1. Embedded Systems Overview

1.1 Developing Embedded Software

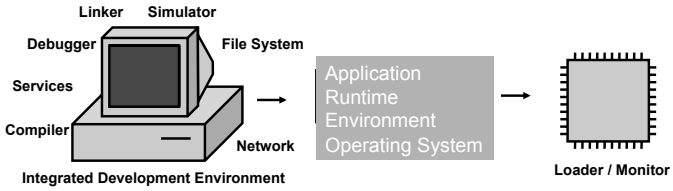
Roadmap for Section 1.1

- Cross-Platform Development
- Microcontroller Development Cycle
- Software Architecture & Control Loop
- The GNU Compiler Collection

Cross-Platform Development

Development Host ⟺ **Embedded Device**

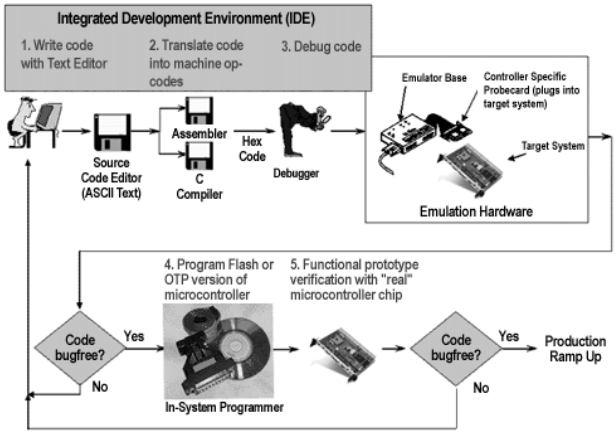
Connections
- Ethernet
- Serial
- BDM/JTAG



Embedded "Hello, World"

```
#include <avr/io.h>

char hello[22] = {1,1,1,1,5,1,3,5,1,3,1,1,5,1,3,1,1,5,3,3,3,5};

void wait(int msec)
{
    int i,j;
    for(i=0;i<10000;i++) for(j=0;j<1*msec;j++);
}

void blink(int time)
{
    int i;
    outp(0,PORTB);
    wait(time);
    outp(1,PORTB);
    wait(1);
}

void main(void)
{
    int i;
    outp(255,DDRB);
    while(1)
    {
        for(i=0;i<22;i++) blink(hello[i]);
    }
}
```

Microcontroller Development Cycle



Embedded System Development Development Techniques

- In-System-Programming
  - Programming in target system
  - Live updates via UART, SPI
- Starter Kits, Evaluation Boards
  - Typically contain In-System-Programmer
  - Sample Processor
  - Assembler, Compiler, Linker, Debugger, IDE
  - Sample Board with simple I/O facilities, connectors
  - AVR : 115€, Arm9 3000$

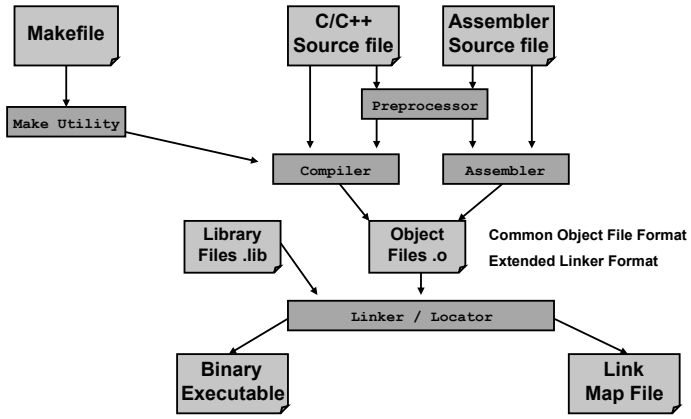
Embedded System Development Emulation, Simulation

- In-Circuit Emulation (ICE)
  - Replace CPU of target system
  - Real-Time Tracing
  - Very, very expensive (powerful host needed)
- Simulation
  - Simulation of complete instruction set
  - Interprets each instruction of target image
  - Mostly no real-time simulation
  - Simulation of external source by “stimulus files”
  - Very cheap, difficult to simulate timing, external hardware

