Compiling Polychronous Programs into Conditional Partial Orders for ASIP Synthesis

Sandeep K. Shukla,
FERMAT Lab,
Virginia Tech.

with
Mahesh Nanjundappa,
FERMAT Lab,
Virginia Tech.

Motivation

Current trends in hardware requirements

- ↑Performance, ↓Latency, ↓Power Consumption, ↓Form Factor
- ↑Programmability for enabling reuse of components
- ↑Flexibility to introduce late specification changes

Motivation

Current trends in hardware requirements

- ↑Performance, ↓Latency, ↓Power Consumption, ↓Form Factor
- ↑Programmability for enabling reuse of components
- ↑Flexibility to introduce late specification changes

Application Specific Instruction-set Processors (ASIPs)

- Designed to exploit special characteristics of class of applications
- Reuse of components based on programmable modes of operation
- Custom instruction sets allow to maintain a level of flexibility
- Balance between ASICs and general purpose processors

Requirements

Design methodologies for ASIPs should provide,

- Compact & efficient way to describe & store instruction sets
- Identify parallelism and express modes of operation
- A way to express available and required resources and map them
- Encoding of instruction sets for various optimization criteria

Requirements

Design methodologies for ASIPs should provide,

- Compact & efficient way to describe & store instruction sets
- Identify parallelism and express modes of operation
- A way to express available and required resources and map them
- Encoding of instruction sets for various optimization criteria

Conditional Partial Order Graphs (CPOGs) offer these facilities!

Outline of the talk

Outline of the talk

1. Motivation
2. Introduction to CPOGs
3. Introduction to MRICDF
4. MRICDF Models to CPOGs
5. Analysis and ASIP Synthesis
6. Conclusion and Future

Conditional Partial Order Graphs

- A compact semantic model to express and compose large partial order sets
- Yields itself very easy for transformations, refinements, optimizations and encodings
- Graphically they can be visualised as hierarchical, annotated, weighted, directed graphs

Formally CPOG is represented as a quintuple $G = \langle V, E, X, \rho, \phi \rangle$

- V is a set of *nodes* which corresponds to events/atomic actions in a system that is being modelled.
- $E \subseteq V \times V$ is a set of directed *edges* between the *nodes*. An edge from node n to node m, indicates action m depends on n.
- X is a set of n Boolean variables. Each Boolean variable could be assigned values $\{0, 1\}$ resulting in unique 2^n possible codes.
- ρ is a *restriction function* defined on the set of Boolean variables in X as $\rho \in \mathcal{F}(X)$, where $\mathcal{F}(X)$ is the set of all Boolean functions on the Boolean variables in X.
- Function $\phi : (V \cup E) \rightarrow \mathcal{F}(X)$. It assigns a Boolean condition $\phi(z)$ to every node and edge z in the graph G.

Example of CPOG

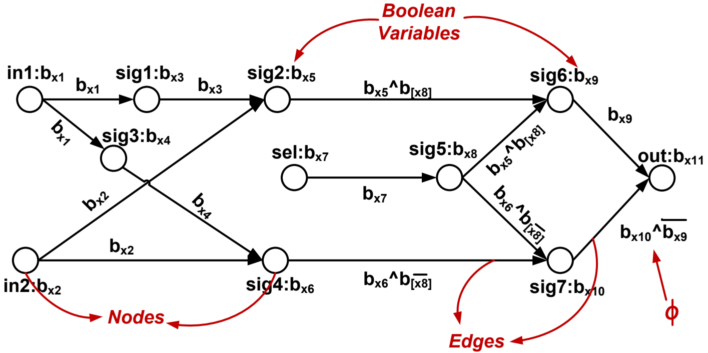


Figure: Graphical representation of CPOG

Example : Simple adder/subtractor application

- Does add or subtract based on select signal
- Table below shows micro-steps of instructions for example

Adder(A=A+B; *select*)	Subtractor(A=A-B; $\overline{select}$)
I_1:Load A	I_1:Load A
I_2:Load B	I_2:Load B
I_3:Compute A+B	I_5:Compute A-B
I_4:Store A	I_4:Store A

Example : Simple adder/subtractor application

- Does add or subtract based on select signal
- Table below shows micro-steps of instructions for example

