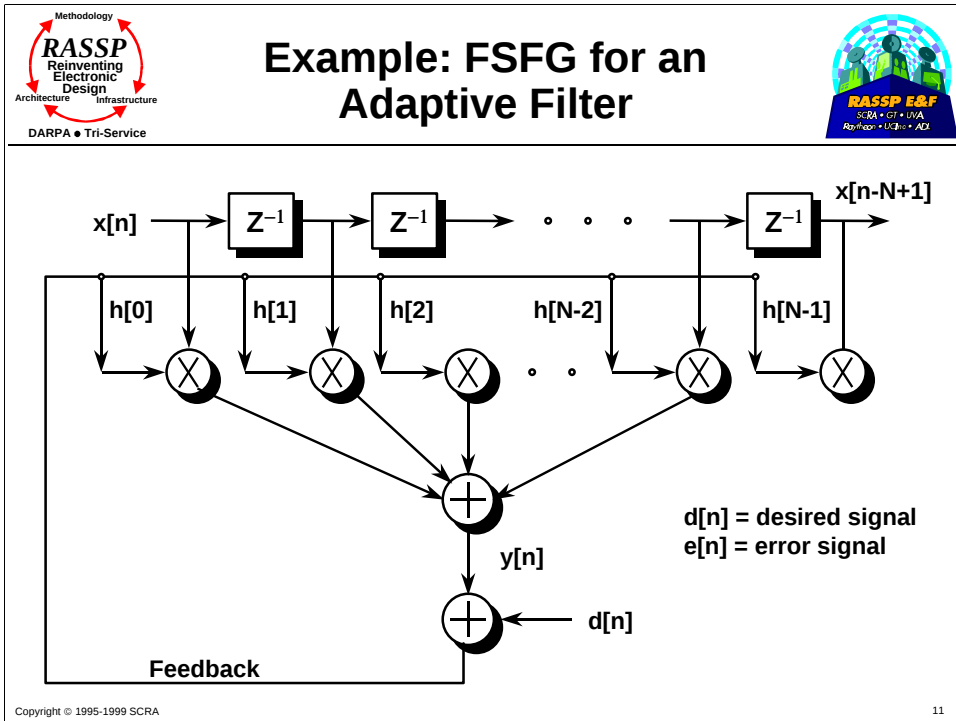
### | IRST - A coarse-grain depiction



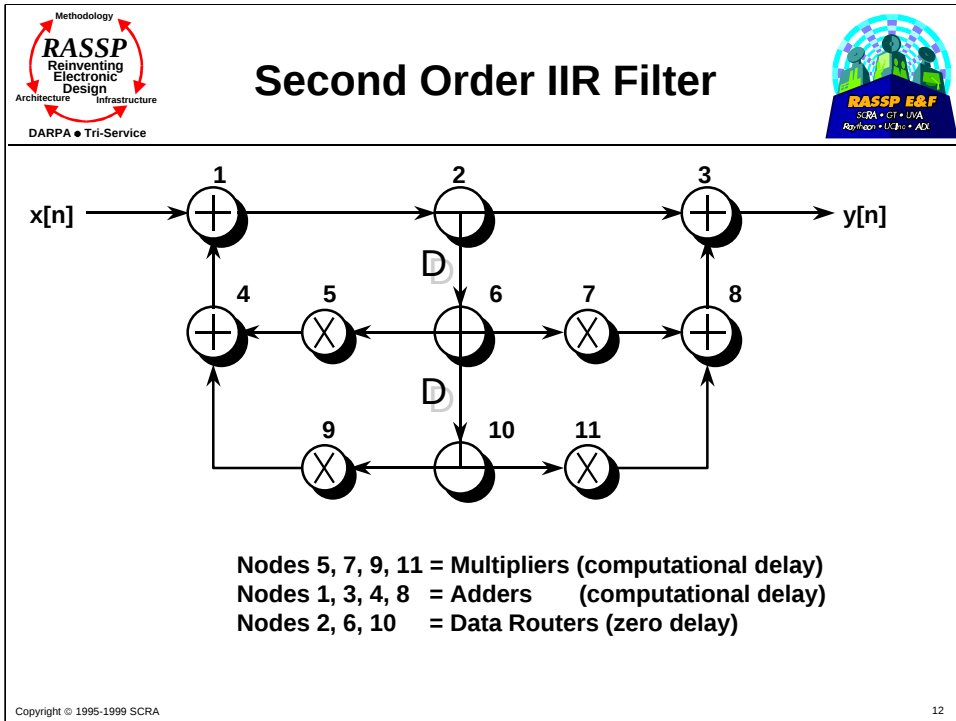
Copyright © 1995-1999 SCRA

10

FSFGs can be used to represent a variety of classes of algorithms.



FSFGs are particularly suited for DSP applications - examples. Note that it is not necessary to have  $N$  multipliers in the eventual implementation, and the FSFG only represents the input specification (including its concurrency) and not the final design.



This example is used throughout the module to represent the various scheduling technologies available. The delays (D) represent logical delays. E.g.,  $x[n]$  when passed through a D operator results in  $x[n-1]$  and represent storage elements for intermediate computation.

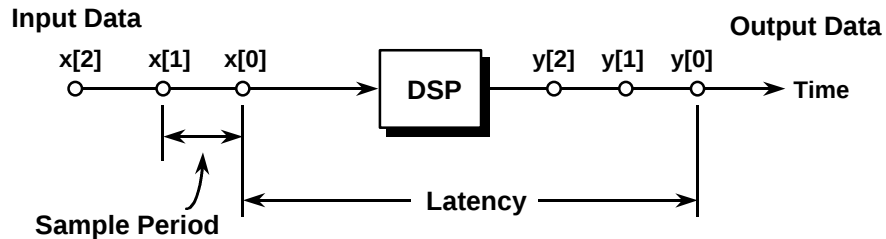
Another type of delay is called the computational delay that represents the wall-clock delay due to the computational time requirements of the datapath operators.

## Module Outline

- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | **Measures of Performance**
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

One cannot define how well a scheduling technology performs without introducing some metrics or measures of performance. These will be described in the following slides together with their bounds.

## Performance Measures

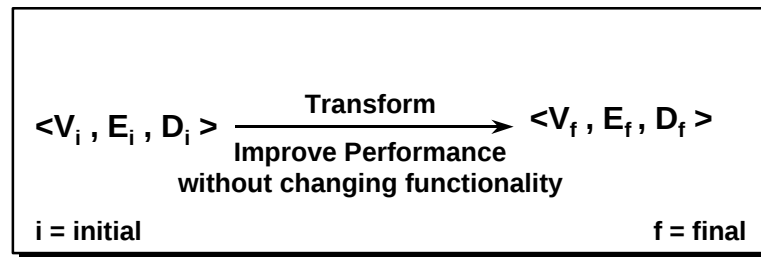


- | **Sample period = average time between arrival/acceptance of two successive data samples**
- | **Latency = the time for the DSP to produce a result  $y[0]$  in response to input  $x[0]$**
- | **In DSP applications, one usually considers minimization of sample period to be more important than latency**

Of the performance measures, the Sample Period is more important than the Latency.

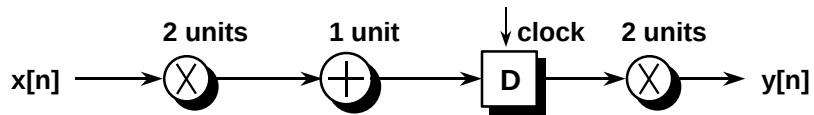
## Fully Specified Flow Graph

- | **FSFG =  $\langle V_i, E_i, D_i \rangle$** 
  - m  $V_i$  - set of computational nodes
  - m  $E_i$  - set of directed edges (implying data precedence)
  - m  $D_i$  - initial assignment of delays
- | **Goal**

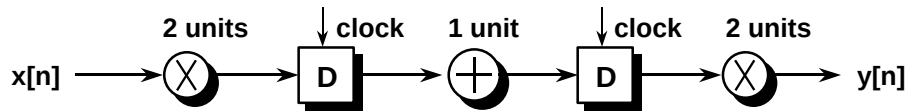


This slide formally defines the FSFG.  
The nodes are non-preemptible.