Download and Debugging

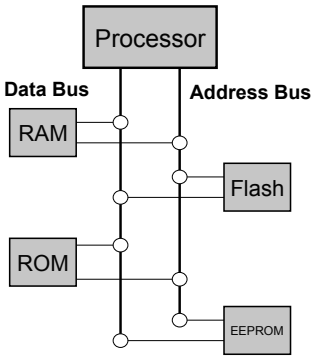
- Program ROM and insert into target system
- (Re-) Program Flash Memory on development boards
- Embedded Loader
  - Typically in ROM, small footprint
  - Downloads image from host system and executes it
- Embedded Monitor
  - Includes all Loader Functions
  - Debugging Facilities (read register, memory, single stepping, breakpoints)
- On-Chip Debugging / Hardware Debug
  - JTAG – Joint Test Action Group
  - BDM – background debug mode

Building Executable Image Files

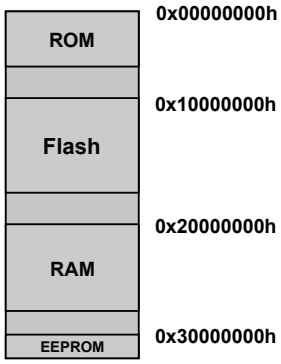


Mapping Executable Images to Target Systems

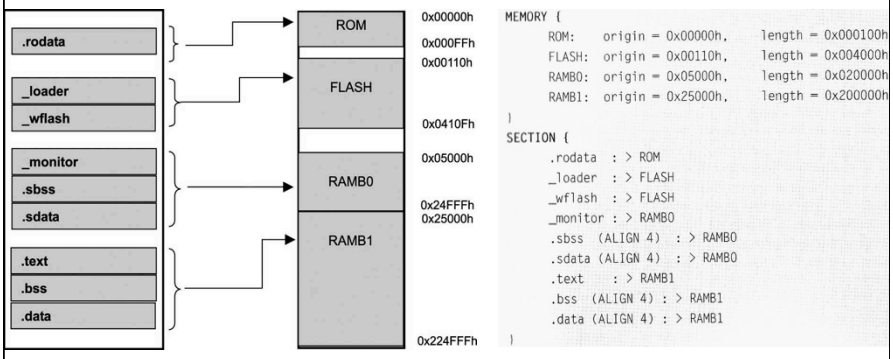
Schematic Target System

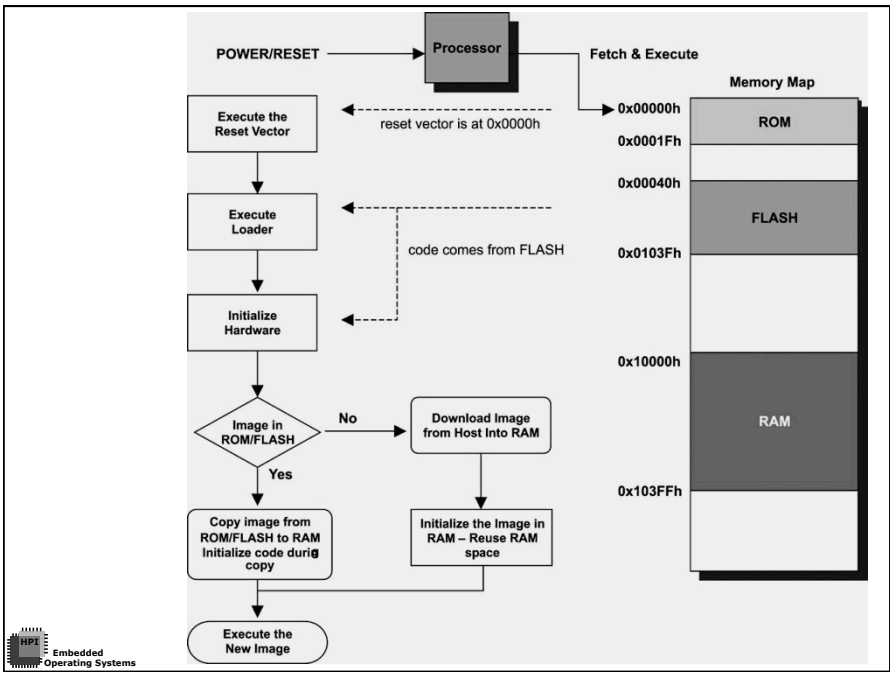


Memory Map



Mapping Executables



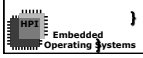


Hardware Initialization

- Start execution at reset vector
- Put processor into a known state
  - Set all registers to initial values
- Disable interrupts and caches
- Initialize memory system
  - Type, Controller, Size, Cache, Tests, Stack
- Load / Decompress code sections from ROM to RAM
- Set up initial interrupt and exception handler
- Initialize internal bus interfaces, peripherals
- Initialize software (RTOS, application)

Software Architectures
Simple Control Loop


```
void main(void)
{
   // read switch - in or out position
   switchPosition = // Read switch value
   while(TRUE)
   {
        switch(switchPosition)
        {
             case IN_POS:
                  temperature = // read int. sensor
                   break;
             case OUT_POS:
                  temperature = // read int. sensor
                   break;
        }
        UpdateDisplay();
   }
}
```



Software Architectures
