Design Methodologies and Circuit Technologies for Embedded Systems

Benny Thörnberg
Associate Professor in Electronics
Goal with this lecture

- Introduce you to the concepts of software hardware co-design
- Stimulate you to analyze and reflect on the Xilinx Vivado system development methodology that you are using in your projects
  - How good is the Vivado workflow to handle co-design?
- Introduce basics concepts of circuit technologies for programmable hardware
- FPGA development towards SoC
Example - Video surveillance of a carpark

- Background estimation
- Dynamic segmentation
- Image component labelling
- Computation of descriptors

Video input

Human, Animal, Car, Motorcycle ...

Classification of intruder

Copyright Benny Thörnberg 2005

Human, Animal, Car, Motorcycle ...

Copyright Benny Thörnberg 2005
Example - Video surveillance of a carpark

- What is the cost of implementing each operation?
  - Cost can be a combination of speed, power, NRE cost, time-to-market …
  - Estimate cost for software and different hw platforms

- How can development work move on in parallel for all operations and for both sw and hw?

- How can we gradually refine a system specification into a synthesizable system model?

- How to verify functionality of sw on a hw platform that is not yet fully developed?

- Can High Level Synthesis be used with satisfactory result?
What is hardware software co-design?

- The meeting of system level objectives by exploiting the tradeoffs between hardware and software in an embedded system through their concurrent design
  - Concurrent design of hardware and software
  - Integrated design environments
  - Whole system functionality is captured in one executable specification
  - No unjustified early assumptions on hardware software partitioning of functionality
  - Partitioning is instead based on a Design Space Exploration where performance parameters are estimated on different HW/SW platforms
Importance of co-design?

• Improves design quality, reduce design cycle time and cost
• Good methodology to handle growing design complexity
• Exploits advances in tools and technologies
  • Processor cores and multi-processor architectures
  • High Level Synthesis (HLS)
  • Asic design, FPGA System on Chip
Why is co-design needed?

• Complexity of embedded systems is increasing
  • Control system for a washing machine
  • Vision based ADAS or autonomous vehicles
  • Large variation of complexity among different tasks
  • Variation of nature of tasks, e.g. reactive and control flow intensive or data dominated tasks

• Most systems today include both custom designed hardware and software running on processors

• Reduce time to market

• Exploits Processor cores and heterogeneous multi-processor architectures
The co-design problem?

- Specification of system
- Partitioning of functionality into smaller tasks executed as software or hardware modules
- Scheduling of software tasks
- Modelling of software and hardware during the design refinement process
Traditional design flow of an embedded system
Design flow for hardware/software co-design

Specifications and requirements
- Behavior specification
  - (1.2) Models and Languages for HW/SW codesign
- HW platforms and IPs
  - (1.4) Processor, memory, and communication architecture design

Platform selection & HW/SW partitioning
- (1.3) Design space exploration

Performance estimation
- (1.5) HW/SW cosimulation and prototyping
- (1.6) Performance estimation, analysis, and verification

Architecture fine tuning

HW/SW synthesis
- (1.7) HW/SW compilation and synthesis

Design Loop
Comparison

![Comparison Diagram]

Reference
The SystemC approach to co-design

- C++ Class library for system modelling
  - Single model for both software and hardware
  - Explore hardware architecture
  - Support for different abstraction levels
    - Untimed Functional (UTF)
    - Timed Functional (TF)
    - Bus Cycle Accurate (BCA)
    - Pin Cycle Accurate (PCA)
    - Register Transfer Accurate (RT)
  - Provides clear separation of communication and behavior
  - Enables gradual refinement of abstraction levels for behavior and communication separately

- SystemC community at [www.accellera.org](http://www.accellera.org)
- SystemC was adopted as standard IEEE 1666-2011
The SystemC approach to co-design

class library and simulation kernel

header files
libraries

Standard C++ development environment

compiler
linker
debugger

"make"