Adder(A=A+B; *select*)	Subtractor(A=A-B; $\overline{select}$)
I_1:Load A	I_1:Load A
I_2:Load B	I_2:Load B
I_3:Compute A+B	I_5:Compute A-B
I_4:Store A	I_4:Store A

- Representing this instructions as CPOG H
  - Create 5 nodes: I_1, I_2, I_3, I_4, I_5
  - Create 6 edges: $I_1 \xrightarrow{select} I_3$, $I_2 \xrightarrow{select} I_3$, $I_3 \xrightarrow{select} I_4$, $I_1 \xrightarrow{\overline{select}} I_5$, $I_2 \xrightarrow{\overline{select}} I_5$, $I_5 \xrightarrow{\overline{select}} I_4$
  - Create Boolean variable set $X = \{select\}$
  - Establish ρ and ϕ functions

Encoding

- Atomic actions I_1 and I_2 can be executed concurrently or sequentially
- Atomic action I_2 has to be executed before I_3
- Partial order on the set of micro-steps/atomic actions
- Assigning values from the set $\{0,1\}$ to variables of X, to get unique Boolean vectors
- Unique vectors can be used as opcodes for instructions

CPOG representing execution of Simple adder/subtractor application

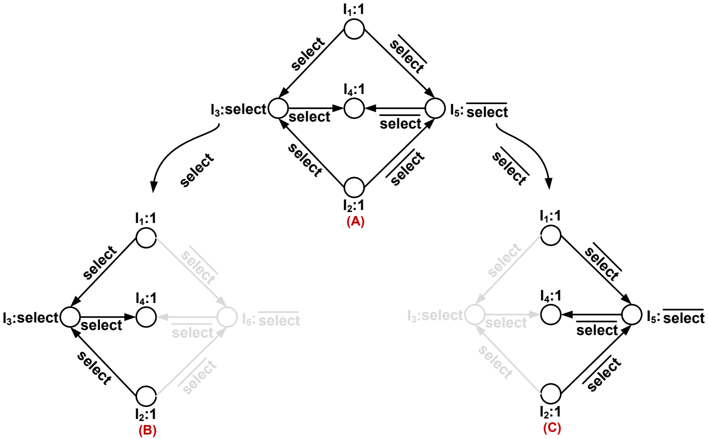


Figure: (A) Graphical representation of CPOG H, (B) $H|_{select}$, (C) $H|_{\overline{select}}$

Composition of Instruction sets

- Instruction is a pair $\mathcal{I} = (\phi, H|_{select})$, where ϕ is the opcode and $H|_{select}$ is the partial order
- Instruction set $\mathcal{IS}$ is a set of instructions - $\mathcal{IS} = \{\mathcal{I}_1, \mathcal{I}_2, ..\}$, such that each $\mathcal{I}_k$ has a different opcode ϕ

Composition of Instruction sets

- Instruction is a pair $\mathcal{I} = (\phi, H|_{select})$, where ϕ is the opcode and $H|_{select}$ is the partial order
- Instruction set $\mathcal{IS}$ is a set of instructions - $\mathcal{IS} = \{\mathcal{I}_1, \mathcal{I}_2, ..\}$, such that each $\mathcal{I}_k$ has a different opcode ϕ
- Composition of 2 instruction sets $\mathcal{IS}_i$ and $\mathcal{IS}_k$
    - Is defined as $\mathcal{IS}_i \cup \mathcal{IS}_k$
    - Is defined only when no instruction in set $\mathcal{IS}_i$ has same opcode as any instruction in $\mathcal{IS}_k$
    - Is not defined if there exists 2 instructions with same opcodes
- Composition of more than 2 instruction sets is done by performing pairwise composition in arbitrary order

Composition of Instruction sets

- Instruction is a pair $\mathcal{I} = (\phi, H|_{select})$, where ϕ is the opcode and $H|_{select}$ is the partial order
- Instruction set $\mathcal{IS}$ is a set of instructions - $\mathcal{IS} = \{\mathcal{I}_1, \mathcal{I}_2, ..\}$, such that each $\mathcal{I}_k$ has a different opcode ϕ
- Composition of 2 instruction sets $\mathcal{IS}_i$ and $\mathcal{IS}_k$
    - Is defined as $\mathcal{IS}_i \cup \mathcal{IS}_k$
    - Is defined only when no instruction in set $\mathcal{IS}_i$ has same opcode as any instruction in $\mathcal{IS}_k$
    - Is not defined if there exists 2 instructions with same opcodes
- Composition of more than 2 instruction sets is done by performing pairwise composition in arbitrary order
- Complexity of composition: Linear with respect to the total number of instructions