## Clock Period



Minimum clock period = 2 + 1 = 3 units



Minimum clock period = 2 units

**Minimum Clock Period = Computational delay of the longest zero-delay path**

This slide describes a simple optimization (visually) of a functional unit chain.

## Clock Period (Cont.)

### | Formally

$$T_c = \max T(p), p \in P_l$$

where

m  $T(u)$  = computational latency (in wall-clock time) of node  $u$

m  $p$  = path of edges

m  $P_l$  = set of paths with zero delays (purely combinatorial)

m  $T_c$  = smallest clock period

**Goal: Optimize the clock period**

Note the difference between delay elements [D] and wall clock computational delays within arithmetic and other operators that contribute to the clock period and latency.

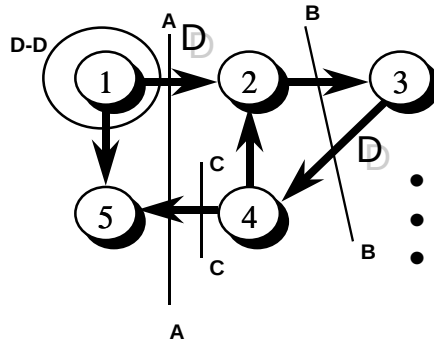
## Module Outline

- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | **Retiming**
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Retiming is the reassignment of delays within a FSFG without changing its algorithmic behavior.

## Cutset Definition

**A Cutset consists of the minimal set of edges that divide an FSFG into two disjoint parts**

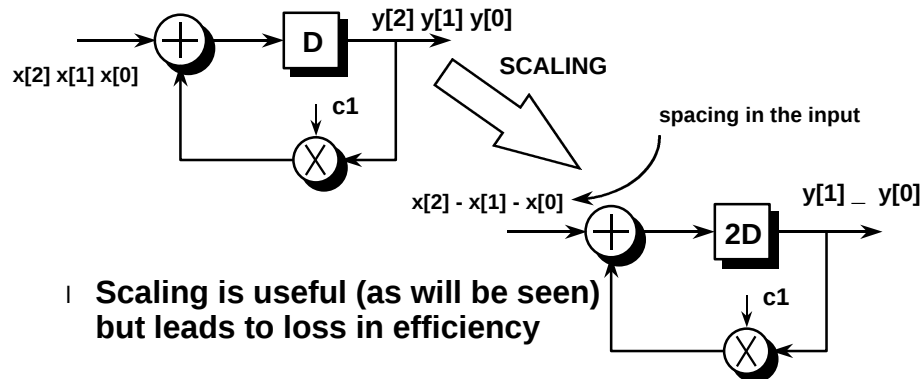


- A-A, B-B, D-D are valid cutsets
- C-C is not a valid cutset
- Edge 1-2 and edge 4-5 are directed in opposite directions with respect to cutset A-A

Cutsets will be important to the scheduling methodologies introduced in the remainder of the module.

## Time Scaling

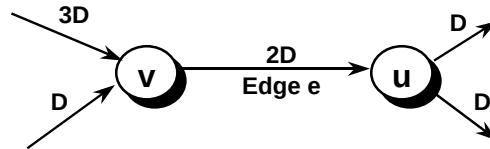
- All delays in an FSFG can be scaled by a positive number “s” without changing the functionality (except the input/output rates)**



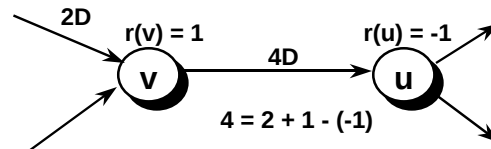
- Scaling is useful (as will be seen) but leads to loss in efficiency**

A number of examples of time scaling can be introduced at this juncture to illustrate time scaling.

## Nodal Transfer



$r(u)$  = number of delays REMOVED from EACH input arc and moved to each output arc (can be +ve or -ve)



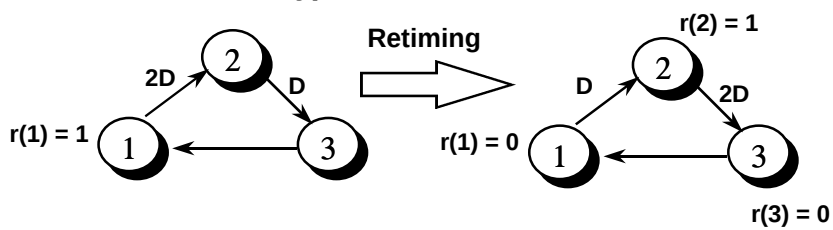
$$D_f(e) = D_i(e) + r(v) - r(u)$$

| Same functionality, different timing - RETIMING

We illustrate the process of transfer of delays through a node without changing functionality (See [Madisetti95] for further details).

## Legal Retiming

- |  $D_f(e) \geq 0$  for all edges
- |  $D_f(e) \geq 1$  for a SYSTOLIC FSFG
  - <sup>m</sup> Systolic FSFGs have minimum clock period, or the highest clock frequency
- | Total number of delays in a loop remains constant with and without retiming (no loop can have a zero delay)

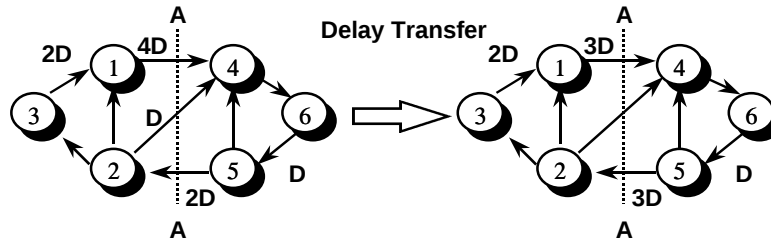


Copyright © 1995-1999 SCRA

22

A neat way to convert FSFGs to systolic arrays.  
 Systolic arrays are not efficient though.

## Delay Transfer Across Cutsets



- | Remove  $k$  delays from positive edges and move them to negative edges ( $k$  can be positive or negative)
- | No algorithmic change in behavior

We will use these properties to optimize schedules.

## Results from Retiming

- | **Theorem I: A systolic FSFG can always be realized**
- | **Theorem II: If the number of delays in a loop is less than the number of arcs, then time scaling is required for obtaining systolic schedules**
- | **Theorem III: The delays in a retimed circuit can be minimized by setting up the following Linear Integer Program for  $r(v)$**

Minimize  $\sum_{r \in v} r(v) \{ \text{Fanout}(v) - \text{Fanin}(v) \}$   
 subject to  $r(u) - r(v) \leq D_i(e) - 1$

- | **Where  $e$  is a directed edge from  $v$  to  $u$**

Proofs of these results are better illustrated by examples.



## Module Outline



- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | **Iteration Period and Iteration Period Bounds**
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

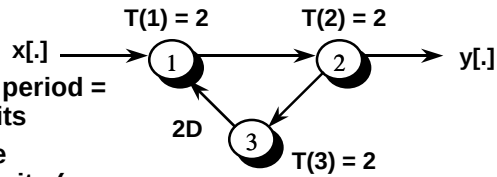
25

A few bounds that relate to the iteration period between successive periods of a DSP algorithm are now covered.

