March 2007 Meeting

S.K. Wood

iemn Recherche C.N.R.S.

University of Kent

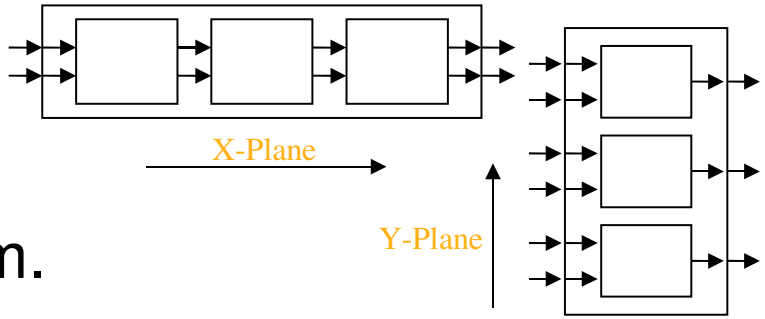
INRIA FUTURS

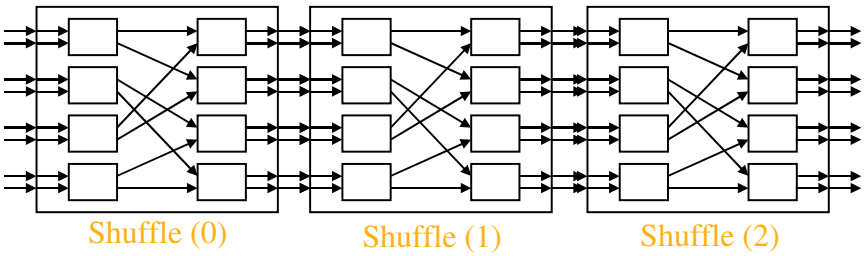
1. Introduction

- A summary of the work carried out on mapping repetitive structures to VHDL.
- What has happened - Following on from the last meeting:
  - Creating VHDL examples and generic templates
- What is happening
  - Work on non-trivial examples for publication
- What to do next ?

2. Past work

- ArrayOL to represent repetitive component structures –
- VHDL implementations carried out to identify required transformations:
  - X and Y-plane repetition of basic elementary tasks.
  - The perfect shuffle algorithm.
  - X-plane repetitions of the perfect shuffle.





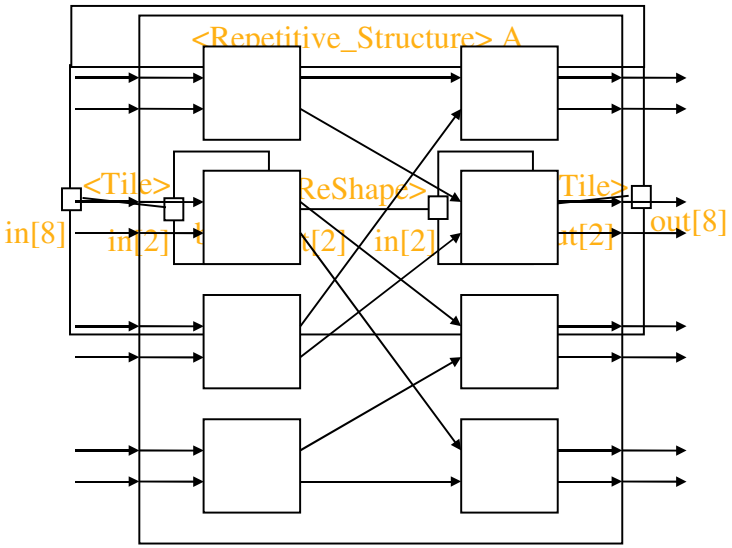
• Perfect shuffle

```
1.  entity shuffle_A is
2.  generic (Origin: Integer:= 0;ArraySize: Positive:= 8; P_Lim: Positive:= 4; F_Lim: Positive:= 2);
3.  Port(
4.          A_IN: in std_logic_vector (0 to ArraySize -1);
5.          A_OUT: out std_logic_vector(0 to ArraySize -1)
6.          );
7.  end shuffle_A;

9.  architecture Behav of shuffle_A is
10.         signal rehape: std_logic_vector (0 to ArraySize -1);  -- interconnects
11.         component B
12.             Port(
13.                 B_IN: in std_logic_vector(0 to 2-1);
14.                 B_OUT: out std_logic_vector(0 to 2-1)
15.                 );
16.         end component;
17. begin
18.         TILERS:for i in Origin to Origin + P_Lim -1 generate
19.         Source:B port map (B_IN(0)=> A_IN (i*F_Lim),B_IN(1)=> A_IN (i*F_Lim+1),B_OUT(0)=>rehape(i*F_Lim),B_OUT(1)=>rehape(i*F_Lim+1));
20.         Target:B port map (B_IN(0)=>rehape(i),B_IN(1)=>rehape(i+P_Lim),B_OUT(0)=> A_OUT (i*F_Lim),B_OUT(1)=> A_OUT (i*F_Lim+1));
21.         end generate TILERS;
22. end Behav;
```

