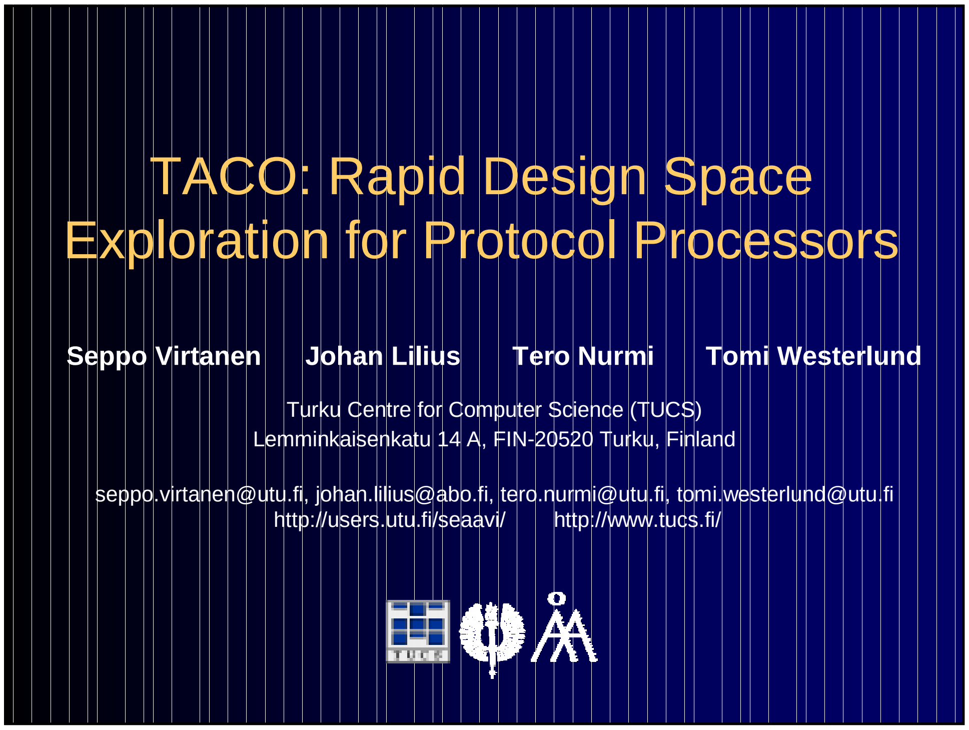
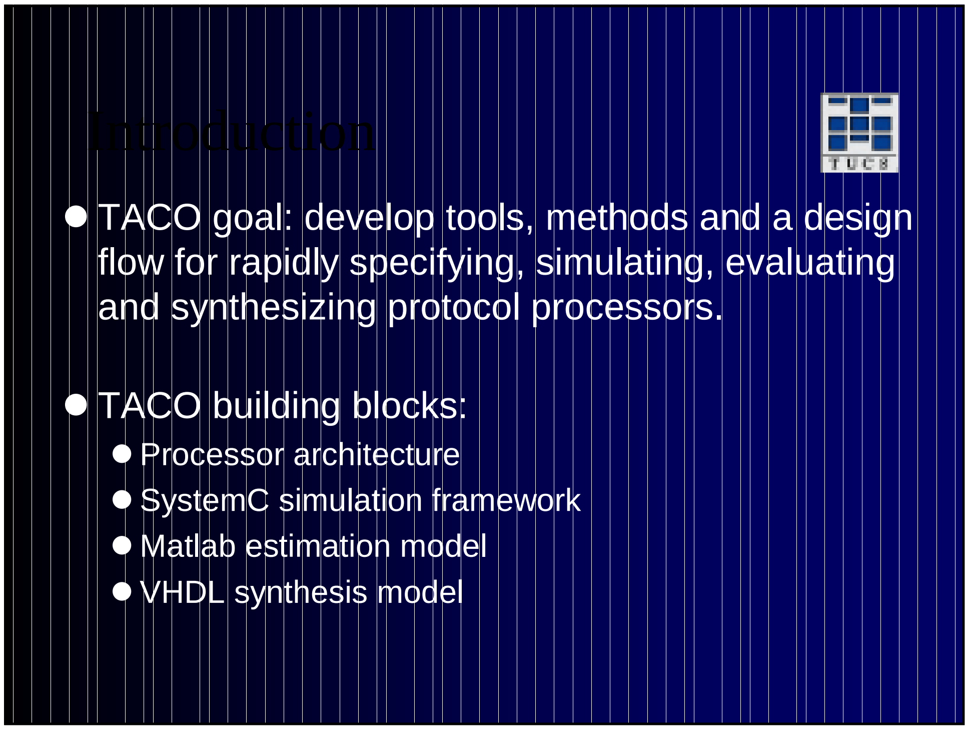


TACO: Rapid Design Space Exploration for Protocol Processors

Seppo Virtanen **Johan Lilius** **Tero Nurmi** **Tomi Westerlund**

Turku Centre for Computer Science (TUCS)
Lemminkaisenkatu 14 A, FIN-20520 Turku, Finland

seppo.virtanen@utu.fi, johan.lilius@abo.fi, tero.nurmi@utu.fi, tomi.westerlund@utu.fi
http://users.utu.fi/seaavi/ http://www.tucs.fi/

