A Design Flow Based on a Domain Specific Language to Concurrent Development of Device Drivers and Device Controller Simulation Models

Edson Lisboa, Luciano Silva, Igino Chaves, Thiago Lima and Edna Barros

Email:{ebl2, lms4, imc, tav1, ensb}@cin.ufpe.br
Outline

• Introduction
• Problem Description
• Objective
• Design Flow
• DevC Language
• Results
• Conclusion
• Future works
Embedded Systems

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

• Electronic products integrate several functions;
• Several kinds of interface
  – Video, keyboards, usb, wireless, etc
• Complex systems

• Examples
  – Cel phones, MP3, MP4, Games
  – Palms, GPRs
  – Automotive systems
  – Industrial control
  – ...
Hardware/Software Components

Introduction
Problem
Objective
Design Flow
DevC Language
Results
Conclusion
Future works

Main Components

Great design effort
Embedded System Design

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- Platform Based Design (PBD)
- Components based on:
  - Simulation Models
- High level of abstraction
  - Languages: SystemC, Archc
  - Standards: TLM
- Simulation speed
- Flexibility

Virtual platform for software development

- Availability
  - Earlier development
- Scalability
  - Easy to modify
- Greater controllability/visibility
  - Easy to debug
- Must be efficient
Device model is not available.
Need to develop and test the device model.
Need to develop and test the driver.
Main Objective

- Methodology for the concurrent and incremental development of devices (focus on the controller) and respective device drivers.

- Mechanisms to specify devices

- Strategy to partially generate device controller models and device drivers for virtual platforms.
Proposed Design Flow
# DevC Descriptions - Features

<table>
<thead>
<tr>
<th>Spec</th>
<th>Device Features</th>
<th>Driver</th>
<th>SO Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Variables</td>
<td>Interface Specification</td>
<td></td>
</tr>
<tr>
<td>Parametrized Services</td>
<td>Sequencing</td>
<td>Connection to Services</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>Verifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E/S Techniques</td>
<td>Interruption</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DevC Descriptions

- Functional description of a device
  - Declare ports;
  - Specify parameters and services;
  - Map services on addresses.

- Example: Simple_IO
  - Print a character;
  - Read a character;
  - Specifications:
    - Services: print, read;
    - Data: 8 bits;
    - Signed.
DevC Description: Refining the Device Specification

- Real world devices use registers for information exchange.
  - Data, control, status.

- Specification of structural elements.
  - Registers formatting (fields).
  - Registers and access rules.
  - Mapping of registers.

DevC(display) {

  dc_scml_port pv;
  dc_format p_type = "%data:8:s";
  dc_service <p_type, WRITE> print;
  dc_format c_type = "%RESERVED:7 %DE:1";
  dc_reg <c_type, WRITE> ctrl;

  DEV_CTOR{
    print.set_address(0);
    ctrl.set_address(1);
  }
}
DevC Description: Refining the Device Specification

- Defining the driver variables according to the registers.
- Specifying the conditions to access the services.
- Specifying assignment.

```cpp
DevC(display){

dc_scml_port pv;

dc_format p_type = "%data:8:s %data1:8:s";
dc_service <p_type, WRITE> print;

dc_format c_type = "%RESERVED:7 %DE:1";
dc_reg <c_type, WRITE> ctrl;

dc_drv_map var{
    DE = ctrl.DE;
};

DEV_CTOR{
    print.set_action(pre){ ctrl.DE == 1;
        var.DE = 1; }
    print.set_address(0);
    ctrl.set_address(1);
}
}
DevC Description: Refining the Device Specification

- Specifying buffers.
- Specifying pooling as I/O technique.
- Defining registers and their connection to the services.
Generating the Controller from DevC

DevC Language

- DevC

