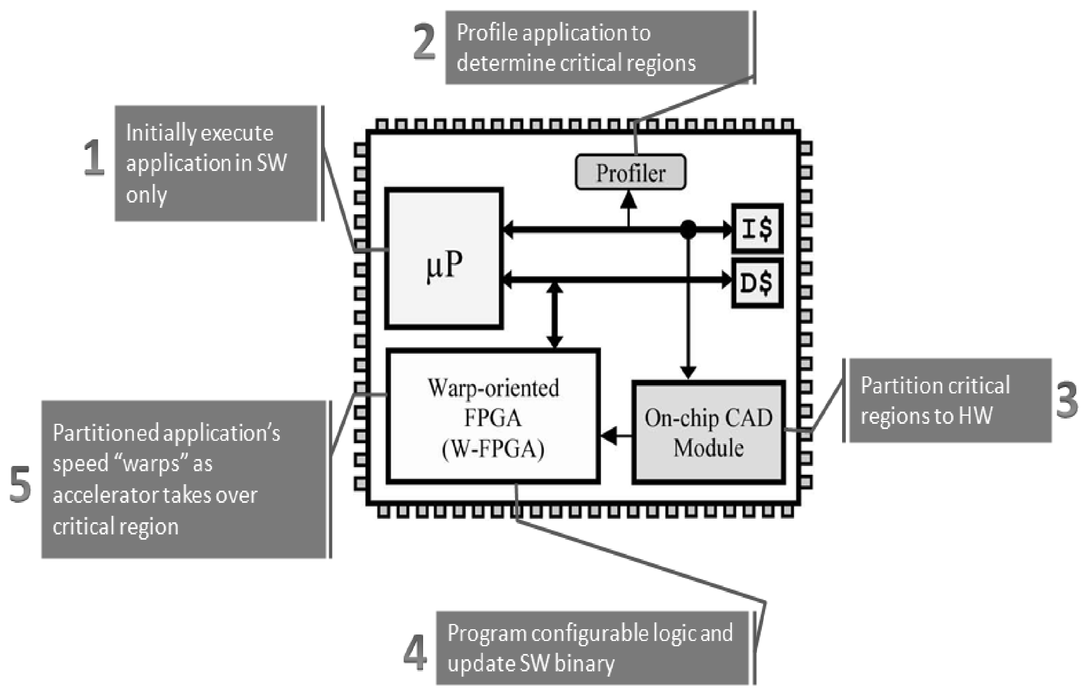
Computer Engineering of TU Delft

Design of CAD Module for JIT Extensible Processor Customized for Placement and Routing

S.H. Daryanavard T. Marconi M. Eshghi

Architecture of Warp Processor



Vahid proposed the first JIT extensible processor named as Warp Processor. In Warp processor, initially applications run on microprocessor which is general propose processor. The profiler detects the binary's kernels of applications, dynamically; then the On-chip CAD Module synthesizes and maps those kernels online to run on FPGA; results are sent back to microprocessor. As a result program's running might suddenly speed up by a factor of 2, 10, or even more. In other words, the running time "warps".

The most Important Challenge

Execution time of CAD Module
FPGA design automation consumes ultra-long time.
Includes the phases to produce a FPGA bitstream: synthesis, technology mapping, place and route

Proposed Idea

In this paper, application-specific instruction set processor (ASIP) has been proposed as a promising solution to speed up CAD algorithm to be used in JIT extensible processor.

