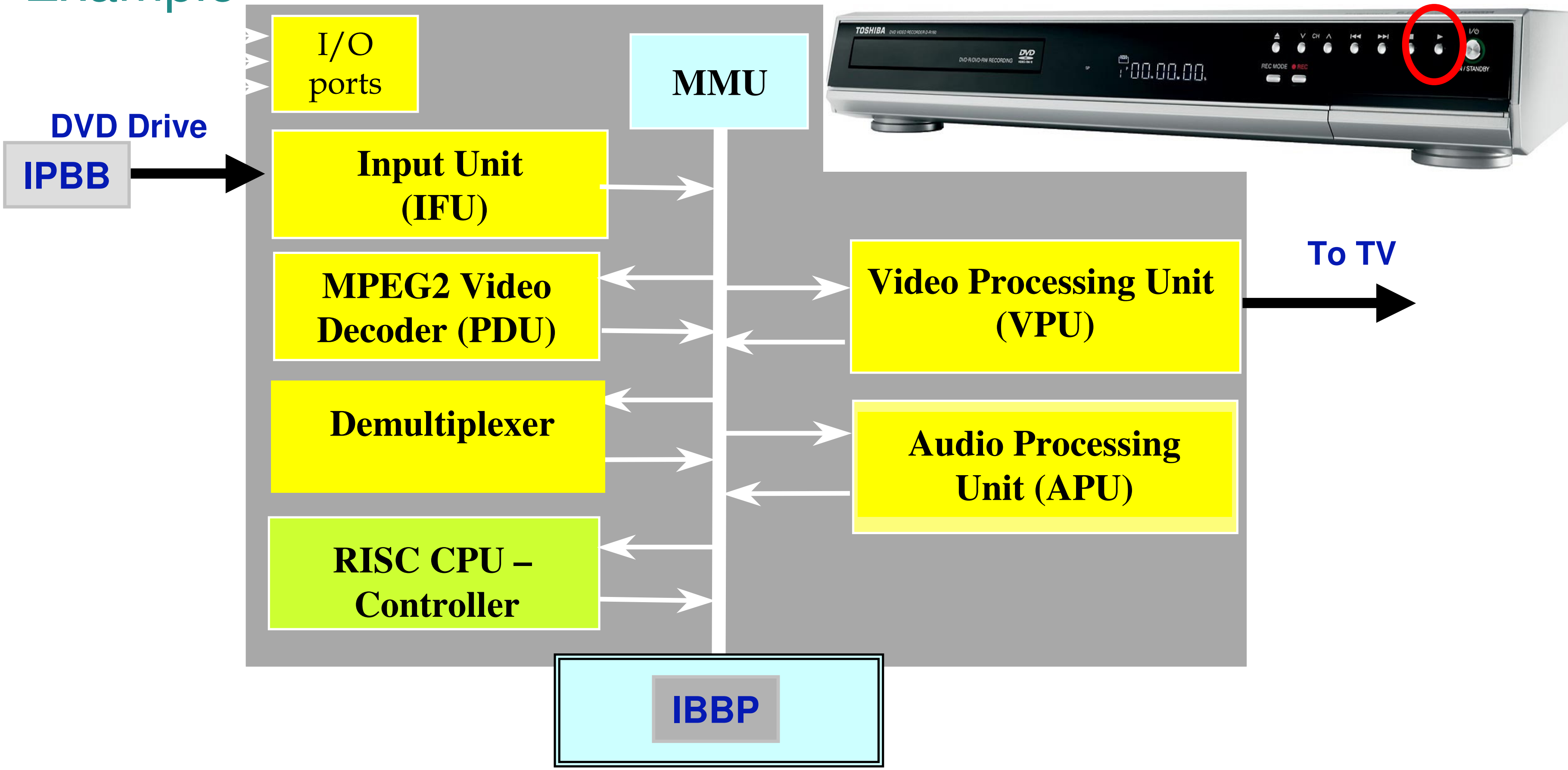
Using Linear Programming Techniques for Scheduling-based Random Test-case Generation