## Iteration Period and Delays in a Loop

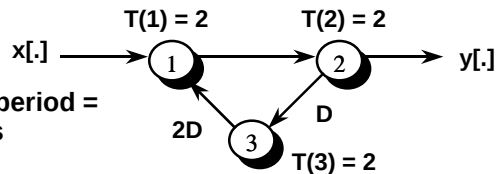
### Case 1

- m Bound on iteration period =  
 $= (2+2+2)/3 = 3$  units
- m Data sample can be  
 accessed every 3 units (on  
 average)



### Case 2

- m Bound on iteration period =  
 $= (2+2+2)/3 = 2$  units



**Case 1 and Case 2 are different algorithms, though!**

Iteration Period is related to the sample period in that the former is suitable for loops.

## Iteration Period Bound

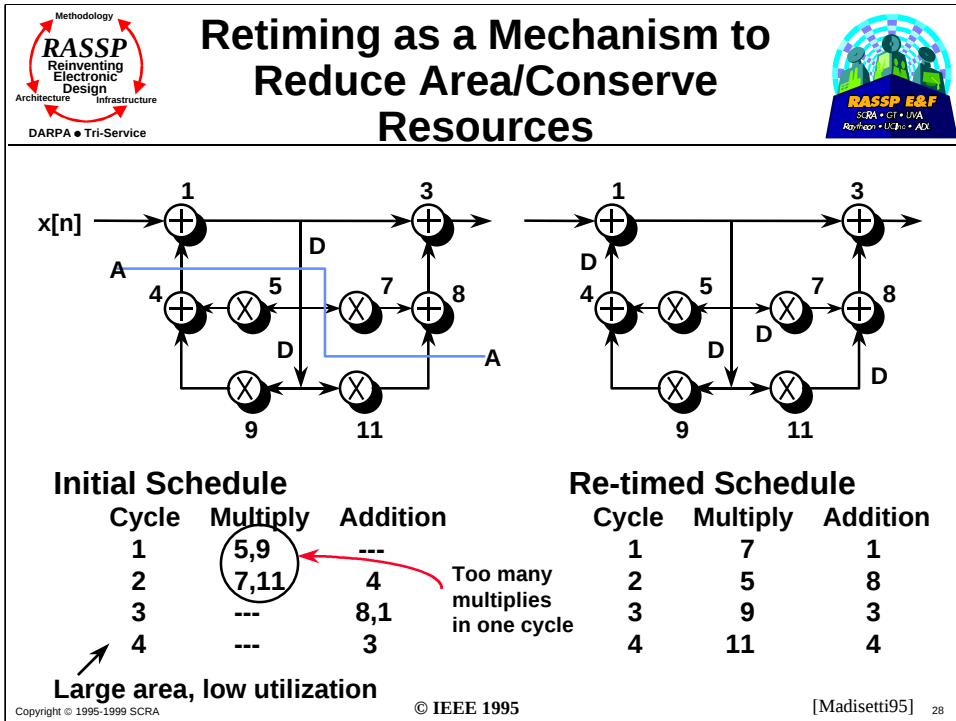
- | An FSFG is bounded below by the Iteration Period Bound (IPB)

$$\text{IPB} = \max_{\text{all loops}} \left[ \frac{\text{Total computational latency of a loop}}{\text{Total number of delays in the loop}} \right]$$

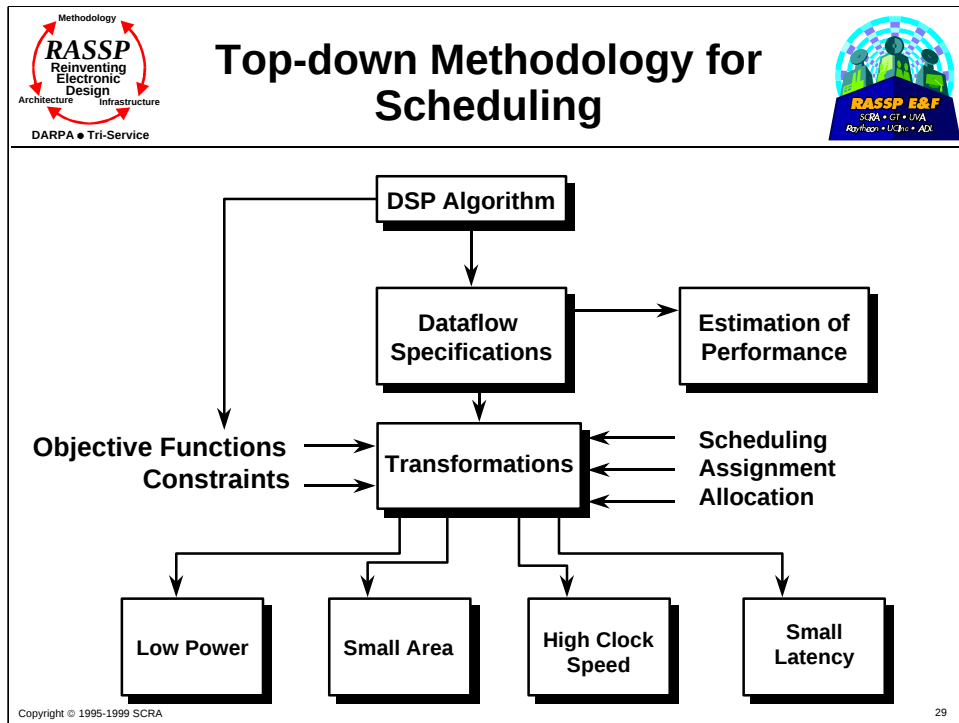
- | Thus the IPB can be reduced by increasing the delays in the loops or by reducing the computational latency of the loops

**Scheduling and assignment efficiency is limited by loops in the DSP flow graph!**

The IPB is a fundamental bound to scheduling efficiency.



We repeat the example to show how retiming “flattens” resource utilization.



The DSP algorithm is first analyzed to derive its bounds and then suitable transformations of the flow graph are considered while keeping the original intent of the scheduling algorithm to suit the implementation objectives of low power, small area, or small latency, etc.



## Partial Summary of Results



- | **Flow graph transformations can result in improved efficiency of implementation for the same algorithm**
- | **Retiming cannot improve upon the bound, but it can help achieve the bound on the iteration bound**
- | **A formal approach is necessary for top down design, optimization, and implementation**

Copyright © 1995-1999 SCRA

30

So far we have shown that retiming and transformations help, without formally proving anything --- so now we move ahead with a formal procedure.



## Module Outline



- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | **Formal Methods for FSFGs**
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

31



# FSFG Transformation and Scheduling



- | **Various transformations exist that lead to efficient implementation of flow graphs**

