Course Information

Web page: http://www.ida.liu.se/~TDTS07
http://www.ida.liu.se/~TDDI08

Examination: written

Lecture notes: available from the web page, latest 24 hours before the lecture.

Recommended literature:


Course Information (cont’d)

Labs&Lessons:

Adrian Lifa
Institutionen för Datavetenskap (IDA)
email: adrl@ida.liu.se
http://www.ida.liu.se/~adrl
phone: 28 1763
B building, 329:206

Embedded Systems and Their Design

1. What is an Embedded System
2. Characteristics of Embedded Applications
3. The Traditional Design Flow
4. An Example
5. A New Design Flow
6. The System Level
7. Course Topics
That’s how we use microprocessors

General purpose systems
Embedded systems

What is an Embedded System?
There are several definitions around!

☞ Some highlight what it is (not) used for:

“An embedded system is any sort of device which includes a programmable component but itself is not intended to be a general purpose computer.”

☞ Some focus on what it is built from:

“An embedded system is a collection of programmable parts surrounded by ASICs and other standard components, that interact continuously with an environment through sensors and actuators.”

What is an Embedded System? (cont’d)

Some of the main characteristics:

- Dedicated (not general purpose)
- Contains a programmable component
- Interacts (continuously) with the environment

A Look at Two Typical Implementation Architectures

Telecommunication System on Chip

Programmable processor
ASIC block (Application Specific Integrated Circuit)
Standard block
Memory
Reconfigurable logic (FPGA)
Distributed Embedded System (automotive application)

The Software Component

Software running on the programmable processors:

- Application tasks
- Real-Time Operating System
- I/O drivers, Network protocols, Middleware

Characteristics of Embedded Applications

What makes them special?

- Like with “ordinary” applications, functionality and user interfaces are often very complex.

But, in addition to this:

- Time constraints
- Power constraints
- Cost constraints
- Safety
- Time to market

Characteristics of Embedded Applications (cont’d)

☞ Time constraints:

Embedded systems have to perform in real-time: if data is not ready by a certain deadline, the system fails to perform correctly.

- Hard deadline: failure to meet leads to major hazards.
- Soft deadline: failure to meet can be tolerated but quality of service is reduced.
Characteristics of Embedded Applications (cont’d)

- **Power constraints:**
  There are several reasons why low power/energy consumption is required:
  - **Cost aspects:**
    High power consumption ⇒ strong power supply, expensive cooling system
  - **Battery life**
    High energy consumption ⇒ short battery life

- **Cost constraints:**
  Embedded systems are very often mass products in highly competitive markets and have to be shipped at a low cost.
  What we are interested in:
  - Manufacturing cost
  - Design cost
  Factors:
    - design time, type of components used (processor, memory, I/O devices), technology, testing time, power consumption, etc.
  - Non-recurring engineering (NRE) costs (such as design cost, masks, prototypes) are becoming very high
    - It is very difficult to come out with low quantity products;
    - Hardware and software platforms are introduced which are used for several products in a family;

- **Safety critical:**
  Embedded systems are often used in life critical applications: avionics, automotive electronics, nuclear plants, medical applications, military applications, etc.
  - Reliability and safety are major requirements.
  In order to guarantee safety during design:
    - Formal verification: mathematics-based methods to verify certain properties of the designed system.
    - Automatic synthesis: certain design steps are automatically performed by design tools.

- **Short time to market:**
  In highly competitive markets it is critical to catch the market window; a short delay with the product on the market can have catastrophic financial consequences (even if the quality of the product is excellent).
  - Design time has to be reduced!
    - Good design methodologies.
    - Efficient design tools.
    - Reuse of previously designed and verified (hardware and software) blocks.
    - Good designers who understand both software and hardware!
Why is Design of Embedded Systems Difficult?

- High Complexity
- Strong time and power constraints
- Low cost
- Short time to market
- Safety critical systems

☞ In order to achieve all these requirements, systems have to be highly optimized.

Both hardware and software aspects have to be considered simultaneously!

An Example

The system to be implemented is modelled as a task graph:
- a node represents a task (a unit of functionality activated as response to a certain input and which generates a certain output).
- an edge represents a precedence constraint and data dependency between two tasks.

Period: 42 time units
- The task graph is activated every 42 time units ⇒ one activation has to be terminated in time less than 42.

Cost limit: 8
- The total cost of the implemented system has to be less than 8.

The Traditional Design Flow

It works like this (or even worse):

1. Start from some informal specification of functionality and a set of constraints (time and power constraints, cost limits, etc.)
2. Generate a more formal model of the functionality, based on some modeling concept (finite state machine, data-flow, etc.).
   This model is in Matlab, Statecharts, C, UML, VHDL. Such a model is our task graph (slide 18).
3. Simulate the model in order to check the functionality. If needed make adjustments.
4. Choose an architecture (µprocessor, buses, etc.) such that the cost limits are satisfied and, you hope, that time and power constraints will be fulfilled.
5. Build a prototype and implement the system.
6. Verify the system: neither time nor power constraints are satisfied!!!

☞ Now you are in great trouble: you have spent a lot of time and money and nothing works!
   - Go back to 4 and choose another architecture and start a new implementation.
   - Or negotiate with the customer on the constraints.

The Traditional Design Flow (cont’d)
The Traditional Design Flow (cont’d)

What are the consequences:

- Delays in the design process
  - Increased design cost
  - Delays in time to market → missed market window
- High cost of failed prototypes
- Bad design decisions taken under time pressure
  - Low quality, high cost products

Example

Let’s come back to the example on slide 18.

- We have the system model (task graph) which has been validated by simulation. What next?
- We decide on a certain processor μp1, with cost 4.
- For each task the worst case execution time (WCET) when executed on μp1 is estimated.

