An Alternative Polychronous Model and Synthesis Methodology for Model-Driven Embedded Software

Bijoy A. Jose
FERMAT Lab
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
e-mail: bijoy@vt.edu

Sandeep K. Shukla
FERMAT Lab
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
e-mail: shukla@vt.edu

Abstract— Multi-clocked synchronous (a.k.a. Polychronous) specification languages do not assume that execution proceeds by sampling inputs at predetermined global synchronization points. The software synthesized from such specifications are paced by arrival of certain inputs, or evaluation of certain internal variables. Here, we present an alternate polychronous model of computation termed Multi-rate Instantaneous Channel connected Data Flow (MRICDF) actor network model\(^1\). Sequential embedded software from MRICDF specifications can be synthesized using epoch analysis, a technique proposed to form a unique order of events without a reference time line. We show how to decide on the implementability of MRICDF specification and how additional epoch information can help in synthesizing deterministic sequential software. The semantics of an MRICDF is akin to that of SIGNAL, but is visual and easier to specify. Also, our prime implicate based epoch analysis technique avoids the complex clock-tree based analysis required in SIGNAL. We experimented with the usability of MRICDF formalism by creating EmCodeSyn, our visual specification and synthesis tool. Our attempt is to make polychronous specification based software synthesis more accessible to engineers, by proposing this alternative model with different semantic exposition and simpler analysis techniques.

I. INTRODUCTION

Synchronous programming languages such as Esterel [1], SIGNAL [2], LUSTRE [3] etc., have been successful in embedded reactive software specification and generation of sequential software code from them. With rigorous formal semantics, a stepwise correctness preserving refinement strategy from specification to implementation, works quite well for them. At the heart of these languages is the synchrony hypothesis that abstracts away the computation and communication time, and hence the term synchronous languages. Even though broadly classified as synchronous languages due to the underlying synchrony assumption, these languages often have very different underlying model of computation (MoC) [4]. Software synthesis and execution semantics depend on these MoCs. For example, Esterel, and LUSTRE programs have the notion of totally ordered global tick (or global clock), which provides a sequence of global and predetermined synchronization points for the reading of inputs and the corresponding reaction to start. Thus during the execution of the software synthesized from these languages, one has to provide some external means to indicate these ticks or synchronization points. The time between two consecutive ticks must also be equal or more than the worst case execution time (WCET) of the computation embodied in a reaction. SIGNAL is a data flow synchronous programming language with the distinction that, it does not assume the existence of any external trigger for reacting to the inputs. It is paced by the rate at which data arrives or some other internal events occur. The freedom from the notion of a global synchronization or ticks leads to a multi-rate system specification and hence this computational model is known as polychronous model [5].

A. Motivating cases for polychronous formalism

A few examples are presented first to explain why multi-rate specifications are effective as opposed to single rate or globally synchronized model of computation.

Example 1 (Mutually Exclusive Signals) Suppose, it is known that every time an event on a signal c arrives with value true, an event on a arrives. When an event on c arrives with a value of false, then only event b arrives. Events on a or b are always accompanied by an event on c.

In a global synchronization based model of computation, every time the global tick occurs, at least c must be sensed and hence the program needs to wake up repeatedly even when no event on c occurred, leading to less efficient implementation (e.g., the system cannot go to sleep mode). In a polychronous model of computation, each signal is operated (read, modified, written) based on its own clock, hence c is read only when a new event has arrived at its port. Thus absent events need not be sensed nor operated on, resulting in a lot of savings in computation.

Example 2 (Implementability) Consider a system that computes a function f on two synchronized input signals a and b. In order to check that the software implementation of f is correct, one can have multiple implementations for f, say P_1 and Q_1. The inputs taken from a and b are simultaneously passed to both these P_1 and Q_1, and outputs are compared. If the outputs are equal always, then P_1 and Q_1 are both implementing the same function f, otherwise not. (c.f. program checking [6]).
If we have a global synchronization based model of computation, by synchrony hypothesis, the outputs from both $P_f$ and $Q_f$ come out instantaneously and they are compared instantaneously. Now consider the case where the implementation of $P_f$ runs much faster than that of $Q_f$. If the specification was in a global tick based formalism, either the inter-tick interval must be $\max(\text{WCET}(P_f), \text{WCET}(Q_f))$, or $P_f$ must be suspended until the tick at which $Q_f$ completes. This requires a suspension code around $P_f$, watching if $Q_f$ has completed etc. In polychronous formalism, one has to just state that $P_f$ and $Q_f$ must synchronize. A ‘completion’ signal from $Q_f$ will be enough for this synchronization, and there is no need to check at predetermined intervals. This might remind the reader about the similarity to synchronous vs asynchronous design in hardware.

