A Distributed Object System Approach for Dynamic Reconfiguration

Ronald Hecht, Stephan Kubisch, Harald Michelsen, Elmar Zeeb, Dirk Timmermann
{ronald.hecht, dirk.timmermann}@uni-rostock.de

University of Rostock,
Institute of Applied Microelectronics and Computer Engineering

April 25th, RAW 2006, Rhodes Island, Greece
Outline

1 Introduction
   - Dynamic Reconfiguration
   - Motivation

2 Distributed Objects
   - Approach
   - Examples
   - Implications

3 System Architecture
   - Middleware
   - Operating System

4 Summary
Outline

1. Introduction
   - Dynamic Reconfiguration
   - Motivation

2. Distributed Objects
   - Approach
   - Examples
   - Implications

3. System Architecture
   - Middleware
   - Operating System

4. Summary
Dynamically Reconfigurable Systems

**FPGA**
- Network-on-Chip
- Reconfigurable tiles with accelerators
- Embedded processor
- Other fixed cores

**Processor**
- Operating system
- Applications
- Decelerators
Tiles, IP Cores and Processes

Hierarchy of Reconfigurable Hardware

- Tile is configured with IP core
- IP Core runs processes
- NoC addressing scheme reflects hierarchy
Simplifying the Design Process

Previous Work
- Object-oriented design
- UML entry
- Design Automation

Looking for new methodologies to . . .
- Unify software and reconfigurable hardware
- Hide dynamic reconfiguration
- Hide relocation
- Borrow from the software world
Outline

1. Introduction
   - Dynamic Reconfiguration
   - Motivation

2. Distributed Objects
   - Approach
   - Examples
   - Implications

3. System Architecture
   - Middleware
   - Operating System

4. Summary
Abstracting the Reconfigurable System

IP Cores . . .

- Are objects
- Have interfaces
- Communicate by the use of messages
- Are distributed in the NoC
Distributed IP Cores

myCore->setKey(key)
myCore->encrypt(plain)
myCore->decrypt(cipher)

Similarities
- Remote method invocation (RMI)
- Client proxy, Server skeleton
- Remote references to access IP cores
C++ Example – Explicit bind

AES Crypto Core

// Declare remote reference
AESCRef myAESC;

// Explicit bind, Trigger dynamic reconfiguration
myAESC = AESCore::getInstance();

// Remote method invocation
myAESC->setKey(aKey);
myAESC->encrypt(aMessage);

// Explicit unbind, Unload IP core
myAESC->releaseInstance();
C++ Example – Implicit bind

AES Crypto Core

```cpp
{
    // Declare remote reference
    AESCoreRef myAESCore;

    // Remote method invocation
    // Implicit bind, Trigger dynamic reconfiguration
    myAESCore->setKey(aKey);
    myAESCore->encrypt(aMessage);

    // Implicit unbind, Unload IP core
}
```
IP Core Relocation

Requirements
- Relocation between hardware and software
- Contexts must be compatible

Object System Approach
- Relocating objects requires serializing objects
- Extend the IP core interface
- Designer knows best about the context save

```plaintext
+setKey()
+encrypt()
+decrypt()
+start()
+stop()
+reset()
```

```
+setKey()
+encrypt()
+decrypt()
+start()
+stop()
+reset()
```

```
+setKey()
+encrypt()
+decrypt()
```

```
+start()
+stop()
+reset()
```

```
AESC0re
```

```
int0e «Interface» IAESC0re
```

```
int0e «Interface» IRelocation
```

```
int0e «Interface» ICrypto
```

```
int0e «Interface» IRelocation
```
Design Flow

Facilitates Automation

- High-level design entry
- Late partitioning
- Border is defined by the designer
- Automatic generation of
  - Messages
  - Proxy
  - Skeleton
- Make use of IDL compilers
Outline

1. Introduction
   - Dynamic Reconfiguration
   - Motivation

2. Distributed Objects
   - Approach
   - Examples
   - Implications

3. System Architecture
   - Middleware
   - Operating System

4. Summary
Middleware simplifies using IP cores
Remote references allow IP cores to access other IP cores
IP core look-up service is visible within the network
Self-triggered Dynamic Reconfiguration

New Approach
- IP cores are allowed to initiate dynamic reconfiguration
- Software, decelerators, and accelerators have equal rights
- Unifies dynamic reconfiguration for hardware and software
Extension of BSD Socket API and `exec()` system call

SystemC FPGA model for evaluation
NoC Integration

- NoC interface is a standard network device
- Easy to use. BSD Socket network programming
OS extensions are accessible through /proc file system
IP cores are loadable in the shell or with exec()
Outline

1. Introduction
   - Dynamic Reconfiguration
   - Motivation

2. Distributed Objects
   - Approach
   - Examples
   - Implications

3. System Architecture
   - Middleware
   - Operating System

4. Summary
Summary

Approach
- Reconfigurable system is a distributed system in the small
- Distributed object system
- Applying a well-known and proven technology

Results
- Perfectly blends with an OOP design approach
- Automation with IDL tools possible
- Self-triggered dynamic reconfiguration
- Middleware and OS extensions