Amir Nahir
Yossi Shiloach
Avi Ziv

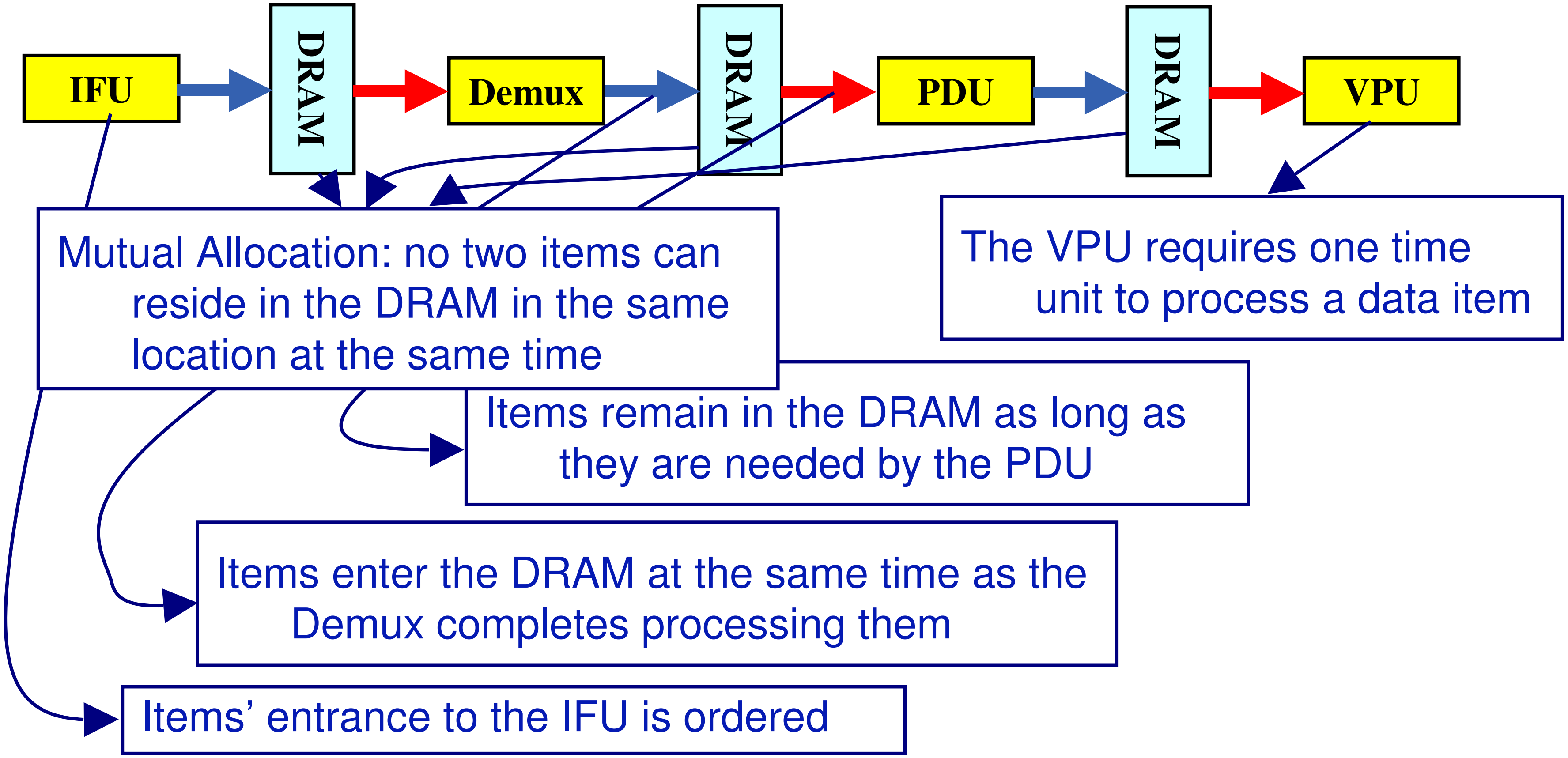
Agenda

- What is Scheduling-based Test-case Generation
- CSP Model
- Mixed Integer Linear Problems (MIP)
- MIP Model
- Results
- Future Directions
  - Combining CSP & MIP