The above examples are to illustrate cases where global synchronization based specifications unnecessarily restrict some obvious optimization opportunities that can be exploited using a polychronous formalism. The relative popularity of Estrel and LUSTRE is due to the simplicity of its globally synchronized execution model. But multi-rate models are not restricted due to their independence from an external global clock and give more freedom for compilers to choose from different schedules for computation. These refinement choices for the designer makes polychronous specification more amenable to optimized code synthesis. Since complex semantic exposition of the SIGNAL language thwarted its widespread use in the industry, our attempt is to make polychronous specification based software synthesis more accessible to engineers, by proposing this alternative model with different semantic exposition and analysis techniques.

In this paper, we provide an alternative for multi-rate specification model, with analysis and synthesis techniques that are very simple to understand for designers. Our prime implicate based implementation model from polychronous specification alleviates the bane of complicated semantic theory of SIGNAL based on clock equations, without compromising the rigor. Specifications must be as abstract as possible, leaving more choice to implementors for optimization. This is the philosophy with which we have developed MRICDF as a more user friendly polychronous modeling formalism, a theory of implementability, and a visual environment to support such modeling and synthesis.

**Contributions of this Paper**

1. An alternative specification language and formalism for multi-rate reactive deterministic embedded software systems which we call MRICDF or Multi-Rate Instantaneous Channel-Connected Data Flow actor network model.

2. Sequential implementability of a polychronous model is analyzed and a static analysis procedure is proposed (Epoch Analysis) to realize MRICDF model into embedded software. Solutions to tackle problems like cyclic dependencies (deadlock), scheduling constraints for MRICDF models are explained.

3. A method for formulating implicit or derived triggering conditions for polychronous models is proposed. Proofs are provided for the existence of a master trigger for deterministic software implementation from a polychronous model. A modified procedure for utilizing exogenous constraints in the event of insufficient information within an MRICDF model is also described.

**II. Preliminary Definitions**

In this section we review some of the structures required to explain the semantics and analysis of MRICDF models. Some of the concepts are based on synchronous structures [7]. We list a few of the preliminary definitions here:

**Definition 3 (Events)** An occurrence of a value on a signal is called an event. We denote with $\Xi$ the set of all events that can occur during the execution of an MRICDF model. If ‘$a$’ is a signal in the MRICDF network, the events on ‘$a$’ is given by $E(a)$.

Let ‘$\preceq$’ be a preorder on $\Xi$ such that for any two events $e, f \in \Xi$, we say $e \preceq f$ iff $e$ occurs before $f$, or if they occur together (synchronous). Let ‘∼’ be the equivalence relation induced by the preorder $\preceq$. Thus $e \sim f$ would mean $e$ does not occur before $f$, and $f$ does not occur before $e$. Formally, $\forall e, f \in \Xi, e \sim f$ iff $e \preceq f$ and $f \preceq e$. We also define a precedence relation $\prec$ on events such that $\forall e, f \in \Xi, e \prec f$ iff $e \preceq f$ and $f \not\preceq e$. Precedence could be due to data dependence between signals. Data dependence is a binary relation on events, and will be represented in this paper using ‘$\rightarrow$’.

**Definition 4 (Instant)** Note that $\sim$ is an equivalence relation on $\Xi$. Hence one can take a quotient of $\Xi$ with respect to $\sim$. This quotient will be sets of events that belong to an Instant, denoted by $\Upsilon = \Xi/\sim$. Each set $S \in \Upsilon$ will contain events which have the property $\forall e, f \in S, e \sim f$.

Thus events belonging to the same instant are grouped together. One can then naturally extend the $\prec$ relation on $\Upsilon$. We can say for any two sets $S, T \in \Upsilon$, $S \prec T$ if and only if for all events $e \in S$, and $f \in T$, $e \prec f$. This provides us with an abstraction of the notion of ‘ordered set of events’ without any reference time line.