Outline of the talk

MRICDF - Multi-Rate Instantaneous Communication Data Flow

- A Visual Language (with a textual substitute) to express a computation over concurrent streams of data
- MRICDF model is hierarchical composition of actors
- Actors are connected using channels
- Signal flows via channels

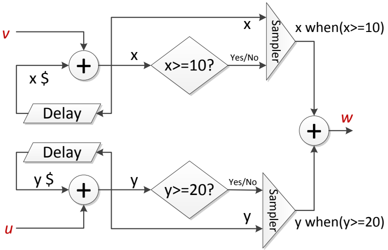


Figure: A simple MRICDF model

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definition (Signal)

A signal is a totally ordered sequence of events

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definition (Signal)

A signal is a totally ordered sequence of events

Definition (Instant set of a signal)

$\sigma(x)$ **– set of all instants where signal** x **has events**

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definition (Signal)

A signal is a totally ordered sequence of events

Definition (Instant set of a signal)

$\sigma(x)$ – **set of all instants where signal x has events**

Definition (Clock of a signal)

Instant set $\sigma(x)$ is also known as clock of signal denoted by $\hat{x}$

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definition (Signal)

A signal is a totally ordered sequence of events

Definition (Instant set of a signal)

$\sigma(x)$ – **set of all instants where signal** x **has events**

Definition (Clock of a signal)

Instant set $\sigma(x)$ **is also known as clock of signal denoted by** $\hat{x}$

Definitions

Definition (Event)

An occurrence of a fresh value on a signal constitutes an event

Definition (Signal)

A signal is a totally ordered sequence of events

Definition (Instant set of a signal)

$\sigma(x)$ **– set of all instants where signal** x **has events**

Definition (Clock of a signal)

Instant set $\sigma(x)$ **is also known as clock of signal denoted by** $\hat{x}$

Definition (Synchronous Signals)

Signals x **and** y **are called synchronous signals iff** $\hat{x} = \hat{y}$

Definitions

Definition (Data Dependence Relation)

• Multiple signals being read/written in an instant have a partial order - Dependency order

A dependency relation $x \xrightarrow{[c]} y$, indicates that the signal y is dependent on x, when condition c is *true*

- Data dependencies are not static, they change based on predicates

MRICDF Actors

- MRICDF consists of 4 primitive actors
- Numerous derived actors, Ex: Logical And, Multiplication, etc
- Every actor has a predefined set of Rate Constraints
- User can specify various synchronization requirements by adding additional clock constraints

Actor definition	Clock Relations	Data Dependency Relations
Function $r = a \star b$	$\sigma(a) = \sigma(b) = \sigma(r)$ $\hat{a} = \hat{b} = \hat{r}$	$a \to r$ $b \to r$
Buffer $y = x\$n\ init\ v_1..v_n$	$\sigma(y) = \sigma(x)$ $\hat{y} = \hat{x}$	No dependency
Sampler $y = x$ when z	$\sigma(y) = \sigma(x) \cap \sigma(z = true)$ $\hat{y} = \hat{x} \wedge [\hat{z}]$	$x \xrightarrow{[z]} y$
Merge $r = a$ default b	$\sigma(r) = \sigma(a) \cup \sigma(b)$ $\hat{r} = \hat{a} \vee \hat{b}$	$a \to r$ $b \xrightarrow{\hat{b}-\hat{a}} r$

Clock Calculus

- Determining relations between clocks and analysing is done in a step called - *Clock Calculus*
- Aim of clock calculus: To determine which signals participate in which reaction
- The signal that participates in each and every reaction - *Master Trigger* - multiple master triggers possible
- Clock of *Master Trigger* signal is *Master Clock*
- Clocks of signals that aren't master triggers can be derived based on predicates of either master clock or clocks of other known signals
- Hierarchically ordering these clocks gives us *Hierarchial Clock Relation Graph* (HCRG)
- Rooted HCRG : *Clock Tree*

