



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

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

Email: {ebl2, lms4, imc, tavl, 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

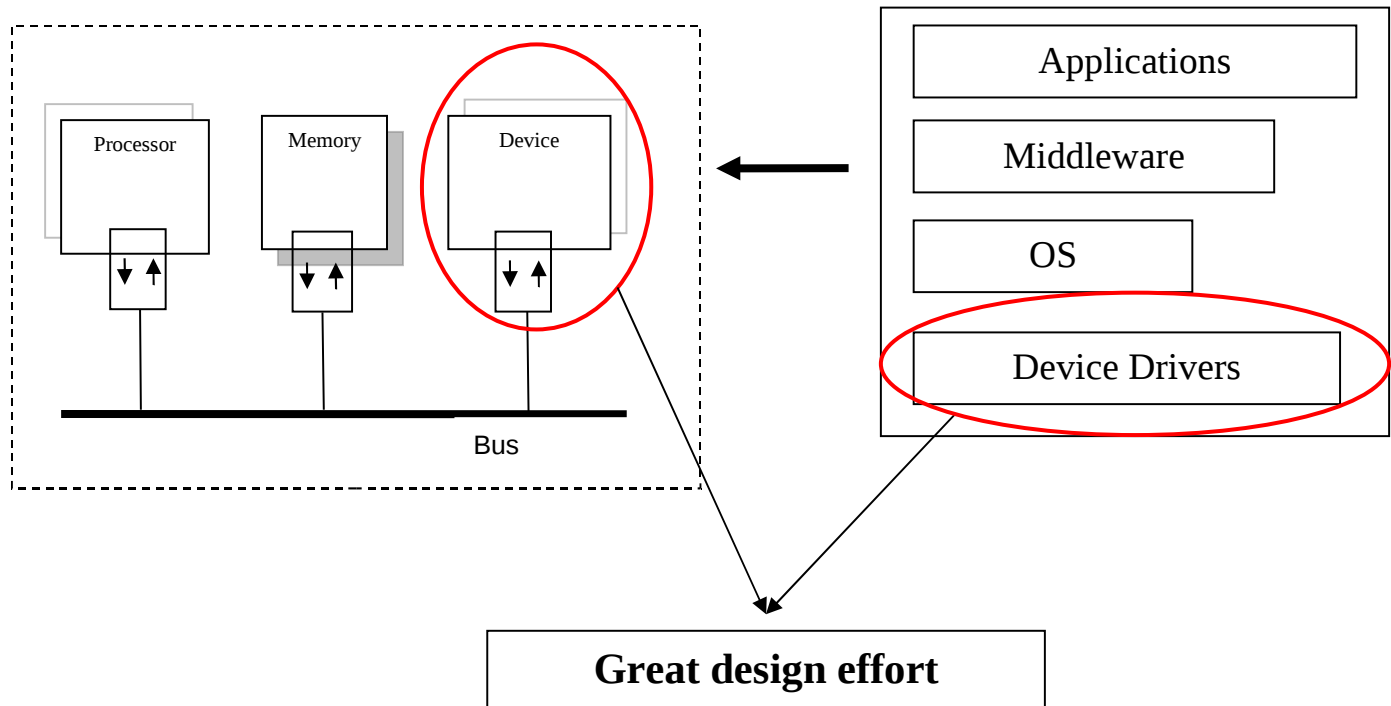
DevC Language

Results

Conclusion

Future works

Main Components



Embedded System Design

Introduction

Problem

Objective

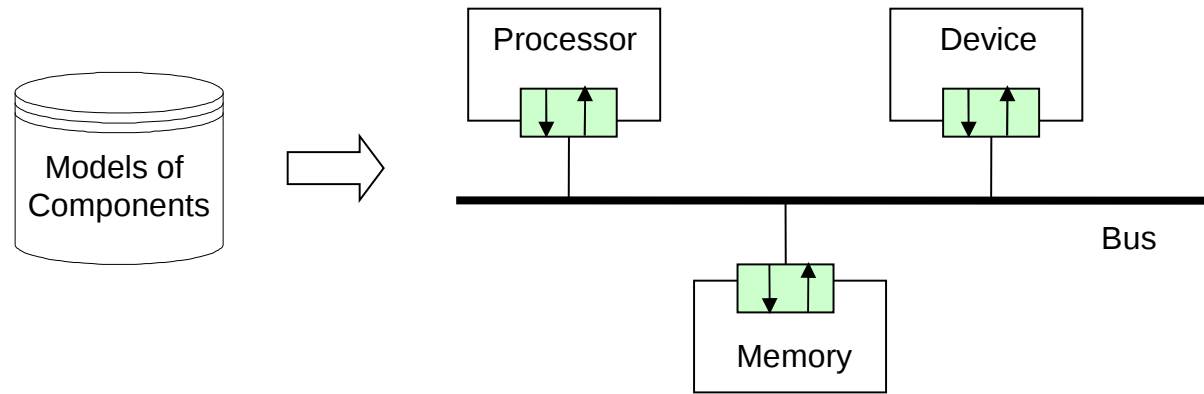
Design Flow

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