**Definition 5 (Epoch)** Given a signal ‘a’, $I(a)$ or $\hat{a}$ is called its Epoch (or rate). It is a possibly infinite set of ordered instants at which the signal has events. Let $I(a) \subseteq \Upsilon$ denote the set of instants of ‘a’ or its epoch. Formally $I(a) = \{I|e \in \Upsilon \land \exists e \in E(a) \exists e \in \Upsilon\}$.

**Definition 6 (Synchronous Signals)** Given two signals ‘a’ and ‘b’, we say they are synchronous signals if and only if $I(a) = I(b)$ or $\hat{a} = \hat{b}$. In other words, the signals have unique events that are part of the same equivalence class $S \in \Upsilon$.

Now consider two events $e$ and $f$ with $e \preceq f$. There could be two reasons why $e$ might be prerequisite of $f$: (i) there is a data dependence between them, $f$ results from a computation of an actor with $e$ as input, (ii) they belong to two distinct instants such that $e \in S, f \in T$ and $S, T \in \Upsilon$ with $S \prec T$. The reason (i) has to be distinguished because during code synthesis, one must make sure event $e$ occurs before $f$. 

...
III. MRICDF: AN ALTERNATIVE POLYCHRONOUS MODEL FOR CODE SYNTHESIS

Multi-Rate Instantaneous Channel-connected Data Flow or MRICDF is a data flow network model consisting of several actors communicating with each other via instantaneous channels. The new formalism focuses on visual representation of synchronous language as actors in a data flow network. An actor in MRICDF has an input interface comprising input signals, an output interface comprising of output signals and several possible internal signals. All actors compute their reaction to a trigger condition within an instant. The actors are not allowed to test the presence of an event on any signal, i.e. absence cannot be recognized as a value at the interface. The communication between actors using an instantaneous-channel is assumed to be completed within an instant. Determinism is required in the execution of an actor and synchronous behavior is guaranteed by the instantaneous-channel. Actors could be of two types. Primitive actors or hierarchically composed composite actors. The four types of primitive actors are shown in the Fig. 1 and their epoch relations are described below:

1. **Buffer actor** $B$: A Buffer is a node which has single input port and single output port. An event arriving into a buffer node is consumed and stored in the buffer node, and the event that was consumed before this particular event is emitted on the output port. However, for the very first input event, a pre-specified default value is emitted. If $i_1$ is the input port name of a Buffer primitive node $B$, and $o_1$ is its output port, then it is implied $\hat{i}_1 = \hat{o}_1$.

2. **Sampler actor** $S$: A Sampler node has two input ports ($i_1, i_2$), and one output port ($o_1$). The second input port must always have Boolean valued events. If, the first input has an event on it, and the second input has a ‘true’ valued Boolean event belonging to the same instant, then the first input event is emitted on the output port. If events appear on its first input port, and no event or a ‘false’ Boolean input appears on its second input port at that instant, then no output is emitted. The epoch relation of Sampler actor is $\hat{o}_1 = \hat{i}_1 \cap \hat{i}_2$. $\hat{x}$ denotes the set of instants at which the port $x$ has an event occurrence which contains the Boolean ‘true’ value. So for a Boolean variable/signal one can write $\hat{x} = [x] \cup [\neg x]$. Here $[\neg x]$ represents an event occurrence with ‘false’ value.

3. **Prioritized Merge actor** $M$: A Merge node also has two input ports and one output port. If an input event appears on the first input port, then it is emitted on the output port irrespective of whether the second input port has any event appearing on it. However, in the absence of an event at the first input port, if the second input port has an event on it, then that event is emitted on the output port. If $i_1$ and $i_2$ are the input port names for a prioritized merge node, and $o_1$ is its output port name, then it is implied that $\hat{o}_1 = \hat{i}_1 \cup \hat{i}_2$.