- m **Unfolding**
- m **Retiming**
- m **Lookahead**
- m **Loop shrinking**

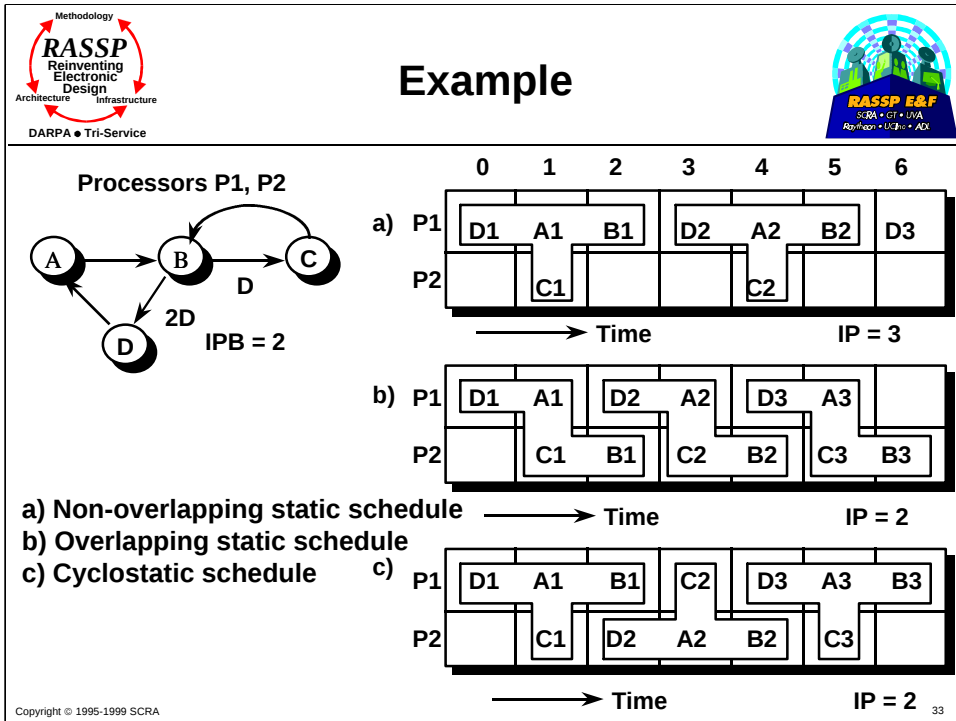
- | **Approach**

- m **Step 1: Classify different types of schedules**
- m **Step 2: Try to find methods to construct these schedules, if they exist**

Copyright © 1995-1999 SCRA

32

We follow a 2 step procedure - first identify what schedules we like, and then look or construct these schedules.



Though the cyclostatic and the static schedules both have the same IP here, the cyclostatic has, in general, a smaller delay although with more complex control.

We prefer schedules of type (b).



## Questions To Be Answered at This Point



- | Can every FSFG achieve the iteration period bound?
- | Can every FSFG achieve the IPB in addition to the processor bound?
- | Can transformations assist in achieving the IPB?
- | Are optimal schedules static or cyclostatic?
- | Can the IPB itself be reduced?
- | Can transformations reduce the processor bound?

Copyright © 1995-1999 SCRA

34

These questions will be answered in the following discussion.



DARPA • Tri-Service

# Module Outline



RASSP E&F  
SCRA • CT • USA  
Rapidgo • USCIB • ADX

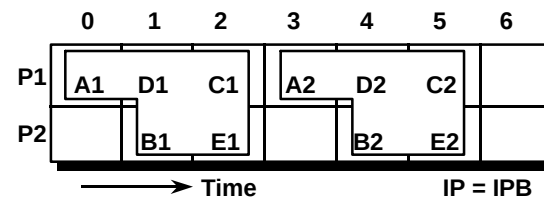
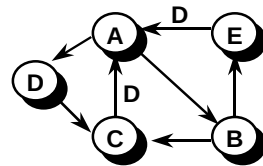
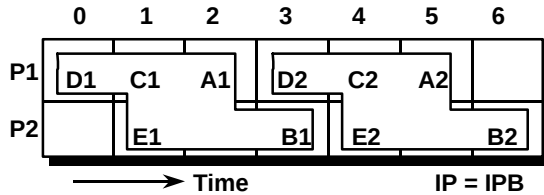
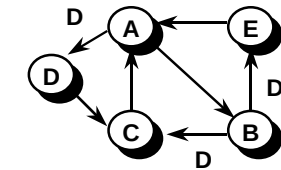
- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | **Perfect-rate FSFGs and Unfolding**
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

35

We now describe those graphs that have optimal rate schedules, called perfect-rate schedules.

## Perfect-rate FSFGs



**Optimal static schedules are possible for  
PERFECT-RATE FSFGs**

A perfect-rate FSFG is one which has only one delay in each loop - no more and no less. Perfect rate graphs on the left have the rate-optimal schedules shown on the right.



## Perfect-rate FSFGs (Cont.)



### Conclusion

- | **Because perfect-rate FSFGs have static rate-optimal schedules, can all FSFGs be transformed to PR-FSFGs?**
- | **Answer: Yes, all FSFGs can be transformed to PR-FSFGs**
  - m **Note communication cost and processor cost is not optimal, only sample period is optimized**

All perfect-rate FSFGs have rate optimal static schedules.



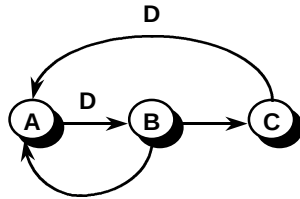
# Unfolding



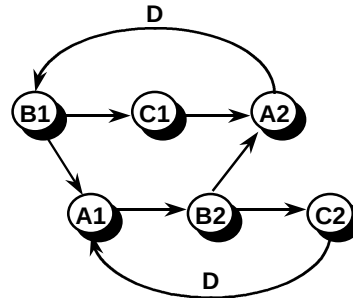
- | **How does one convert FSFGs to perfect-rate FSFGs?**
- | **Answer: Through UNFOLDING! (method 1)**

Unfolding is a method by which one executes two or more iterations of an FSFG in one step.

## Example of Unfolding



Original FSFG



Unfolded-by-2 FSFG

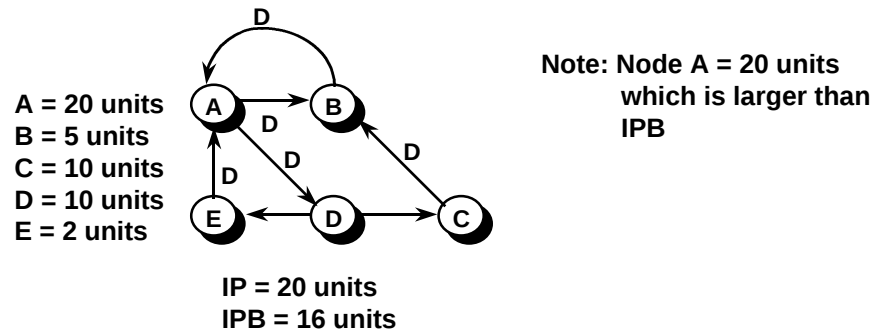
**By unfolding-by-2 we expose two iterations of the DSP algorithm at the same time**

Here note that the unfolded graph is perfect-rate.

Also the IPB is 4, but two iterations get done in one step.

## Optimum Unfolding

- Unfold the flow graph by the LCM of the number of delays in each loop
- Example: Unfold by 6 results in IP = 16 units on 4 processors



This example shows that unfolding is sufficient but not necessary for rate optimal schedules. Unfolding also is NOT as efficient.



## Module Outline



- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | **Lookahead**
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

41

## Method 2: Lookahead

- | **Lookahead actually reduces the iteration period bound**
  - m Pipelining and unfolding reduce the iteration period, but do not affect the iteration period bound
- | **Design Flow**
  - m **Step 1:** Check if sample period is satisfied by IPB
  - m **Step 2:** Lookahead can be used to reduce IPB
  - m **Step 3:** Retiming, unfolding, or pipelining can then be used to reduce the iteration period to that of the IPB

Lookahead is best illustrated by the example that follows.

## What is Lookahead?

**Statement 1:**  $x[n] = ax[n-1] + u[n]$

**Statement 2:**  $x[n+1] = ax[n] + u[n+1]$

**Statement 2 cannot be executed until  $x[n]$  is computed**

**Rewriting the equations again (with a lookahead of two)**

**Statement 1:**  $x[n] = bx[n-2] + au[n] + u[n-1]$

**Statement 2:**  $x[n+1] = a(ax[n-1] + u[n]) + u[n+1]$   
 $= a^2x[n-1] + au[n] + u[n+1]$   
where  $b = a^2$

**Note that  $x[n+1]$  depends on  $x[n-1]$  and not on  $x[n]$**

**Therefore, these statements could be executed in parallel**

Note that in lookahead, we precompute intermediate steps of future values, thus reduces the iteration period bottleneck through dependencies between successive iterations.

