ESL Design using TLM

Yan-Tse Chuang 莊彥澤
DSD/CIC
2007/03/01

Outline

- **TLM Concepts**
  - ESL Tools Overview
- **Platform Based Environment Tools**
  - CoWare – Platform Architect
  - ARM – SoC Designer
- **Mapping Tools**
  - Carbon – VSP
- **Conclusion**
ESL Design Methodology

- Electronic System-Level Design
- System Design Crisis
  - Productivity Gap
  - Complexity
    - ITRS, SIA, 1999
  - Billion-Transistor
- Modeling Methodology
  - High simulation speed
  - Model efficiency
  - Reusability

Transaction-Level Modeling

- Use function calls for communication
  - Minimize the number of events and amount of information that have to be processed during simulation
- Bus model for platform simulation
  - Accuracy
  - Simulation speed
- Advantages of using TLM
  - Fast and compact
  - Integrate HW and SW models
  - Early platform
    - for SW development
    - exploration and verification
  - Verification reuse
TLM Use Cases

- Application/Algorithm Model
  - No timing, point-to-point communication
  - Import from system application domain

- Embedded Software Design
  - Memory map correct (bit-true)
  - Functionally complete and correct system
  - Timing and concurrency only to the level a SW designer is interested

- Architectural Exploration
  - Focus on communication (data rates, data sizes)
  - Application mapping
  - Approximate timing

- ESL Verification
  - 100% accurate profiling
  - Link to hardware description

TLM API Goals

- Common API foundation for TLM SystemC
- Support design & verification IP reuse
- Ease of use, Safety, Speed
- Generality
  - Abstraction Levels
  - Different communication architecture (bus, packet, NoC, …)
  - Different protocols
Abstraction Levels

- **Transaction Level**
  - Layer 3: Message Layer
    - Model (un-)timed functionality
    - Point-point communication
  - Layer 2: Transaction Layer
    - Analyze SoC architecture, early SW development
    - Estimated timing
  - Layer 1: Transfer Layer
    - Cycle true but faster than RTL
    - Detailed analysis, develop low-level SW

- **Pin Level**
  - Layer 0: Register Transfer Level

---

SystemC Language Architecture

<table>
<thead>
<tr>
<th>Core language</th>
<th>Predefined channels</th>
<th>Utilities</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>Signal, clock, FIFO, mutex, semaphore</td>
<td>Vectors, strings, tracing</td>
<td>4-valued logic type, 4-valued logic vectors, bit vectors, arbitrary-precision integers, fixed-point types</td>
</tr>
<tr>
<td>Ports</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interfaces</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Application

Written by the end user

Methodology- and technology-specific libraries

SystemC verification library, bus models, TLM interfaces

Programming language C++
SystemC Features

- A collection of C++ classes
  - Add hardware elements to the C++ language
  - Hierarchical support, Modules, Ports, Signals, Logic types, Multiple clocks, Resets, Parallel execution of events
- Upper layers are cleanly built on top of the lower layers
- Core Language provides only a minimal set of modeling constructs
  - structural description, concurrency, communication, and synchronization
- Data types and user-defined data types are fully supported
- Built on top of the core language
  - Commonly used communication mechanisms (fifo, signal)
  - Commonly used models of computation

---

Design Flow with ESL
Design Case – CBEmacro 3G modem

- Modem solution
  - GSM
  - GPRS
  - EDGE
  - WCDMA
  - HSDPA
- ARM1156T-S

Outline

- TLM Concepts
  - ESL Tools Overview
- Platform Based Environment Tools
  - CoWare – Platform Architect
  - ARM – SoC Designer
- Mapping Tools
  - Carbon – VSP
- Conclusion
ESL Classification Framework

- A unified **Framework** based on “*Platform-Based Design concepts*”
  - Functionality (F)
    - Functional representations of a design
  - Platform (P)
    - Modules used to implement
  - Mapping (M)
    - Identify the elements

