Concurrent Embedded System Model of Computation

Xun Yang,
Computer Science Department,
UCLA
Requirements for Model of Computation

- **Hierarchy:** The hierarchy maybe behavioral or structural.
- **Concurrency:** The concurrency granularity of the computation may be at bit level, operation level or processor level. It can be expressed using the execution order of computation or the data flow.
- **Communication:** There are two kinds, message passing and shared memory. Remote procedure call is a hybrid model allowing to model both message and shared memory.
- **Synchronization:** There are two kinds of synchronization mode, synchronous mode and asynchronous mode.
Requirements for Model of Computation

- Timing aspects are very important, because embedded system have some strict requirements concerning the timing. Functional timing represents the consumed time for executing a behavior. Timing constraints specify a range of elapsed time for executing a special behavior.

- Non-determinism is useful if the designer does not want to specify some special aspects of the system leaving details to the implementation step.

- Environment characteristics: Environment should be specified by a set of properties, related to operational conditions such as timing specifications of incoming data (frequency, timing, waveform type).
Requirements for Model of Computation

- Exceptions may occur in embedded system, because it implements reactive systems communication with environment.
- State Transitions are required to model the internal state of an embedded system which is important for control dominated or reactive systems.
- Verifiability and formal analysis are key issues of analytic aspects.
- Model executability is another possibility beside formal analysis.
Model of Computation

- Formal models
  - Can be classified into two categories: descriptive and operational approaches, the former are mainly based on process algebra or different temporal logics. The latter is based on different kinds of automata.
  - Extension of process algebra,
  - Extension of temporal logic,
  - Extension of state level models,
  - Extension of Petri Nets.

- Flowgraph models
  - DFG
  - CDFG
Different Kinds of MOC

- System level and co-design models
- Sequencing graph model based
- Synchronous language based
- Concurrent state machine based
- Task graph based
- Concurrent communicating process based
A multi-thread graph $M$ is defined as a 11-tuple $(O, E, V, D, \Theta, t, \Lambda, \Gamma^{\text{lat}}, \Gamma^{\text{resp}}, \nabla^{i}, \nabla^{\text{av}})$.

- $O$ is the set of operation nodes, an operation has a type \{thread, hier-thread, or, event, thread, syncho, sema, source, sink\}.

- $E$ is the set of control edges, A control edge has the following attributes \{g_{i,j}, r_{p_{i,j}}, r_{c_{i,j}}, d_{i,j}\}, which are respectively the conditional guard, production and consumption rate, time weight.
Multi-Thread Graph Model

- $\mathcal{G} = \mathcal{G}_o \cup \mathcal{G}_h$ is the set of all (shared memory) data ports. $\mathcal{G}_o$ is the set of primitive data ports of operation node $o$, $\mathcal{G}_h$ is the set of hierarchical data ports on the board.
- $I$ is the set of all system I/O nodes.
- $V$ is the set of local shared memory variable nodes.
- $D$ is the set of all shared memory data edges.
- $\Lambda$ is a time function associating an execution latency with each thread.
Multi-Thread Graph Model

- $\Gamma_{\text{lat}} = \Gamma_{\text{lat}}^{\text{min}} \cup \Gamma_{\text{lat}}^{\text{max}}$, representing the minimum and maximum latency timing constraints.
- $\Gamma_{\text{resp}} = \Gamma_{\text{resp}}^{\text{min}} \cup \Gamma_{\text{resp}}^{\text{max}}$, representing the minimum and maximum response timing constraints.
- $\nabla_i = \nabla_{i,\text{min}} \cup \nabla_{i,\text{max}}$ the set of minimum and maximum instantaneous (operation rate constraints)
- $\nabla_i = \nabla_{i,\text{min}} \cup \nabla_{i,\text{max}}$ the set of average rate (operation) constraints.
Multi-Thread Graph Model

- Behavioral node
- A program thread $T_i \in O$ is a maximum set of connected operations with a deterministic execution latency. $cdfg(T_i)$ is the CDFG representation of this set of operations.

- Control flow node
- A source node has $\Lambda(o_{src})=[0,0]$. It has only an exit port, and indicates the starting point of an MTG.
A sink node has $\Lambda(o_{src})=[0,0]$. It has only an start port, and indicates the completion point of an MTG.