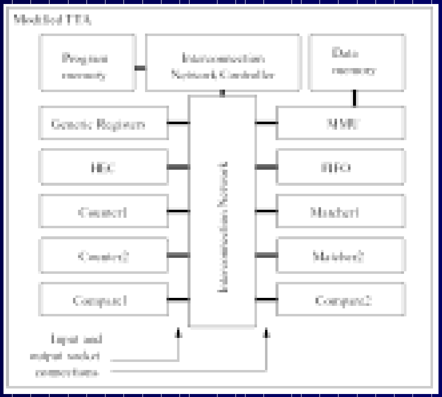


Introduction

- TACO goal: develop tools, methods and a design flow for rapidly specifying, simulating, evaluating and synthesizing protocol processors.
- TACO building blocks:
  - Processor architecture
  - SystemC simulation framework
  - Matlab estimation model
  - VHDL synthesis model

TACO Processor Architecture

- Based on TTA architecture
- Processors constructed of functional units, sockets, interconnection buses, control blocks and memory
- Data moves trigger operations
- Functional units of processor
  - Optimized for protocol processing
  - Alike in structure and connectivity
- One instruction: move
- FU's and interconnection network can be designed independently as long as both follow socket interface spec.

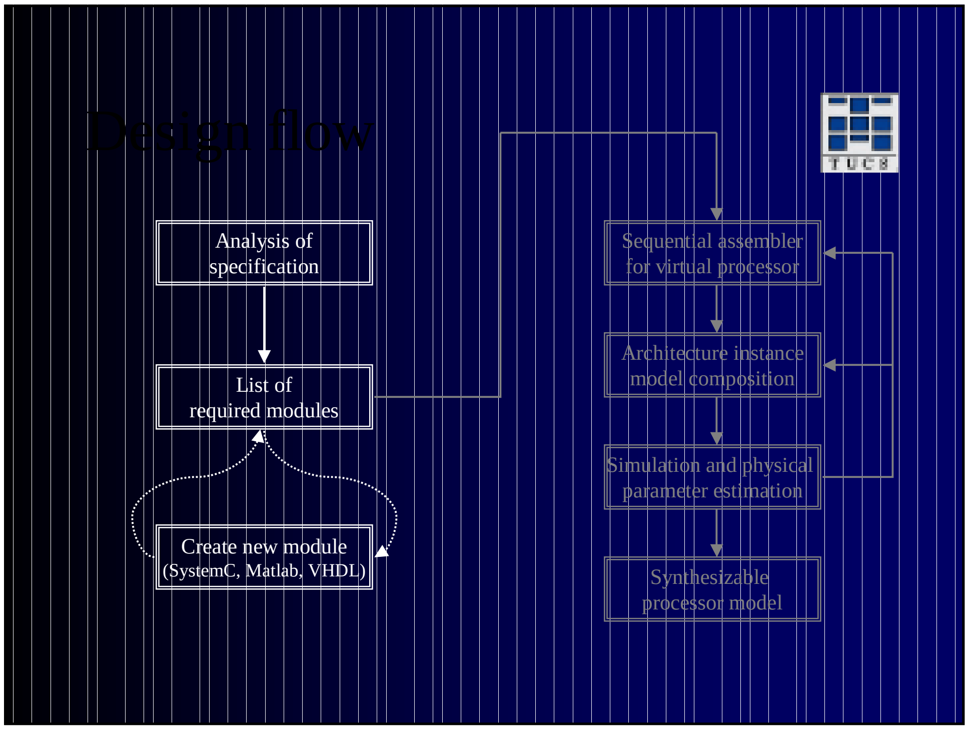


Design flow

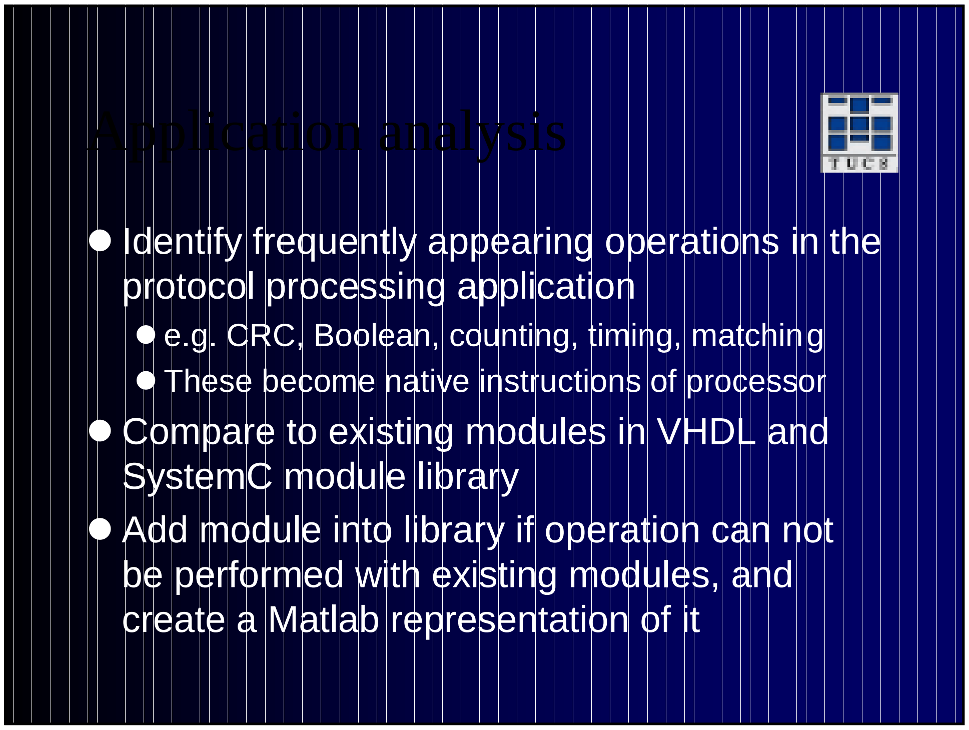
- Derive gate-level synthesized protocol processor models from a high level application description or specification
- Use TACO tools to rapidly:
  - design architecture instances
  - simulate instances
  - estimate physical characteristics of instances
  - analyze instance quality based on simulation and estimation results
  - generate a synthesizable VHDL model