Scheduling-based Test-case Generation – DVD Player SoC Example



Constructing a scheduling problem



Scheduling Solution

<table>
<tr><td></td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
<tr><td>IFU</td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td colspan="2">B</td><td></td><td></td><td></td><td></td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td colspan="2">B</td><td></td><td></td><td></td></tr>
<tr><td>Demux</td><td></td><td></td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td colspan="2">B</td><td></td><td></td><td></td><td></td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td colspan="2">B</td><td></td></tr>
<tr><td>PDU</td><td></td><td></td><td></td><td></td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td colspan="2">B</td><td></td><td></td><td></td><td></td><td colspan="2">I</td><td colspan="2">P</td><td colspan="2">B</td><td>B</td></tr>
<tr><td>VPU</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>I_B</td><td>I_T</td><td>B_B</td><td>B_T</td><td>B_B</td><td>B_T</td><td>P_B</td><td>P_T</td><td>P_B</td><td>P_T</td><td>P_B</td><td>P_T</td><td>I_B</td><td>I_T</td></tr>
<tr><td>VPU-N</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>I_T</td><td>B_B</td><td>B_T</td><td>B_B</td><td>B_T</td><td>P_B</td><td>P_T</td><td>P_B</td><td>P_T</td><td>P_B</td><td>P_T</td><td></td><td>I_T</td><td>B_B</td></tr>
</table>

Scheduling Rules

- Basic rules:
  - Duration of processing
  - Order within the processing core
  - Order between processing cores
- More Complex rules:
  - Mutual exclusion: two items cannot be processed by the same core at the same time
  - Mutual allocation: two items cannot be stored in the same location at the same time

Scheduling Rules (cont.)

- Special DVD rules:
  - Whenever the VPU has no new data items to process, the controller instructs the VPU to display the last two data items again
    - Causing the image on the viewer's screen to freeze
  - The VPU-Next core processes the data item that will be processed by the VPU at the following cycle
    - Unless the VPU is idle, or the data item to be processed is from a different group. In these cases, VPU-Next should be idle

VPU										I_B	I_T	B_B	B_T	B_B	B_T	P_B	P_T	P_B	P_T	P_B	P_T	I_B	I_T
VPU-N										I_T	B_B	B_T	B_B	B_T	P_B	P_T	P_B	P_T	P_B	P_T		I_T	B_B

Constraint Satisfaction Problems - Definition

[Mackworth, Freuder, Montanari, Dechter, Rossi, ...]

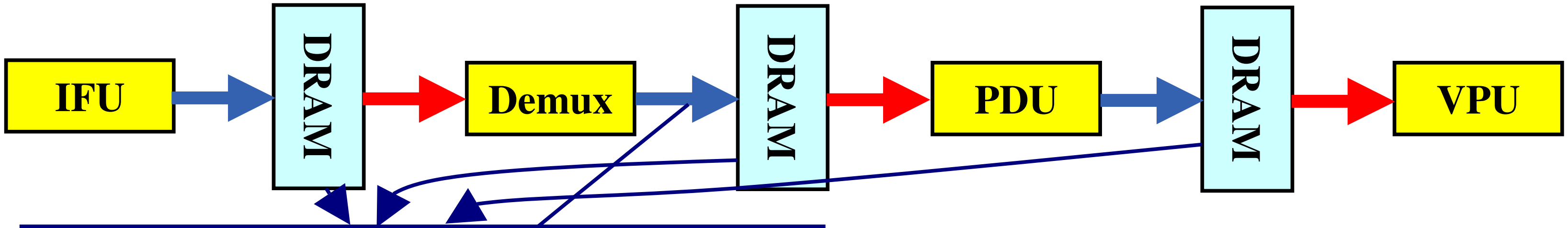
- CSP P = {V, D, C}
- Variables
  - Address, register_value
- Domains (finite sets) for each variable
  - Address: 0x0000 - 0xFFFF
  - Number of bytes in a 'load': { 1, 2, 4, 8, 16 }
- Constraints (relations) over variables
  - (load n bytes) ➔ (align address to n bytes boundary)
  - value(base_reg) + displacement = address
- Solution for a CSP
  - Every variable is assigned a value from its domain, such that all constraints are satisfied
    - All solutions are born equal. There is no better or best solution!

