Agenda

- ➢ Introduction
- ➢ Implementation graph
- ➢ Translation of a graph
- ➢ Translation of nodes
- ➢ An example

Introduction

- Signal
- SynDEx

Introduction

- Objectives:
  - To give access to the functionalities of SynDEx to Signal designs, especially the possibility to get and to prototype distributed implementations obtained from quantitative criteria.
  - To allow the use of SynDEx on applications developed in formalisms for which it is of interest to have a Signal intermediate representation.
  - To provide Signal as a possible input formalism for SynDEx users.

Introduction

- Hierarchical Conditional Dependence Graph (HCDG)
  - A HCDG can be defined as a six-tuple:

    $< G, C, \Sigma, f_N, f_\Gamma, \delta_T >$

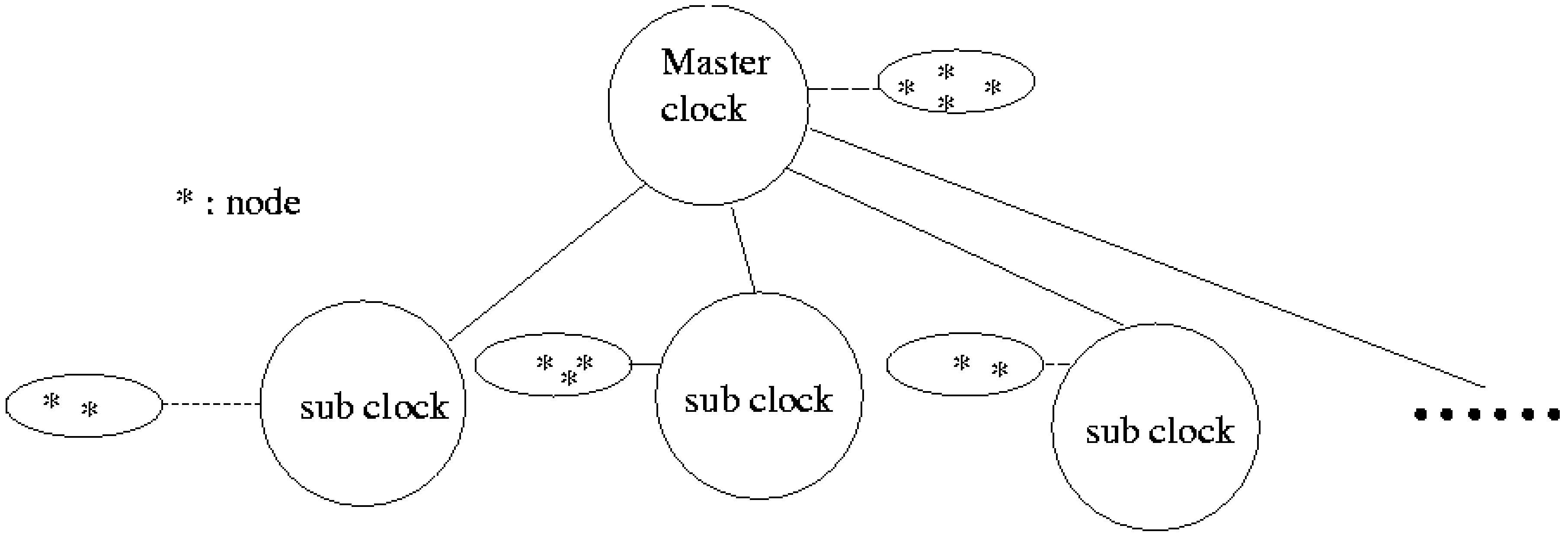
In one word, it's a hierarchical tree-like representation with each branch stands for a sub-clock, for each clock, there are several nodes attach to it, which means such expressions appear under this clock.

Introduction

- The different levels of representation of clock hierarchy
  - the DC+ level
    - clocks are represented as signals of event type
  - the bDC+ level
    - boolean DC+: the clocks are represented as Boolean signals
  - the STS level
    - Symbolic Transition System(flat bDC+): it has at most two levels

Introduction

- The STS (Symbolic Transition System) level



Implementation graph

- A graph P consists of the following elements:
  - its name
  - its signals
  - its clock hierarchy
  - its nodes
  - a function (associates node and its signal)
  - another function(associates clock and its nodes).

Implementation graph

- Different sorts of nodes:
  - Mathematic equation (eg: X := a+b)
  - Memorization (eg: a := b $2 init [3,4])
  - Cell (eg: X := Y cell A)
  - Default (eg: X := Y when c default Z)
  - Sub-graph (eg: e::(| Q |))
  - External call (eg: (s1,...,s_q) := Q(e1,...,e_p))
  - Process call (eg: (s1,...,s_q) := Q(e1,...,e_p))
  - Array of processes (eg: array i to n of Q end)

Translation of a graph

- Interface
  - Parameters
    - parameters of Signal are translated to parameters of SynDEx
  - Interface signals
    - For all read and write nodes: translate to sensor & actuator
    - For the interconnection of sub-graphs: translate to inputs and outputs

Translation of a graph

- Nodes
  - Constant
    - def constant ...
    - int/constant <2> cstint_1;
  - Equation
    - mathematic functions
    - others (eg: default, cell)
  - Memorization
    - delay, window
  - *External call: analogous,only that the body of algorithm is empty*
  - *Process call:analogous,with a new defined algorithm*
  - *Sub-graph: analogous to process call*
  - *Array of processes:later*