Outline of the talk

1. Motivation
2. Introduction to CPOGs
3. Introduction to MRICDF
4. MRICDF Models to CPOGs
5. Analysis and ASIP Synthesis
6. Conclusion and Future

CPOG for Function Actor

Operation: $y = f(x_1, x_2, .., x_n)$, Clock relation: $\hat{y} = \hat{x}_1 = \hat{x}_2 = .. = \hat{x}_n$

- $V = \{y, x_1, x_2, ..., x_n\}$
- $E = \{x_i \rightarrow y \mid x_i \in (x_1, x_2, ..., x_n)\}$
- $X = \{\{b_y\} \cup \{b_{xi} | x_i \in (x_1, x_2, ..., x_n)\}\}$, $b_y = b_{x1} = b_{x2} = ... = b_{xn}$
- Function ϕ

$$\begin{aligned} \phi(y) &= b_y, \\ \phi(x_1) &= b_{x1}, \\ &\dots \\ \phi(x_n) &= b_{xn}, \\ \phi(x_1 \rightarrow y) &= b_{x1}, \\ \phi(x_2 \rightarrow y) &= b_{x2}, \\ &\dots \\ \phi(x_n \rightarrow y) &= b_{xn} \end{aligned}$$

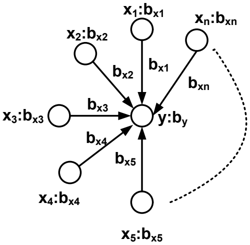


Figure: CPOG for Function Actor

CPOG for Buffer Actor

Operation: $y = x \ \$ \ 1 \ init \ c$
Clock relation: $\hat{y} = \hat{x}$

- $V = \{y, x\}$
- $E = \{\}$
- $X = \{b_y, b_x\}$
- $\rho = \{b_y = b_x\}$
- Function ϕ

$$\phi(y) = b_y$$
$$\phi(x) = b_x$$

Figure: CPOG for Buffer Actor

CPOG for Sampler Actor

Operation: $y = x \textit{ when } c$

Clock relation: $\hat{y} = \hat{x} * [c]$

- $V = \{y, x, c\}$
- $E = \{x \rightarrow y, c \rightarrow y\}$
- $X = \{b_y, b_x, b_c, b_{[c]}, b_{[\bar{c}]}\}$
- $$\rho = \left\{ \begin{array}{l} \{b_y = b_x \wedge b_{[c]}\} \cup \\ \{b_c = b_{[c]} \vee b_{[\bar{c}]}\} \cup \\ \{b_{[c]} \wedge b_{[\bar{c}]} = \mathit{false}\} \end{array} \right\}$$
- Function ϕ

  $$\phi(y) = b_y$$
  $$\phi(x) = b_x$$
  $$\phi(c) = b_c$$

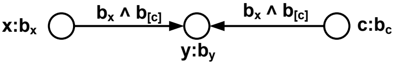


Figure: CPOG for Sampler Actor

CPOG for Merge Actor

Operation: $y = x \; default \; z$

Clock relation: $\hat{y} = \hat{x} + \hat{z}$

- $V = \{y, x, z\}$
- $E = \{x \to y, z \to y\}$
- $X = \{b_y, b_x, b_z\}$
- $\rho = \{b_y = b_x \vee b_z\}$
- Function ϕ

$$\begin{aligned} \phi(y) &= b_y \\ \phi(x) &= b_x \\ \phi(z) &= b_z \\ \phi(x \to y) &= b_x \\ \phi(z \to y) &= b_z \wedge \overline{b}_x \end{aligned}$$

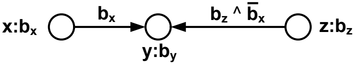


Figure: CPOG for Merge Actor

Observations

Observation

For each primitive actor A*, if* g_A *represents the CPOG derived using the steps described above, then* g_A *contains all the necessary information for control of scheduling the execution of* A*.*

Observation

For primitive actors A_1 *and* A_2*, if* g_{A_1} *and* g_{A_2} *represents the corresponding CPOGs then for composition* $A_1 \mid A_2$*, the corresponding CPOG is the* $g_{A_1} \cup g_{A_2}$*.*

