Chapter 6
Co-synthesis Techniques
Cosynthesis

- Methodical approach to system implementations using automated synthesis-oriented techniques
- Methodology and performance constraints determine partitioning into hardware and software implementations
- The result is “optimal” system that benefits from analysis of hardware/software design trade-off analysis
Cosynthesis Approach to System Implementation

- Memory
- Behavioral Specification and Performance criteria
- System Input
- System Output

Mixed Implementation

Pure SW

Pure HW

Performance vs. Cost

© IEEE 1993 [Gupta93]
Co-synthesis

- Implementation of hardware and software components after partitioning
- Constraints and optimization criteria similar to those for partitioning
- Area and size traded-off against performance
- Cost considerations
**Synthesis Flow**

- HW synthesis of dedicated units
  - Based on research or commercial standard synthesis tools
- SW synthesis of dedicated units (processors)
  - Based on specialized compiling techniques
- Interface synthesis
  - Definition of HW/SW interface and synchronization
  - Drivers of peripheral devices
Co-Synthesis - The POLIS Flow

“ESTEREL” for functional specification language
Hardware Design Methodology

Hardware Design Process:
Waterfall Model
Hardware Design Methodology (Cont.)

- Use of HDLs for modeling and simulation
- Use of lower-level synthesis tools to derive register transfer and lower-level designs
- Use of high-level hardware synthesis tools
  - Behavioral descriptions
  - System design constraints
- Introduction of synthesis for testability at all levels
**Hardware Synthesis**

- **Definition**
  - The automatic design and implementation of hardware from a specification written in a hardware description language

- **Goals/benefits**
  - To quickly create and modify designs
  - To support a methodology that allows for multiple design alternative consideration
  - To remove from the designer the handling of the tedious details of VLSI design
  - To support the development of correct designs
Hardware Synthesis Categories

• Algorithm synthesis
  – Synthesis from design requirements to control-flow behavior or abstract behavior
  – Largely a manual process

• Register-transfer synthesis
  – Also referred to as “high-level” or “behavioral” synthesis
  – Synthesis from abstract behavior, control-flow behavior, or register-transfer behavior (on one hand) to register-transfer structure (on the other)
  – Logic synthesis
  – Synthesis from register-transfer structures or Boolean equations to gate-level logic (or physical implementations using a predefined cell or IC library)
Hardware Synthesis
Process Overview

- Specification
  - Behavioral Simulation
  - Optional RTL Simulation
- Implementation
  - Behavioral Synthesis
  - Synthesis & Test Synthesis
- Verification
  - Gate-level Simulation
  - Gate-level Analysis
- Silicon Vendor
  - Place and Route
- Silicon

Behavioral Functional
RTL Functional
Gate
Layout
HW Synthesis

1. Specification Analysis
2. Concurrent Design
3. System Integration
Software Design Methodology

Software Design Process:
Waterfall Model

Software Requirements → Software Design → Coding → Testing → Maintenance → Maintenance → Coding → Software Design → Software Requirements
Software Design Methodology (Cont.)

- Software requirements includes both
  - Analysis
  - Specification
- Design: 2 levels:
  - System level - module specs.
  - Detailed level - process design language (PDL) used
- Coding - in high-level language
  - C/C++
- Maintenance - several levels
  - Unit testing
  - Integration testing
  - System testing
  - Regression testing
  - Acceptance testing
Software Synthesis

- Definition: the automatic development of correct and efficient software from specifications and reusable components

- Goals/benefits
  - To Increase software productivity
  - To lower development costs
  - To Increase confidence that software implementation satisfies specification
  - To support the development of correct programs
Why Use Software Synthesis?

- Software development is becoming the major cost driver in fielding a system

- To significantly improve both the design cycle time and life-cycle cost of embedded systems, a new software design methodology, including automated code generation, is necessary

- Synthesis supports a correct-by-construction philosophy

- Techniques support software reuse
Why Software Synthesis?

