1. Introduction to Embedded Systems

1.1 Definition

Definition for: Embedded System

A combination of hardware and software which together form a component of a larger machine. An example of an embedded system is a microprocessor that controls an automobile engine. An embedded system is designed to run on its own without human intervention, and may be required to respond to events in real time.

Source: [www.computeruser.com/resources/dictionary](http://www.computeruser.com/resources/dictionary)

1.2 Application Areas

Where in our daily life do we use embedded systems?

Consumer Products

- TV
- stereo
- remote control
- phone / mobile phone
- refrigerator
- microwave
- washing machine
- electric tooth brush
- oven / rice or bread cooker
- watch
- alarm clock

- electronic musical instruments
- electronic toys (stuffed animals, handheld toys, pinballs, etc.)
- medical home equipment (e.g. blood pressure, thermometer)
- ...

- [PDAs ??
  More like standard computer system]
Application Areas

- **Medical Systems**
  pace maker, patient monitoring systems, injection systems, intensive care units, …

- **Office Equipment**
  printer, copier, fax, …

- **Tools**
  multimeter, oscilloscope, line tester, GPS, …

- **Banking**
  ATMs, statement printers, …

- **Transportation (Planes/Trains/[Automobiles] and Boats)**
  radar, traffic lights, signalling systems, …

Application Areas

- **Automobiles**
  engine management, trip computer, cruise control, immobilizer, car alarm, airbag, ABS, ESP, …

- **Building Systems**
  elevator, heater, air conditioning, lighting, key card entries, locks, alarm systems, …

- **Agriculture**
  feeding systems, milking systems, …

- **Space**
  satellite systems, …

Facts:

- 1997: The average U.S. household has over 10 embedded computers (source: www.it.dtu.dk/~jan)
- 2001: The Volvo S80 has 18 embedded controllers and 2 busses (source: Volvo)

1.3 Performance of Embedded Systems

State of the Art for Embedded Controllers

System on chip – custom solution
(e.g. incorporating ARM processor kernel)
- Feasibility study $ 100,000
- Production setup up to $1,000,000

System on board (PCB)
- More economical for smaller production runs
- Custom or off-the-shelf
Performance of Embedded Systems

State of the Art for Embedded Controllers

- Embedded PC (incl. Ethernet, frame grabber, color LCD, CAN bus, 5/12V) about 50% standard PC performance
- Systems with extended temperature range about 25% standard PC performance
- Systems with ext. temp. and electromagnetic compatibility about 10% standard PC performance

1.4 Automobiles

How many embedded systems are in a modern car?

- trip computer (fuel cons., etc.)
- electronic ignition
- airbag
- immobiliser, keyless entry
- ABS
- ESP
- …

Automobiles

2002: Opel Vectra has over 40 sensors (25 types)

Autonomous cars:
- Electronic gas
- Electronic brake
- Electronic steering

See: The Daimler Story

Sensors: Stereo-cameras, speedometer, accelerometers, signaling

Actuators: gas, steering, brake

Embedded System CAN bus
1.5 Design Goals

- Reliability
  *Continuous operation, no reboot*
- Cost-effectiveness
  *Matching requirements, no over-design*
- Space
  *Limited memory, reduce program code and data*
- Energy
  *Limited energy consumption, select appropriate processor, sometimes even optimize software!*
- Performance
  *Use of assembly where necessary, multi-tasking*

1.6 Microcontrollers

- Microprocessor
- Microcontroller
- Data Representation
- von-Neumann Computer Architecture

![Diagram of CPU, Mem, I/O, Bus, DMA]

| CPU: Central Processing Unit |
| Mem: RAM or ROM Memory (for program and data) |
| I/O: Input/Output (e.g. keyboard, CRT screen, disk drive, CD-ROM, network, modem, etc.) |
Microcontrollers

CPU = \textit{ALU} + \textit{CU}

Components

- ALU: Arithmetic Logic Unit \textit{(Maths)}
- CU: Control Unit \textit{(Logic)}

Tasks

- Central clock \textit{(provide system timing)}
- Control bus access
- Execute program

1.7 Memory

Memory \rightarrow \text{ basically 2 types }

- ROM: "\textit{Read Only Memory}" \textit{(Read only)}
- RAM: "\textit{Random Access Memory}" \textit{(Read/Write)}

ROM

- Cannot be overwritten \textit{(contents is never changed)}
- Used for system boot
- Variety of different versions