## What is Lookahead? (Cont.)

### Original first order Filter

|             | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------|---|---|---|---|---|---|---|---|---|
| Read Ops    | x |   |   |   | y |   |   |   |   |
| Multiply    |   | x |   |   | y |   |   |   |   |
| Add         |   |   | x |   |   | y |   |   |   |
| Store State |   |   |   | x |   |   | y |   |   |

First instruction ←

→ Second Instruction  
begins after x[n] is  
ready

### Filter with lookahead of 2

**Note:**  
Factor sample  
period due to  
increased  
lookahead

|             |   |   |   |   |   |   |  |  |   |
|-------------|---|---|---|---|---|---|--|--|---|
| Read Ops    | x | y |   |   | z |   |  |  |   |
| Multiply    |   | x | y |   | z |   |  |  |   |
| Add 1       |   |   | x | y |   | z |  |  |   |
| Add 2       |   | x | y |   | z |   |  |  |   |
| Store State |   |   |   | x | y |   |  |  | z |

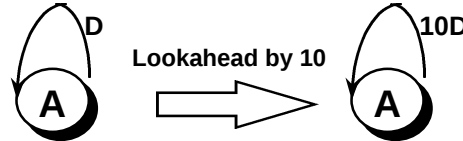
Copyright © 1995-1999 SCRA

44

The crosses mark the time units (along x-axis) when the operation is active.

## What is Lookahead? (Cont.)

- | Lookahead reduces the IPB, but at the cost of additional hardware (adders and multipliers)
- | Lookahead adds delay (or memory) into the feedback loop
  - m E.g.,  $x[n]$  depends on  $x[n-2]$  => Two delays
  - $x[n]$  depends on  $x[n-1]$  => One delay
- | Iteration period bound =  $\frac{\text{Total Computational Latency in Loop}}{\text{Total Number of Delays}}$ 
  - m Lookahead increases the denominator
- | Alternative

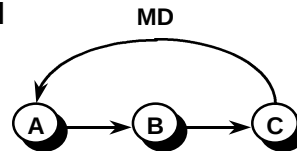


Lookahead improves the sample period bound at the cost of additional hardware (similar to the carry lookahead adder).

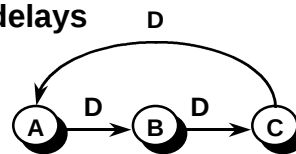
## Flowchart

### Flowchart for optimization with Lookahead and Retiming

| **Step 1: Use Lookahead by M**



| **Step 2: Retime to distribute delays**



| **Step 3: Obtain a static rate-optimal schedule**

The above three-step procedure leads to optimal schedules.



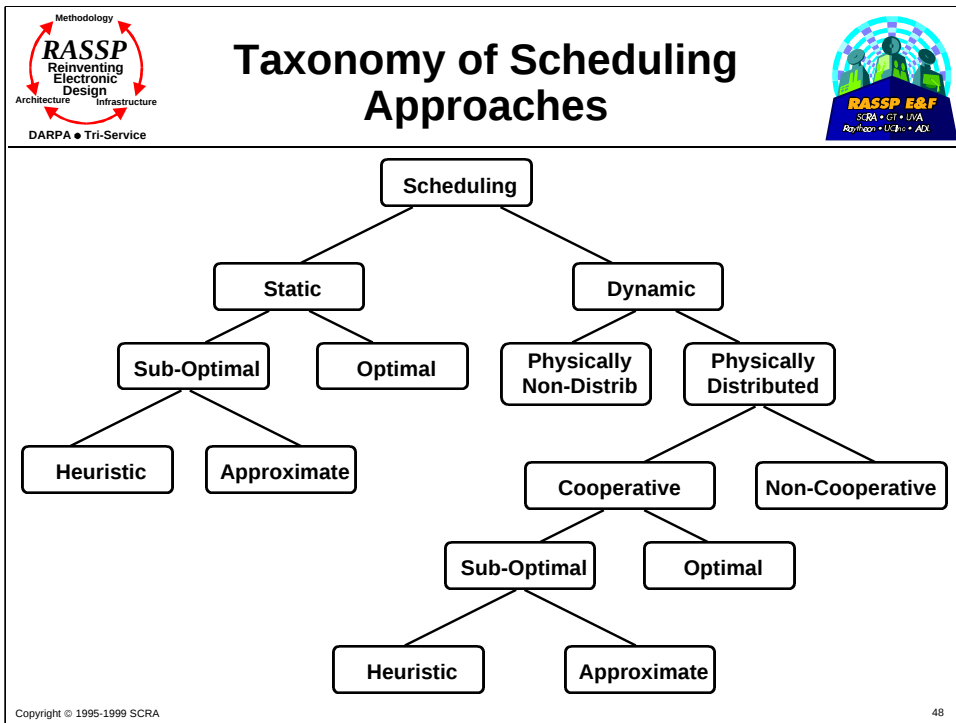
## Module Outline



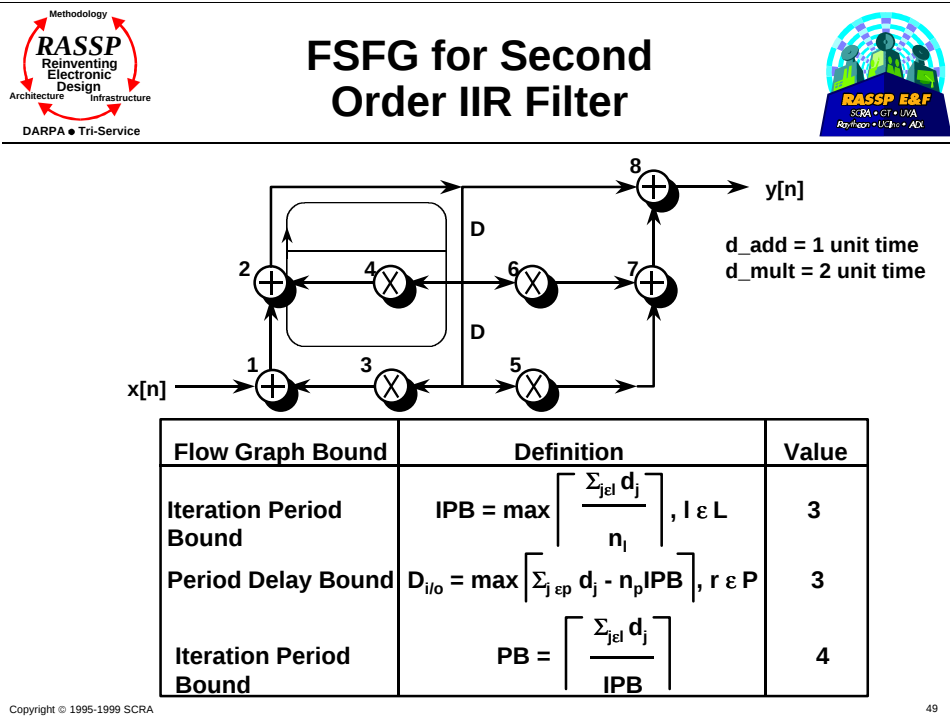
- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | **Scheduling Approaches**
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | Summary

Copyright © 1995-1999 SCRA

47

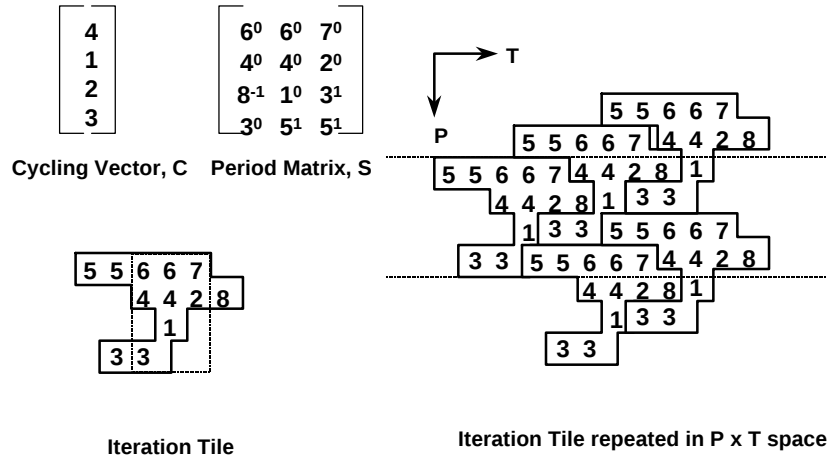


This chart lists various possible heuristic and non-heuristic methods.