Translation of a graph

- Dependences
  - Data flow connections
    - In one subclock ,from the ith output of node p_k to the jth input of node p_l

      strong_precedence_data p_k_s.o_i -> p_l_t .i_j
      - eg: plus1.o - > minus1.i1

        (x := a+b; y := x-2)
  - Inputs and outputs
    - the ith output of subclock p_k is the jth input of subclock p_l

      strong_precedence_data p_k_s.o_i -> p_l_t .i_j
      - eg: C_A_1.o_i -> C_B_1. i_j

Translation of a graph

- Clock hierarchy
  - STS has at most two levels.
    - the most frequent clock (master clock)
    - the less frequent clock, the sons of the master clock (sub-clock)
  - Translate the master clock, translate its sons one by one.
    - For each clock, define one algorithm with the name of the clock
    - Translate the nodes one by one under this clock.
    - Set the algorithm of master clock to main algorithm

Translation of nodes (predefined)

- Predefined types and operations:
  - The basic static monochronous operators
  - Dynamic monochronous operators(delay,window)

- Define libraries for different types.
  - int.sdx
  - float.sdx
  - boolean.sdx
  - ...

Translation of nodes (predefined)

- Mathematic expressions:
  - Reference to the algorithm defined in the library
  - Add dependences to each operand
- Delay,cell
  - Define "memory/algorithm" and "constant" in libs
  - Add dependences to each operand (also initial value)
- Default
  - Define algorithm in libs
  - For every "default" operator, set reference and dependences to the predefined algorithm

Translation of nodes (predefined)

- Examples:
  - int zx , x , y;

    zx := x + y ;
  - in library int.sdx,there is:
    - def algorithm add :
      ? int i1;
      ? int i2;
      ! int o;
      description : "Integer add, o= i1 + i2"
  - The translated result is:

```
...
...
references :
        int/add add_1;
        ...
dependences :
strong_precedence_data  x ->add_1.i1;
strong_precedence_data  y ->add_1.i2;
strong_precedence_data  add_1.o -> zx;
...
...
```

Translation of nodes (predefined)

- Non elementary types.
  - enumerated types, structures, external types.
    - declare the operations as *algorithms*.
  - array
    - use array constructor provided by SynDEx.

(later)

An Example

- In Signal

```
process Merge1 =
    ( ? B;
        real A;
      ! X3, Y;
    )
   (| X3 := A default B | Y:= X3 + 4.0 |)
;
```

An Example

- In SynDEx

```
syndex_version : "6.4.1"

include "boolean.sdx";
include "float.sdx";

def sensor inputboolean :
! boolean  o;

def sensor inputfloat :
! float  o;

def algorithm CLK_B_0 :
? boolean  C_B;
! float  B;

conditions: C_B = 1;
references:
  inputfloat     inputfloat_1;
dependences:
  strong_precedence_data inputfloat_1.o -> B;

def algorithm CLK_A_0 :
? boolean  C_A;
! float  A;

conditions: C_A = 1;
references:
  inputfloat     inputfloat_2;
dependences:
  strong_precedence_data inputfloat_2.o -> A;

def actuator outputfloat :
? float  i;

def algorithm CLK_X3_0 :
? float  B;
? float  A;
? boolean  C_A;
? boolean  C_Y;
? boolean  C_37;

conditions: C_Y = 1;
references:
  float/default     default_1;
  float/Constant  <4;0>    Constant_1;
  float/add         add_1;
  outputfloat     outputfloat_1;
  outputfloat     outputfloat_2;
```

An Example

```
dependences:
  strong_precedence_data default_1.o ->
       outputfloat_1.i;
  strong_precedence_data add_1.o -> outputfloat_2.i;
  strong_precedence_data A -> default_1.i1;
  strong_precedence_data C_A -> default_1.i2;
  strong_precedence_data B -> default_1.i3;
  strong_precedence_data default_1.o -> add_1.i1;
  strong_precedence_data Constant_1.o -> add_1.i2;

def algorithm CLK_C_B :

conditions: true;
references:
  inputboolean     inputboolean_1;
  inputboolean     inputboolean_2;
  boolean/or     or_1;
  boolean/not     not_1;
  boolean/and     and_1;
  CLK_B_0     CLK_B_0_1;
  CLK_A_0     CLK_A_0_1;
  CLK_X3_0     CLK_X3_0_1;
dependences:
  strong_precedence_data inputboolean_1.o -> or_1.i1;
  strong_precedence_data inputboolean_2.o -> or_1.i2;
  strong_precedence_data inputboolean_2.o -> not_1.i1;
  strong_precedence_data not_1.o -> and_1.i1;
  strong_precedence_data inputboolean_1.o -> and_1.i2;
  strong_precedence_data inputboolean_1.o -> CLK_B_0_1.C_B;
  strong_precedence_data inputboolean_2.o -> CLK_A_0_1.C_A;
  strong_precedence_data CLK_B_0_1.B -> CLK_X3_0_1.B;
  strong_precedence_data CLK_A_0_1.A -> CLK_X3_0_1.A;
  strong_precedence_data inputboolean_2.o -> CLK_X3_0_1.C_A;
  strong_precedence_data or_1.o -> CLK_X3_0_1.C_Y;
  strong_precedence_data and_1.o -> CLK_X3_0_1.C_37;

#Architectures

#Main Algorithm /Main Architecture
 main algorithm CLK_C_B;

#Extra durations

#Software components

#Constraints
```

End

Thank you