RAM

- Can be changed any time
- Store programs or data
- Writing data to a location \textit{("address")(destroys old data contents}
- All RAM data is lost when power is turned off

- Static RAM
  - Static RAM keeps data as long as power is on

- Dynamic RAM
  - Dynamic RAM "leaks" memory contents over short time period, needs periodic "refresh" \rightarrow periodically read/write all memory locations
Address

- RAM/ROM contains a large number of memory cells per chip
- Size of memory cell is usually 1 Byte = 8 bits
- Size of RAM, e.g. 128K*8 bits = 128K Bytes
  
  Note: k (lower case) ≠ K (capital)
  
  k = 1000 but K = 2^10 = 1024
  
  so the exact memory size is 131,072 bytes
- Size of RAM corresponds to number of address lines

How many address lines are needed to access 128K?

- 128K = 2^7 * 2^10 = 2^17 → 17 address lines required
- RAM/ROM chips always come in sizes of 2^n bits

Example Memory

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0</td>
</tr>
<tr>
<td>1</td>
<td>$1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>131,071</td>
<td>$1FFF</td>
</tr>
</tbody>
</table>

Bräunl 2006

Address

- Each memory location contains one word, typically 8 bits (1 byte), can also be 16 or 32 bits.
- Writing in one location does not affect other locations
- Memory Sizes
  
  1KB = 1024 Byte Kilo 10 address bits
  1 MB = 1024 KB Mega 20 address bits
  1 GB = 1024 MB Giga 30 address bits
- The number of memory addresses available to CPU is called address space.
- Max. memory size = address space * word length (e.g. 8)

Example: 8 bit controller

- 16 address bits → max. 64 KB memory size

Example: 32 bit controller (e.g. 68332)

- 24 address bits, addressing single bytes (or 23 address bits, addressing 16 bit words) → max. 16 MB memory size
- 27 address bits (24 + 3 function codes) → max. 128 MB memory size
ROM Implementation

Example: ROM with 8 bytes contents

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

ROM contents
- mem[0] = $01
- mem[1] = $02
- mem[2] = $05
- mem[5] = $F0
- mem[6] = $78
- mem[7] = $3C

Decoder

0 1 2 3 4 5 6 7

Decoder

0 1 2 3 4 5 6 7

RAM Implementation

Similar to ROM, but replace connection element by active switch

1. DRAM (dynamic RAM, needs ‘refresh’ ~ 500 times/sec, 1 FETTransistor)

2. SRAM (static RAM, no ‘refresh’ required, 6 FETTransistors)

CMOS = complementary metal oxide silicon

1.8 Input and Output

Required to communicate with outside world

- PC System:
  - Keyboard
  - Monitor
  - Parallel port (printer port)
  - Serial port + USB

- Embedded System:
  - Sensors (e.g. in automobile: acceleration sensor, seat sensor)
  - Actuators (e.g. in automobile: valves for airbags)
Input and Output

Input / output device implementation can be:

- Memory-mapped
- I/O mapped (ports)
- DMA (direct memory access)

Memory-Mapped vs. I/O-Mapped

Memory-Mapped
- I/O device looks like regular memory cell
- Often: addresses for control register + data register
- E.g. used for Eyebot
  - output: `MOVE.B D0, OutBase`
  - input: `MOVE.B InBase, D0`
- E.g. used for video display (text mode) on IBM-PC
  - address B800:0000 to B800:0F9F, 2 bytes per char for blinking, underline, graphics, etc.

IO-Mapped
- System provides special I/O ports with:
  - port-addresses
  - port-commands
- Commonly used for printers and modems
- E.g. serial port COM1 on IBM-PC
  - I/O port address $3F8 to $3FF
  - send ‘A’: `outportb(0x3F8, 'A')`
  - receive: `c = inportb(0x3F8)`
  - status: `$3F0, bit 0 = byte received ready = inportb(0x3FD) & 0x01`

Direct Memory Access (DMA)

- Allows I/O devices to access memory directly without involving the CPU
- DMA - device acquires bus-control and reads/writes data while CPU is doing arithmetic
  → parallel processing = higher speed
- If both CPU and DMA device want bus access, one has to wait a few cycles
- Useful for block storage devices, e.g. disk drives
  - For data transfer, we have to wait for the right sector to come around
  - DMA: CPU does not have to wait
  - DMA interrupts when data is ready, transfers data, sets flag for completion