Modeling our scheduling problem using CSP

[Based on a well-known Scheduling->CSP reduction]

- Variables
  - <core , data-item, enter-time / exit-time / address>
    - IFU.I-Frame.Enter-time
    - DRAM.B-Frame.Exit-time
    - DRAM.B-Frame.Address
- Domains
  - For enter/exit variables – discrete time (clock cycles)
  - For address variables – DRAM addresses

CSP Constraints



Mutual Allocation: no two items can reside in the DRAM in the same

not (((DRAM.Item(i).Address + item(i).size) < DRAM.item(j).Address) or ((DRAM.item(j).Address + item(j).size) < DRAM.item(i).Address))) ->
((DRAM.item(i).Exit-time <= DRAM.item(j).Enter-time) or
(DRAM.item(j).Exit-time <= DRAM.item(i).Enter-time))

This constraint has to be applied to every pair of items in the DRAM

Items enter the DRAM at the same time as the Demux co

DRAM.Item(i).Enter-time = Demux.Item(i).Exit-time

Modeling our scheduling problem using CSP – the harder cases…

- Special DVD rules:
  - Whenever the VPU has no new data items to process, the controller instructs the VPU to display the last two data items again

How can we express these rules over Enter / Exit / Address variables?

  - The VPU-Next core processes the data item that will be processed by the VPU at the following cycle
    - Unless the VPU is idle, or the data item to be processed is from a

some auxiliary variables and complex constraints were added to the model…

VPU																						
VPU-N										I_T	B_B	B_T	B_B	B_T	P_B	P_T	P_B	P_T	P_B	P_T		I_T	B_B

Mixed Integer Linear Problems - Definition

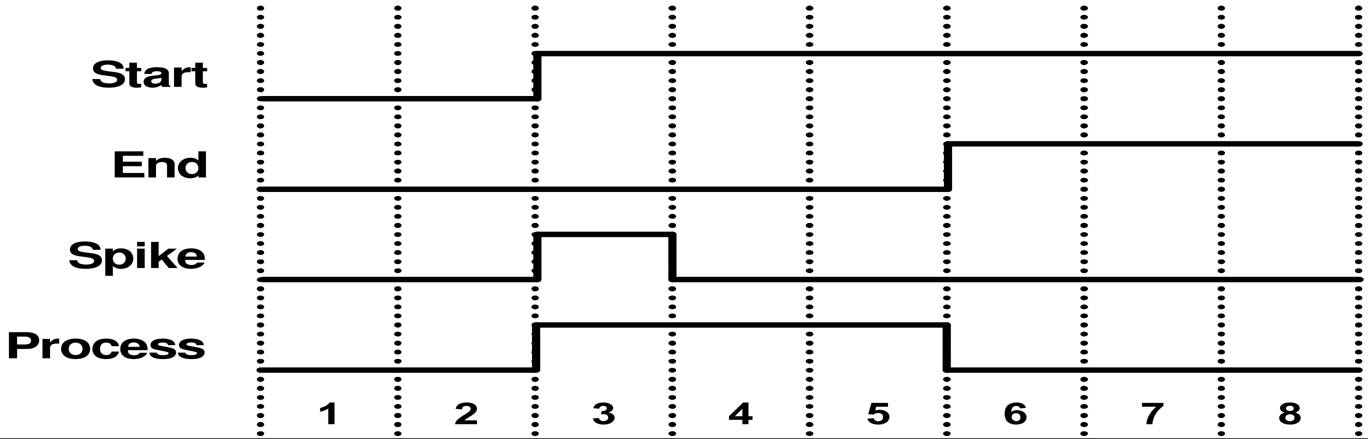
- MIP P = {V,R,C,f}
- Variables
- Ranges
  - Upper bound & Lower bound
- Constraints
  - Linear constraints $\sum_{v_i \in V} a_i \cdot v_i \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b$
  - Constrain certain variables to receive integer values only
- A linear objective function

Mixed Integer Linear Problems – Definition (cont.)