4. **Function actor** $F(n, m)$: A Function node $F(n, m)$ has input arity $n$, output arity $m$ and computes function $F$ on $n$ events, one each arriving on each of its input ports, and produces $m$ output events, one each on each of its $m$ output ports. If the input port names of a Function $F(n, m)$ node are $i_j$ for $j = 1..n$, and output port names are $o_k$ for $k = 1..m$, then usage of the $F(n, m)$ node implies that the events on these signals occur at the same abstract instant. $\hat{i}_1 = \hat{i}_2 = .. = \hat{i}_n = \hat{o}_1 = \hat{o}_2 = .. = \hat{o}_m$.

Fig. 1.: MRICDF primitive operators

Fig. 2.: MRICDF network example

The primitives of MRICDF are very similar in structure to those in SIGNAL, but with a visual representation as actors in a data flow network. An example for an MRICDF network along with its textual representation is shown in Fig. 2. Here we see a network with all four primitive actors with input signals $a, b, c$ and output signal $g$. The epoch relations of the composite actor network1 is as follows:

\[
\hat{f} = \hat{b} \cap \hat{c}, \quad \hat{e} = \hat{a} = \hat{d}, \quad \hat{d} = \hat{g}, \quad \hat{g} = \hat{e} \cup \hat{f}
\]

IV. IMPLEMENTATION OF MRICDF MODELS AS EMBEDDED SOFTWARE

For implementing an MRICDF as embedded software, an abstract instant has to be mapped to real world time periods. Some of the issues that might hinder the mapping of MRICDF into executable software are discussed now. (i) If there are two events in the same instant such that there is cyclic data dependency between them directly or indirectly (for e.g. $e \rightarrow f$ and $f \rightarrow e$, while $e \sim f$), the equivalent embedded software can result in a causal cycle. (ii) Epoch Analysis technique, described later in the paper is used to analyze the MRICDF and obtain the right precedence order. It might indicate that there is no deterministic sequential implementation possible from the given MRICDF model. (iii) Mapping the abstract instant into real world time periods require a special signal which has instants in each equivalence class $\Upsilon$. This special signal called master trigger, needs to be found for a possible sequential implementation of MRICDF model. Cases (i) and (iii) will result in the
EmCodeSyn software tool rejecting the MRICDF design for software implementation, while case (ii) can be remedied by providing exogenous information regarding the input signals. The need for master trigger, proofs for identifying a signal as master trigger and examples for using implicit or exogenous epoch information to satisfy sequential implementability will be explained in the following subsections.

For sequential software synthesis, it is required that \( \prec \) is a total order on \( \Upsilon \). We want the software to execute in rounds, each round corresponding to one instant \( S_t \in \Upsilon \). Within each \( S_t \), there will be data dependence between events implied by \( \rightarrow \). To find this unique behavior, we define the condition for existence of master trigger for an MRICDF model.

**Definition 7 (Master Trigger)** Let \( M \) be an MRICDF model. Let \( t \) be a signal with \( E(t) \) as its set of events, with the property that for each \( S_t \in \Upsilon \), there exists an event \( e_S \in E(t) \), such that \( e_S \in S \), and there is no \( S \in \Upsilon \) which has a causal cycle with respect to the relation \( \rightarrow \). Then \( t \) will be termed the master trigger for \( M \) and \( M \) is said to be sequentially implementable.

Informally, sequential implementability requires that there should be at least one signal (\( t \)) in a cycle-free MRICDF model \( M \), having events in every equivalence class of instants (\( S_t \in \Upsilon \)) to form a master trigger. If such a \( t \) exists, since \( E(t) \) is totally ordered, it is implied that \( \Upsilon \) is totally ordered with respect to \( \prec \). This is a necessary, yet not a sufficient condition for implementability.

**A. Prime Implicate based master trigger identification**

For deterministic sequential software synthesis, mapping from the instances of rounds to software code can be performed. Signal epoch is be divided into two disjoint subsets which are Boolean signals. Recall that for a Boolean signal \( x, [x] \) denotes instances where \( x \) has an event and carries ‘true’ value and \( [\neg x] \) denotes instances where \( x \) has an event and carries ‘false’ value. In Boolean logic, \( b_x = b[x] \lor b[\neg x] \) and \( b[x] \land b[\neg x] = false \) since they are mutually exclusive. For the MRICDF epoch relations, equivalent system of Boolean equations can be formed. \( \{ \hat{z} = \hat{y}, \hat{x} = \hat{y} \cup \hat{z}, \hat{x} = \hat{y} \cap \hat{z} \} \) transforms to \( \{ b_x = b_y, b_{[x]} = b_{[y]} \lor b_{[z]}, b_{[z]} = b_{[y]} \land b_{[z]} \} \).

