Transformational Design Basics

1. Introduction
2. Unified design representations
3. ETPN
4. The ETPN transformation process.
5. Basic ETPN transformations
Transformational Approach to HLS

- Optimization heuristics
- Correctness by construction
- Integration of several synthesis tasks
- Several criteria can be considered simultaneously:
  - performance/cost trade-offs
  - power consumption and testability
  - geometry information
  - pre-specified partial structure (design re-use)
Unified Design Representation

• Used to capture the intermediate results of the transformational process.

• Data flow and control flow information must be *explicitly* represented.

• **Expressiveness**: It should represent both the structural and behavioral aspects of a design.

• **Concurrency**: It must be able to deal naturally with concurrency of computations.

• **Modularity**: It should support decomposition of systems in a clear and well-defined way.

• **Interface**: It should represent a design to the designers in a convenient way, best by means of graphics.

• **Formalness**: It should have a precisely defined semantics so that an equivalent relation between different designs can be proved.
Extended Timed Petri Nets (ETPN)

Control Part

- Control Signals
- Conditional Signals

Data Path

Scheduling

Timed Petri net

Microprogram

Allocation and Binding

Directed graph

RTL net-list
The Basic Petri Net Model

PN = (P, T, A, M)

P = \{p_1, p_2, ..., p_n\}, a set of places;
T = \{t_1, t_2, ..., t_m\}, a set of transitions;
A \subseteq (P \times T) \cup (T \times P), a set of input and output arcs;
M = \{m_1, m_2, ..., m_n\}, the initial marking.

Execution rule: an enabled transition can be fired at any time to generate a new marking.

Ex.

- parallel start
- sequential
- synchronization
- non-deterministic choice
Partial Ordering in ETPN

Floating point multiplication example

\[ 8.4 \times 10^2 \times 9.4 \times 10^3 = 78.96 \times 10^5 = 7.896 \times 10^6 \]

S1: addition of exponents
S2: multiplication of mantissa
S3: normalization of the results
Formal Definition of ETPN

A data path, \( D = (V, I, O, A, B) \):

\[ V = \{ V_1, V_2, ..., V_n \} \] is a finite set of vertices each of which represents a data manipulation or storage unit;

\[ I = I(V_1) \cup I(V_2) \cup ... \cup I(V_n) \] with \( I(V_i) \) = the set of input ports associated with vertex \( V_i \);

\[ O = O(V_1) \cup O(V_2) \cup ... \cup O(V_n) \] with \( O(V_j) \) = the set of output ports associated with vertex \( V_j \);

\[ A \subseteq O \times I = \{ <O, I> \mid O \in O, I \in I \} \] is a finite set of arcs each connecting an output port to an input port;

\[ B : O \rightarrow 2^{OP} \] is a mapping from output ports to sets of operations; \( OP = \{ OP_1, OP_2, ..., OP_m \} \) is a finite set of operations which is divided into the sequential subset \( SEQ \) and the combinatorial subset \( COM \).
Formal Definition of ETPN (Cont’d)

A data/control flow system, $\Gamma = (D, S, T, F, C, G, R, M_o)$:

$D = (V, I, O, A, B)$ is a data path;

$S = \{S_1, S_2, ..., S_n\}$ is a finite set of $S$-elements;

$T = \{T_1, T_2, ..., T_m\}$ is a finite set of $T$-elements;

$F \subseteq (S \times T) \cup (T \times S)$ is a binary relation, called the control flow relation;

$C : S \rightarrow 2^A$ is a mapping from control places to sets of arcs of the given data path; an arc $A_i$ is controlled by a control place $S_j$ if $A_i \in C(S_j)$;

$G : O \rightarrow 2^T$ is a mapping from output ports of data path vertices to sets of transitions; a transition $T_i$ is guarded by an output port $O_j$ if $T_i \in G(O_j)$;

$R : S \rightarrow 2^{(O \times OP)}$ is a mapping from control places to sets of pairs consisting of an output port and an operation; an output port/operation pair $<O_i, OP_j>$ is selected by $S_k$ if $<O_i, OP_j> \in R(S_k)$;

$M_o : S \rightarrow \{0, 1\}$ is an initial marking function.
An ETPN Example

**Control Part**

**Data Path**


BEGIN
  y := 0;
  x := 1;
  WHILE x > 0 DO
    BEGIN
      Read(x);
      y := y + x;
    END;
    Write(y);
  END;
END;
**Execution Rules**

1) A marking is an assignment of *tokens* to the places. Initially there is a token in each of the initial places, i.e., place $S$, such that $M_o(S) = 1$.