from the system level.

Design flow


Analysis of specification
List of required modules
Create new module (SystemC, Matlab, VHDL)
Sequential assembler for virtual processor
Architecture instance model composition
Simulation and physical parameter estimation
Synthesizable processor model
TUCS

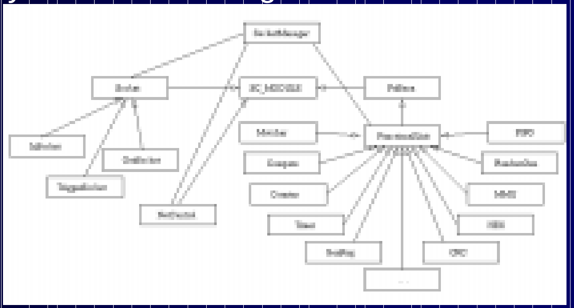
Application analysis



- Identify frequently appearing operations in the protocol processing application
  - e.g. CRC, Boolean, counting, timing, matching
  - These become native instructions of processor
- Compare to existing modules in VHDL and SystemC module library
- Add module into library if operation can not be performed with existing modules, and create a Matlab representation of it

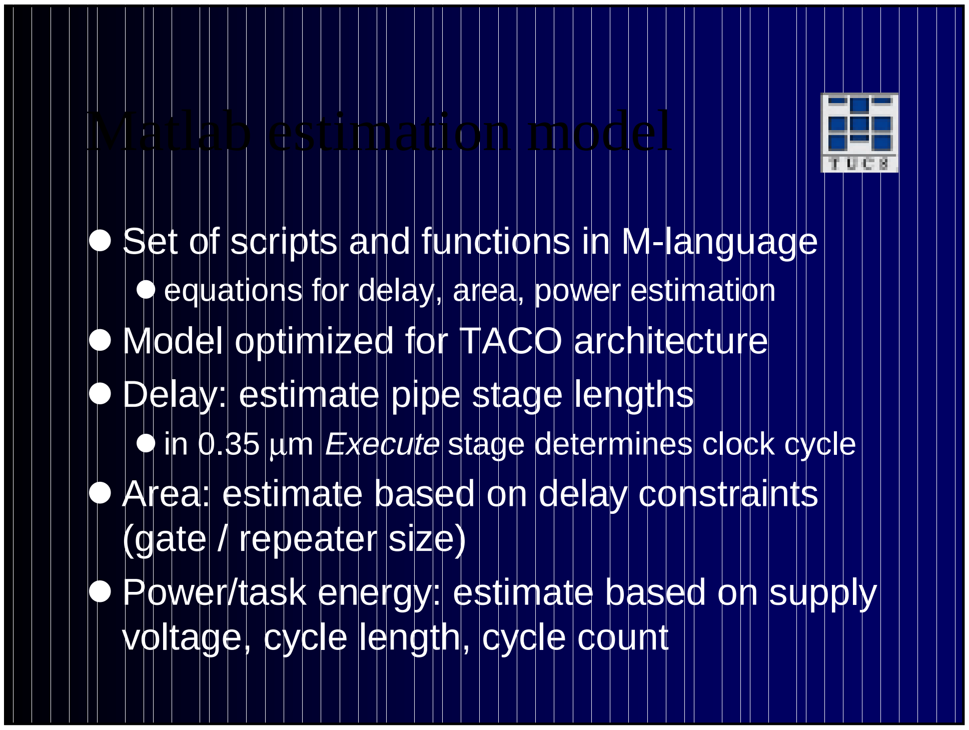
SystemC Simulation framework

- Implementations of FU’s, sockets, interconnection buses, dispatch logic written in SystemC 1.0.1
- Heterogenous level of abstraction
  - Inter-module communication at RTL level
  - Internal functionality of modules at higher levels
- Object oriented techniques used:
  - Inheritance
  - Polymorphism