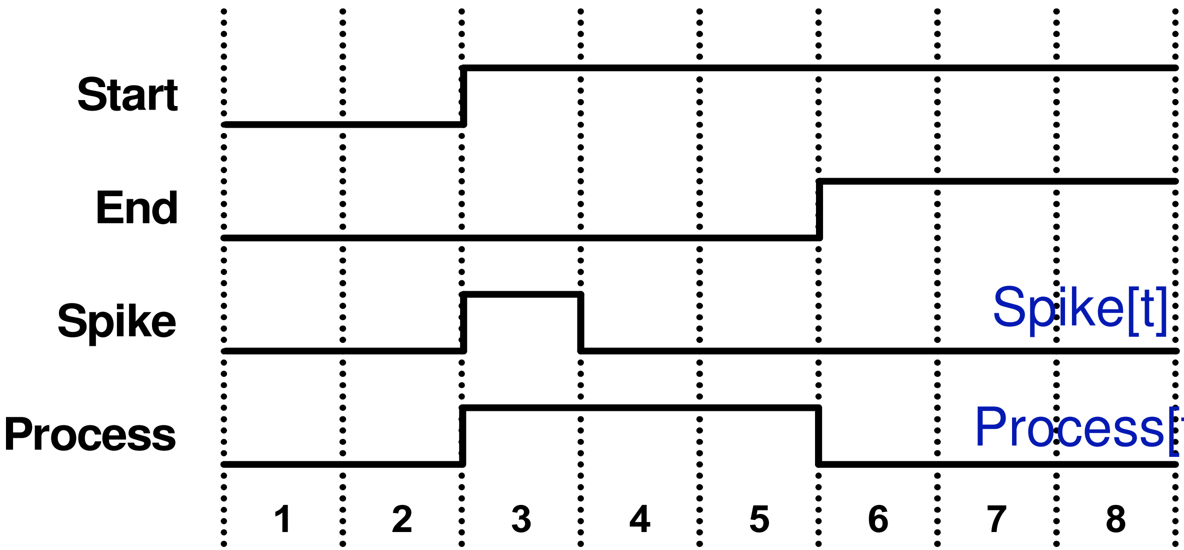
- Feasible solution for a MIP
  - Every variable is assigned a value from its range, such that all constraints are satisfied
- Optimal solution for a MIP
  - A feasible solution, such that there exists no other solution with lower value to the objective function
- MIP Solvers find <u>optimal</u> solutions

Modeling our scheduling problem using MIP

- Ranges
  - Only [0,1]
  - Decision Problems
- Variables
  - For each <core , data-item>:



Constructing the MIP "functions"



