A HW/SW co-design case

Using UML for Embedded Systems Development
Methods and process
Linux and device drivers
SystemC
Used in a FPGA project

Kim Bjerge, 31 Jan. 08

Using UML in Embedded Systems

Informally
• Sketch a few diagrams – from where the coding proceeds

Structured model
• Tool to generate code and the developer fills in the rest using the model as a guide

Executable model
• The source code is generate based on the UML model.

Translatable with action language (MDA)
• UML model with action language that is executable and can be translated to any implementation. Different structure for model and implementation.
Using UML what do I need to do?

1. Get Training
   - Books and articles
   - Courses in basis UML, design patterns
   - Hire UML experienced architects, consultant or employ
2. Select a methodology
   - With focus on embedded development
   - Adaptation of the methodology to your needs
3. Select a UML Development Tool
   - From simple Visio drawings to advanced tools like Rhapsody
4. Start with a simple project

Development process and methodology

- A process is the specification of a sequence set of activities performed by a collaborating set of workers resulting in a coherent set of project artifacts, one of which is the desired system.

- A methodology consists of a language to specify elements and relations of interest and a process that tells the developer what parts of the language to use, how to use them, and when to use them.

(Bruce Powel Douglass)
UML basis

• UML 2.1
  – Diagrams for structure, behavior and interaction (business process and data structures)
  – Extended with Component, Interfaces, Ports and connectors and Timing diagrams
  – OCL - Object Constraint Language
  – XMI - XML Metadata Interchange of UML models
  – Profile mechanism for extension of the UML standard
  – UML is methodology-independent and defines a graphical language to describe different model views of a system.

What is a UML profile?

– Standard extension mechanism to customize the UML so that any system could be modeled

– Metamodel of the modeling elements
– Set of stereotypes for specific metaclases
– Constraints (OCL semantic)
– Tagged values
Some profiles for system development

- **SPT** Schedulability, Performance and Timing Analysis (OMG 01-2005)
- **SysML** (System Modeling Language, OMG 04-2006)
  - profile to systems which are not purely software based. (Requirement and Block diagrams)
- **UML Profile for SoC** (OMG 08-2006)
  - Class and structure diagrams with special symbols for modules, channels and ports targets concepts from SystemC TLM
- **MARTE** (Modeling and Analysis of Real-Time Embedded Systems) (OMG 2008)
- **UML 2.0 profile for SystemC**
  - STMicroelectronics and Universities in Italy

UML 2.0 diagrams

- **Structure**
  - Composite Structure diagram*
  - Package diagram*
  - Component diagram*
  - Class and Object model diagram*
  - Deployment diagram*
- **Behavior**
  - Statechart*
  - Activity diagram
  - Use case diagram*
- **Interaction**
  - Sequence diagram*
  - Timing diagram*
  - Communication diagram
SysML (customization of UML 2.0)

- Requirements
  - Requirements diagram
  - Use case diagram
- Structure
  - External block diagram
  - Internal block diagram
- Behavior
  - Statechart
  - Activity diagram
  - Sequence diagram
- Constraints
  - Parametric diagram

TUP - UML Methodology

- Teknologisk Unified Process (TUP)
- SPU-UML Finn Overgaard Hansen
- S-Model
  - Rapid Object-oriented Process for Embedded Systems (ROPES and Harmony)
- U-Model
  - Use Case driven development
- W-Model
  - Iterative delivery model (Alistair Cockburn)
- T-Model
  - Development activities for test by use of the V model
**HARMONY® a revision of ROPES**

*Systems Engineering Workflow (UML+SysML)*

**HARMONY®**

*Essential Systems Engineering Model Artifacts*
Harmony - Planning with risks

- Maintain a risk management plan
- Each risk should be identified and ranked
- A risk-mitigation strategy must be described
- In each party phase, the risk plan is reviewed
- Newly risks are added and minimized risks are updated
**W-model – focus on delivery**

9 måneder

Ekstern synlige leverancer

4 måneder
3 måneder
2 måneder

Interne udviklings-iterationer og leverancer

Evaluering, Leverance

Kilde: A. Cockburn

---

**T-Model for test activities**

Diagram af testaktiviteterne i context:

- Næste iteration
- Accept-testspecifikation
- Implementering af SW & HW

Kilde: A. Cockburn
MBC – UML System Analyse

- Use Case specification
- Domain analysis
- Application analysis
- Class Diagrams
- Sequence Diagrams
DIMMS - Data Interpretation and Model Management System