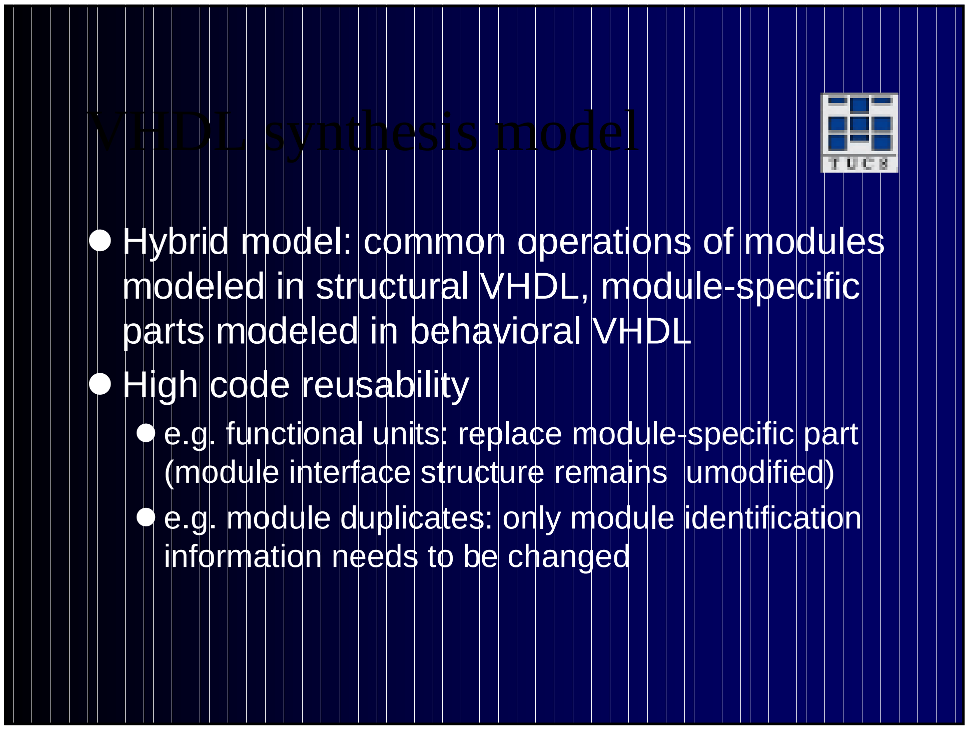
SystemC Model Details

- Parent class encompasses all mutual features of subclasses, both functionality and interfaces
- Leaf classes contain distinctive additions to functionality and interface -> leaf classes remain relatively simple
- Benefits: more compact and readable code, fewer errors, design of new FU’s is faster