$Start[t] \geq Start[t-1]$

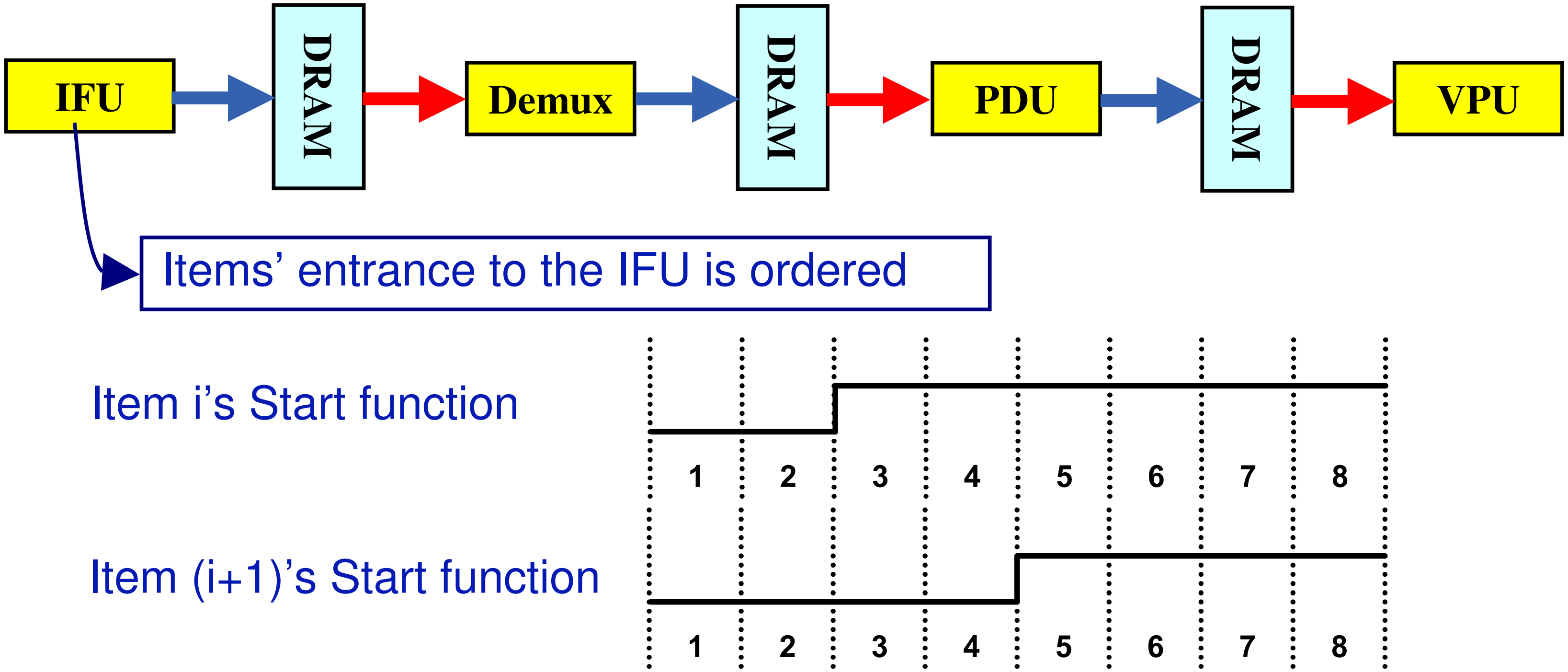
$End[t] \geq End[t-1]$

$Spike[t] = Start[t] - Start[t-1]$

$Process[t] = Start[t] - End[t]$

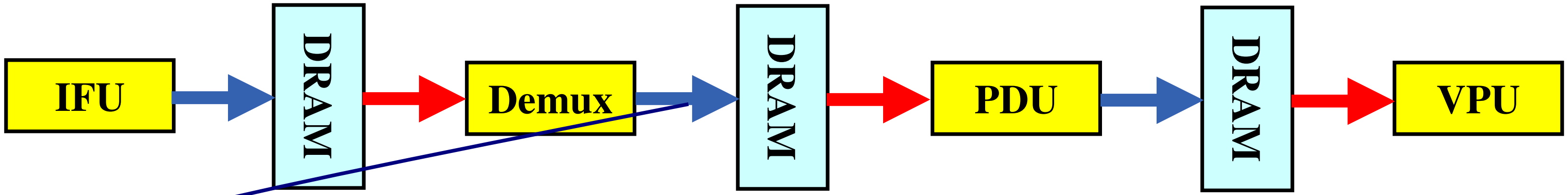
Not all variables are constrained to integers (but all variables will eventually be assigned with integer values)

MIP Constraints



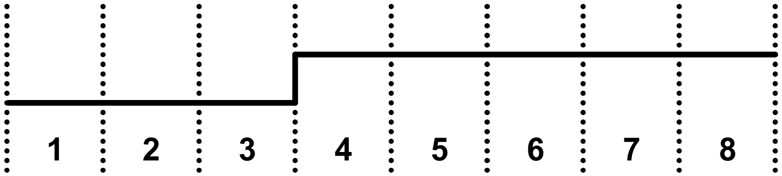
Item i's Start[t] ≥ Item (i+1)'s Start[t]

MIP Constraints (cont.)

