Modelling Latency-Insensitive Systems in CSP
(extended abstract)

Hemangee K. Kapoor
Dhirubhai Ambani Institute of Information and Communication Technology,
Gandhinagar, India
hemangee.kapoor@daiict.ac.in

1 Introduction

With the advance in semiconductor technology we are able to pack more and more devices on a single chip [5]. However, the threat comes from the long interconnect wires [6] whose delays dominate in deep-submicron (DSM) CMOS. To handle the increased latency due the long interconnects, we require the IP cores to be latency-insensitive (LI). Design and validation of LI design is studied in [1, 2, 11]. Generalised latency-insensitive systems, design of connecting FIFOs and other communication protocols appear in [3, 4, 8, 10].

Process algebras provide a well-studied framework for modelling and verifying concurrent systems. In this work we try to address the problem of long interconnects by modelling the latency insensitive protocol in the discrete time version of CSP [7, 9]. Time is modelled in terms of events occurring at regular intervals, modelled by the event \( \text{tock} \).

2 Modelling a LI Computational Block

A process in CSP is associated with an input (\( \mathcal{A} \)) and output (\( \mathcal{B} \)) alphabet. Input of a value \( v \) on a channel \( c \) is denoted by \( c?v \), while an output is denoted by \( c!v \). Other language constructs include: the guarded choice \( [ x \rightarrow P \circ b \rightarrow Q ] \), interleaving \( P \parallel Q \), \( \text{skip} \), \( \text{stop} \) and \( \text{error} \). Conditional actions are given by \( P \downarrow \text{cond} \uparrow Q \), where \( P \) is performed if the condition \( \text{cond} \) holds, otherwise \( Q \) is executed. Passage of time is denoted by a special \( \text{tock} \) event.

A LI computational module is described by a process with input and output ports (Figure 1). This process being synchronous waits for the \( \text{tock} \) event before accepting inputs and producing outputs. However, the process may/may-not be in a position to produce output at each \( \text{tock} \) because all the necessary inputs may not be available.

Note that if the given process is unable to generate output at every \( \text{tock} \), the receiving module may not get the required inputs at every \( \text{tock} \). This leads to the necessity for a mechanism to inform the validity of data on the channels. We therefore divide each (input or output) port into three parts: (a) Channel carrying the data to the process: \( \text{a.dataIn} \) or \( \text{b.dataOut} \); (b) Channel carrying the information about data (in)validity: \( \text{a Void} \) or \( \text{b.Void} \); (c) Channel carrying the information that data is not consumed: \( \text{a.stall} \) or \( \text{b.stall} \).

2.1 Process algebraic model

A latency-insensitive block can be defined by a process (with its associated state) as given below. As a process represents a synchronous block, at the \( n^{th} \) \( \text{tock} \) it emits the outputs produced during the \((n-1)^{th}\) \( \text{tock} \) and consumes inputs to compute the output for the \((n+1)^{th}\) \( \text{tock} \).

\[
P_{(x,y,v,s,xnw,snw,ynw,snw)} = \text{tock} \rightarrow
\]
\[
(\text{STALL} \rightarrow (R01 \downarrow \exists j.s[j] = 1 \uparrow \text{R01}))
\]
\[
\downarrow \text{st} = 1 \uparrow
\]
\[
(\text{OUT} \rightarrow (\text{NXTSTALL} \downarrow \exists j.s[j] = 1 \uparrow \text{REPEAT}))
\]

\[
\text{STALL} = (||| j \in \mathcal{B} \ j.\text{void}!1 \rightarrow (||| i \in \mathcal{A} \ i.\text{stall}!1) \rightarrow \text{IN})
\]
\[
\text{OUT} = (||| j \in \mathcal{B} \ j.\text{dataOut}!y[j])
\]
\[
\downarrow \text{ynw} = 1 \uparrow
\]
\[
((||| j \in \mathcal{B} \ j.\text{void}!1 \rightarrow (||| i \in \mathcal{A} \neg xnw[i]=1 \ i.\text{stall}!1)) \rightarrow \text{IN}
\]
\[
\text{IN} = (||| i \in \mathcal{A} \ (\exists \text{void}!v[i]
\]
\[
\Square \text{dataIn}??x[i] \rightarrow \neg xnw[i] = 1))
\]
\[
\rightarrow (||| j \in \mathcal{B} \ (\exists j.\text{stall}!s[j] \text{skip}))
\]
\[
\text{NXTSTALL} = P_{11} \downarrow \forall i.xnw[i] = 1 \uparrow P_{10}
\]
\[
\text{REPEAT} = P_{01} \downarrow \forall i.xnw[i] = 1 \uparrow P_{00}
\]
\[
R01 = P_{(x,y,v,s,xnw,0,1)} \quad R11 = P_{(x,y,v,s,xnw,1,1)}
\]
\[
P_{00} = P_{(x,y,v,s,xnw,0,0)} \quad P_{01} = P_{(x,y',v,s,xnw',0,1)}
\]
\[
P_{10} = P_{(x,y,v,s,xnw,1,0)} \quad P_{11} = P_{(x,y',v,s,xnw',1,1)}
\]

Here \( y' = f(x) \) is the result of the computation performed by the block and \( \neg xnw = \forall i.xnw[i] = 0 \). The
A method was presented to model latency-insensitive modules in a process algebraic framework. It was shown that such modules can be described in the well established process calculus CSP. This makes it possible to use standard tool such as FDR (http://www.fsel.com/) (with modifications) for equivalence and refinement checking. Such a model paves a way for translating it to an implementation. Synchronisation of events during composition helps to understand the behaviour of a larger composition.

3 Conclusion

Some properties enjoyed by a latency-insensitive process are: There is a lock event between two consecutive transitions on the same channel; Stalls ripple through a pipeline of computational blocks; Data is not over-written.

References