Problem

Introduction

Problem

Objective

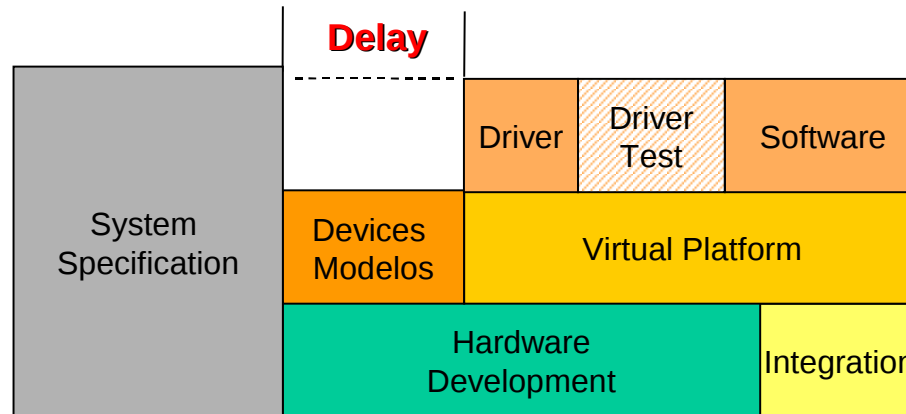
Design Flow

DevC Language

Results

Conclusion

Future works



- Device model is not available.
- Need to develop and test the device model.
- Need to develop and test the driver.

Main Objective

Introduction

Problem

Objective

Design Flow

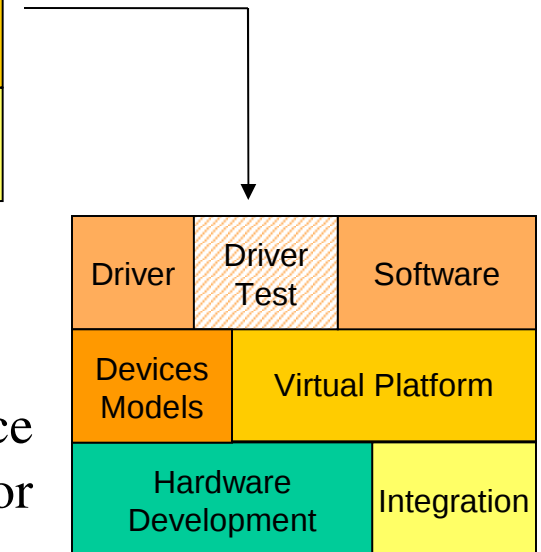
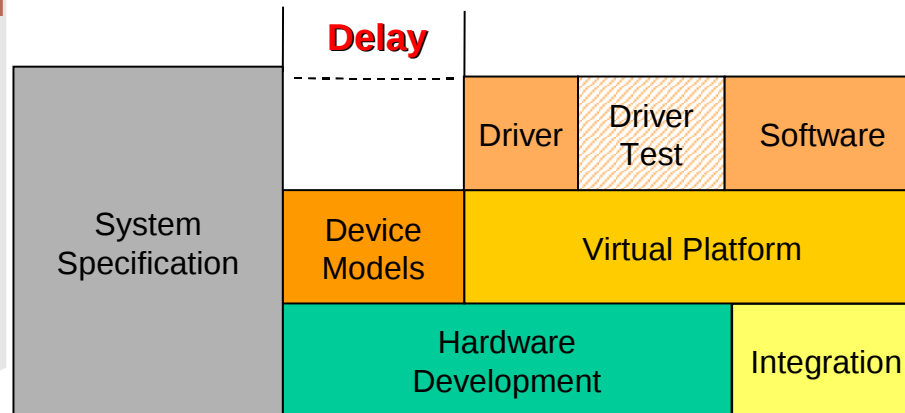
DevC Language