MBC - UML System Analyse (Domain – Use Case)
4.4 Use Case descriptions

**Use Case descriptions**
MBC - UML System Analyse
(Domain and Application – Sequence diagram)

ADC Interface
Controller
Sample Block
Power Spectrum Block
Welch PSD

Loop: ADC Input
Loop: Collect Input Samples [n < Sample Block Size]
Loop: Sum Processed Sample Blocks [n < Num Sample Blocks]

Interrupt controller timing calculation:
interruptRate = sampleRate / (Sample Block Size * Num Sample Blocks)
= 250.000 / (512 * 41) = 12 times/second or 83 ms between interrupts

MBC – UML System Design

Architecture & design
- Logical View
  - Refinement of class diagram
  - Packages
- Process View
  - Adding concurrency
- Deployment View
  - Components
  - Partitioning
  - Deployment
- Implementation View
  - Signals and Timing
  - Sequence and states
- HW/SW Interface View
  - IP memory map
DMW: 606402

MBC - UML System Design View
(Logical - Packages)

DMW: 606402

MBC - UML System Design View
(Process - Refinement with concurrency)
MBC - UML System Design View
(Implementation - Timing)
MBC - UML System Design View
(Implementation - States, Sequence)

WelchPSD IP-Core memory map
(Implementation – Interface)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Words (32bit)</th>
<th>Direction</th>
<th>Reset value</th>
<th>Range</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumSampleBlocks</td>
<td>Number of sample blocks to be used in calculation of a PowerSpectrumBlock.</td>
<td>W/R</td>
<td>4</td>
<td>32-128</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>SampleBlockSize</td>
<td>Size of sample block. (PowerSpectrumBlock)</td>
<td>W/R</td>
<td>512</td>
<td>256-1024</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>WindowWidth</td>
<td>Window width of the Welch curve.</td>
<td>W/R</td>
<td>1024</td>
<td>512-2048</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>Control</td>
<td>Control register for the Welch PSD.</td>
<td>W/R</td>
<td>0</td>
<td></td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>OverflowStatus</td>
<td>Overflow counter. Incremented every time a block overflow is detected.</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>Status</td>
<td>Status register for the Welch PSD.</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>BlockCount</td>
<td>Incremented each time a new block is updated.</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>PowerSpectrumBlock</td>
<td>Processed PowerSpectrumBlock. Valid words in block depends on the SampleBlockSize. There will be new data in block when Status register DataReady bit is 1. Buffer is emptied by reading NumSampleBlocks.</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td>Window</td>
<td>Welch curve. Must be updated during initialization before enable of WelchPSD. WindowWidth words are written to this field.</td>
<td>W/R</td>
<td>0</td>
<td></td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>RawSample</td>
<td>Raw input sample. Updated at a rate of 250 kps.</td>
<td>W/R</td>
<td>0</td>
<td></td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>ScaleValue</td>
<td>Scaling value for the calculated PSD spectrum</td>
<td>W/R</td>
<td>0</td>
<td></td>
<td>10</td>
<td>45</td>
</tr>
</tbody>
</table>
MBC - System Implementation

- VHDL PSD algorithm from Simulink
- ADC interface SystemC to VHDL
- SystemC test bench
- Memory register interface in VHDL
- Linux device driver in C
- Linux C++ application

MBC – UML System Verification
Key Features for MDA (Rhapsody) (Model-Driven Architecture with UML)

- Visual Modeling*
- Model Execution*
- Platform-independent (PIM)*
- Platform-specific (PSM) translation of PIM to technology platform
- Real-Time Framework (language and OS)
- Model-Code Associativity*
- Rules-Based Code Generation*
- Automated Requirements-Based Testing
- Reverse Engineering

MDA Rhapsody example (Demo)
UML research for HW/SW codesign

- UML as the language for HW/SW people in a project to give a common understanding of the system under construction
- Agile methods using UML will ensure that focus is put building the system in smaller steps ensuring fast verification of specification and reducing risks
- Executable model driven development forces you to prove you architecture and design before detailed implementation
- Still haven’t found the ultimate UML tool to automate the partitioning of the HW/SW codesign

UML Inspiration, books and links

- UML Distilled, Martin Fowler
- Real Time UML Third Edition, Bruce Powel Douglass
- Real-Time Design Patterns, Bruce Powel Douglass
- UML for SOC Design, Grant Martin and Wolfgang Müller
- Object Management Group, home [www.omg.org](http://www.omg.org)
- UML Courses by Technological Institute, link [http://www.teknologisk.dk/kurser/8304-0?category_id=108](http://www.teknologisk.dk/kurser/8304-0?category_id=108)
Linux and device driver

What is Linux?