Deriving CPOG for Composite Actor

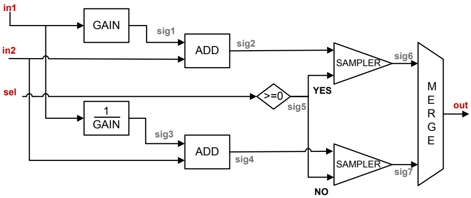
- Combination of primitive actors that are used to express modular and hierarchical behavior
- First we derive the CPOGs of composite actors and then compose ($\cup$) it with the CPOG of the rest of the model
- Algorithm 1 lists the method used to derive a CPOG for a composite actor

Algorithm 1: Algorithm to derive CPOG for a Composite Actor

```
Input: Composite Actor $CA$, Model $M$
Output: CPOG $G = \langle V, E, X, \rho, \phi \rangle$ for $CA$
Initialize $G = \langle \{\}, \{\}, \{\}, \{\}, \{\} \rangle$;

Let $A_{NC}$ & $A_C$ be partition of actors in $CA$ into sets of Primitive(Non-composite) and Composite actors resp. (present immediately under $CA$);
Let $I_{CA} = \{p_1, p_2, .., p_n\}$ be the inports of $CA$;
Let $O_{CA} = \{p_1, p_2, .., p_m\}$ be the outports of $CA$;

foreach composite actor $a \in A_C$ do
    //recursive call, ∪ represents composition of CPOGs
    $G \leftarrow G \cup composite\_cpog(a, M)$;
end
foreach primitive actor $a \in A_{NC}$ do
    //∪ represents composition of CPOGs
    $G \leftarrow G \cup primitive\_cpog(a)$;
end
foreach $p_i \in I_{CA} \cup O_{CA}$ do
    Let $ch_{in}$ be the in-coming channel connected to $p_i$;
    Let $p_{e_{in}}$ be source port of the channel $ch_{in}$;

    foreach out-going channel $ch_{out}$ from $p_i$ do
        Let $p_{e_{out}}$ be destination port of channel $ch_{out}$;
        Let $e_{new} = createEdge(p_{e_{in}}, p_{e_{out}})$;
        $E \leftarrow E \cup \{e_{new}\}$;
        $\phi(e_{new})$ = Constraints on $ch_{in}$ && Constraints on $ch_{out}$;
    end
end
return $G$;
```

Example MRICDF model

- Sample MRICDF model & its SIGNAL code
- ADD, Comparator, GAIN & $\frac{1}{GAIN}$ are predefined function actors



```
function process = (?int i1, i2, sel; !int out;)
(| sig1 = GAIN(in1)
 | sig3 = 1/GAIN(in1)
 | sig2 = ADD(sig1, in2)
 | sig4 = ADD(sig2, in2)
 | sig5 = (sel >= 0)
 | sig6 = sig2 when sig5
 | sig7 = sig2 when not sig5
 | out  = sig6 default sig7
 |)
where
  integer sig1,sig2,sig3,sig4,sig5,sig6,sig7;
end;
```