2) A transition $T$ is *enabled* at a marking $M$ if and only if for every $S$ such that $(S, T) \in F$, we have $M(S) \geq 1$.

3) A transition $T$ may be *fired* when it is enabled and the guarding condition is true.

4) Firing an enabled transition $T$ removes a token from each of its input places and deposits a token in each of its output places, resulting in a new marking.

5) $\nu(P)$ denotes the data value present at input or output port $P$. When a control place holds a token, its associated arcs in the data path will open for data to flow.

6) For each output port $O$ of a vertex $V$, $\nu(O) = OP(\nu(I(V)))$, where $OP \in B(O)$, $<O, OP>$ is selected and $OP \in COM$. If $OP \in SEQ$, it is assumed that $O$ has memory capability and $\nu(O)$ remains the same until it is changed.

7) If none of the connected arcs of an input port $I$ is active, $\nu(I)$ is *undefined*. 
Compiling VHDL to ETPN

- VHDL specification
- syntactic & semantic analysis
- program graph
- transformation & optimization
- data/control flow description
- parallelization
- parallelized data/control flow description
- ETPN generation
  - data path
  - control
An Example

... 

S1: A:=1;  
S2: X:=1;  
S3: while 100>=X loop  
S4: V:=V*2;  
S5: X:=X+1;  
end loop;  
S6: R:=V+A;  
...
IF Statements

••• if C then
    sequence_1
else
    sequence_2
end if;

••• if C1 then
    sequence_1
elsif C2 then
    sequence_2
else
    sequence_3
end if;

controls evaluation of condition C

controls evaluation of condition C1

controls evaluation of condition C2

statement sequence following the if statement

statement sequence following the if statement
ETPN to RTL Mapping

- Each data path node is implemented by a RTL component from a module library.
- The control part is mapped into a FSM notation, which can then be implemented as microprogram, PLA, or random logic.

FSM Generation (ASAP Scheduling)

\[ M_0 : S_0 \text{ exit TRUE } \rightarrow M_1; \]
\[ M_1 : S_1,S_2,S_3 \text{ exit TRUE } \rightarrow M_2; \]
\[ M_2 : S_4,S_5,S_6 \text{ exit TRUE } \rightarrow M_3; \]
\[ M_3 : S_4,S_7,S_8 \text{ exit C}_1 \rightarrow M_4; \]
\[ \text{exit NOT C}_1 \rightarrow M_5; \]
\[ M_4 : S_4,S_7,S_9 \text{ exit TRUE } \rightarrow M_5; \]
\[ M_5 : S_4,S_7,S_{10} \text{ exit TRUE } \rightarrow M_6; \]
\[ M_6 : S_{11} \text{ exit C}_2 \rightarrow M_7; \]
\[ \text{exit NOT C}_2 \rightarrow M_8; \]
\[ M_7 : S_{12},S_{13} \text{ exit TRUE } \rightarrow M_8; \]
\[ M_8 : S_{14} \text{ exit}; \]
ETPN Transformations

- Compiler oriented transformations
- Operation scheduling oriented transformations
- Data path oriented transformations
- Control oriented transformations
- Advanced transformations (e.g., pipelining)
Compiler Oriented Transformations

- Algebraic transformations — make use of basic algebraic laws, such as associativity, commutativity and distribution.

- Common subexpression elimination
- Dead code elimination
- Constant and variable propagation
- Loop unrolling
- ...

Tree-height reduction transformation example.
Operation Scheduling Oriented Transformations

- Determination of the serial/parallel nature of the design.

- Division or grouping of operations into time steps.

- Change of the order of operations.
Operation Scheduling Oriented Transformations (Cont’d)

- Rescheduling transformation — introduction of dummy places into the Petri net to change the default scheduling of the operations
Data Path Oriented Transformations

- Vertex merger folds two combinatorial vertices into one
- Vertex splitter is an inverse transformation.

[Diagram of data path oriented transformations with nodes and edges labeled with variables and states (S1, S2, S3, S4)]
Control Oriented Transformations

- *Control merger* folds a set of Petri net places, that are strictly parallel (they all have the same input transition), into one place.
Control/Data Path Exchange

- Conditional statement transformation

\[
\text{if } \text{cond} \text{ then } a := b + c; \text{ else } a := d + e; \text{ end;}
\]
Summary

- An intermediate design representation is critical to a transformational approach.

- A formal notation of semantic equivalence must be defined in order to prove that the design transformations are semantics-preserving.

- Explicit representation of parallel computations is important.

- Timing information should be explicit to facilitate performance estimation.

- If the design representation captures both scheduling and allocation information, these tasks can be integrated in a single algorithm.

- Advanced optimization techniques are required for the selection of transformations.