Now we claim that from such a system of Boolean equations we can show that there exists a variable \( b_x \) such that, if \( b_x = false \), then all variables in the equation system have to be \( false \) for the equations to be satisfied, then \( x \) is the coveted signal that we look for as master trigger. If no such \( b_x \) exists then there is no such signal, and hence we cannot sequentially implement the MRICDF specification.

**Theorem 8 (Existence of Master Trigger)** Given an MRICDF model \( M \), let \( B_M \) denote the system of Boolean equations obtained by the method described above. Then a signal \( x \) in \( M \) is a master trigger if and only if there exists an input signal \( x \) in the model \( M \) such that its corresponding Boolean variable \( b_x \) in \( B_M \) has the property that if \( b_x \) is false, every other variable is false.

**Proof sketch:** Recall that a master trigger is one which has one event in every instant of the set of abstract instants \( \Upsilon \). The instants are constructed by partitioning the union of events from all signals. Thus if \( x \) has to have an event in every instant, its instant set must be superset of all instants in the model. In other words, every signal’s instant set must be a subset of its instant set. By definition \( b_x \) should then be implied by every variable \( b_y \) for all signals \( y \) in the system. \( b_x \) encodes presence of an event of \( x \) in an arbitrary instant \( T \), and if \( b_y \Rightarrow b_x \), then if \( y \) has an event in \( T \), so does \( b_x \). So if the solution set of this equation system implies that for any signal \( y \) in \( M \), \( b_y \Rightarrow b_x \), then when we set \( b_x \) to false, all lefthand sides of those implications have to be set to false.

Theorem 8 is a precursor to our informal implementation method for identifying the master trigger. Let us consider \( F \) as a set of clauses in a Boolean equation system containing the signals \( \{ x, y, z, ... \} \). \( F \) is the logical AND of all equations, similar to the CNF form as in SAT solvers. The prime implicate of the system of equations, by definition, is a variable or a logical AND of variables that covers the whole max terms of a function. If there exists a single variable \( x \) as prime implicate to the system \( F \), \( b_x = false \) will imply all the SAT equations being false. Thus a single variable prime implicate to the system is the master trigger of the MRICDF model. By Shannon decomposition, \( F = x_{F_x=1} + not(x)_{F_x=0} \). The prime implications for \( F_{x=1} \) is found next using the same procedure. We are not interested in \( F_{x=0} \), since that will not trigger any other set of signals. The next set of prime implicates (literal or an equation) are called the immediate follower set. The repeated execution of this procedure will build us the next follower set which determines the order of computation of signals. We use the prime implicate generator provided in [8] in our EmCodeSyn tool. The example in Listing 1 will demonstrate the importance of the immediate follower set.

**B. Epoch Analysis**

Epoch analysis is the process by which we determine if an MRICDF specification is sequentially implementable. In other words, we determine if the set of instants in the semantic model is totally ordered. Two possible situations can happen in this process: (i) from implicit epoch information from the MRICDF model, the master trigger is identified (ii) Exogenous constraints are required from the user to form a master trigger.

**B.1 Implicitly derived epoch relations between Signals**

If a hierarchical MRICDF is flattened by unrolling each composite actor in terms of primitive actors, implicit epoch relations can be obtained from the epoch relations of individual primitive actors. Solving the system of boolean equations can help us identify the viability of the MRICDF model. For a Function or Buffer actor, any input signal can act as master trigger, because the epoch of all input and output signals are the same. When a complex network with Sampler or Merge actors is analyzed, master trigger signal may not be found from implicit epoch relations. Consider an MRICDF textual specification given below:

\[
|X := \text{Sampler}(Y,Z) |W := \text{Sampler}(U,V)
\]
Utilizing the rules for the epoch relations of primitive actors $S, M$ outlined earlier we obtain:

$$\hat{X} = \hat{Y} \cap [\hat{Z}], \quad \hat{W} = \hat{U} \cap [\hat{V}], \quad \hat{S} = \hat{X} \cup \hat{W}\$$

