

Programming embedded systems

Nicholas Wickström

IDE, Högskolan i Halmstad

2009

Outline

- 1 Introduction
 - Goal
 - Background
 - Issues
 - Tools

- 2 C

Outline

- 1 Introduction
 - Goal
 - Background
 - Issues
 - Tools

- 2 C

Goal

Program embedded systems

- Details
- Principles and techniques

Goal

Program embedded systems

- Details
- Principles and techniques

Goal

Program embedded systems

- Details
- Principles and techniques

Background

- 98% of computing devices are embedded in all kinds of electronic equipment and machines
- Over 4 billion embedded processors were sold last year and the global market is worth 60 billion Euro with annual growth rates of 14%. Forecasts predict more than 16 billion embedded devices by 2010 and over 40 billion by 2020.
- they all need code!

Background

- 98% of computing devices are embedded in all kinds of electronic equipment and machines
- Over 4 billion embedded processors were sold last year and the global market is worth 60 billion Euro with annual growth rates of 14%. Forecasts predict more than 16 billion embedded devices by 2010 and over 40 billion by 2020.
- they all need code!

Background

- 98% of computing devices are embedded in all kinds of electronic equipment and machines
- Over 4 billion embedded processors were sold last year and the global market is worth 60 billion Euro with annual growth rates of 14%. Forecasts predict more than 16 billion embedded devices by 2010 and over 40 billion by 2020.
- they all need code!

Diverging trends

Diverging trends:

- More powerful nodes, OS (eg. cell phones)
- Less powerful nodes, Distributed systems (eg. automobiles)

Diverging trends

Diverging trends:

- More powerful nodes, OS (eg. cell phones)
- Less powerful nodes, Distributed systems (eg. automobiles)

Embedded systems programming

- Sometimes no OS
- Hardware, 1D vs. 2D

Embedded systems programming

- Sometimes no OS
- Hardware, 1D vs. 2D

Issues

- Reentrancy
- Portability
- Dynamic memory

Issues

- Reentrancy
- Portability
- Dynamic memory

Issues

- Reentrancy
- Portability
- Dynamic memory

Tools

- Lint
- IDE
- Code Profiling
- Documentation

Tools

- Lint
- IDE
- Code Profiling
- Documentation

Tools

- Lint
- IDE
- Code Profiling
- Documentation

Tools

- Lint
- IDE
- Code Profiling
- Documentation

Outline

- 1 Introduction
 - Goal
 - Background
 - Issues
 - Tools

- 2 C

C

C is the most common programming language for embedded systems

C

- Pros and Cons
- Call by value
- Structure of a C program

C

- Pros and Cons
- Call by value
- Structure of a C program

C

- Pros and Cons
- Call by value
- Structure of a C program

Def. Pointer

A pointer is a variable containing the adress to another variable, function or adress.

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Def. Pointer

- Simple example
- Pointer arithmetics
- Vector
- Pointers to pointers
- Pointers in function calls
- Pointers to functions

Bit operators

- Set/reset a single bit
- Masks

Bit operators

- Set/reset a single bit
- Masks

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Data structures

- *struct*
- Pointers to structs
- Linked lists
- Trees
- Bit fields

Setting registers with C

- A pointer is a variable containing an address
- Prettifying, define and macros
- Hidding in data stuctures

Setting registers with C

- A pointer is a variable containing an address
- Prettifying, define and macros
- Hidding in data stuctures

Setting registers with C

- A pointer is a variable containing an address
- Prettifying, define and macros
- Hidding in data stuctures

direct way

```
*((volatile unsigned int *)0xFFFFC18) =  
0x00000040;
```

direct way (comments help some)

```
*((volatile unsigned int *)0xFFFFFC18) =  
0x00000040;  
/* enable usart0 peripheral clock */  
/* by setting bit 6 in PMC PCER*/  
/* Power Management Controller */  
/* Peripheral Clock Enable Register */  
/* see page 192 of the Data Sheet */
```

direct way (more readable)

```
#define PMC_PCERr 0xFFFFFC18  
#define PMC_PCER_USART0 0x00000040  
*((volatile unsigned int *)PMC_PCERr) =  
PMC_PCER_USART0;
```

direct way (perhaps even more readable)

```
#define PMC_PCERr 0xFFFF FC18  
#define PMC_PCER_USARTO 0x00000040  
#define PMC_PCER (*((volatile unsigned int *)PMC_PCERr))  
PMC_PCER = PMC_PCER_USARTO;
```

using data structures

```
#include "AT91SAM7S256.h"  
// pointer to PMC data structure  
volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;  
// enable usart0 peripheral clock  
pPMC->PMC_PCER = (1<<AT91C_ID_US0);
```

Tool chain

- Compiler
- Linker
- Relocation
- Optimization

Tool chain

- Compiler
- Linker
- Relocation
- Optimization

Tool chain

- Compiler
- Linker
- Relocation
- Optimization

Tool chain

- Compiler
- Linker
- Relocation
- Optimization