---

Classifying ESL Tools

- **MetaBin PM**
  - Platform Based Environment Tools
    - CoWare – Platform Architect
    - ARM – RealView SoC Designer
    - …
  - Mapping Tools
    - Carbon – VSP
    - …
Outline

- TLM Concepts
  - ESL Tools Overview
- Platform Based Environment Tools
  - CoWare – Platform Architect
  - ARM – SoC Designer
- Mapping Tools
  - Carbon – VSP
- Conclusion

Transaction, Transfer and Attributes

- Transaction - a group of Transfers
- Transfer - a set of Attributes
TLM API Syntax in PA

- Sending and Receiving Transactions
  - `<port>.getTransaction()`
  - `<port>.canSendTransaction()`
  - `<port>.sendTransaction()`

- Sending and Receiving Transfers
  - `<port>.getTrfName()`
  - `<port>.canSendTrfName()`
  - `<port>.canReceiveTrfName()`
  - `<port>.sendTrfName()`
  - `<port>.sendDelayedTrfName(delay)`

- Accessing attributes from a Transaction
  - `<port>.Transaction->getAttributeName()`
  - `<port>.Transaction->setAttributeName(value)`

- Accessing attributes from a Transfer
  - `<port>.TrfName->getAttributeName()`
  - `<port>.TrfName->setAttributeName(value)`

- Static sensitivity
  - `<port>.getSendTrfNameEventFinder()`
  - `<port>.getReceiveTrfNameEventFinder()`

- Dynamic sensitivity
  - `<port>.getSendTrfNameEvent()`
  - `<port>.getReceiveTrfNameEvent()`

JPEG using ARM926EJ-S

![JPEG Encoding Diagram]
DCT_AHB_TLM.h

```c
SC_MODULE(DCT_AHB_TLM) {
    // Constructor
    DCT_AHB_TLM(sc_module_name name_) : p_AHB(name_),
        // Constructor code

    static sensitivity:
    port.getSendTrfNameEventFinder();
    port.getReceiveTrfNameEventFinder();

    Transfer Name: WriteDataTrf, ReadDataTrf, EotTrf
}
```

DCT_AHB_TLM.cpp

```c
void DCT_AHB_TLM::receiveWriteData() {
    p_AHB.getWriteDataTrf();
    Block[1] = data; // send data is at first
    // receiveWriteData code
}
```

```c
void DCT_AHB_TLM::sendReadData() {
    p_AHB.getReadDataTrf();
    Block[1] = data; // send data is at first
    // sendReadData code
}
```

```c
void DCT_AHB_TLM::sendEot() {
    p_AHB.sendEotTrf();
    // sendEot code
}
```

```c
void DCT_AHB_TLM::computation() {
    int w, n, y, y; // computation code
}
```

```c
void DCT_AHB_TLM::sendData() {
    p_AHB.sendDataTrf();
    // sendData code
}
```
Outline

- TLM Concepts
  - ESL Tools Overview
- Platform Based Environment Tools
  - CoWare – Platform Architect
  - ARM – SoC Designer
- Mapping Tools
  - Carbon – VSP
- Conclusion

Component

- Modeling the behavior of Hardware element encapsulated in a C++ class.
- Computation – behavior modeling
- Communication – port for connection
  - Master ports
    - Access shared resources from other components
    - SystemC – ports
  - Slave ports
    - Define the access behavior for internal shared resources
    - SystemC – channels

Connection

- Established between Master and Slave Ports
  - Signal based
    - Establish between Master and Slave Ports
    - Connect
  - Transaction based
    - Transaction based
    - Encapsulate a group of signals into one read or write transaction
Capture of System

![Diagram of system components and connections]

