ESE Back-End

D. Gajski

(with contributions from
S. Abdi, R. Ang, A. Gerstlauer,
J. Peng, D. Shin, R. Doemer)

Center for Embedded Computer Systems
University of California, Irvine
Technology advantages

- **No basic change in design methodology required**
  - ES methodology follows present manual design process
- **Productivity gain of more then 1000X demonstrated**
  - Designers do not write models
- **Simple change management: 1-day change**
  - No rework for new design decisions
- **High error-reduction: Automation + verification**
  - Error-prone tasks are automated
- **Simplified globally-distributed design**
  - Fast exchange of design decisions and easy impact estimates
- **Benefit through derivatives designs**
  - No need for complete redesign
- **Better market penetration through customization**
- **Shorter Time-to-Market through automation**
ES Environment

ESE Front – End
System Capture + Platform Development

ESE Back – End
SW Development + HW Development

Decision User Interface (DUI)
- Create
- Select
- Partition
- Map
- Compile
- Replace

Validation User Interface (VUI)
- Compiler
- Debugger
- Stimulate
- Verify
- TIMED
- CYCLE
- ACCURATE
- Compile
- Check
- Simulate
- Verify

Application Tools: Compilers/Debuggers

Commercial Tools: FPGA, ASIC

Copyright ©2006, CECS
Input: Transaction Level Model (TLM)

TLM is generated automatically from ESE Front-End for given Spec/Platform
TLM Features

- **Universal Bus Channel (UBC)**
  - Bus is modeled as universal channel with send/recv, read/write functions
  - Well defined functions for routing, synchronization, arbitration and transfer

- **SW modeling**
  - Application SW is modeled as processes in C
  - A RTOS model or real RTOS is used for dynamic scheduling of processes
  - Communication with peripherals, memory or other IP is done using UBC

- **HW modeling**
  - Application HW is modeled as processes written in C
  - Communication with processor, memory or other IP is done using UBC

- **Memory modeling**
  - Memory is modeled as array in C
  - Controller is modeled by function in UBC
Cycle-Accurate Software Synthesis

CPU

Program

EXE

RTOS

HAL

HW

Compile

RTOS Synthesis

CPU Bus

Bridge

IP Bus

IP

HAL

B1

B2

B5

OS

HAL

Compile

RTOS Synthesis

Program

EXE

RTOS

HAL
SW Synthesis Issues

• **Compiler selection**
  • The designer specifies which compiler is used for the SW

• **Library selection**
  • Libraries are selected for SW support such as file systems, string manipulation etc.
  • Prototype debugging requires selection of additional libraries

• **RTOS selection and targeting**
  • Designer selects an RTOS for the processor
  • RTOS model is replaced by real RTOS and SW is re-targeted

• **Program and data memory**
  • Address range for SW program memory is assigned
  • Address range for data memory used by program is assigned
  • For large programs or data, off-chip memory may be allocated
Cycle-Accurate Hardware Synthesis

CPU

Mem

CPU Bus

Bridge

IP Bus

B6

B3

Cycle-accurate Synthesis

HW (RTL)

Behavior in C

IP (RTL)
HW Synthesis Issues

- **IP insertion**
  - C model of HW is replaced with pre-designed RTL IP, if available

- **RTL synthesis tool selection**
  - RTL synthesis tool must be selected for custom HW design

- **C code generation**
  - C code for input to RTL synthesis tool is generated

- **Synthesis directives**
  - RTL architecture and clock cycle time is selected
  - UBC calls are treated as single cycle operations, to be later expanded during interface synthesis

- **HDL generation**
  - RTL synthesis result in cycle accurate synthesizable Verilog code
Cycle-Accurate Interface Synthesis

CPU

Arbiter

IC

Custom HW

Interface Synthesis

Mem

Bridge

IP
**Interface Synthesis Issues**

- **Synchronization**
  - UBC has unique flag for each pair of communicating processes
  - Flag access is implemented as polling, CPU interrupt or interrupt controller

- **Arbitration**
  - Selected from library or synthesized to RTL based on policy