A OR node has $\Lambda(o_{src})=[0,0]$. It has a single entry and a single exit control port.

An event node is an operation node. It has an exit port. It acts as a token injection point by the environment.

An syncho node is an operation node. It has an entry port. It acts as a token injection point from the system to its environment.
Multi-Thread Graph Model

- A control edge $e_{i,j}$ between the operation nodes $o_i$ and $o_j$ is an edge such that:
  - $e_{i,j} = \text{exit}(o_i)$ and $e_{i,j} = \text{entry}(o_j)$
  - $t^s(o_i) \geq t^e(o_j)$


Multi-Thread Graph Model

- Operational semantics
- Firing of an enabled node $o_i \in O$ in marking $\mu$ at time $t^s$ results in the following phases with associated markings.
- Firing: at $t^s$ tokens are consumed from the input edges, resulting in

$$\mu_{i,j} = \begin{cases} 
\mu_{i,j} & \text{if } e_{i,j} \in E \setminus \text{pre}(o_i), \\
\mu_{i,j} - 1 & \text{if } e_{i,j} \in \text{pre}(o_i) \}
\end{cases}$$
Execution: this takes $\lambda(o_i)$ time, with $\delta(o_i) \in \lambda(o_i) \in \Delta(o_i)$, and does not result in a marking change. The marking is thus:

- $\mu_{i,j}$, $\forall t \in [t^s, t^s + \lambda(o_i)]$, during execution, the elapsed firing time of $o_i$, $\text{EFT}(o_i, t)$ is equal to $t - t^s$

Completion: at time $\lambda(o_i)$, tokens are produced on the output edges, resulting in:

- $\mu_{i,j} = \begin{cases} \mu_{i,j} & \text{if } e_{i,j} \in E \setminus \text{post}(o_i) \\ \mu_{i,j} - 1 & \text{if } e_{i,j} \in \text{post}(o_i) \end{cases}$
start_latency(..);
if c1 then
    ....
else
    ....
if c2 then
    ....
else
    ....
end if;
end if;
max_latency(..);
start_latency(..);
if c3 then
    ....
else
    ....
end if;
min_latency(..);
MTG Model Extended with Data Communication

- Intra-graph and inter-graph shared memory communication
MTG Model Extended with Timing

- Operation node start & end times
- Operation node execution rate
- Pipelined execution of an MTG
- Latency constraints
- Response time constraints
- Execution rate constraints
MTG Model Extended with Timing

- Minimum latency constraints
- Maximum latency constraints
MTG Model Extended with Hierarchy
FunState Model
FunState Model

Definition: The basic FunState component consists of a network $N$ and a finite state machine $M$. The network $N=(F,S,E)$ itself contains a set of storage units $s \in S$, a set of functions $f \in F$, and a set of directed edges $e \in E$, where $E \subseteq (FXS) \cup (SXF)$. 
FunState Model

- Storage units: Queues have FIFO behavior and unbounded length. $q#$ represents the number of tokens in the queue, $q$$1, q$$2, \ldots$ represent the value of the tokens. Registers are linear arrays of limited length $n$ of pairs of address and values.

- Functions: The function objects $f \in F$ are uniquely named and operate on tokens or values when firing. It can be in idle or run state.
FunState Model

- State machine control the activation of embedded components or functions. A synchronous/reactive model is used. Transactions are labeled with conditions and actions. Conditions are predicates on storage units. A transaction is enabled if the corresponding predicate is true. The action consists of a set of names of function objects. Events are sent to these functions when the transaction is taken.
FunState Model

- Operational semantics of the model
- 1) Initialization
- 2) Check for progress
- 3) Function object termination
- 4) State machine reaction
FunState Model

- Extension of the model with hierarchy
- States can be hierarchical, they can contain other automata.
- The network \( N=(F,S,C,I,O,E) \) of a hierarchical component contains a set of functions \( F \), a set of storage units \( S \), input ports \( I \), output ports \( O \), embedded components \( C \) with input and output ports \( I_C \) and \( O_C \) respectively, and directed edges \( E \subseteq ((F \cup O) \times (O \cup S)) \cup ((I \cup S) \times (F \cup I_C)) \).
FunState Model
FunState Model