We now introduce methods for non-ideal systems (e.g., communication delays and finite resources).

## Second Order Filter- A Cyclostatic Schedule



Recapitulation of scheduling. The cycling vector and the period matrix together determine the P x T tiling.

The superscripts represent the iteration numbers.

Methodology  
  
**RASSP**  
 Reinventing  
 Electronic  
 Design  
 Architecture Infrastructure  
 DARPA • Tri-Service

# Comparison of Several Scheduling Algorithms



**RASSP E&F**  
 SCRA • CT • USA  
 Reprogram • U.S. • ADX

| Scheduling                           | Iteration Period | No. of Processors | Throughput Delay        | Comm. Considered |
|--------------------------------------|------------------|-------------------|-------------------------|------------------|
| Single Iteration                     | not optimal      | not optimal       | not optimal             | no               |
| Direct Blocking                      | not optimal      | not optimal       | not optimal             | no               |
| Maximum-Spanning tree                | not optimal      | optimal           | not optimal             | no               |
| Optimum Unfolding                    | optimized        | optimal           | not optimal             | no               |
| Cyclo-Static                         | optimal          | optimal           | optimal                 | no               |
| CSPP                                 | optimal          | optimal           | optimal                 | no               |
| Generalized PSSIMD                   | optimal          | optimal           | optimal                 | yes              |
| Scheduling-Range   Fixed rate Max TP | optimized fixed  | optimal optimized | not optimal not optimal | yes yes          |
| DSMP-C1                              | optimal          | optimal           | optimal                 | yes              |
| MULTIPROC                            | optimal          | optimal           | optimal                 | yes              |

Copyright © 2006-2010 RASSP

51

Note that we now have a formal approach of comparing various scheduling methods based on our optimality criteria (See [Madiseti95]).



## Module Outline



- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | **Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources**
- | Summary

Copyright © 1995-1999 SCRA

52



## Optimal Scheduling and Assignment



- | **Focus on an optimal scheduling and assignment methodology**
- | **Optimize objective function subject to**  
m **Constraints**

Copyright © 1995-1999 SCRA

53

This optimization-based methodology can be found in  
[Madisetti95] V.Madisetti, "VLSI Digital Signal Processors", IEEE Press, 1995.

## IPB Model

| **Minimize IPB subject to**

m **Precedence constraints:** Node v precedes node w (arc v->w)  
 $t_w - t_v > d_w - n_{vw} \text{ IPB, for } e(v,w) \in E$

m **Non-negativity constraints:** IPB,  $t_i > 0$ , for  $i = 1, \dots, N$

| **For the second order IIR filter, this problem becomes**

m **Minimize IPB**

m **Subject to:**

q **Precedence Constraints**

$$\begin{array}{llll} t_1 - t_3 > 1 & t_2 - t_1 > 1 & t_2 - t_4 > 1 & t_4 - t_2 + \text{IPB} > 2 \\ t_3 - t_2 + 2 \text{ IPB} > 2 & t_8 - t_2 > 1 & t_8 - t_7 > 1 & t_7 - t_6 > 1 \\ t_7 - t_5 > 1 & t_6 - t_2 + \text{IPB} > 2 & t_5 - t_2 + 2 \text{ IPB} > 2 & \end{array}$$

q **Non-negativity constraints**

$$t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, \text{IPB} > 0$$

We illustrate the typical scheduling problem as an integer programming problem. Efficient methods exist to solve integer programs.

## Finite Resources Model

- | **Minimize the number of processors  $P$  and periodic throughput delay  $D_{i/o}$**

$$T \sum_{j=0}^{2^j} p_j + D_{i/o}$$

- | **Subject to**

m **Precedence constraints:** Node  $v$  precedes node  $w$  (arc  $v \rightarrow w$ ).  $t_w - t_v > d_w - n_{vw} IP$ , for  $e(v,w) \in E$  where  
 $t_i = \sum_{t=1}^T x_{it}$

m **Job completion constraints:** All nodes must be scheduled  $\sum_{t=1}^T x_{it} = 1$ , for  $i = 1, \dots, N$

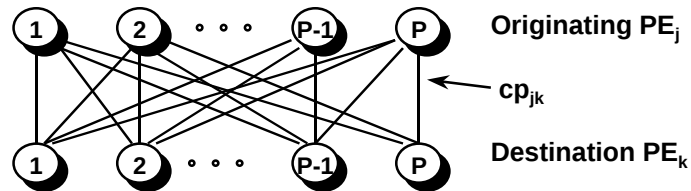
m **Processor modulo and non-preemption constraints:**  
**At most  $P$  processors can be scheduled at a time**  
 $\sum_{i=1}^N \sum_{q \in Q_n} x_{iq} < \sum_{j=0}^{2^j} p_j$ ,  $t=1, \dots, T$   $n = 0, \dots, IPB-1$   
 where  $Q_n = \{q \mid q \bmod IP = n, \text{ for } q = t \text{ to } t+d_i-1\}$

m **0-1 constraints:**  $x_{it} \in \{0,1\}$  for  $i = 1, \dots, N$  and  $t = 1, \dots, T$   
 $p_j \in \{0,1\}$  for  $j = 1, \dots, u$

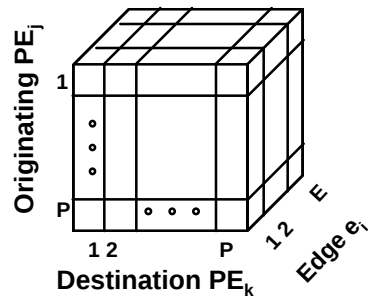
We can also solve the scheduling problem for a finite number of processors.