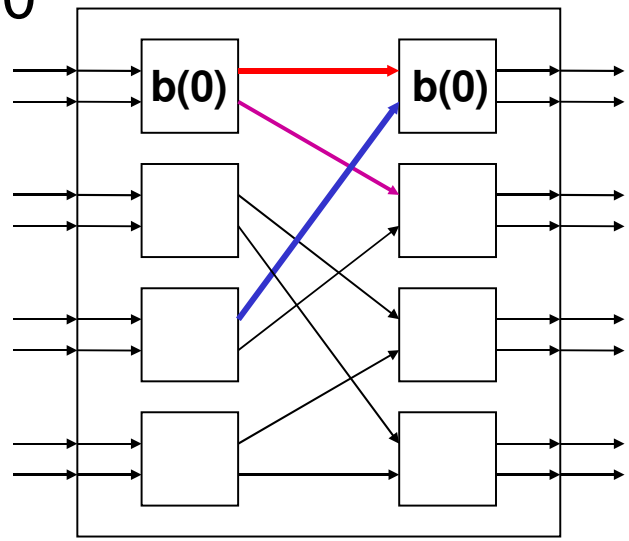
• Simplified Tilers for Shuffle

17. TILERS:**for** i **in** 0 **to** 4 -1 **generate**
18. Source:B **port map** (
    B_IN(0)=> A_IN (i*2),
    B_IN(1)=> A_IN(i*2+1),
    B_OUT(0)=>rehape(i*2),
    B_OUT(1)=>rehape(i*2+1));
19. Target:B **port map** (
    B_IN(0)=>rehape(i),
    B_IN(1)=>rehape(i+4),
    B_OUT(0)=> A_OUT (i*2),
    B_OUT(1)=> A_OUT (i*2+1));
20. **end generate** TILERS;