$X$ and $W$ are defined by the first two equations and the final equation merges them to form a faster signal $S$. It is easy to show that we cannot conclude the existence of a master trigger in this model by the following reasoning. Let $y \in Y$ and $z \in Z$ belong to an instant $T$ such that there is no signal $w \in W$ for the instant $T$. The same argument can be used for an event belonging to $U$ and $V$ in another instant $R$, but not belonging to $X$. So there is no assured common signal for $X$ and $W$ to form a master trigger and deliver a unique behavior and absence of master trigger results in a sequentially unimplementable MRICDF model.

### B.2 Exogenous constraints for code synthesis

In situations where sufficient information is not implicitly present in the network, exogenous constraints or signals have to be provided to the system as shown below using an example. Consider a simple MRICDF textually represented as follows:

$$z := \text{Merge}(a, b).$$

Since there is no information available on relating $a$ and $b$, no master trigger can be identified (refer Example 1 and case B.1). Exogenous constraints $\hat{x} = \hat{y}$; $\hat{a} = [x]$ and $\hat{b} = [y]$ are added to the system for creating a master trigger. The Boolean epoch relations are formed using the variables $b_a, b_b, b_x, b_y, b_z, b_{[x]}, b_{[y]}$ for an arbitrary instant $T$. The epoch relations are as follows: $b_a = b_{[x]}, b_b = b_{[y]}, b_x = b_y, b_x = b_{[x]} \lor b_{[y]}, b_y = b_{[y]} \lor b_{[-y]}$ and $b_z = b_a \lor b_b$. It is trivial that $x$ and $y$ form the master trigger condition since $b_x = b_y = 0$ would make every variable 0. Repeating prime implicate based technique, immediate follower set is formed for master trigger which will contain $a$ and $b$. Order of whether to read $a$ or $b$ first is not left ambiguous since they depend on values of $x, y$. The equivalent C code for this example will be as in Listing 1. This solves the priority-merge execution problem as described in Example 1.

#### Listing 1: C code for $z := \text{Merge}(a, b)$

```c
while (true)
    read(x);
    if (x == true)
        read(a);
        z = a;
    else if (y == true)
        read(b);
        z = b;
```

Let us reiterate. For a given MRICDF model if epoch analysis proves that (i) there exists a signal $T$ in the model such that for every instant $t \in T$, there is an event $t \in T$ such that $t \in I$ (ii) there is no cyclic data dependence within an instant, then the entire MRICDF can be statically scheduled into a sequential program whose behavior is equivalent to the behavior of the MRICDF model. In the polychrony literature, this property of an MRICDF model is called endochrony [9].

#### V. MRICDF Modeling Using EmCodeSyn

EmCodeSyn is a visual framework for code synthesis from MRICDF specifications. The goal of the EmCodeSyn project is to provide embedded system designers with a model driven software environment for code synthesis. Currently EmCodeSyn is capable of generating sequential C code from MRICDF specifications. The design methodology and information about the transforming an MRICDF network to C code can be found in [10]. Causal cycle detection, epoch analysis and schedulability tests are performed using EmCodeSyn. An external prime implicate generator [8] is used to assist in identifying the master trigger and the immediate follower set. The sample MRICDF models, intermediate files and synthesized C code can be found in the technical report [11].

MRICDF modeling in EmCodeSyn is demonstrated using a simple case study of a classic multi-process synchronization problem. A producer-consumer model with one stage buffer which requires exogenous constraints on its inputs is analyzed. Since the producer and consumer are two independent entities, the events on their signals are independent. Hence there is no implicit or derived master trigger in the system. Once additional constraints are provided, a master trigger can be formed and the model becomes implementable.