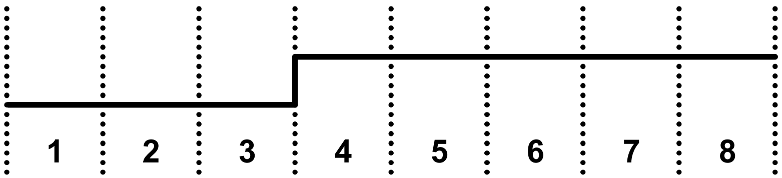


Items enter the DRAM at the same time as the Demux completes processing them

Item i's DRAM Start function

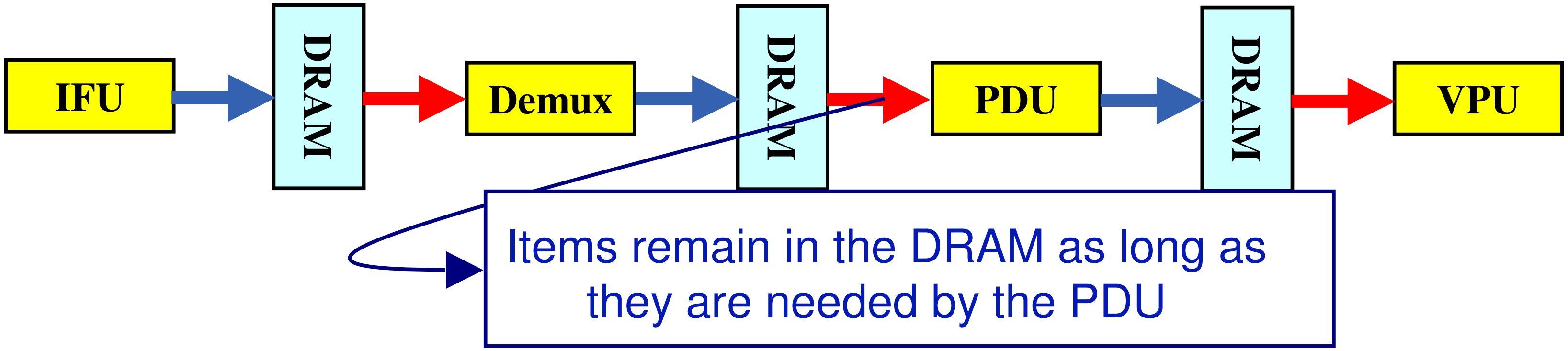


Item i's Demux End function



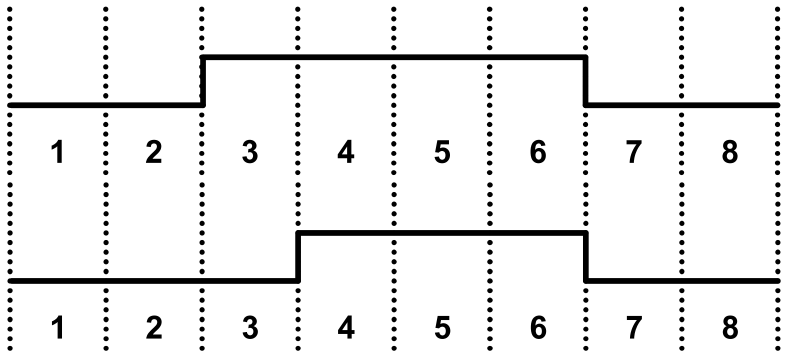
Item i's DRAM Start[t] = Item i's Demux End[t]

MIP Constraints (cont.)



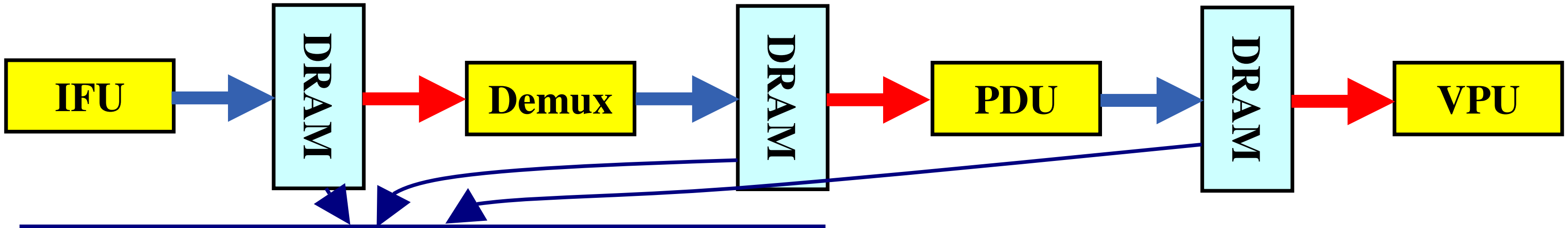
Item i’s DRAM Processing function

Item i’s PDU Processing function



Item i’s DRAM Processing[t] ≥ Item i’s PDU Processing[t]

MIP Constraints (cont.)



Mutual Allocation: no two items can reside in the DRAM in the same location at the same time

Sum(Item i's size x Item i's DRAM Processing[t]) ≤ DRAM size

Modeling our scheduling problem using MIP – the harder cases…

Remember these?