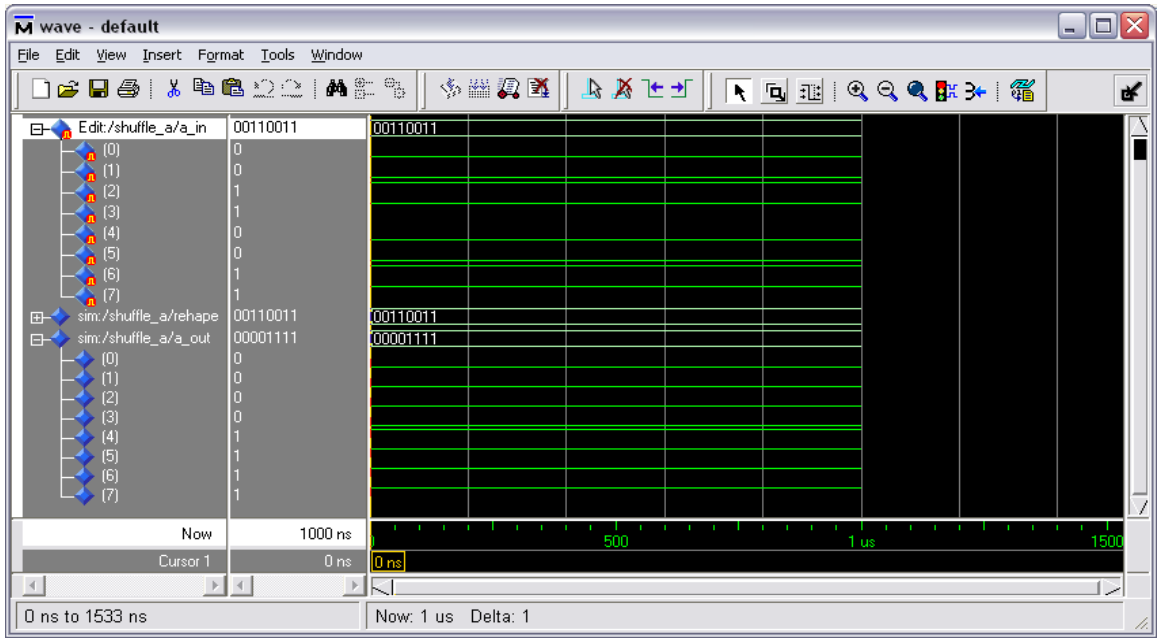
Simplified Tilers for Shuffle

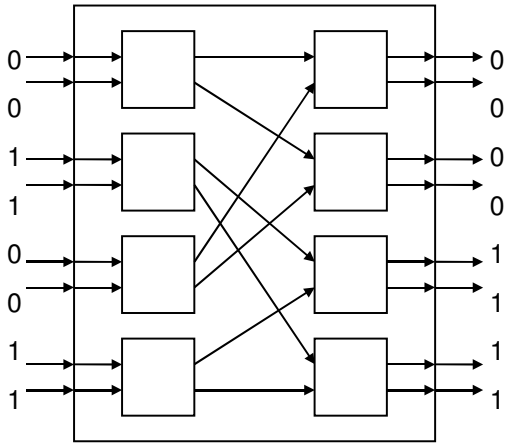
- When index **i** = 0



B_OUT(0)=>**rehape**(i*2), **= 0** B_IN(0)=>**rehape**(i), **= 0**
B_OUT(1)=>**rehape**(i*2+1),**= 1** B_IN(1)=>**rehape**(i+4), **= 4**

- Test-bed simulations of these examples
  - To verify their operation against a set of conventionally coded port-map examples.





- The examples considered so far are considered as minor manipulations of structures comprised of basic elementary components; and as such, not of significant scientific interest to the world of embedded system design.

- Such examples are generated with minimal effort by VHDL software engineers.

- The benefit of alternative method of describing them is not apparent.

- A non-trivial example is required to demonstrate the use of the ArrayOL extension in the modelling tool to describe repetitive component structures.

- Need to present the case of an example that is significant in the design of embedded systems.

4. Current Work

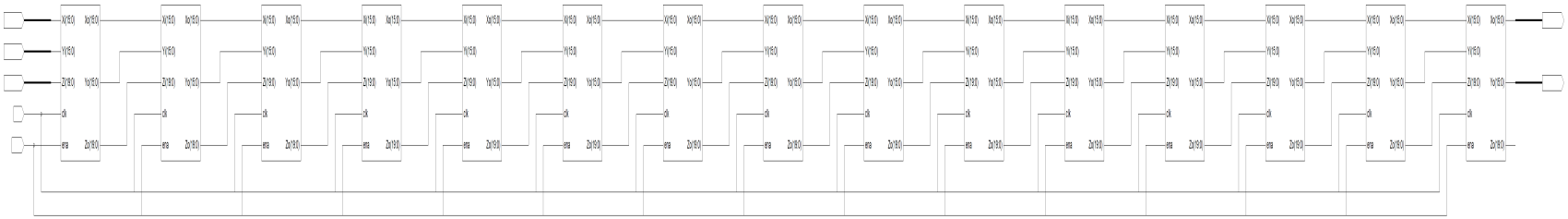
- In search of a more challenging example: I have considered a CORDIC pipeline processing stage, as used in implementations of Fast Fourier Transforms.

- CORDIC is an acronym for **CO**ordinate **R**otation **DI**gital **C**omputer, which forms part of a class of shift-add algorithms for rotating vectors in a plane.

- Vector rotation is useful in a host of DSP applications including modulation and Fourier Transforms.