Example (cont’d)

Using this architecture we got a solution with:

- Execution time: 58 > 42
- Cost: 4 < 8
- We have to try with another architecture!
Example (cont'd)

☞ We look after a µprocessor which is fast enough: µp2.
☞ For each task the WCET, when executed on µp2, is estimated.

Using this architecture we got a solution with:

- Execution time: 28 < 42
- Cost: 15 > 8
☞ We have to try with another architecture!

Example (cont'd)

☞ Now we have to look to a multiprocessor solution.
In order to meet cost constraints we try two cheap (and slow) µps:
µp3: cost 3
µp4: cost 2
interconnection bus: cost 1
☞ For each task the WCET, when executed on µp3 and µp4, is estimated.

Example (cont'd)

☞ Now we have to map the tasks to processors.
µp3: T1, T3, T5, T6, T7, T8,
µp4: T2, T4.
☞ If communicating tasks are mapped to different processors, they have to communicate over the bus. Communication time has to be estimated; it depends on the amount of bits transferred between the tasks and on the speed of the bus.

Estimated communication times:
C_{1,2}: 1
C_{4,8}: 1

Example (cont'd)

☞ A scheduler:

We have exceeded the allowed execution time (42)!
Example (cont'd)

☞ Try a new mapping; move $T_5$ to $\mu_4$, in order to increase parallelism.

Two new communications are introduced, with estimated times:

- $C_{3,5}: 2$
- $C_{5,7}: 1$

☞ A schedule:

<table>
<thead>
<tr>
<th>Time</th>
<th>$\mu_3$</th>
<th>$\mu_4$</th>
<th>bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T_1</td>
<td>T_2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T_3</td>
<td>T_4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T_5</td>
<td>T_6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T_7</td>
<td>T_8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T_9</td>
<td>T_10</td>
<td></td>
</tr>
</tbody>
</table>

The execution time is still 62, as before!

Example (cont'd)

☞ There exists a better schedule!

Using this architecture we got a solution with:

- Execution time: 52 > 42
- Cost: 6 < 8

Example (cont'd)

☞ Possible solutions:

- Change $\mu$-processor $\mu_3$ with a faster one $\Rightarrow$ cost limits exceeded
- Implement some part of the functionality in hardware as an ASIC

Example (cont'd)

☞ A schedule:

Using this architecture we got a solution with:

- Execution time: 41 < 42
- Cost: 7 < 8
Example (cont'd)

What did we achieve?
- We have selected an architecture.
- We have mapped tasks to the processors and ASIC.
- We have elaborated a a schedule.

Extremely important!!!
Nothing has been built yet.
All decisions are based on simulation and estimation.

☞ Now we can go and do the software and hardware implementation,
with a high degree of confidence that we get a correct prototype.

The Design Flow

The Design Flow (cont'd)

What is the essential difference compared to the flow on slide 20?
- It is the inner loop which is performed before the effective hardware/software implementation.
  This loop is performed several times as part of the design space exploration. Different architectures, mappings and schedules are explored, before the actual implementation and prototyping.
- We get highly optimised good quality solutions in short time.
  We have a good chance that the outer loop, including prototyping, is not repeated.

Some additional remarks:
- **Formal verification**
  It is impossible to do an exhaustive verification by simulation!
  Especially for safety critical systems (but not only) formal verification is needed.
- **Simulation**
  Simulation is used not only for functional validation.
  It is used also after mapping and scheduling in order to test, for example, timing aspects.
  It is used also during the implementation steps; especially interesting: hardware/software cosimulation.
The Design Flow (cont'd)

- **Hardware/software codesign**
  
  During the mapping step we also decide what is going to be executed on a programmable processor (software) and what is going into hardware (ASIC, FPGA).

  During the implementation phase, hardware and software components have to be developed in a coordinated way, keeping their consistency (hardware/software cosimulation is important here).

The “Lower Levels”

- **Software generation**:
  - Encoding in an implementation language (C, C++, assembler).
  - Compiling (this can include particular optimizations for application specific processors, DSPs, etc.).
  - Generation of a real-time kernel or adapting to an existing operating system.
  - Testing and debugging (in the development environment).

- **Hardware synthesis**:
  - Encoding in a hardware description language (VHDL, Verilog).
  - Successive synthesis steps: high-level, register-transfer level, logic-level synthesis.
  - Testing and debugging (by simulation).

The “Lower Levels” (cont’d)

- **Hardware/software integration**:
  - The software is “run” together with the hardware model (cosimulation).

- **Prototyping**:
  - A prototype of the hardware is constructed and the software is executed on the target architecture.
The “Lower Levels” (cont’d)

There are available tools on the market which automatically perform many of the low level tasks:

- Code generators (software model → C, hardware model → VHDL)
- Compilers
- Test generators and debuggers
- Simulation and cosimulation tools
- Hardware synthesis tools
  - High level synthesis
  - RT-level synthesis
  - Logic synthesis
  - Layout and physical implementation

The System Level

- This is a hot research area.
- Very few commercial tools are offered.
- Mostly experimental and academic tools available.

Huge efforts and investments are currently made in order to develop tools and methodologies for system level design. Ad-hoc solutions are less and less acceptable.

It is the system level we are interested in, in this course!

Course Topics at a Glance

- Introduction: Embedded Systems and Their Design (just finished!)
- Models of Computation and Specification Languages
  - Dataflow Models, Petri Nets, Discrete Event Models, Synchronous Finite State Machines & Synchronous Languages, Globally Asynchronous Locally Synchronous Systems, Timed Automata, Hybrid Automata.
- Architectures and Platforms for Embedded Systems Design
- Task Scheduling
- System-Level Power/Energy Optimization