## Adding Communications

### Communication paths



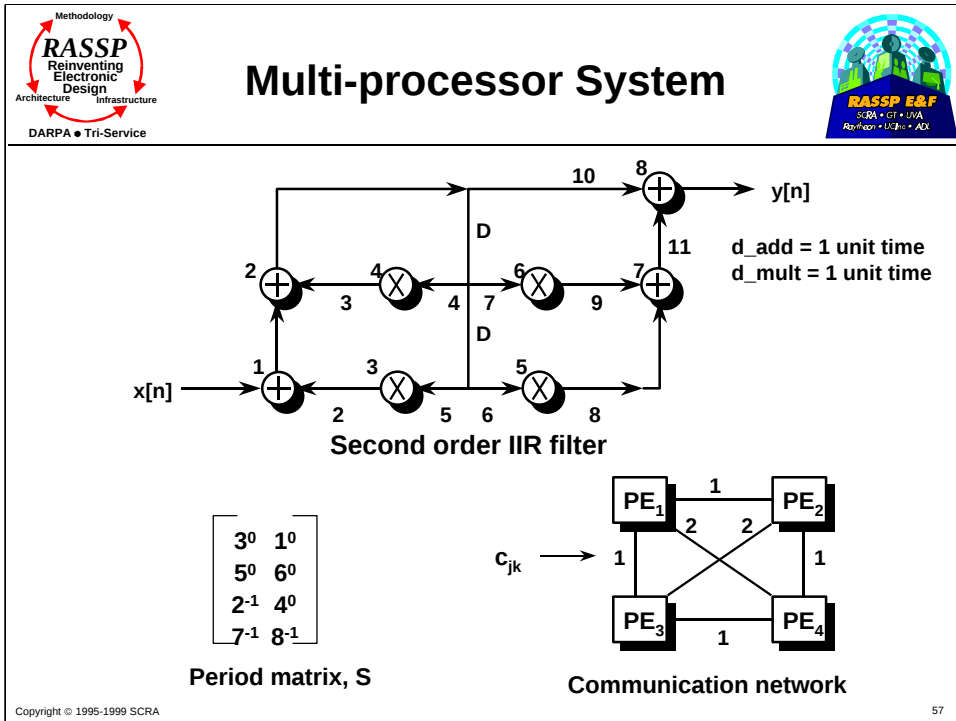
Decision variable  $x_{ejk}$



Copyright © 1995-1999 SCRA

56

The real benefit of this approach comes when it handles non-zero communication costs.



We now show how one can map a schedule on a target multiprocessor architecture with known costs.

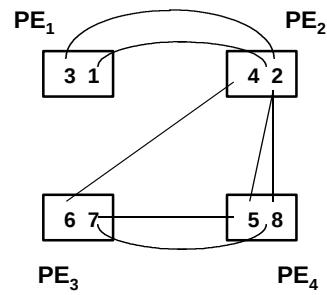
## Second Order IIR Filter

| Edge | Comm. Path |
|------|------------|
| 1    | 1 --> 2    |
| 2    | 1 --> 1    |
| 3    | 2 --> 2    |
| 4    | 2 --> 2    |
| 5    | 2 --> 1    |
| 6    | 2 --> 4    |
| 7    | 2 --> 3    |
| 8    | 4 --> 3    |
| 9    | 3 --> 3    |
| 10   | 2 --> 4    |
| 11   | 3 --> 4    |

Edge mapping

| PE | Nodes |
|----|-------|
| 1  | 3, 1  |
| 2  | 4, 2  |
| 3  | 6, 7  |
| 4  | 5, 8  |

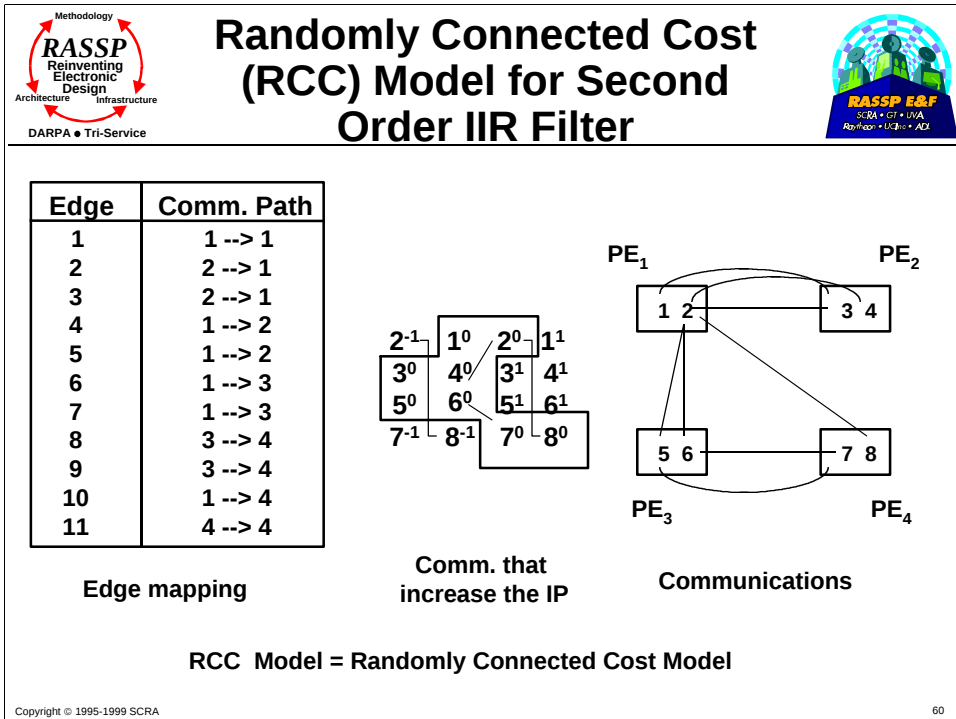
PE assignment



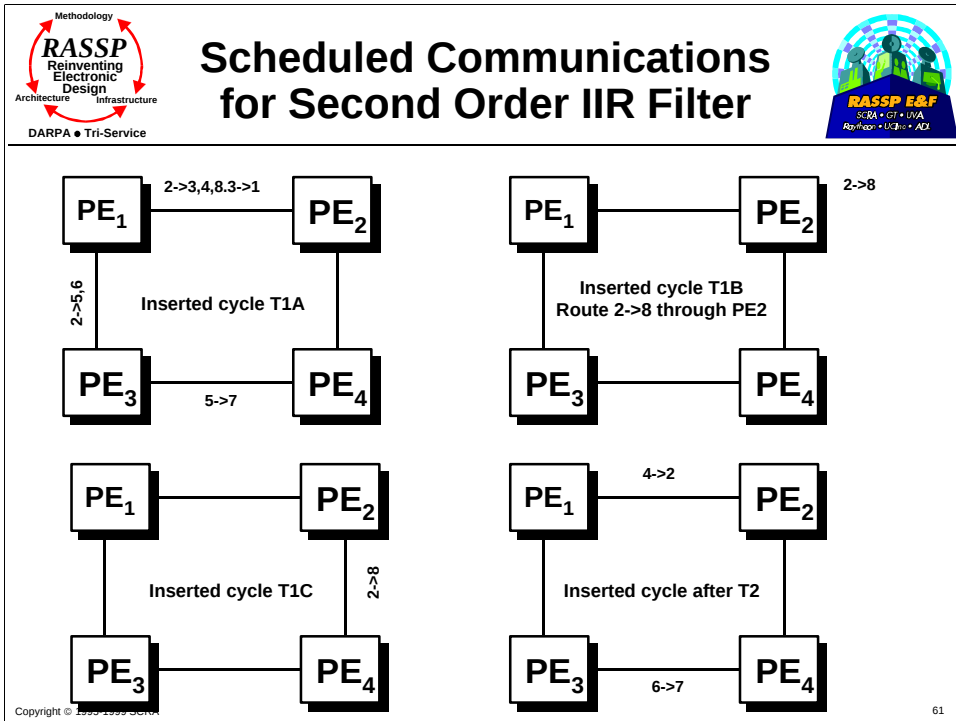
Communications

Note the two steps involved in the edge mapping and PE assignment.



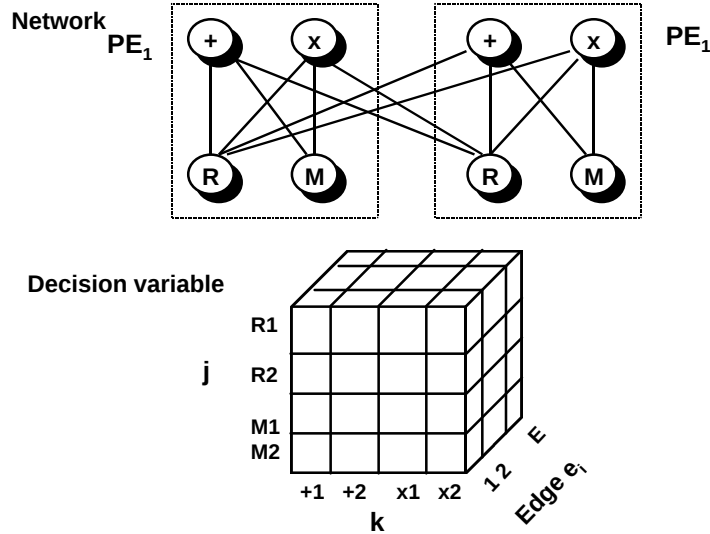


RCC = Randomly Connected Cost Model.