The CORDIC pipeline structure

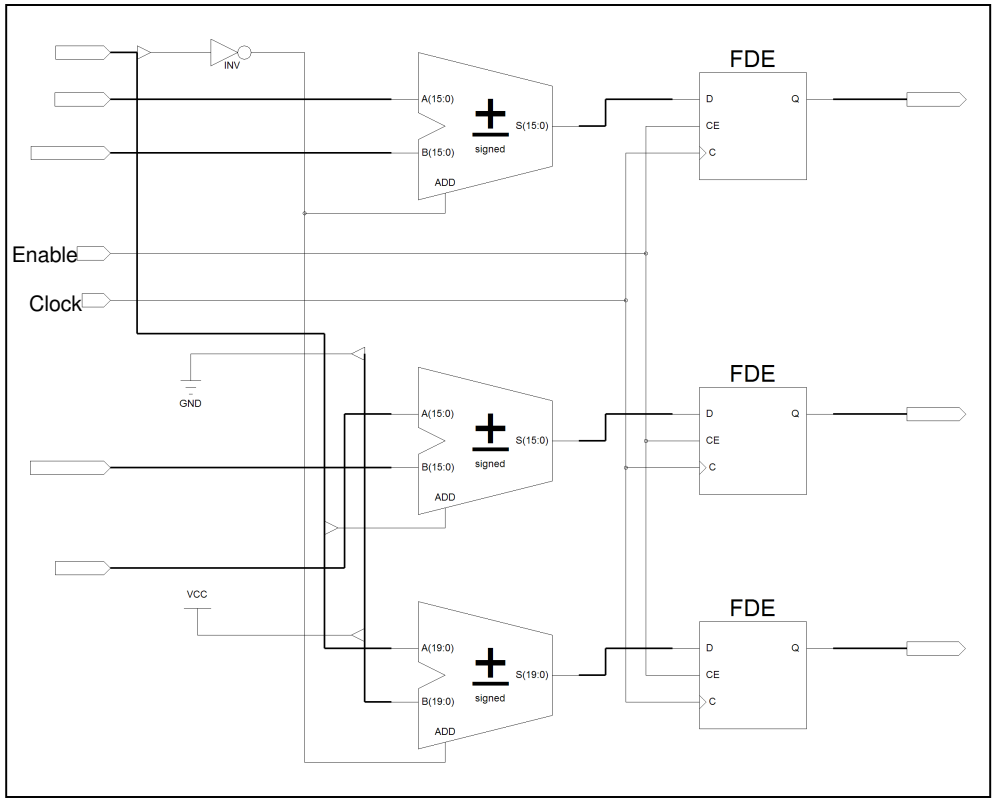
- The COORDIC processor stage can be described as an array of interconnected ‘adder-subtractor’ stages.



- Thus, this component repetition in the X-Plane is ideally suited for an ArrayOL description.

The elementary components

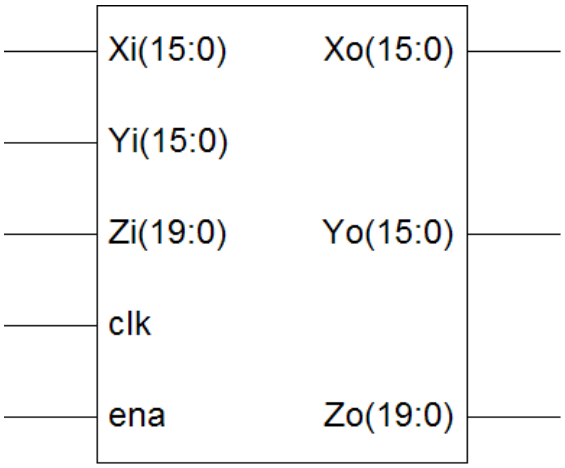
- A single ‘Adder-Subtractor’ stage



Not really concerned with the function of this stage at this time.

Description of the elementary components (*continued*)

- Component entity description of the ‘Adder-Subtractor’ stage



- We can consider the stage as a ‘black box’ with identified ports.

- **BUT . . .**
- When the ‘Adder-Subtractor’ stage is considered as an elementary component in this way. . .

- This example degenerates into little more than those previously considered ☹

- The complexity of this example lies not with the repetition of components, but with the architecture of the component entities.

- ~~Have I been wasting my time~~ ?

- The real question is . . .
  - Is it in fact that repetitive structures are themselves trivial and as such require only trivial examples ??

5. Future work

- Having identified how ArrayOL can be used to describe the VHDL port map constructs
  - And what attributes need to be handled . . .
- The next stage would be to write the necessary transformation rules.
- To replicate a component, we first need to be able to model it as an entity with definable ports.
  - This is an issue which came apparent when attempting to integrate state machines into other architectures, and is currently being addressed.

- That concludes my section of this mornings presentations

Thank You