E.g. Component with Slave Ports

**Transaction Slave Port 1**

```plaintext
read(addr, value) {
    ...
}
write(addr, value) {
    ...
}
```

**Transaction Slave Port 2**

```plaintext
read(addr, value) {
    ...
}
write(addr, value) {
    ...
}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>FFFF</td>
</tr>
<tr>
<td>2</td>
<td>FFFF</td>
</tr>
<tr>
<td>3</td>
<td>0001</td>
</tr>
<tr>
<td>4</td>
<td>FFFF</td>
</tr>
<tr>
<td>5</td>
<td>FFFF</td>
</tr>
<tr>
<td>6</td>
<td>FFFF</td>
</tr>
<tr>
<td>7</td>
<td>FFFF</td>
</tr>
<tr>
<td>8</td>
<td>FFFF</td>
</tr>
</tbody>
</table>
SoC Designer Interfaces

MxSI – Simulation Interfaces
- MxSI enables *inter-component communication* in hierarchical systems
  - How *components* are called by the scheduler
  - How components *communicate* with each other

MxDI – Debug Interface
- Permit you to view and modify the memory and register for the component
- Creating the MxDI Interface
  - Automatically by the Component Wizard

MxPI – Profiling Interface
- A generic implementation of profiling
  - Collect different types of data
  - Manually implemented in each component
Build Component – JPEG Decoding

- Component Wizard
  - Microsoft Visual C++ (.NET 2003)

SoC Designer Canvas & Simulator

SoC Designer Simulator

SoC Designer Canvas
Interaction

ARM SW

```c
void DCT_2D()
{
    volatile UINT32 *src_buf = volatile UINT32(0x0);
    volatile UINT32 *src_before = volatile UINT32(0x0);
    volatile UINT32 *src_after = volatile UINT32(0x0);
    volatile UINT32 *src_next = volatile UINT32(0x0);
    volatile int i;

    // Read data to DCT
    for(i=0;i<4L;i++)
    {
        *src_before = Bigend(16); // 8 bytes data
        *src_etrl = 1;
    }

    // DCT System Mode executing - BEGIN
    *src_etrl = 2;

    // Read the DCT value after computation
    for(i=0;i<4L;i++)
    {
        Block[i] = *src_after; // sraw data out
        *src_etrl = 3;
    }
    *src_etrl = 4;
}
```

DCT Component

```c
void DCT_1D()
{
    switch (c_level) // Read the data to be computed
    {
        case 1:
            Block[0] = (Block[0] * 0) + (Block[1] * 0);
            break;
        case 2:
            // Remain DCT
            index = 0;
            s4ps2SIG = 1;
            vmvk;
            case 3:
                // Assign the computed value for DCT
                r_after = Block[4];
                vmvk;
                case 4:
                    // Finish
                    vmvk;
                    break;
    }
}
```

Run...

Waveform Viewer

Memory Window

Register Window
Debuggers Support

Outline

- TLM Concepts
  - ESL Tools Overview
- Platform Based Environment Tools
  - CoWare – Platform Architect
  - ARM – SoC Designer
- Mapping Tools
  - Carbon – VSP
- Conclusion
Carbon Design Flow

Design Flow

- Design Flow
- VSP Compiler
  - Options files
    - provide control and guidance to VSP Compiler
  - Directives files
    - control how VSP Compiler interprets and builds an object
  - RTL design files
    - **Golden** RTL of the hardware design

Virtual Hardware Model (VHM)

- A high-performance linkable software object
  - generated by VSP Compiler directly from RTL design files.
- The object contains a cycle and register-accurate model of the hardware design in the form of
  - software object file
  - header file
  - supporting binary database

Validation Environment
Outline

- **TLM Concepts**
  - ESL Tools Overview

- **Platform Based Environment Tools**
  - CoWare – Platform Architect
  - ARM – SoC Designer

- **Mapping Tools**
  - Carbon – VSP

- **Conclusion**
Conclusion

- **TLM**
  - “Some people assume that the TLM is just one thing, ‘transaction-level modeling.’ But what has materialized thru. the Working Group is that it’s actually a number of different styles of models being used at different abstraction levels. It covers the gamut of everything above RTL.”
  - ARM’s Burton, chair of OSCI’s TLM WG
  - “We only care about one thing, the interface between blocks in a system…, the interface at many levels of abstraction.”
  - Sonics’ Wingard, OCP-IP

References

- Carbon, “VSP™ Compiler User Reference”, 2005