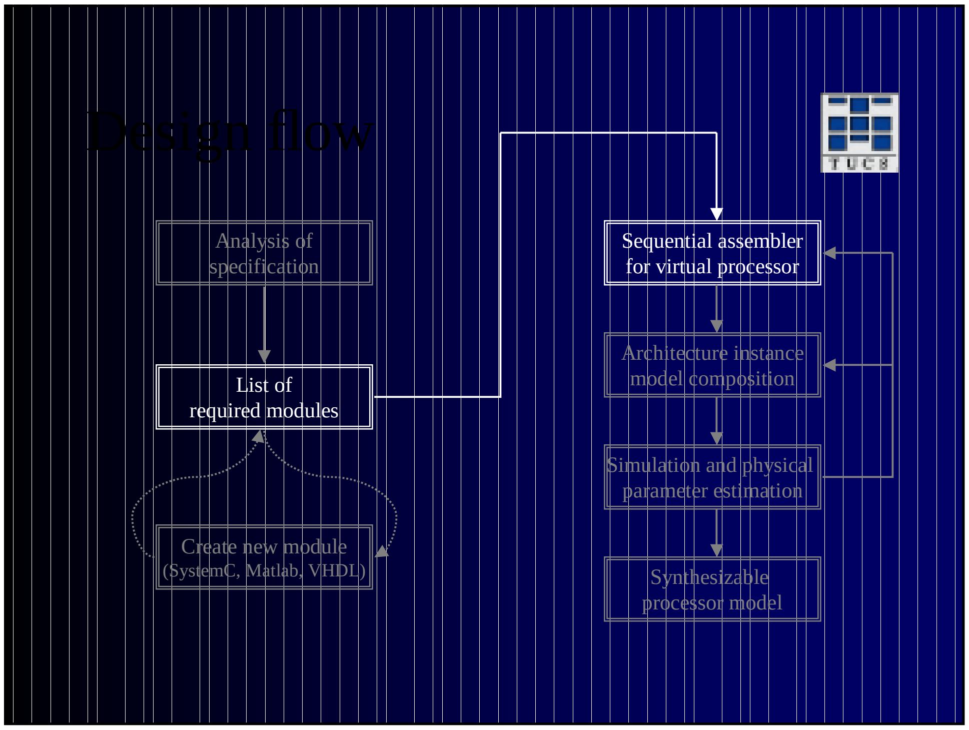
Matlab estimation model

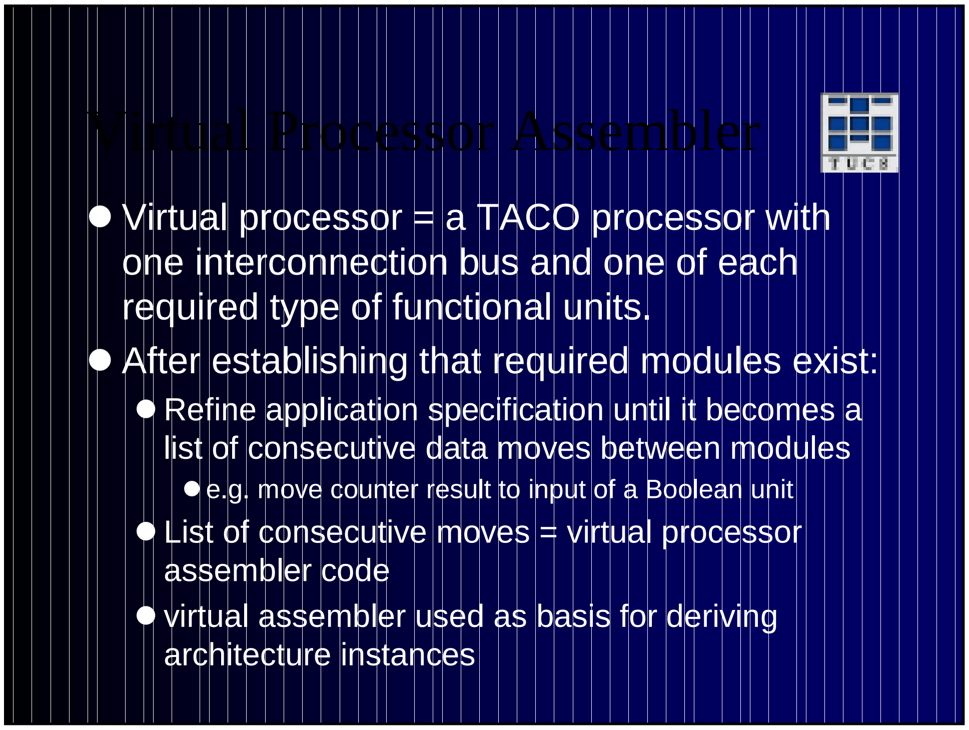
- Set of scripts and functions in M-language
  - equations for delay, area, power estimation
- Model optimized for TACO architecture
- Delay: estimate pipe stage lengths
  - in 0.35 µm *Execute* stage determines clock cycle
- Area: estimate based on delay constraints (gate / repeater size)
- Power/task energy: estimate based on supply voltage, cycle length, cycle count



VHDL synthesis model

- Hybrid model: common operations of modules modeled in structural VHDL, module-specific parts modeled in behavioral VHDL
- High code reusability
  - e.g. functional units: replace module-specific part (module interface structure remains umodified)
  - e.g. module duplicates: only module identification information needs to be changed

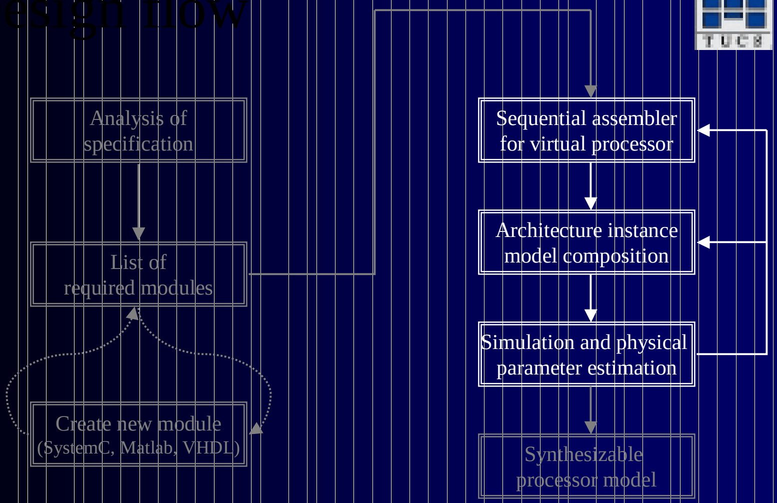

Design flow
Analysis of specification
List of required modules
Create new module (SystemC, Matlab, VHDL)
Sequential assembler for virtual processor
Architecture instance model composition
Simulation and physical parameter estimation
Synthesizable processor model
TUCS