- More software ➞ high complexity ➞ need for automatic design (synthesis)
- Eliminate human and logical errors
- Relatively immature synthesis techniques for software
- Code optimizations
  - size
  - efficiency
- Automatic code generation
Software Synthesis Flow Diagram for Embedded System with Time Petri-Net

Vehicle Parking Management System → System Software Specification

Modeling System Software with Time Petri Net

Task Scheduling

Code Generation

Code Execution on an Emulation Board

Feedback and Modification

Test Report
Automatically Generate CODE

- Real-Time Embedded System Model?
  Set of concurrent tasks with memory and timing constraints!

- How to execute in an embedded system (e.g. 1 CPU, 100 KB Mem)?
  Task Scheduling!

- How to generate code?
  Map schedules to software code!

- Code optimizations?
  Minimize size, maximize efficiency!
Software Synthesis
Categories

- Language compilers
  - ADA and C compilers
  - YACC - yet another compiler compiler
  - Visual Basic

- Domain-specific synthesis
  - Application generators from software libraries
Software Synthesis Examples

- Mentor Graphics Concurrent Design Environment System
  - Uses object-oriented programming (written in C++)
  - Allows communication between hardware and software synthesis tools
- Index Technologies Excelerator and Cadre’s Teamwork Toolsets
  - Provide an interface with COBOL and PL/1 code generators
- KnowledgeWare’s IEW Gamma
  - Used in MIS applications
  - Can generate COBOL source code for system designers
- MCCI’s Graph Translation Tool (GrTT)
  - Used by Lockheed Martin ATL
  - Can generate ADA from Processing Graph Method (PGM) graphs
**Software Synthesis Tool:**

The Design of Synthesis Tool for Interrupted-based Embedded Software
Motivation

- System Complexity
- Time to Market
- Hardware-Software Co-design Methodology
- Embedded Software Synthesis Tools
  - Text-Edit User-Interface
- System Model
  - Interrupt Behavior
Designing of a Synthesis Tool

- System Model
  - Interrupt Time Petri Net, ITPN
- Scheduling
  - Interrupt-Based Quasi-Dynamic Scheduling, IQDS
- Code Generation
  - Microcontroller(89c51) C Program Code
- Real-Time Embedded Software Synthesis Tool
  - Graphical User-Interface
System Framework

GUI

Software Synthesis
- ITPN
- IQDS
- Code Generation
Definition for ITPN

- ITPN Definition

\[
ITPN = (P, T, I, O, \Omega)
\]

- A ITPN is a 5-tuple.
- \(P\) is a non-empty finite set of places.
- \(T\) is a non-empty finite set of transitions.
- \(I(t_i)\) is a input function.
- \(O(t_j)\) is a output function.
Definition for ITPN

- $\Omega: \Omega(t) = (\alpha, \beta, \gamma)$
  - $\alpha$: Earliest Firing Time, EFT.
  - $\beta$: Latest Firing Time, LFT.
  - $\gamma$: The type of interrupt for 8051.
A ITPN Example

\[ t_1(\alpha_1, \beta_1, \gamma_1) \rightarrow p_1 \rightarrow t_2(\alpha_2, \beta_2, \gamma_2) \rightarrow p_2 \rightarrow \text{End Place} \]

\[ p_1 \rightarrow t_3(\alpha_3, \beta_3, \gamma_3) \rightarrow p_3 \rightarrow \text{End Place} \]
**Interrupt-Based Quasi-Dynamic Scheduling (IQDS)**

- **Step1:** Find the Initial Place, End Place, and Choice Block
- **Step2:** Decompose the ITPN into two parts: static scheduling and choice block (CB)
- **Step3:** Search the routing path (choice clock set, CBS) for each CB
- **Step4:** Derive all routing path from Initial Place
- **Step5:** Check the real-time constraints for all routing path
An example of IQDS
An example of IQDS (cont.)