"executable specification"

a.out

source files for system and testbenches

executable = simulator
SystemC abstractions and design flow

Design Exploration
Performance Analysis
HW/SW partitioning

Multi-tasking
Abstract RTOS
Inter process comm.
Scheduling/priority

Target RTOS

Software
Abstr. RTOS
target code

Hardware
Bus Cycle Accurate
Cycle Accurate
Synthesizable

Bus Cycle Accurate
Refine

UnTimed Functional
Refine

Timed Functional
HW/SW Partition

RTL
BCA

SystemC abstractions and design flow

Design Exploration
Performance Analysis
HW/SW partitioning

Multi-tasking
Abstract RTOS
Inter process comm.
Scheduling/priority

Target RTOS

Software
Abstr. RTOS
target code

Hardware
Bus Cycle Accurate
Cycle Accurate
Synthesizable

RTL
BCA

UnTimed Functional
Refine

Timed Functional
HW/SW Partition

RTL
BCA
Example of RTL modelling using VHDL D Flip-Flop with Asynchronous Reset

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity dffa is
  port(  clock : in std_logic;
          reset : in std_logic;
          din   : in std_logic;
          dout  : out std_logic);
end dffa;

architecture rtl of dffa is
begin
  process(reset, clock)
  begin
    if reset = '1' then
      dout <= '0';
    elsif clock'event and clock = '1' then
      dout <= din;
    end if;
  end process;
end rtl;
```
Example of SystemC modelling D Flip-Flop with Asynchronous Reset

```cpp
#include "systemc.h"

SC_MODULE(dffa) {
    sc_in<bool> clock;
    sc_in<bool> reset;
    sc_in<bool> din;
    sc_out<bool> dout;

    void do_ffa() {
        if (reset) {
            dout = false;
        } else if (clock.event()) {
            dout = din;
        }
    }

    SCCTOR(dffa) {
        SC_METHOD(do_ffa);
        sensitive(reset);
        sensitive_pos(clock);
    }
};
```
void generate_data (int &out)
{
    for (int i = 0; i < 10; i++)
    {
        accumulate(out);
    }
}

void accumulate (int &in1)
{
    sum += in1;
    cout << "Sum = " << sum << endl;
}`
Producer

Concurrent process

```
SC_MODULE(producer)
{
    sc_outmaster<int> out1; // master port
    sc_in<bool>      start;  // to kick-start the producer

    void generate_data()
    {
        for (int i = 0; i < 10; i++)
        {
            out1 = i; // this will invoke the slave;
        }
    }

    SC_CTOR(producer)
    {
        SC_METHOD(generate_data); // concurrent process
        sensitive << start;
    }
```
Consumer

SC_MODULE(consumer)
{
    sc_inslave<int> in1; // slave port
    int sum; // declare as a module state variable

    void accumulate ()
    {
        sum += in1;
        cout << "Sum = " << sum << endl;
    }

    SC_CTOR(consumer)
    {
        SC_SLAVE(accumulate, in1); // slave process
        sum = 0; // initialize the accumulator
    }
}
Putting it all together

SC_MODULE(top) // structural module
{
    producer *A1;
    consumer *B1;
    sc_link_mp<int> link1;

    SCCTOR(top)
    {
        A1 = new producer("A1");
        A1.out1(link1);
        B1 = new consumer("B1");
        B1.in1(link1);
    }
};
In-Lined Execution

The execution order in this example is A, A11, A12, A13, A21, A22.

Top most master port is assumed being accessed first.
Sequential execution semantics in multi-point communication links

- A master performs an access to its master port, which will invoke all compatible slaves. Compatible slaves are explained below.
- Slaves are invoked sequentially but in an undefined order.
- Each slave executes and returns. Then, the next slave is invoked in a depth-first order.
- If a slave blocks at a wait(), then the remaining slaves in the list block as well because of the depth-first ordering.
The refinement procedure

- Manual Conversion Creates Errors
- Disconnect Between System Model and HDL Model
- Multiple System Tests
The refinement procedure

• Sequential communication channels are refined to concurrent cycle accurate communication channels with a bus protocol.
• Functional processes are refined to synchronous processes, which have clocks and resets.

SystemC comes with three pre-defined bus protocols:
• No-handshake
• Enable-handshake
• Full-handshake
Ex. Module refinement of producer to RTL