CPOG for the Example MRICDF model

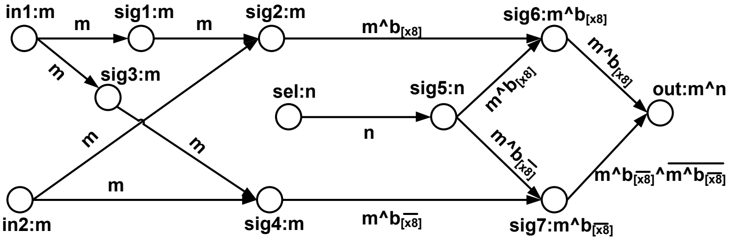


Figure: CPOG for the MRICDF Model

Formal representation CPOG for the Example MRICDF model

Quintuple Element	Set Elements
V	$\{in1, in2, sel, out, sig1, sig2, sig3, sig4, sig5, sig6, sig7\}$
E	$\{in1 \rightarrow sig1,\ in1 \rightarrow sig3,\ in2 \rightarrow sig2,\ in2 \rightarrow sig4,$ $sig3 \rightarrow sig4,\ sig1 \rightarrow sig2,\ sig2 \rightarrow sig6,\ sig4 \rightarrow sig7,$ $sel \rightarrow sig5,\ sig5 \rightarrow sig6,\ sig5 \rightarrow sig7,\ sig6 \rightarrow out,\ sig7 \rightarrow out\}$
X	$\{b_{x1}, b_{x2}, b_{x3}, b_{x4}, b_{x5}, b_{x6},\ b_{x7}, b_{x8}, b_{[x8]}, b_{[\overline{x8}]}, b_{x9},\ b_{x10}, b_{x11}\}$
ρ	$\{b_{x1} = b_{x2} = b_{x3} = b_{x4} = b_{x5} = b_{x6} = \mathbf{m},\ b_{x7} = b_{x8} = \mathbf{n},$ $b_{x8} = b_{[x8]} \vee b_{[\overline{x8}]},\ false = b_{[x8]} \wedge b_{[\overline{x8}]},\ b_{x9} = b_{x5} \wedge b_{[x8]},$ $b_{x10} = b_{x6} \wedge b_{[\overline{x8}]},\ b_{x11} = b_{x9} \vee b_{x10}\}$
ϕ	$\{\phi(in1) = b_{x1},\ \phi(in2) = b_{x2},\ \phi(sig1) = b_{x3},\ \phi(sig2) = b_{x5},$ $\phi(sig3) = b_{x4},\ \phi(sig4) = b_{x6},\ \phi(sel) = b_{x7},\ \phi(sig5) = b_{x8},$ $\phi(sig6) = b_{x9},\ \phi(sig7) = b_{x10},\ \phi(out) = b_{x11},\ \phi(in1 \rightarrow sig1) = b_{x1},$ $\phi(in1 \rightarrow sig3) = b_{x1},\ \phi(in2 \rightarrow sig2) = b_{x2},\ \phi(in2 \rightarrow sig4) = b_{x2},$ $\phi(sig1 \rightarrow sig2) = b_{x3},\ \phi(sig3 \rightarrow sig4) = b_{x4},$ $\phi(sig2 \rightarrow sig6) = b_{x5} \wedge b_{[x8]},\ \phi(sig4 \rightarrow sig7) = b_{x6} \wedge b_{[\overline{x8}]},$ $\phi(sel \rightarrow sig5) = b_{x7},\ \phi(sig5 \rightarrow sig6) = b_{x5} \wedge b_{[x8]},$ $\phi(sig5 \rightarrow sig7) = b_{x6} \wedge b_{[\overline{x8}]},$ $\phi(sig6 \rightarrow out) = b_{x9},\ \phi(sig7 \rightarrow out) = b_{x10} \wedge \overline{b_{x9}}\}$

Outline of the talk

- Initial CPOG needs to be simplified before transformations are applied
- Aim is to reduce the number of variables in set X
- Use the equivalence relations in set ρ
- Algorithm 2 lists the simplification step

Algorithm 2: *simplify*(G): Simplify CPOG

```
Input: Un-simplified CPOG $G=\langle V, E, X, \rho, \phi \rangle$
Output: Simplified CPOG $G=\langle V, E, X, \rho, \phi \rangle$
Let $\mathcal{E} = \{$Set of all Boolean equalities among single literals in $\rho\}$;
Let $\langle b_{x1}, b_{x2}, \ldots., b_{xn} \rangle$ represent the vector $X$;
foreach $b_{xi} \in V$ do
    if $(b_{xi} = b_{xj}) \in \mathcal{E}$ then
        replace all occurrences of $b_{xj}$ in $\rho$ and $\phi$ and simplify with idempotence and other Boolean
        simplification laws to obtain new $\rho$, and new $\phi$.
        $X = X - \{b_{xj}\}$;
    end
end
```

Proposition

Algorithm 2 converges and reduces the number of control states of the resulting system

Proof: Convergence is based on number of equivalence classes of control variables in X, and its reduction in each step

- Number of control states can be reduced further by proving more Boolean equivalences using powerful solvers like SMT solver
- Another way to reduce control states is by eliminating equivalent behaviors

Proposition

Algorithm 2 converges and reduces the number of control states of the resulting system

Proof: Convergence is based on number of equivalence classes of control variables in X, and its reduction in each step

- Number of control states can be reduced further by proving more Boolean equivalences using powerful solvers like SMT solver
- Another way to reduce control states is by eliminating equivalent behaviors
- X is simplified

Proposition

Algorithm 2 converges and reduces the number of control states of the resulting system

Proof: Convergence is based on number of equivalence classes of control variables in X, and its reduction in each step