Linux is a full-blown 32-bit operating system.
Linux is GPL.
Free? NO!

Our statement is “There is no free Operating System”. You must at any time invest in infrastructure and education.

The educational investment is the largest so:
“don’t expect to save money, expect to save problems!”
**Which flavours of Linux do we use at DTI?**

- **2.4** Currently only used for demonstration purposes. Will be phased out.
- **2.6** Used in TI/Århus.
- **uClinux** TI/Århus, the commercial model is changing; the license is not free anymore.

---

**Why Linux**

- **Pros**
  - Linux has a lot of support for networking, devices at different boards.
  - Linux is widely accepted; a lot of developers are supporting you “out there”.
  - There is also Commercial support.
  - If you know linux on one board, your knowledge is portable to another board
- **Cons**
  - Linux is BIG, that means you cannot use it for very small systems.
  - As with all OS’es you have to learn how.
  - Who is responsible.
History of UNIX/Linux

Early History

- Linus Torvalds creates Linux in 1991
  - Based on Minix (Andrew S. Tanenbaum)
    - From Minix 16bits to Intel386 32bits architecture
- 09-1991 Release 0.01
- 12-1991 Release 0.10
  - Support for AT-disks, no login -> bash
- 06-1993
  - Slackware – First distribution
- 03-1994
  - Linux kernel 1.00
Which kernel shall I choose?

- If you “google” for recipes most literature covers 2.4, so there is a lot of support.
- If you examine it a bit further, the 2.6. The kernel is more mature, with exception of Ethernet support. But there is not much of support.
- Xilinx EDK (9.1i) does not support the standard 2.6 kernel structure very well.

DIMMS Linux implementation

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PSDProxyReader</td>
<td></td>
</tr>
<tr>
<td>Linux Application API</td>
<td>open, read, ioctl, close, (select, write)</td>
</tr>
<tr>
<td>WelchPSD Driver</td>
<td>structure operations</td>
</tr>
<tr>
<td>WelchPSD Device</td>
<td>Resources Memory address map Interrupt</td>
</tr>
<tr>
<td>Xilinx IP memory map</td>
<td></td>
</tr>
</tbody>
</table>
Linux Kernel

Linux Kernel space & user space
Linux Device Driver

- Application API contains functions to open, read, ioctl and close
- Driver handles initialization and access to Xilinx IP core memory map, operations to read PSD blocks of from IP core and copy data from kernel to user space
- Device handles resources like memory map, interrupt, device instantiation in Linux kernel and installing interrupt

Linux Device Driver Model (Char Special)

- The module parameters are configurable at module load time
- The symbols are visible from outside
- The IRQ handler handles the interrupts from HW.
- Init is called when the module is initialized (loaded)
- Probe is called by the kernel when needed, it is used to get the resources (interrupts and memory) which the driver uses to communicate the kernel, and to do the setup.
- The Methods are standard device driver operations such as Open/Read/Write/Close.
- Register and Unregister are used when the device driver is loaded or removed by the Linux kernel.
- Exit is called when the device driver(module) is removed.
Linux boot sequence

- Register resources (Memory, Interrupt)
- Load modules
- Driver probe() allocates resources, initialization of hardware and install/enable interrupts
- Start_kernel()
  - Kernel_thread() -> init (Process PID 1 starts)

Linux Application API (PSDProxyReader)

```cpp
int fd = open(fileName_c_str(), O_RDONLY);
if (fd > -1)
{
    size_t count = read(fd, reinterpret_cast<char*>(intBuffer), sizeof(intBuffer));
    if(count != sizeof(intBuffer))
    {
        close(fd);
        throw PSDError(“Couldn't retrieve data...”);
    }
}

int overflow;
ioctl(fd, WPSD_IOC_OVERFLOW, &overflow);
std::cout << “Overflow “ << overflow << std::endl;
close(fd);
```
Device resources