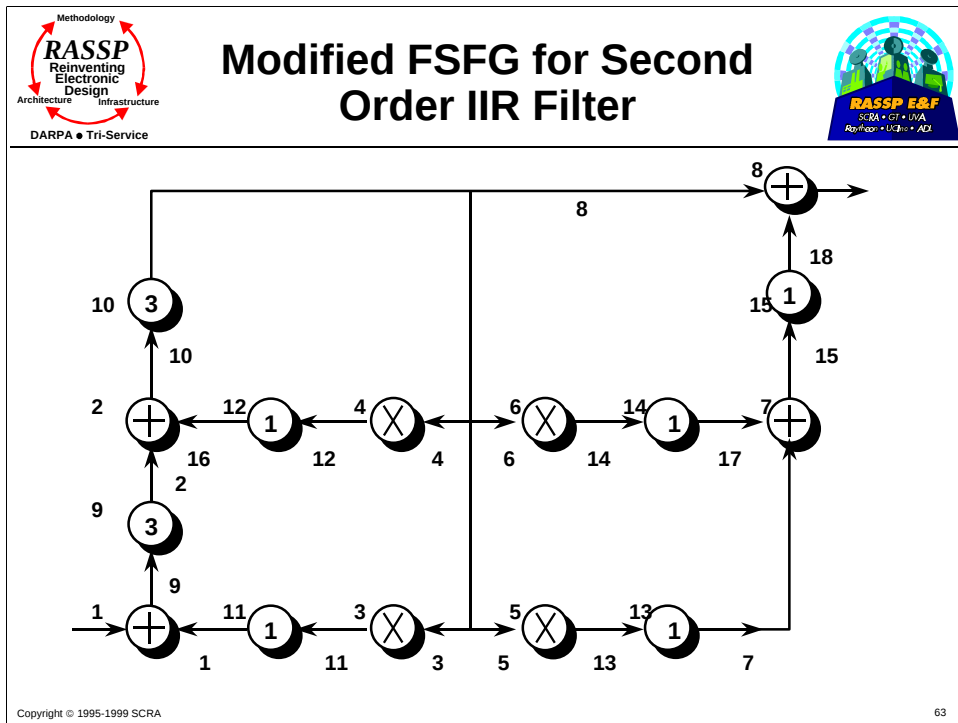


All communications are strictly scheduled due to the deterministic nature.

## Including Memory and Registers



Memory and storage can also be included. This implies that the same framework can be used for synthesis.



We now arrive at this intermediate FSFG that can be used as the input to the next steps in the scheduling process.

## Comparison of Processor Mapping Models

| Model                                                              | Min Increase IP | Max Num of Comm | Proc  | Network | Multiple | Reg & Mem |
|--------------------------------------------------------------------|-----------------|-----------------|-------|---------|----------|-----------|
| Fully connected cost                                               | Yes             | No              | Given | Full    | No       | No        |
| Fully connected cost capacity                                      | Yes             | Yes             | Given | Full    | No       | No        |
| Randomly connected cost                                            | Yes             | No              | Given | Random  | No       | No        |
| Randomly connected cost & capacity                                 | Yes             | Yes             | Given | Random  | No       | No        |
| Randomly connected cost & capacity with Multiple FU                | Yes             | Yes             | Given | Random  | Yes      | No        |
| Randomly connected cost & capacity with Multiple FU & Memory & Reg | Yes             | Yes             | Given | Random  | Yes      | Yes       |
| Combined Scheduler                                                 | Min IP          | No              | Given | Random  | No       | No        |
| Combined Scheduler with Multiple FU                                | Min IP          | Yes             | Given | Random  | Yes      | Yes       |

Summary of the constrained scheduling problem.



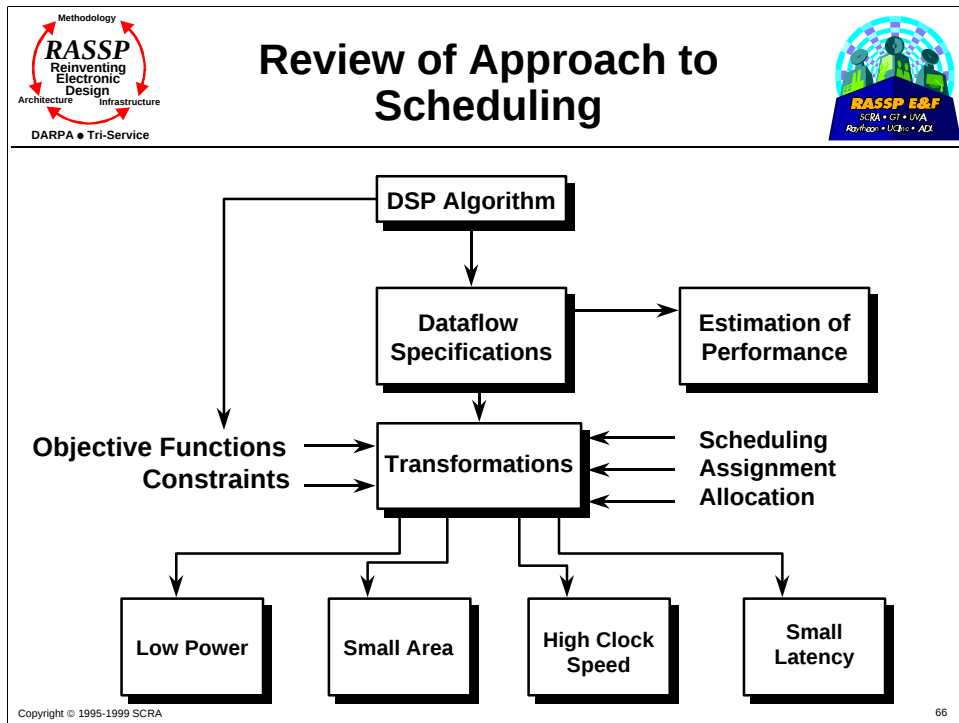
# Module Outline



- | Introduction
- | FSFG as a Representation of DSP Algorithms
- | Measures of Performance
- | Retiming
- | Iteration Period and Iteration Period Bounds
- | Formal Methods for FSFGs
- | Perfect-rate FSFGs and Unfolding
- | Lookahead
- | Scheduling Approaches
- | Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- | **Summary**

Copyright © 1995-1999 SCRA

65



We have presented a concise but complete introduction to the scheduling, assignment, and allocation problem for multiprocessor systems (both ideal and non-ideal cases).



## Summary



- | **A formal approach to recent results to the scheduling and assignment of non-ideal systems has been presented**
- | **This framework is very powerful and flexible for a variety of computational granularities found in signal processing**
- | **RASSP is currently developing a number of constraint-based schedulers**

Copyright © 1995-1999 SCRA

67

RASSP has utilized the use of graphical driven front-end approaches that capture the algorithmic specification. In RASSP the building blocks within the FSFG are of a coarser granularity (e.g., FFTs as opposed to multiply/adds), but the results developed in this module do not change.



## References



[IEEE] All referenced IEEE material is used with permission.

[Madisetti95] V.Madisetti, "VLSI Digital Signal Processors", *IEEE Press*, 1995; © IEEE 1995

[Richards97] Richards, M., Gadiant, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997.