

## **Partial Synchronization:**

In this Document we will try to explain the idea behind partial synchronization. For that I will explain the translation of a “simple” core program to LOTOS.

### The Core Code:

```
HOLD_PREVIOUS

VAR
globalVariable: boolean INITVAL TRUE;

MODULE M1 () {
  VAR
  temp: boolean INITVAL TRUE;

  TRANS A1:
    enable: temp ;
    assign: temp' := TRUE;

  TRANS B1:
    enable: temp ;
    assign: temp' := TRUE;

}

MODULE M2 () {
  VAR
  temp: boolean INITVAL TRUE;

  TRANS A2:
    enable: temp ;
    assign: temp' := TRUE;

  TRANS B2:
    enable: temp ;
    assign: temp' := TRUE;

}

MODULE SYSTEM () {
  (M1 () | (A1,A2) | M2 ())
}
```

## LOTOS Translation Code:

```
specification _SYSTEM :noexit
  hide [A1,A2,B1,B2,globals] in
    (SYSTEM [A1,A2,B1,B2,globals] || Global [globals] (true) )
where

process M1 [A1,B1,globals] (temp:Bool) :noexit :=
  // before execution receive the current value of globalVariable.
  globals !"send" ?globalVariable:Bool;
  // check if the pre-condition of the transition is satisfied,
  // if yes the we could proceed and choosing a random value of
  // temp_
  ( [ (temp) ]-> choice temp_:Bool []
  // now after we choose the random value, check if the value that
  // we choose satisfy the post-condition, if yes then output the
  // globalVariable to the "A1" gate
  [ (temp_= true) ]-> A1 !globalVariable;
  // also output the current value of the globalVariable on the
  // "globals" gate
  globals !"receive" !globalVariable;
  // now recursively call the process M1
  M1 [A1,B1,globals] (temp_)
  []
  [ (temp) ]-> choice temp_:Bool []
  [ (temp_= true) ]-> B1 !globalVariable;
  globals !"receive" !globalVariable;
  M1 [A1,B1,globals] (temp_) )
endproc

process M2 [A2,B2,globals] (temp:Bool) :noexit :=
  globals !"send" ?globalVariable:Bool;
  ( [ (temp) ]-> choice temp_:Bool []
  [ (temp_= true) ]-> A2 !globalVariable;
  globals !"receive" !globalVariable;
  M2 [A2,B2,globals] (temp_)
  []
  [ (temp) ]-> choice temp_:Bool []
  [ (temp_= true) ]-> B2 !globalVariable;
  globals !"receive" !globalVariable;
  M2 [A2,B2,globals] (temp_) )
endproc

process SYSTEM [A1,A2,B1,B2,globals] :noexit :=
  globals !"send" ?globalVariable:Bool;
  ( ( (M1 [A1,B1,globals] (true) |[ [A1,globals] ]| HelpProcess0
  [A1,A2,globals] ) ) |[ [A2] ]| M2 [A2,B2,globals] (true) )
endproc
```

```

process Global [globals] (globalVariable:Bool) :noexit :=
  // This process used to receive and send the current value of
  // globalVariable.
  globals !"send" !globalVariable;
  Global [globals] (globalVariable)
  []
  globals !"receive" ?globalVariable_:Bool;
  Global [globals] (globalVariable_)
endproc

process HelpProcess0 [A1,A2,globals] :noexit :=
  // this process used to support the partial synchronization.
  // we placed the |(A1,A2) | in the CORE with this process.
  (A1 !globalVariable;
   A2 !globalVariable;
   globals !globalVariable (*))
  []
  globals !globalVariable) (**))
endproc

endspec

```

### How this will work ?

We will explain the interaction of M1, M2 and HelpProcess0, any misunderstanding of the purpose of the global process plz. Refer to implementations design document.

There are three possibilities to occur:

- The process M1 execute transition B1 (The situation of executing B2 by process M2 identical), this mean that we output on gate “B1” and gate “globals” the value of globalVariable. Because process M1 isn’t synchronized with the HelpProcess0 on gate B1 there will be no problem, but when we execute the instruction that output the value of globalVariable on gate “globals” it should synchronized with HelpProcess0 on gate “globals”, but in this case the instructions that will be chooses is `globals !globalVariable) (**)` .[this explain the importance of the `(**)` ] => till now there is no problem.
- The process M1 execute the action on gate A1 => he should synchronized with HelpProcess0 on this action => the set of instruction that will be chooses in the HelpProcess0 are:

```

A1 !globalVariable;
A2 !globalVariable;
globals !globalVariable (*)

```

Because the next instruction of process A1 is an action on “globals” gate and because it is synchronized with Global process on this gate => the next instruction that the CPU will execute will the action on gate “A2” => on this way we are sure that the transition A1 and transition A2 occur at the same time.

[This explain why process A1 and HelpProcess0 are synchronized on gate “globals” and the need of `(*)` instruction].

- The process M2 execute the action on gate “A2” => the HelpProcess0 should also execute this action on gate “A2” => from the code of HelpProcess0 we already

execute an action on gate A1 => we are sure that transitions A1, A2 are occurred at the same time.