Round Robin

```
void main(void)
{
   while(TRUE)
   {
        if(device A needs service)
               // Handle A -> 5ms
        if(device B needs service)
               // Handle B -> 5ms
        if(device C needs service)
               // Handle C -> 2ms
                              must be serviced all 8 ms
   }
}
```

- Advantage : Simplicity
- Problem No priorities, I/O overhead

Software Architectures
Round Robin with Interrupts

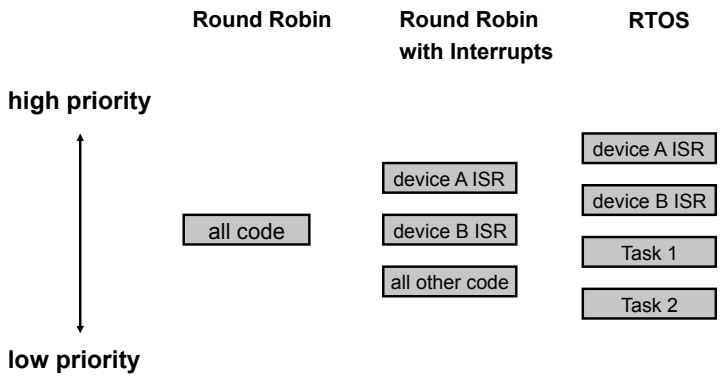
```
bool handleA,handleB;
void interrupt HandleDeviceA(){
   handleA = true;
}
void interrupt HandleDeviceB(){
   handleB = true;
}
void main(void)
{
   while(true)
   {
        if(handleA){
            handleA = false;
            // handle A }
         if(handleB){
            handleB = false;
            // handle B }
   }
}
```

Software Architectures
Real-Time Operating System

```
void interrupt HandleDeviceA(){
   // set signal X
}
void interrupt HandleDeviceB(){
   // set signal Y
}

void Task1(){
   while(true){
        // wait for signal X
        // handle device A
   }
}
```

```
void Task2(){
       while(true){
              // wait for signal Y
              // handle device B
       }
}
```

Software Architectures
Priorities



GNU Compiler Collection

- Multiple language frontends, for parsing many languages (C,C++,Ada, some Ecma-IL)
- Compile / Cross-compile for many architectures (x86, Sparc, Itanium, AVR, …)
- Cross Compiler Gcc binary : TARGETNAME-gcc
- Gcc invocation stages
  - preprocessing (to expand macros)
  - compilation (from source code to assembly language)
  - assembly (from assembly language to machine code)
  - linking (to create the final executable)

Embedded Operating Systems