Placement algorithm and Functional Units

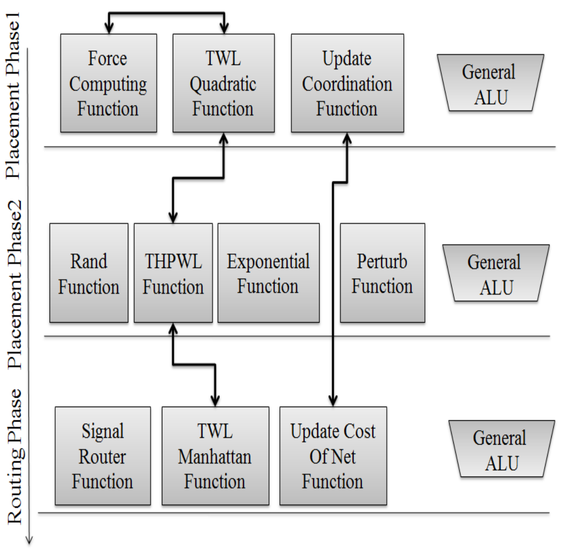
Proposed FPGA Placement: The presented algorithm consists of two stages. In first stage is force-directed based, Second stage is a revised SA placement. At the end of second stage we have 2.33X speedup with the same quality of VPR. First and second stages are called Long Wire Reduction (LWR) and Low Temperature SA (LTSA), respectively, which pseudo code of stages was shown in below figure.

First Stage of Placement Algorithm (LWR) {

1. *For (outer loop)*
2. *For (inner loop)*
3. *F_{Place}=Compute Forces for all Cells;*
4. *Place=Update Position of Cells regarding F_{Place};*
5. *End for;*
6. *Place=Integer Rounding (Place);*
7. *Place=Overlap Remove (Place, F_{Place});*
8. *End for; }*

Second Stage of Placement Algorithm (LTSA) {

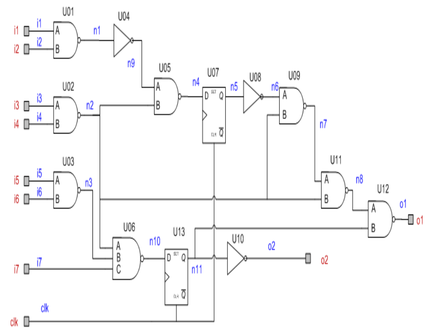
1. *D=initial d T=Initial T --D is high and T is Low Value*
2. *Old Cost=Cost (Place);*
3. *While (T>0)*
4. *For inner loop*
5. *C_1=select cell by uniform random function.;*
6. *C_2=select cell by normal random funct. Regarding C_1 & D;*
7. *New Place=Perturb (C_1, C_2);*
8. *New Cost=Cost (Place);*
9. *Delta Cost=New Cost- Old Cost;*
10. *If (Delta Cost<0)*
11. *Old Cost=New Cost;*
12. *Place =New Place;*
13. *Else if (rand (0, 1) <$e^{-Delta\ Cost/T}$)*
14. *Old Cost=New Cost;*
15. *Place =New Place;*
16. *End if;*
17. *End for;*
18. *D=schedule (D); T=schedule (T);*
19. *End While; }*



We used coarse grained structure. Considering algorithm, the final instructions set is shown in above figure. Double-arrow connector shows resource sharing between instructions which are time isolated. These instructions are extracted manually in terms of functions utilization and profiling of running algorithms in software mod by sim-profile and Dlite! debugger which are commands of Simplescalar simulator tool.

Experimental Results on Small Benchmark

Circuit Benchmark



Code Size Results

Functions	Code Size (Byte)			
	Software Running	Modify Software Running	Fine Grain Inst. ASIP	Coarse Grain Inst. ASIP
Rand_FUN	192	384	272	112
Perturb _FUN	488	-	376	104
THPWL_FUN	960	-	560	112
Exponent_FUN	376	400	224	120
Main T=10:1:0 Inner loop=100	912	912	912	912

Execution Cycle Results

Functions	Execution Cycle (Clock)			
	Software Running	Modify Software Running	Fine Grain Inst. ASIP	Coarse Grain Inst. ASIP
Rand_FUN	48	43	29	13
Perturb _FUN	60	-	46	12
THPWL_FUN	3730	-	1997	247
Exponent_FUN	162	39	27	14
Main T=10:1:0 Inner loop=100	4567868 Ref	4447539 1.03X	2393903 1.9X	347609 13.1X