Results

Conclusion

Future works

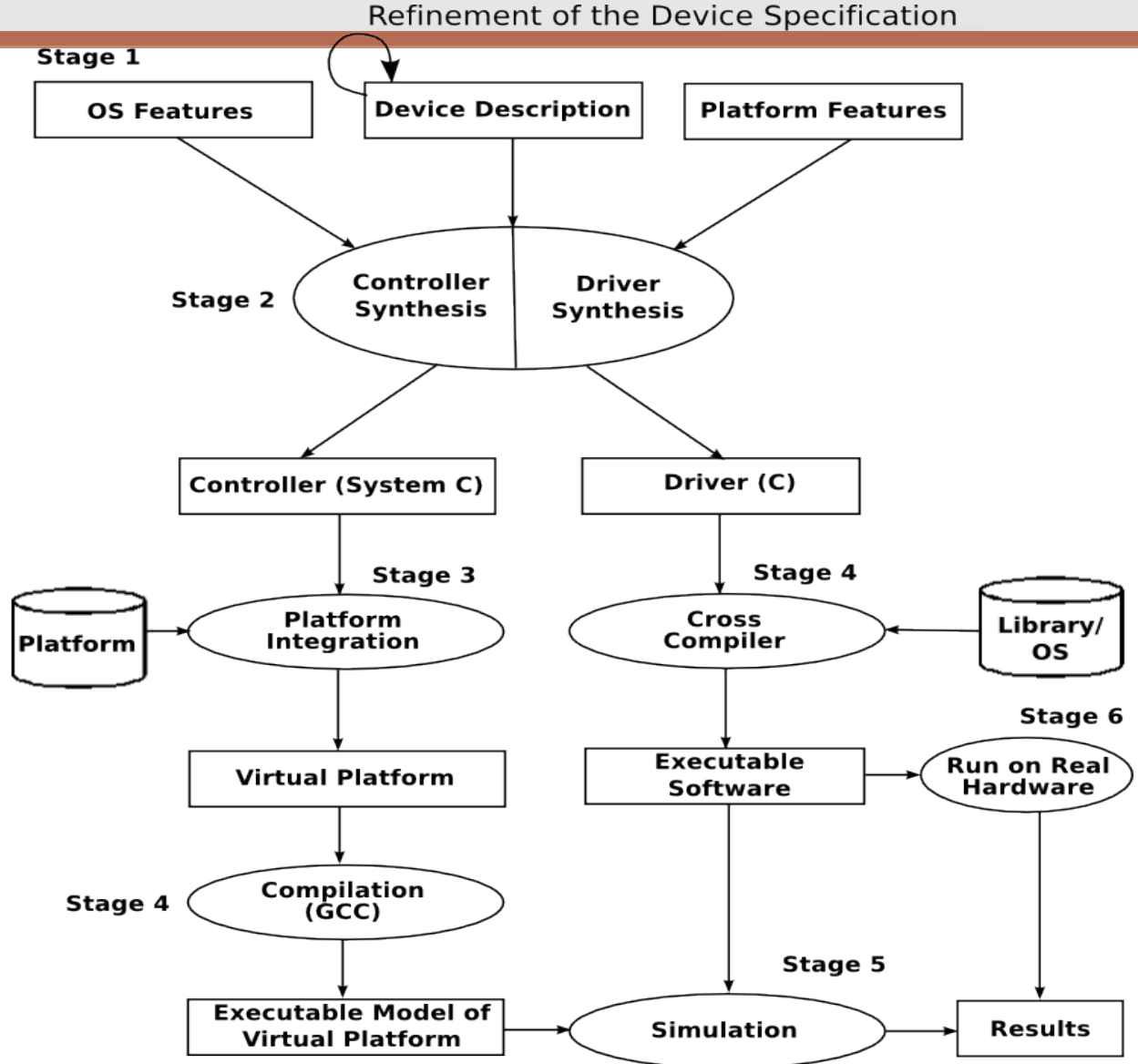
- Methodology for the concurrent and incremental development of devices (focus on the controller) and respective device drivers.



- Mechanisms to specify devices
- Strategy to partially generate device controller models and device drivers for virtual platforms.