- Number of control states can be reduced further by proving more Boolean equivalences using powerful solvers like SMT solver
- Another way to reduce control states is by eliminating equivalent behaviors
- X is simplified

Set of assignments for variables in X that results in feasible behaviors : 1101, 1110

Propagate feasible behavior assignments onto CPOGs to get feasible CPOGs

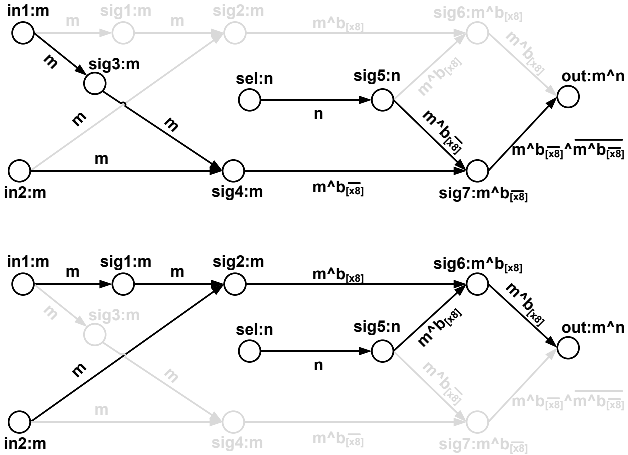
- Nodes and edges with value 0 are eliminated
- Node is excluded, if all the incoming edges to a node are excluded
- Node is excluded, if all the outgoing edges of a node are excluded
- All edges originating from an excluded node are also excluded
- All edges terminating on an excluded node are also excluded
- All other nodes and edges are left as such

Algorithm 3 provides the set of feasible CPOGs

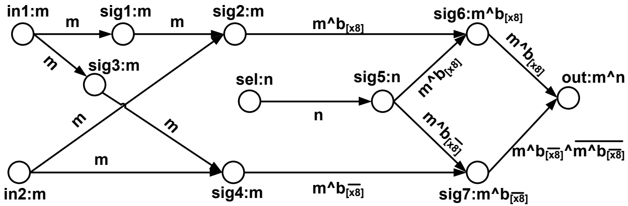
Algorithm 3: *getFeasibleCPOGs*$(G, \mathcal{F})$

```
Input: Simplified CPOG $G=\langle V, E, X, \rho, \phi\rangle$, Feasible behavior assignments for $X$ as $\mathcal{F}=\{\langle f_1\rangle,..., \langle f_k\rangle\}$ //Ex:
    $\mathcal{F}$={<1101>,<1110>}
Output: Set of CPOGs $\mathscr{V}=\{G_1, G_2, \ldots, G_k\}$
Let $\mathscr{V} = \{\}$;
foreach feasible behavior $f_i \in \mathcal{F}$ do
    Let $G_i$ be an instance of $G$;
    foreach node or edge $z \in G_i$ do
        //Evaluate $\phi(z)$ based on $f_i$ value
        if $\phi(z)|_{f_i} == 0$ then
            $G_i = G_i - \{z\}$; //Remove $z$ from CPOG
            //Remove unused edges
            if $z$ is node then
                Remove all incoming edges to $z$ and Remove all outgoing edges from $z$;
            end
        end
    end
    //Remove isolated nodes
    foreach remaining node $z \in G_i$ do
        Let $I_z$ be the set of incoming edges to $z$ and let $O_z$ be the set of outgoing edges from $z$;
        if $I_z == \{\}$ or $O_z == \{\}$ then
            $G_i = G_i - \{z\}$; //Remove node $z$ from CPOG
        end
    end
    $\mathscr{V} \leftarrow \mathscr{V} \cup G_i$; //Add $G_i$ to set $\mathscr{V}$
end
return $\mathscr{V}$;
```