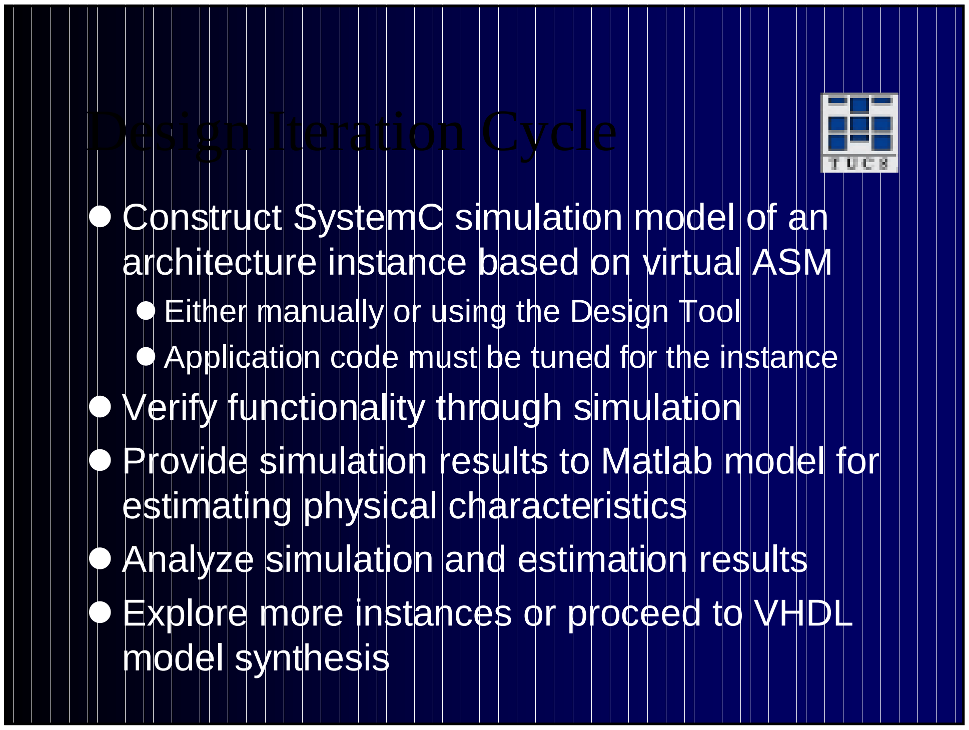


Virtual Processor Assembler

- Virtual processor = a TACO processor with one interconnection bus and one of each required type of functional units.
- After establishing that required modules exist:
  - Refine application specification until it becomes a list of consecutive data moves between modules
    - e.g. move counter result to input of a Boolean unit
  - List of consecutive moves = virtual processor assembler code
  - virtual assembler used as basis for deriving architecture instances

Design flow


Analysis of specification
List of required modules
Create new module (SystemC, Matlab, VHDL)
Sequential assembler for virtual processor
Architecture instance model composition
Simulation and physical parameter estimation
Synthesizable processor model

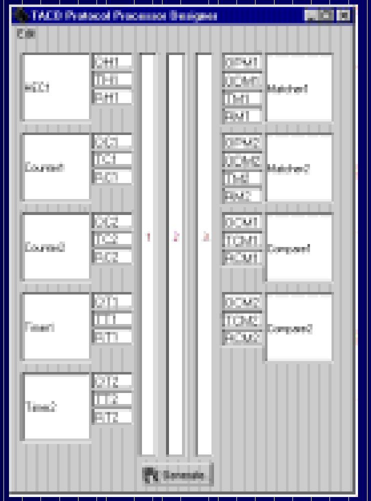


Design Iteration Cycle

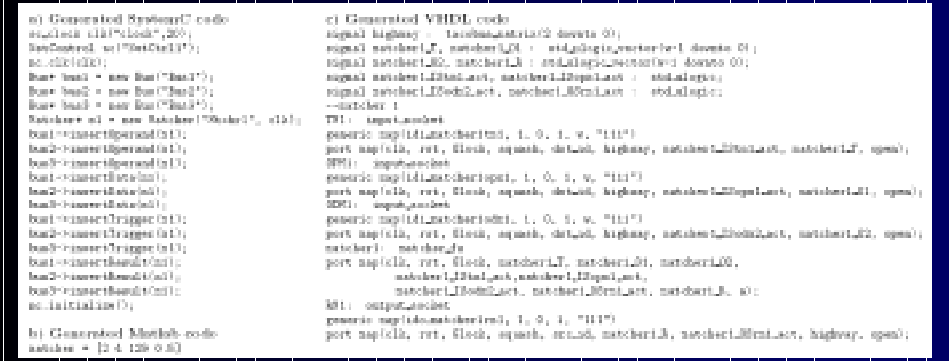
- Construct SystemC simulation model of an architecture instance based on virtual ASM
  - Either manually or using the Design Tool
  - Application code must be tuned for the instance
- Verify functionality through simulation
- Provide simulation results to Matlab model for estimating physical characteristics
- Analyze simulation and estimation results
- Explore more instances or proceed to VHDL model synthesis

Design Tool

- Visual processor design
- Generates SystemC, Matlab and VHDL top files
- Currently only in Windows
- Next version in Java
  - simulation front-end
  - component browsing
  - programming
  - Support for analysis of simulation results