Proposed Design Flow

- Introduction
- Problem
- Objective
- Design Flow**
- DevC Language
- Results
- Conclusion
- Future works



DevC Descriptions - Features

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

Specifications		
Device Features	Driver	SO Features
Ports	Variables	Interface Specification
Parametrized Services	Sequencing	Connection to Services
Storage	Verifications	
E/S Techniques	Interruption	

DevC Descriptions

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- 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(Simple_IO){  
    dc_scml_port pv;  
    dc_format p_type = "%data:8:s";  
    dc_format r_type = "%:r:8";  
    dc_service <p_type, WRITE> print;  
    dc_service <r_type, READ> read;  
    DEV_CTOR{  
        print.set_address(0);  
        read.set_address(1);  
    }  
}
```

DevC Description: Refining the Device Specification

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- 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

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

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

```
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

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- Specifying buffers.
- Especificing pooling as I/O technique.
- Defining registers and their connection to the services.

```
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 %RD:1";
    dc_format d_type = "%data:8";
    dc_reg <d_type, WRITE> data;
    dc_reg <c_type, READ> status;
    dc_buffer buf:512:8;
    dc_drv_map var{
        RD = status.RD;
        VDATA = data;
    };

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

Generating the Controller from DevC Description

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

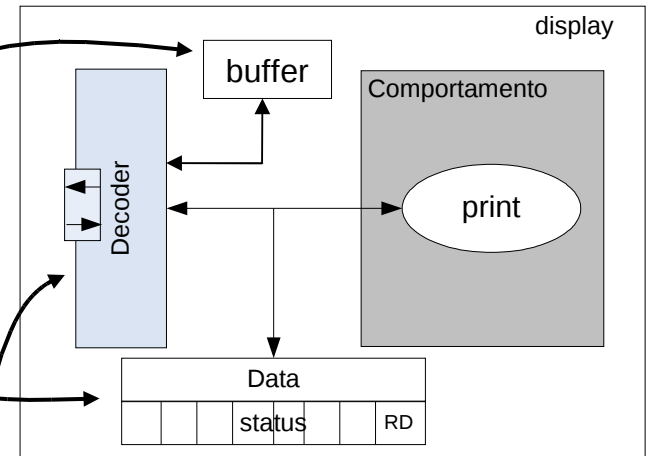
- DevC

```
DEVC(display){
  dc_scm1_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
  };
  DEV_CTOR{
    print.set_action(pre){ polling(var.RD == 1); };
    data.set_address(0);
    print.bindTo(data);
    print.bindTo(VDATA);
    status.set_address(1);
  };
};
```

- Controller



Generating the Driver from DevC Description

- Introduction
- Problem
- Objective
- Design Flow
- DevC Language**
- Results
- Conclusion
- Future works

- DevC

```
DEVC(display){
    dc_scm1_port pv;

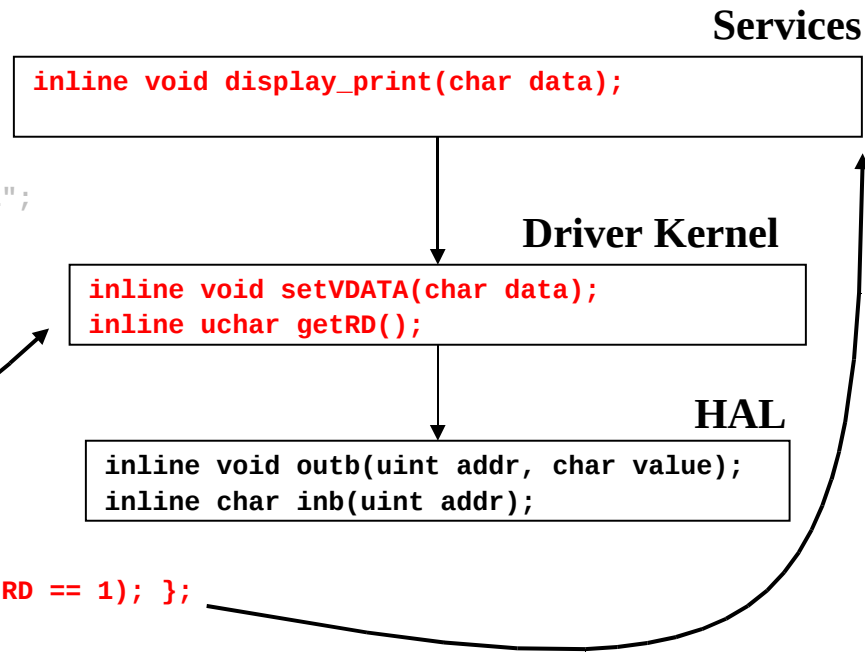
    dc_format p_type = "%data:8:s";
    dc_service <s_type, WRITE> print;

    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;
    };

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

- Driver



Generating the Driver from DevC Description

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

```
DEVC(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;
```

```
DEV_CTOR{
```

```
    VDATA.set_action(pre){
```

```
        interrupt(status.RD == 1); }
```

```
    status.set_address(1);
```

```
};
```

```
};
```