- Special DVD ru
  - Whenever the VPU has no new data items to process, the controller instruc … again
    - Ca

$$\sum_{j\in VPU} \mathbf{Process}_j[t] \ge \sum_{j\in VPU} \mathbf{Process}_j[t-1]$$

  - The VPU-Next core processes the data item that will be processed

$$\sum_{\substack{j\in VPU\text{-}Next \\ j\in Group_i}} \mathbf{Process}_j[t] = \mathbf{Min}\ \{\sum_{\substack{j\in VPU \\ j\in Group_i}} \mathbf{Process}_j[t]\ ,\ \sum_{\substack{j\in VPU \\ j\in Group_i}} \mathbf{Processj}[t+1]\}$$

VPU										I_B	I_T	B_B	B_T	B_B	B_T	P_B	P_T	P_B	P_T	P_B	P_T	I_B	I_T
VPU-N										I_T	B_B	B_T	B_B	B_T	P_B	P_T	P_B	P_T	P_B	P_T		I_T	B_B

MIP & Randomness

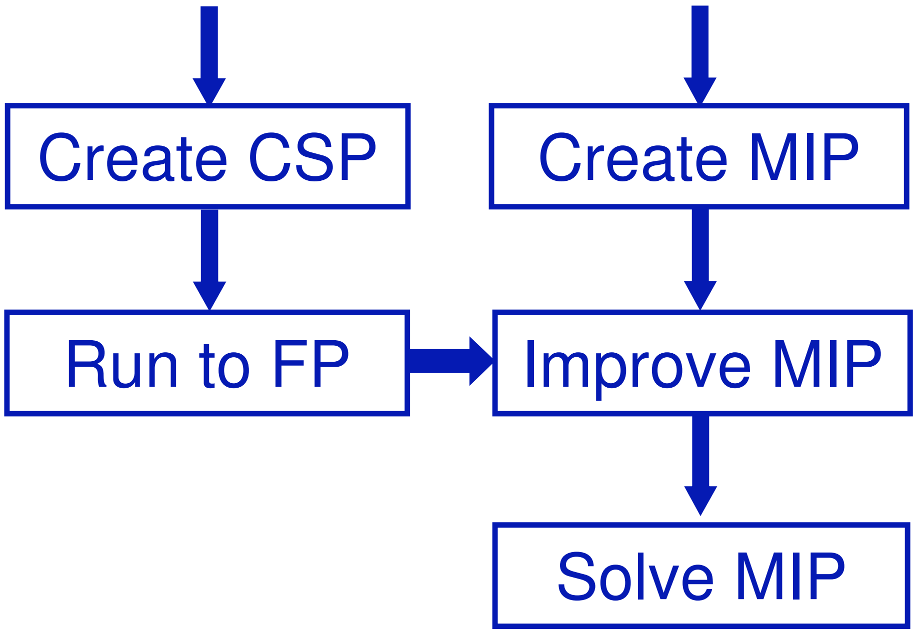
- Test-case generators are required to generate many different solutions from the same test specification
- MIP solvers are optimizers – they seek the <u>optimal</u> solution
- Our solution: we randomly select a set of variables and add them to the objective function
  - In every execution, the solver is driven towards a different solution

Experimental Results

Test name	Framework	Variables	Constraints	Success Rate	Time to Success
Play(IPBB,18)	CSP	552	3077	100%	40.76
	MIP	9412	14429	100%	1.73
Play(IPBB,23)	CSP	582	3260	90%	129.42
	MIP	12002	18469	100%	18.99
Play(IPBBPBB,28)	CSP	945	7562	90%	529.99
	MIP	25410	39225	100%	90.75
Play(IPBBPBB,33)	CSP	975	7795	40%	2181.20
	MIP	29920	46265	100%	400.08

- An in-house solver was used for the CSP framework
- ILOG's CPLEX 10.0 was used for the MIP framework

Future Directions – Combining CSP & MIP



- x5 better than MIP, x20 Better than CSP!!!

Summary

- CSP is the prominent framework for test-case generation, and for good reasons
  - Expressiveness
  - Embedded randomness
- MIP is more efficient than CSP
  - Because of its strong mathematical foundations
- MIP can't replace CSP, but can assist it in difficult problems
  - By replacing CSP over specific problems
  - Through hybrid technologies

Questions Please