Route Extended

<table>
<thead>
<tr>
<th>Present Route</th>
<th>CBS₁ :</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t₁ \rightarrow t₂$</td>
<td>$t₃$ $t₄$ $t₅$</td>
<td>$t₁ \rightarrow t₂ \rightarrow t₃$ $t₁ \rightarrow t₂ \rightarrow t₄$ $t₁ \rightarrow t₂ \rightarrow t₅$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present Route</th>
<th>Static Route</th>
<th>CBS₂ :</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t₁ \rightarrow t₂ \rightarrow t₃$ $t₁ \rightarrow t₂ \rightarrow t₄$ $t₁ \rightarrow t₂ \rightarrow t₅$</td>
<td>$t₆$ $t₇$</td>
<td>$t₁ \rightarrow t₂ \rightarrow t₃ \rightarrow t₆$ $t₁ \rightarrow t₂ \rightarrow t₃ \rightarrow t₇$ $t₁ \rightarrow t₂ \rightarrow t₃ \rightarrow t₈$ $t₁ \rightarrow t₂ \rightarrow t₃ \rightarrow t₉$</td>
<td></td>
</tr>
</tbody>
</table>
An example of IQDS (cont.)

Result
- \( t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_6 \)
- \( t_1 \rightarrow t_2 \rightarrow t_7 \rightarrow t_5 \)
- \( t_1 \rightarrow t_2 \rightarrow t_8 \rightarrow t_9 \)
- \( t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_9 \)

Real Time Check

Schedulable

Not Schedulable

0 \( \alpha \) \( \beta \) 0
RouteTimeMax SystemPeriod
Code Generation

- **Step1**: Differentiate ISR, main_function, and sub_function
- **Step2**: Print transition’s content from Initial Place
- **Step3**: Print “if then else”
- **Step4**: Combine main_function, sub_function, and ISR
Specification

- **Environment:**
  - CPU: Pentium4 1.4GHz
  - Memory: 256MB DDR
  - OS: Windows XP
  - Programming Language: Visual Basic 6.0

- **Input:** Graphical ITPN
- **Output:** Keil C
Graphical User Interface

ITPN Edit, Scheduling and Code Generation
Real-time Stepping Motor Control (RSMC)

- **System Function:**
  - Control Stepping Motor speed and direction.
  - INT0, Timer1: Motor’s speed control.
  - Timer0: Motor’s direction control.
Block Diagram of RSMC

- Stepping Motor
- Driving Circuit
- Microcontroller 89C51
- Display
- Input Device
ITPN Model for RSMC

- **Main Function**

- $t_1$: Capture Keypad register.
- $t_2, t_3$: Turn on Stepping Motor.
- $t_4$: Shut off Stepping Motor.
- $t_5$: Set positive direct.
- $t_6$: Set opposite direction.
- $t_7$: Stop LED on.
ITPN Model for RSMC (cont.)

- Timer0 ISR

\[ t_8(2, 2, 32) \quad t_9(6, 6, 32) \]

\[ p_9 \quad p_{10} \quad p_{11} \]

\[ t_8 : \text{Reset} \quad t_9 : \text{Scan Keypad} \]
ITPN Model for RSMC (cont.)

- **Timer1 ISR**

- \(t_{10}\) : Reset
- \(t_{12}\) : Reset count
- \(t_{14}\) : Control stepping motor

- \(t_{11}\) : Increase counter
- \(t_{13}\) : Undo
ITPN Model for RSMC (cont.)

- **INT0 ISR**

  - \( t_{16}(2, 2, 32) \) : Get the key (Speed up or down)
  - \( t_{16} \) : Check Max speed
  - \( t_{17} \) : Check Min speed
  - \( t_{18} \) : Undo
  - \( t_{19} \) : Speed up
  - \( t_{20} \) : Speed down
  - \( t_{21} \) : Undo
Scheduling and Code Generation
Real Time Constraints

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>ISR’s time</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT0</td>
<td>8</td>
</tr>
<tr>
<td>Timer0</td>
<td>8</td>
</tr>
<tr>
<td>Timer1</td>
<td>15</td>
</tr>
</tbody>
</table>

ISR’s time : $\beta - \alpha > 31$ ($8 + 8 + 15 = 31$)
Cycle Time $> 105$
Emulation

Platform Architecture

- FPGA/CPLD Chip
- Single Chip Microcontroller
- Memory
- Keyboard
- LCD Display
- LED and 7-Segment Display
- Input Switch
Summary

- Extended system model called “Interrupt Time Petri Net, ITPN”
- An Interrupt-Based Quasi-Dynamic Scheduling, IQDS is proposed
- Generate a microcontroller (8051) C program code
- Graphical user interface make the tool more user-friendly