Services

```
void display_intr_handler(int intr);
```

Driver Kernel

```
inline void display_print_ctrl(char data);
```

HAL

```
inline void outb(uint addr, char value);
```

```
inline char inb(uint addr);
```

```
void register_handler(int ID, void (*handler) (int));
```

Results

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

DevC Description

Device Name	Model	Number of Services	DevC Description	Code		Behaviour		Total		Perc (%)	
				Dev	Drv	Dev	Drv	Dev	Drv	Dev	Drv
Uart	Funcional	3	14	72	36	16	2	88	38	81	94
	Estrutural	3	21	100	58	15	3	115	61	86	95
LCD	Funcional	9	28	85	76	125	16	210	86	40	88
IP-Select-Map	Estrutural	2	21	41	42	10	6	51	46	80	87

Simulation Results for the UART

Scenary		Access to the Device		Simulation		
Without SO	With SO	Direct	Via SO	Time	Number of Instructions	Speed
X		X		0.52	38738	74.5 k ins/s
	X	X		15.50	1193070	77.07 k ins/s
	X		X	26.12	1865190	71.68 k ins/s

Hardware prototype

Introduction

Problem

Objective

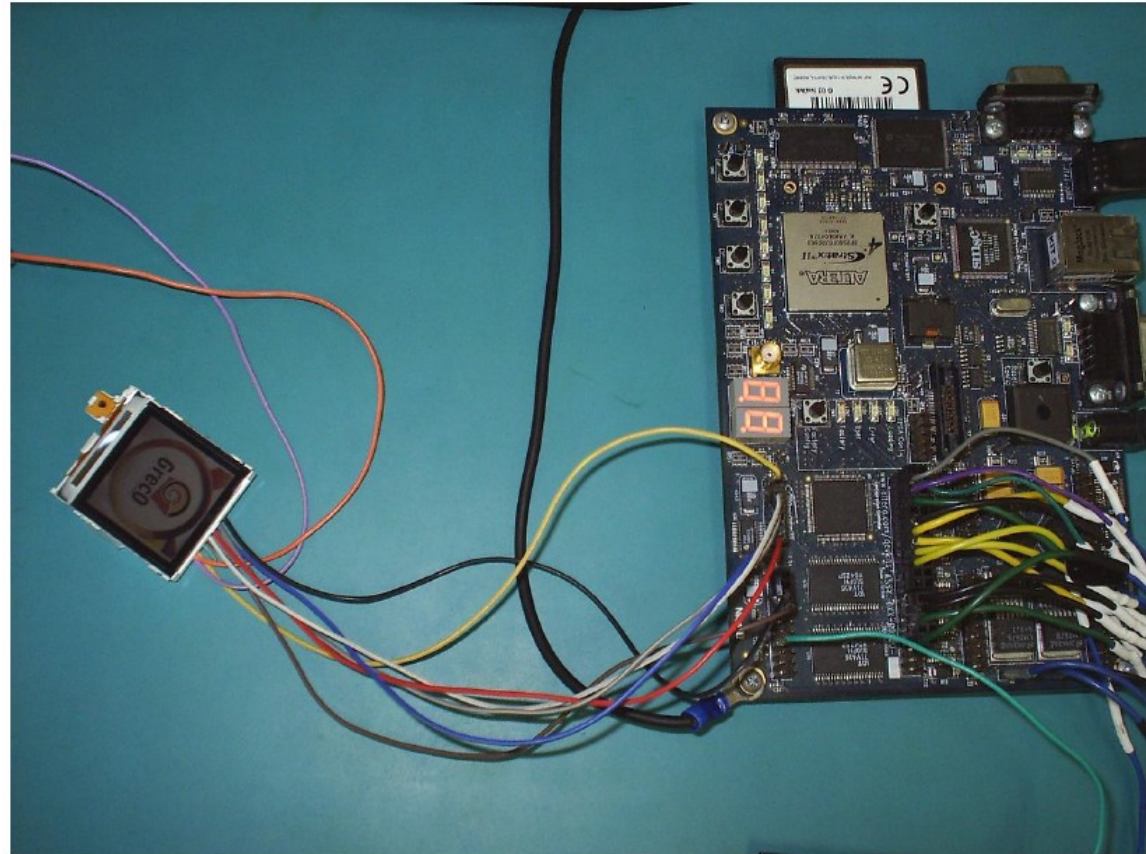
Design Flow

DevC Language

Results

Conclusion

Future works



Conclusions

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- 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

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

Future works

- 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 e it has a similar performance to the manually produced driver.

Future Works

Introduction

Problem

Objective

Design Flow

DevC Language

Results

Conclusion

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.