```
SC_MODULE(producer)
{
    sc_out master<int> out1;
    sc_in<bool> start; // to kick-start the producer

    void generate_data()
    {
        for (int i = 0; i < 10; i++)
        {
            out1 = i; // this will invoke the slave;
        }
    }

    SC_CTOR(producer)
    {
        SC_METHOD(generate_data);
        sensitive << start;
    }
};
```

```python
SC_MODULE(producer)
{
    sc_out master<int, sc_fullHndshk<int> > out1;
    sc_in<bool> clk;

    void generate_data()
    {
        for (int i = 0; i < 10; i++)
        {
            wait();
            out1.req = true;
            out1.data = i;
            while (!out1.ack) {
                wait();
                // wait for the clock in the producer
                out1.req = false;
            }
        }
    }

    SC_CTOR(producer)
    {
        SC_THREAD(generate_data);
        sensitive_pos << clk;
    }
};
```
Xilinx Vivado Design Suite High Level Design Flow

Reference: Xilinx User Guide 892
Xilinx Vivado handoff to software development in SDK

Reference: Xilinx User Guide 898
Xilinx Vivado is an IP centric system development suite

- IP core – Intellectual Property is a bundle of:
  - Synthesizable hardware description
  - Software driver
  - Test bench
  - An IP-XACT xml-file description of meta-data
- IP-XACT is standardized in IEEE 1685-2009
Vivado High Level Synthesis (HLS)

Reference: Xilinx User Guide 902
Circuit Technologies for Programmable Logic

• Historical review – Programmable Array Logic
• Field Programmable Gate Arrays
  • CLB – Configurable Logic Block
  • IOB – I/O Block
• Programmability
• From FPGA to SoC
History - Programmable Array Logic – PAL

- Pre-fabricated building blocks with many AND/OR gates
  - Is really built on NOR or NAND gates
  - The circuit can be configured (programmed) by breaking interconnection
  - Block diagram for programmable circuit implementing logic expressions on sum-of-product form

![Diagram of PAL circuit](image)
Before Programming

- All possible connections are available before programming
Connections are removed
  - 'Fuse' normally connected, remove not needed ones
  - 'Anti-fuse' (normally no-connection, create connections)

After Programming

A B C

F0 F1 F2 F3
Example

- Simplified notation – all wires are not drawn
  - Show that there is a connection to the input
  - Implement $F_0 = AB + A'B'$ and $F_1 = CD' + C'D$
Building large programmable circuits

- **Alternative 1: ”CPLD” (Complex Prog. Logic Device)**
  - Put lots of PALs onto the same chip
  - Add programmable interconnects between the PALs

- **Alternative 2: ”FPGA”**
  - Pool of logic building blocks and pool of signal routing
  - Called *Field Programmable Gate Array (FPGA)*
  - That requires
    - A way to implement configurable logic gates
    - A way to make the configurable interconnections
Field Programmable Gate Array - FPGA

- Logic block
  - Implements combinatorial and sequential logic
- Interconnects
  - Wires connecting in- and outputs to logic blocks
- I/O block
  - Special block for external connection of signals
Configurable Logic Block - CLB

- Simplified model for a CLB
  - Lookup tables – LUTs are memories
Configurable I/O Block - IOB

• Simplified model for an IOB
Programmability

• Volatile configuration
• Non Volatile configuration

SRAM
  • Xilinx
  • Altera

FLASH
  • Actel
  • QuickLogic
  • Lattice

One Time Programmable - OTP
  • Actel
  • QuickLogic
  • MicroChip
Programmability

**Volatile**

SRAM memory cell

**Non Volatile**

Floating gate transistor → FLASH

Example of OTP metal-to-metal connection
From FPGA towards System on Chip - SoC

- FPGAs have evolved towards a heterogeneous platform for computation
  - Programmable Logic Area – PLA
    - CLBs
    - Block-RAMs
    - Pool of ALU blocks
      - Adders
      - Multipliers
    - Dynamic memory controllers
  - Configurable Processor System
    - Memory controller
    - Communication interfaces
    - System bus connection to PLA