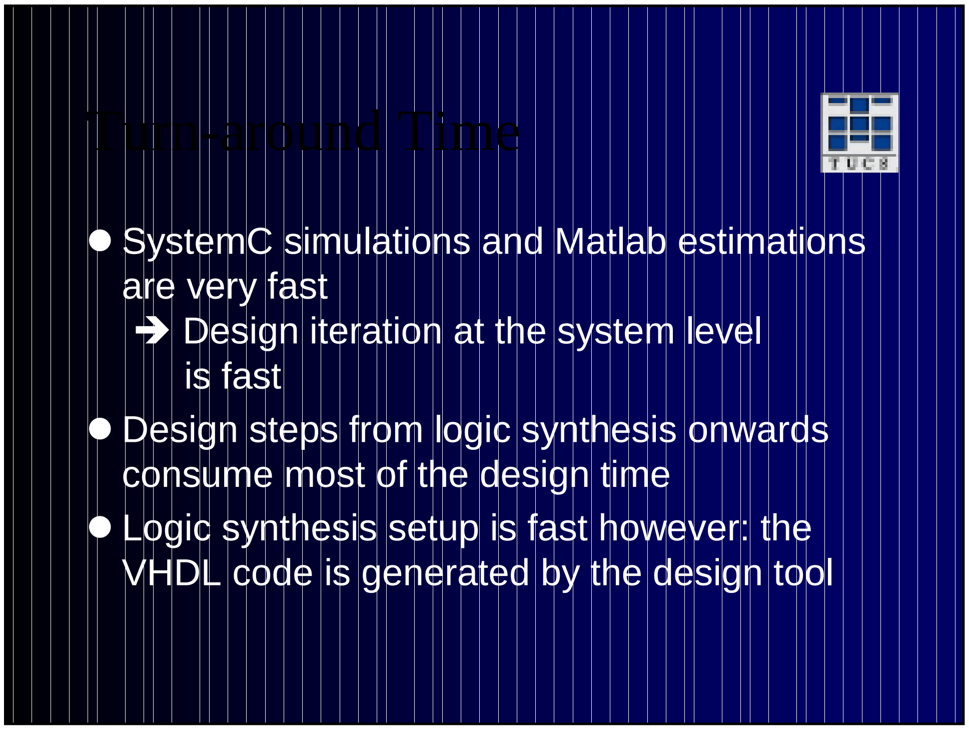


Generated Code



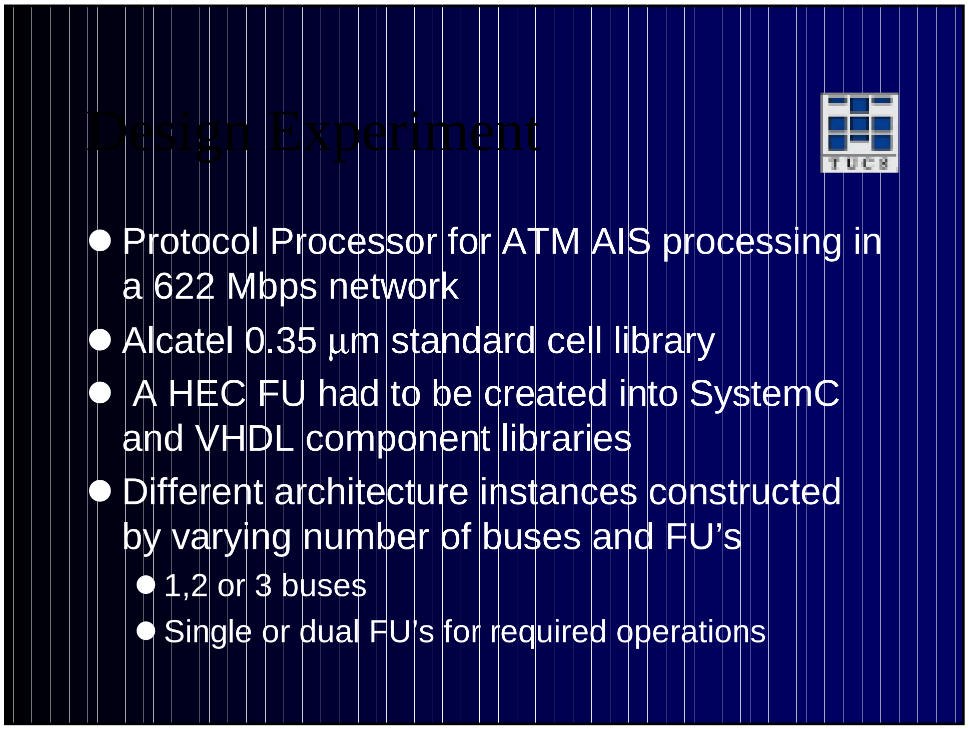
```
a) Generated SystemC code
sc_clock clk("clock",20);
NetControl nc("NetCtrl1");
nc.clk(clk);
Bus* bus1 = new Bus("Bus1");
Bus* bus2 = new Bus("Bus2");
Bus* bus3 = new Bus("Bus3");
Matcher* m1 = new Matcher("Match1", clk);
bus1->insertOperand(m1);
bus2->insertOperand(m1);
bus3->insertOperand(m1);
bus1->insertData(m1);
bus2->insertData(m1);
bus3->insertData(m1);
bus1->insertTrigger(m1);
bus2->insertTrigger(m1);
bus3->insertTrigger(m1);
bus1->insertResult(m1);
bus2->insertResult(m1);
bus3->insertResult(m1);
nc.initialize();

b) Generated Matlab code
matcher = [2 4 128 0.5]
```

```
c) Generated VHDL code
signal highway : [illegible](2 downto 0);
signal matcher1_T, matcher1_O1 : std_ulogic_vector(w-1 downto 0);
signal matcher1_O2, matcher1_R : std_ulogic_vector(w-1 downto 0);
signal matcher1_IDtm1_act, matcher1_IDopm1_act : std_ulogic;
signal matcher1_IDodm1_act, matcher1_RDrm1_act : std_ulogic;
--matcher 1
TM1: input_socket
generic map(id_matchertm1, 1, 0, 1, w, "111")
port map(clk, rst, Clock, squash, dst_id, highway, matcher1_IDtm1_act, matcher1_T, open);
OPM1: input_socket
generic map(id_matcheropm1, 1, 0, 1, w, "111")
port map(clk, rst, Clock, squash, dst_id, highway, matcher1_IDopm1_act, matcher1_O1, open);
ODM1: input_socket
generic map(id_matcherodm1, 1, 0, 1, w, "111")
port map(clk, rst, Clock, squash, dst_id, highway, matcher1_IDodm1_act, matcher1_O2, open);
matcher1: matcher_fu
port map(clk, rst, Clock, matcher1_T, matcher1_O1, matcher1_O2,
         matcher1_IDtm1_act, matcher1_IDopm1_act,
         matcher1_IDodm1_act, matcher1_RDrm1_act, matcher1_R, s);
RM1: output_socket
generic map(id_matcherrm1, 1, 0, 1, "111")
port map(clk, rst, Clock, squash, src_id, matcher1_R, matcher1_RDrm1_act, highway, open);
```