#define XPAR_WELCH_PSD(num) { \n    .name = "welch_psd", \n    .id = XPAR_WELCH_PSD_##num##_DEVICE_ID, \n    .num_resources = 2, \n    .resource = (struct resource[]) { \n        { \n            .start = XPAR_WELCH_PSD_##num##_BASEADDR, \n            .end = XPAR_WELCH_PSD_##num##_HIGHADDR, \n            .flags = IORESOURCE_MEM, \n        }, \n    }, \n    { \n        .start = XPAR_INTC_0_WELCH_PSD_##num##_VEC_ID, \n        .flags = IORESOURCE_IRQ, \n    }, \n};

WelchPSD driver structure

static struct platform_driver welch_psd_platform_driver = { 
    .probe = welch_psd_probe, 
    .remove = welch_psd_remove, 
    .driver = { 
        .owner = THIS_MODULE, 
        .name = "welch_psd", 
    }, 
};

MODULE_AUTHOR("Danish Technological Institut");
MODULE_LICENSE("GPL");
module_init(welch_psd_init);
module_exit(welch_psd_exit);
WelchPSD driver API operations

static int welch_psd_open(struct inode *inode, struct file *file);
static int welch_psd_read(struct file *file, char __user *buffer, size_t count, loff_t *ppos);
static ssize_t welch_psd_write(struct file *file, const char __user *buf, size_t count, loff_t *ppos);
static int welch_psd_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg);
static int welch_psd_release(struct inode *inode, struct file *file);

static const struct file_operations welch_psd_fops = {
    .owner = THIS_MODULE,
    .open = welch_psd_open,
    .read = welch_psd_read,
    .write = welch_psd_write,
    .ioctl = welch_psd_ioctl,
    .release = welch_psd_release,
};

Driver read operation

ssize_t dev_read(struct file *file, char *buf, size_t count, loff_t *ppos);

Kernel Space (non-mpageable)

User Space ( pageable)

Buffer (in the drive)

copy_to_user()

Buffer (in the application or lttcb)
Driver files - Linux Kernel 2.6

Source files that are relevant for Xilinx Linux driver development

- **ML405 parameters for memory address space**
  \texttt{linux-2.6.23\_welch\_3/arch/ppc\_platforms/4\_xx\_parameters/xparameters\_ml403.h}

- **Kernel creation of device drivers for ML405**
  \texttt{linux-2.6.23\_welch\_3/arch/ppc\_syslib/virtex\_devices.c}

- **Xilinx common files**
  \texttt{linux-2.6.23\_welch\_3/driver\_activities/xilinx\_common\_util.h, xio.h, xbasic\_types.h}

- **WelshPSD driver files**
  \texttt{linux-2.6.23\_welch\_3/drivers/char/welch\_psd}
  \texttt{welch\_psd\_driver.c, welch\_psd\_c, welch\_psd.h}

---

Lessons Learned

- Linux is running well on FPGA’s
- A new version of MicroBlaze has been released – with MMU.
- That means Full Linux is available also for MicroBlaze!
- In our case the best way is to use the main trunk 2.6
- Linux is simple to handle if you know what to do.
- As soon as Linux is running a world of possibilities opens!
More about Linux and drivers

- **Understanding the Linux Kernel,**
  - O’REILLY, Daniel P. Bovet & Marco Cesati
- **Building Embedded Linux Systems,**
  - O’REILLY, Karim Yaghmour
- **Linux Device Driver, 3rd edition,**
  - O’REILLY, Jonathan Corbet, Alessandro Rubini & Greg Kroah-Hartman
  [http://lwn.net/Kernel/LDD3/](http://lwn.net/Kernel/LDD3/)
- **Linux Device Driver Template/Skeleton with Interrupt Handler and Device Read Blocking (Kernel Module Example)**

SystemC
Open SystemC Initiative (OSCI)

- Independent, not-for-profit association
- Open industry standard for system-level modeling, design and verification
- SystemC approved by IEEE Standards Association as IEEE Std, 1666-2005
- OSCI launched in 1999 for defining a language for electronic system-level design (ESL) using standard C/C++
- Purpose is to use a single model for the system-to-silicon design flow
  - Rich system-level language “SystemC”
  - Reuse of IP, tools and methods based on “SystemC”
  - Growth of the SystemC community

SystemC User Group Survey Trends

- World-wide SystemC Usage Growth

Production Project Design Experience:
15% > 10 designs
38% 1 to 9 designs
31% next design

Using SystemC
Under consideration
Not planned
Why use SystemC for model verification?