- **Bridge**
  - Selected from library or synthesized using universal bridge generator

- **Addressing**
  - All communicating processes are assigned unique bus addresses

- **SW communication synthesis**
  - UBC functions are replaced by RTOS functions and assembly instructions

- **HW communication synthesis**
  - DMA controller in RTL is created for each custom HW component
  - Send/Recv operations are replaced by DMA transfer states
Pin-Accurate Model

PAM is downloaded automatically for fast prototyping with FPGAs
MP3 Player Prototyping with ESE Back-end

- **TLM Input**
  - TLM is generated by ESE front-end for MP3 application and platform

- **Synchronization and Arbitration synthesis**
  - Polling or interrupt mechanism is selected
  - Arbiter is selected for busses with multiple masters

- **SW synthesis**
  - Compiler/RTOS for SW is selected and addresses are generated for memories

- **HW and Bridge synthesis**
  - RTL is generated for custom HW cores by NISC compiler
  - Bridge between CPU bus and peripheral bus is created by Bridge Generator

- **Export to FPGA design tools**
  - Files are generated for creating complete project for FPGA tools

- **FPGA download and test**
  - FPGA design tools create bit-stream for programming the board
  - MP3 player prototype runs directly on FPGA board
- MP3 encoder mapped to SW (MicroBlaze), filters and PCM to HW
- Mem1 (on OPB bus) for data, Mem2 (on LMB bus) for program
- Custom HWs on DoubleHdshk (DH) bus, with bridge to OPB
• **Interrupt signals and connections are selected**
• Interrupt signals and connections are created
Arbiter Selection

- Arbiter is selected and request / grant pins are connected
Model with Arbitration

- The selected arbiter is instantiated and signals are added to create arbiter connections.
SW synthesis

- Compiler, RTOS and libraries are selected for SW
- Default addresses for all addressable memory/bus is generated by ESE
Model after SW synthesis

- SW application and drivers are targeted for RTOS and ready for compilation on MicroBlaze
• RTL code for HW components is generated using NISC compiler
NISC compilation

- **NISC compiler generates synthesizable RTL verilog from application code for selected architecture**

```c
Pixel sum;
//First Temp=Input*COS2
for (i=0; i<8; i++)
{
    sum = Input[i][0]*COS2[0][j]
}
```

ESE Back-End
Model after HW synthesis

- Model is updated with RTL code for HW units
• RTL code for Bridge is generated using BridgeGenerator
Model after Bridge synthesis

- Design is ready for prototyping after SW, HW and Interface synthesis
Export to FPGA Design Tools

- Platform and SW specification files are created for FPGA design tools
- C code for Microblaze and Verilog for HWs and Bridge is exported
Benefit: FPGA Prototype in 1 Week

- Bit stream from FPGA design environment is downloaded to board
- Implemented prototype is tested with MP3 music files
Manual Design Quality

- **Area**
  - % of FPGA slices and BRAMS
- **Performance**
  - Time to decode 1 frame of MP3 data
Design Quality with NISC components

- **Area**
  - NISC uses fewer FPGA slices and more BRAMs than manual HW
- **Performance**
  - NISC comparable to manual HW and much faster than SW
• **Model Development time**
  • Includes time for C, TLM and RTL Verilog coding and debugging
Development Time with ESE

- ESE drastically cuts RTL and Board development time
  - Models can be developed at Spec and TL
  - Synthesizable RTL models are generated automatically by ESE
Validation Time

- Simulation time measured on 3.3 GHz processor
- Emulation time measured on board with Timer
Validation Time with ESE

- **ESE** cuts validation time from hours to seconds
  - No need to verify RTL models
  - Designers can perform high speed validation at TLM and board
ESE Back-end Advantages

• HW synthesis in ESE removes the need to code and debug large RTL HDL models

• Transducer and interface synthesis allows flexibility to include heterogeneous IP in the design

• SW driver synthesis removes the need for SW developers to understand HW details

• SW and HW application can be easily upgraded at TL and validated on board

• C and graphical input of TL model allows even non-experts to develop and test HW/SW systems with ESE