Turn-around Time

- SystemC simulations and Matlab estimations are very fast
  - ➔ Design iteration at the system level is fast
- Design steps from logic synthesis onwards consume most of the design time
- Logic synthesis setup is fast however: the VHDL code is generated by the design tool

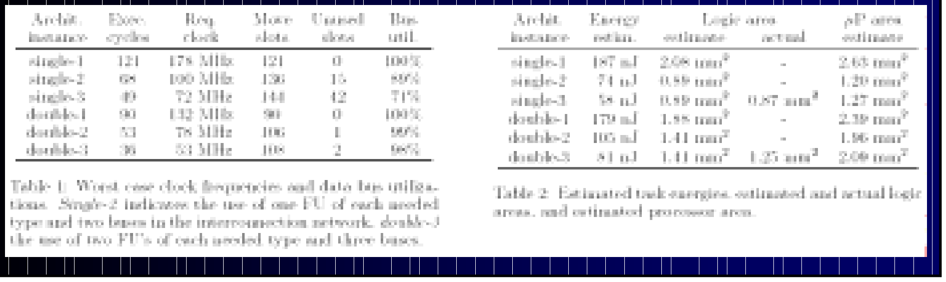


Design Experiment

- Protocol Processor for ATM AIS processing in a 622 Mbps network
- Alcatel 0.35 µm standard cell library
- A HEC FU had to be created into SystemC and VHDL component libraries
- Different architecture instances constructed by varying number of buses and FU’s
  - 1,2 or 3 buses
  - Single or dual FU’s for required operations

Results of Experiment

- Add buses and/or FU's ➔ reduce clock cycles
- Add FU's ➔ consume more energy and area, use buses more efficiently
- Add buses ➔ consume less energy (in the 0.35 μm technology generation)



Archit. instance	Exec. cycles	Req. clock	Move slots	Unused slots	Bus util.
single-1	121	178 MHz	121	0	100%
single-2	68	100 MHz	136	15	89%
single-3	49	72 MHz	144	42	71%
double-1	90	132 MHz	90	0	100%
double-2	53	78 MHz	106	1	99%
double-3	36	53 MHz	108	2	98%

Table 1: Worst case clock frequencies and data bus utilizations. *Single-2* indicates the use of one FU of each needed type and two buses in the interconnection network, *double-3* the use of two FU's of each needed type and three buses.

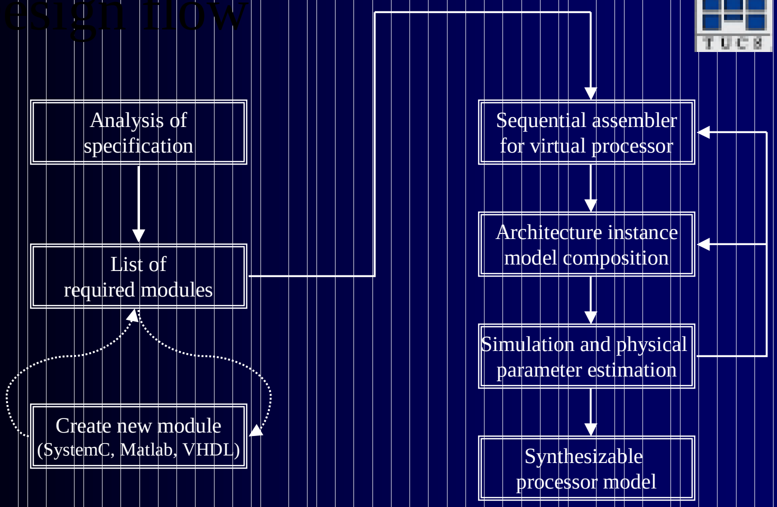
Archit.	Energy	Logic area		μP area
instance	estim.	estimate	actual	estimate
single-1	187 nJ	2.08 mm^2	-	2.63 mm^2
single-2	74 nJ	0.89 mm^2	-	1.20 mm^2
single-3	58 nJ	0.89 mm^2	0.87 mm^2	1.27 mm^2
double-1	179 nJ	1.88 mm^2	-	2.39 mm^2
double-2	105 nJ	1.41 mm^2	-	1.86 mm^2
double-3	81 nJ	1.41 mm^2	1.25 mm^2	2.09 mm^2

Table 2: Estimated task energies, estimated and actual logic areas, and estimated processor area.

Conclusions

- TACO system level simulations and estimations provided reliable results when compared to post-synthesis simulation results
- The TACO design methodology supported this kind of application-specific system design and system level design space exploration well
- Combining results of system level simulation and system level physical parameter estimation gives the designer reliable design feasibility feedback at early stages of the design process

Design flow


Analysis of specification
List of required modules
Create new module (SystemC, Matlab, VHDL)
Sequential assembler for virtual processor
Architecture instance model composition
Simulation and physical parameter estimation
Synthesizable processor model