- **Ultra light-weight cycle-based simulation** kernel high-speed

- **Multiple abstraction levels**
  - high-level functional models
  - detailed clock cycle accurate RTL models

- **Iterative refinement of high level models into lower levels of abstraction**

- **Reuse of SystemC test bench for verification** on more levels can help to reduce the verification gap

### The verification gap

- Brian Bailey, Chief Technologist, Design Verification and Test Division, Mentor Graphics Corp.
SystemC Language Architecture

Methodology-specific channels

Elementary Channels
Signal, Timer, Mutex, Semaphore, FIFO, etc.

- Time
- Concurrency
- Modules
- Processes
- Interfaces
- Ports
- Channels
- Events
- Event-driven sim. kernel

Data Types
- 4-valued logic (0, 1, X, Z)
- 4-valued logic-vectors
- Bits and bit-vectors
- Arbitrary-precision integers
- Fixed-point numbers
- C++ user-defined types
- C++ built-in types (int, char...)

C++ Language Standard
**Why use SystemC?**

- Main purpose is **description and simulation** of larger systems where software and hardware coexist on equal terms — like FPGA with embedded processor.

- SystemC is strong for fast **prototyping and verification**, Reuse of SystemC test bench on different levels.

- SystemC **saves a lot of time** wasted on constant translations between C descriptions and HDL.

- You can get **started with SystemC for free**.
  - (SystemC synthesis for only $2995)
What is SystemC?

- C++ class library and a methodology
  - Create a cycle-accurate model of software algorithms,
  - Hardware architecture
  - Interfaces of a SoC (System On a Chip) on FPGA’s
  - System-level designs

- Use SystemC and other C++ development tools
  - To create a system-level model
  - Simulate to validate and optimize the design
  - Explore various algorithms
  - Provide an executable specification of the system

- An executable specification is a C++ program that is executed with the same behavior as the final system.

SystemC Basis #1 (Module, Process)

- Description of hardware, software, and interfaces in a C++ environment. Concurrent simulation kernel.

- Modules
  - Hierarchical entity
  - Can have other modules or processes contained

- Processes (Thread, Method)
  - Describe functionality and are contained inside modules
  - Methods must never suspend. They are called repeatedly on dynamic or static sensitivity.
  - (Like Verilog always@ and VHDL process)
**SystemC Basis #2** (Port, Channel)

- **Ports** *(In, Out)*
  - Modules have ports through which
  - they connect together using channels
  - Single-direction and bidirectional ports

- **Exports**
  - Exports the channel inside a module and use the
  - exported port externally as though it were a channel.

- **Channels** *(Signals, Fifo, Buffer)*
  - Resolved signals have more than one driver (bus)
  - Unresolved signals can have only one driver
  - Fifo's and buffers for higher level of simulation
  - Mutex and semaphores to share common resources

---

**SystemC Basis #3** (Data types, Clocks, Events)

- **Rich set of data types**
  - Native C types and sc_*_. Four state logic types.
  - Fixed precision data types allow for fast simulation,
  - Fixed-point data types can be used for DSP applications
  - Two-valued and four-valued data types.

- **Clocks**
  - Motion of clocks (as special signals)
  - Multiple clocks, with arbitrary phase relationship, supported.

- **Events**
  - Event is something that happens at a specific point in time
  - (no value and no duration)
SystemC Language UML view

SystemC test bench (Class diagram)
ModelSim Kernel and SystemC (V6.0<)

With ModelSim 6.3 it is possible to mix between VHDL, Verilog and SystemC

SystemC test bench with ModelSim
More about SystemC

- Guide for Getting Started with SystemC development
  - [http://www.teknologisk.dk/gswsystemc](http://www.teknologisk.dk/gswsystemc)
- Learning more about the SystemC language
- A SystemC Primer, *(Hardware view)*
  - J. Bhasker
- SystemC: From the Ground Up, *(System view)*
  - David C. Black and Jack Donovan
- Transaction-Level Modeling with SystemC,
  - Frank Ghenassia
- Teknologisk Institut
  - Fundamentals of SystemC - Doulos 5 days course