#### Listing 2: Producer-Consumer Model pseudo code

```c
Actor Producer (input boolean ptick; 
    output integer dvalue) =
    ( | counter := sampler((prevcount +1) mod 7, ptick) 
      | prevcount := Buffer(counter,0) )
Actor Consumer (input boolean ctick, u; 
    integer bvalue; 
    output integer v) =
    ( | v := PriorityMerge(Sample(true,u),false) 
      | v := PriorityMerge(Sample(bvalue,v),-1) )
Actor PMain (input boolean p, c; 
    output integer pcdata, boolean pcvalid) =
    ( | d := producer(p) 
      | b := PriorityMerge(c, Buffer(d,0)) 
      | pc := PriorityMerge(Buffer(p, false), false) 
      | (pcdata, pcvalid) = consumer(c,pc,b) )
```

![Fig. 3.: Hierarchical actor model for Producer Consumer in EmCodeSyn](image)

The hierarchical top level MRICDF design of the producer-consumer model is shown in Fig. 3 and its textual representation is given in Listing 2. The `Producer` uses a modulo
7 counter, and whenever \( p_{\text{tick}} \) input event occurs, it emits an event with the value of the counter. The composite actor \( \text{Cell} \) contains the one stage buffer for storing values written by \( \text{Producer} \). The \( \text{Consumer} \) reads from a buffer at \( e_{\text{tick}} \) which is independent from \( p_{\text{tick}} \) and hence no master trigger is possible. To resolve this problem, a new \( \text{Activate} \) actor is formed which synchronizes \( p_{\text{tick}} \) and \( e_{\text{tick}} \) in the form of the output signal \( p_{\text{c}} \). Listing 2 combines \( \text{Cell} \) and \( \text{Activate} \) actors into a \( \text{PCmain} \). Code synthesis is performed after identification of the master trigger and the immediate follower set. The design of all composite actors, the SAT files for implicate generation, the synthesized C code of the producer-consumer model and other examples are available in [11].

VI. RELATED WORK

The major synchronous programming languages like \textsc{Esterverl} [1], \textsc{LUSTRE} [3] and \textsc{SIGNAL} [2] and their distinct ways of reacting to events have been discussed in the introductory section. Among synchronous languages, SIGNAL is polychronous in nature and is the inspiration behind this work. SIGNAL [12] and its toolset Polychrony [5] provides a framework for multi-rate data flow specification and obviously the inspiration for our framework. Polychrony analyzes the system by solving a set of clock equations which only apply to Boolean signals, and cannot be easily extended to other data types [13]. This is due to the fact that they use ROBDDs for creating canonical form for all clock variables, and then construct clock hierarchy in the form of tree-like data structures. Tree like data structures formed in this technique is far more complex than the SAT based prime implicate technique of MRICDF. MRICDF is a more intuitive exposition for not only modeling but also for characterizing implementability of the specifications as software programs.

Model driven code generation environments like SIMULINK/Stateflow [14] provide visual frameworks with an underlying state machine based formalism for simulation and code synthesis. In academia, Ptolemy [15] is a significant contribution for actor based networks for code generation. But the specification in Java for Ptolemy makes verification of the model harder. Model of computation is chosen by a ‘director’ which specifies the rules for the design. This is not favorable for code optimization and task scheduling purposes. \textsc{EmCodeSyn} on the other hand is based on a concurrent, multi-rate, synchronous data flow formalism (MRICDF), fit for optimization purposes, which is essential for embedded software. \textsc{SHIM} [16] is an imperative approach to code synthesis with an asynchronous communication model. The idea is inspired by the KPN model, but the buffers in KPN model was replaced by a rendezvous mode of communication. Our MRICDF model is quite different in the sense that our communication model is fully instantaneous and synchronous, and the computation in each actor is synchronous.

VII. CONCLUSIONS

In this paper we have provided an alternate polychronous model of computation, MRICDF. This visual model is more intuitive than SIGNAL, the well known polychronous specification formalism for embedded software. Our formalism is based on a network of actor models which are connected by instantaneously communicating channels. Epoch analysis or the transformation of MRICDF actors into Boolean equations for checking the sequential implementability of MRICDF specification has been proposed. For a sequentially implementable MRICDF model, the existence of a ‘master trigger’ signal amongst signals that make up the system of epoch relations, is proved. A prime implicate based implementation technique for identification of master trigger and immediate follower set was demonstrated. The \textsc{EmCodeSyn} framework based on MRICDF formalism was introduced as a tool for synchronous software code synthesis. MRICDF-\textsc{EmCodeSyn} project aims to extend the synchronous formalism into multi-threaded real-time software domain.

REFERENCES