```c
DEVC(display){
    dc_scml_port pv;

    dc_format p_type = "%data:8:s";
    dc_service <p_type, WRITE> print;
    dc_buffer buf:512:8;
    dc_format d_type = "%data:8";
    dc_reg <d_type> data;
    dc_format c_type = "%RESERVED:7 %RD:1";
    dc_reg <c_type, READ> status;

    dc_drv_map var{
        RD = status.RD;
        VDATA = data
    };
    DEVCTOR{
        print.set_action(pre){ polling(var.RD == 1); }
        data.set_address(0);
        print.bindTo(data);
        print.bindTo(VDATA);
        status.set_address(1);
    }
}
```

- Controller

![Diagram of controller and DevC code](image)
Generating the Driver from DevC Description

DevC Language

- DevC

DEVC(display){
  dc_scml_port pv;

  dc_format p_type = "%data:8:s";
  dc_service <s_type, WRITE> print;

  dc_format d_type = "%data:8";
  dc_reg <d_type> data;
  dc_format s_type = "%RESERVED:7 %RD:1";
  dc_reg <c_type, READ> status;

  dc_drv_map var{
    RD = status.RD;
    VDATA = data;
  };

  DEVCTOR{
    print.set_action(pre){ polling(var.RD == 1); };
    data.set_address(0);
    print.bindTo(data);
    status.set_address(1);
  };
};

• DevC

Driver

• Driver

Services

inline void display_print(char data);
inline void setVDATA(char data);
inline uchar getRD();
inline void outb(uint addr, char value);
inline char inb(uint addr);

HAL

Driver Kernel

• Driver Kernel

inline void display_print(char data);
inline void setVDATA(char data);
inline uchar getRD();
inline void outb(uint addr, char value);
inline char inb(uint addr);
Generating the Driver from DevC Description

**Introduction**

**Problem**

**Objective**

**Design Flow**

**DevC Language**

**Results**

**Conclusion**

**Future works**

```c
DEV_C(display){
    dc_scml_port pv;

    dc_format p_type = "%data:8:s %data1:8:s";
    dc_service <s_type, WRITE> print;

    dc_format c_type = "%RESERVED:7 %RD:1";
    dc_reg <c_type, READ> status;

    DEVCTOR{
        VDATA.set_action(pre){
            interrupt(status.RD == 1); }

        status.set_address(1);
    }
```

**Services**

- void display_intr_handler(int intr);
- inline void display_print_ctrl(char data);
- inline void outb(uint addr, char value);
- inline char inb(uint addr);
- void register_handler(int ID, void (*handler)(int));

**Driver Kernel**

**HAL**

```c
dc_scml_port pv;
```

```c
dc_format p_type = "%data:8:s %data1:8:s";
```

```c
dc_service <s_type, WRITE> print;
```

```c
dc_format c_type = "%RESERVED:7 %RD:1";
```

```c
dc_reg <c_type, READ> status;
```

```c
DEVCTOR{
    VDATA.set_action(pre){
        interrupt(status.RD == 1); }

    status.set_address(1);
}
## Results

### DevC Description

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Model</th>
<th>Number of Services</th>
<th>DevC Description</th>
<th>Code</th>
<th>Behaviour</th>
<th>Total</th>
<th>Perc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uart</td>
<td>Funcional</td>
<td>3</td>
<td>14</td>
<td>72</td>
<td>16</td>
<td>88</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Estrutural</td>
<td>3</td>
<td>21</td>
<td>100</td>
<td>15</td>
<td>115</td>
<td>95</td>
</tr>
<tr>
<td>LCD</td>
<td>Funcional</td>
<td>9</td>
<td>28</td>
<td>85</td>
<td>125</td>
<td>210</td>
<td>88</td>
</tr>
<tr>
<td>IP-Select-Map</td>
<td>Estrutural</td>
<td>2</td>
<td>21</td>
<td>41</td>
<td>10</td>
<td>51</td>
<td>87</td>
</tr>
</tbody>
</table>

### Simulation Results for the UART

<table>
<thead>
<tr>
<th>Scenery</th>
<th>Access to the Device</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without SO</td>
<td>With SO</td>
<td>Direct</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- The proposed methodology includes
  - Specification mechanisms
  - Strategy for automatic generation of the controller model and the driver.

- DevC allows abstracting details about the implementation of the driver and of the device controller, making development easier.
Conclusions

• The proposed approach allows an effective support to platform based design because it allows an incremental and concurrent flow for developing device drivers;
  – Earlier error detection – more robust drivers;
  – Reduced development time.

• LCD case study demonstrated that the proposed methodology is valid.

• Synthesized driver was validated with a controller implemented on hardware and it has a similar performance to the manually produced driver.
### Future Works

- Specify more complex devices (as a USB controller);
- Provide support to more complex data structures on drivers;
- Provide support to other embedded operating systems;
- Validate monitors generating traces of the access to the device.
- Integration on the *Pdesigner* Framework.