Feasible CPOGs with Boolean vector 1101 and 1110

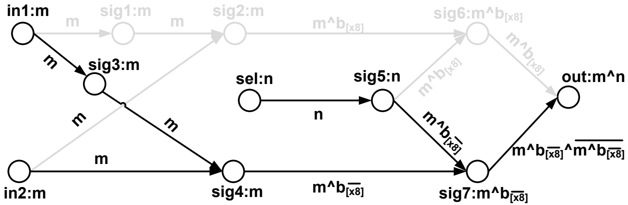


Resources needed



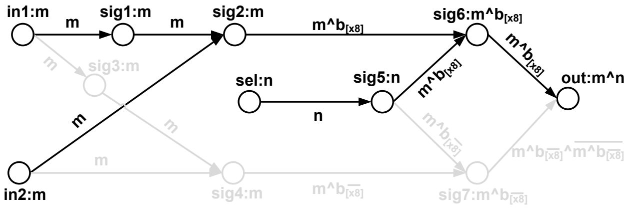
- CPOG has 11 nodes
- Assuming each node requires a computation resource, we need 11 computation resources

Resources needed



- Feasible CPOG with $X = 1101$ has 8 nodes
- We only require 8 computation resources

Resources needed



- Feasible CPOG with $X = 1110$ has 8 nodes
- We only require 8 computation resources

The assignments $X = 1101, 1110$ can be used as opcodes and one can measure latency, power consumption etc in each mode.

- Propagate feasible behaviors assignment values to the CPOG
    - CPOG remains still *connected* and *rooted*
    - No causal loops
- Then the CPOG is sequentially implementable
- Algorithm 4 checks the implementability

Algorithm 4: *isImplementable*(G)

```
Input: Simplified CPOG $G=\langle V, E, X, \rho, \phi\rangle$, Feasible behavior assignments for $X$ as $\mathcal{F}=\{\langle f_1\rangle,..,\langle f_k\rangle\}$
Output: True if implementable, else false
Let $\mathscr{V} = getFeasibleCPOGs(G, \mathcal{F})$;
foreach CPOG $G_i \in \mathscr{V}$ do
    if $G_i$ has causal loops OR $G_i$ is not weakly connected then
        return false;
    end
end
return True;
```

Outline of the talk

1. Motivation
2. Introduction to CPOGs
3. Introduction to MRICDF
4. MRICDF Models to CPOGs
5. Analysis and ASIP Synthesis
6. Conclusion and Future

Conclusion and Future Work

Conclusion

- Proposed a new compilation scheme for SIGNAL/MRICDF polychronous specifications based on CPOGs
- Provided algorithms to derive CPOGs from SIGNAL/MRICDF specifications

Future Work

- Explore the aspect of sequential and concurrent implementability by applying transformations on the CPOGs
- Formal proofs

Further Reading for CPOGs and ASIPs

- Mokhov, A., Sokolov, D., Rykunov, M., Yakovlev, A.
  Formal modelling and transformations of processor instruction sets – MEMOCODE 2011
- Mokhov, A., Yakovlev, A.
  Conditional Partial Order Graphs: Model, Synthesis and Application – IEEE Transactions on Computers 2010
- Kountouris, A.A., Wolinski, C.
  Hierarchical conditional dependency graphs as a unifying design representation in the CODESIS high-level synthesis system – ISSS 2000
- Mokhov, A.,Iliasov, A.,Sokolov, D.,Rykunov, M.,Yakovlev, A.,Romanovsky, A.
  Synthesis of Processor Instruction Sets from High-Level ISA Specifications – IEEE Transactions on Computers 2013
- Singh, S.
  Hardware/Software Synthesis and Verification Using Esterel – CPA 2007
- Mathworks Inc.
  HDL Coder: Generate Verilog and VHDL code for FPGA and ASIC Designs

Further Reading for MRICDF and Polychrony

- Paul Le Guernic, Jean-Pierre Talpin, Jean-Christophe Le Lann
  Polychrony for system design – Journal for Circuits, Systems and Computers 2003
- Bijoy A. Jose, Sandeep K. Shukla
  An alternative polychronous model and synthesis methodology for model-driven embedded software – ASP-DAC 2010
- M Nanjundappa, M Kracht, J Ouy and SK Shukla
  A novel technique for correct-by-construction concurrent code synthesis from polychronous specifications – ACSD 2013
- Bijoy A. Jose, Jason Pribble, Sandeep K. Shukla
  Faster Software Synthesis Using Actor Elimination Techniques for Polychronous Formalism – ACSD 2010
- J. Brandt, M. Gemunde, K. Schneider, S. Shukla, and J.-P. Talpin.

  Embedding polychrony into synchrony – In IEEE Transactions on Software Engineering, 2012.

Any Questions??

Thank You!!