Laurea Specialistica in Ingegneria dell'Automazione

Sistemi in Tempo Reale

Giuseppe Lipari

Introduzione
Embedded control systems

- Typical structure of a feedback controller
Embedded control systems

- Objective
  - Design, implement, analyse, validate and test an embedded control system
Embedded Systems

• An embedded system differs from a normal PC
  – Because it is “embedded” in the system
  – Examples:
    ● Airplane flight controller
    ● Engine control for automobiles
    ● Robot controllers
  – It is not perceived as a computer

• Possible implementations
  – Only hw: examples
    ● Hard disk controller
    ● Low level motor control
  – HW/SW
    ● One generic microcontroller plus I/O devices
    ● A SW that implements the controller
Problems

- From continuous time to discrete time
  - Sampling
- From continuous variables to finite discrete variables
  - Quantization
- hw/sw platform
  - Delays (stochastic)
  - Structure of the program, interaction with the drivers
- Analysis and Testing
  - Conformance between specification and implementation
  - Performance analysis
  - Testing correctness
Design cycle

• Specification of the controller
  – In terms of differential equations
  – For example, with Matlab/Simulink

• Platform Design
  – Choose the microcontroller, the devices, etc.
  – Choose the RTOS

• Mapping on a HW/SW platform
  – Define number of tasks, interrupts, device drivers
  – Map functionality (i.e. simulink blocks) on tasks and interrupts

• Performance analysis
  – Check that delay is bounded
  – Measure controller performance
After analysis, performance is not satisfying:
- Change controller design (i.e. sampling rates, controller algorithm, etc.)
- Change platform (i.e. a faster processor, more memory, etc.)
- Change mapping (i.e. slow down some activity, change number of tasks etc.)
Current practice

- The control engineer designs the system making some assumption
- The he/she passes everything to the software engineer that produces the main code
  - Implementation of the control functions, filters, etc.
- The he/she passes everything to the firmware engineer that writes the glue code
  - Device drivers, interrupt handlers, BIOS routines, etc
- Problem:
  - At every passage, there is the danger of a potential catastrophe! Why?
  - Because there is no formal way of passing the information between the different steps
Existing tools

- From control design:
  - Simulink, stateflow/statecharts, Ascet SD
- To code generation
  - RT Workshop / Embedded Coder / Target Link
- Problems with existing tools
  - The designer does not have much control and insight on the process of code generation
  - It may happen that the final system is *semantically different* from the simulated system
- In this course you will learn what is behind
  - What is a RTOS
  - How to (manually and automatically) implement an ERTCS
Topics Coverage

• Control theory
  – Discrete digital system design, sampling effects
  – Quantization effects
  – Delay effects
  – Multi-rate, multi-level
  – Mode change

• RTOS
  – Real-time systems and multi-tasking
  – Devices – Polling and interrupts
  – Scheduling and resource sharing
  – Schedulability analysis

• Design practice
  – Design and implementation of a controller in the Shark OS
Systems to analyse

- Single rate systems
  - One sampling rate only, MIMO

- Multi-rate systems
  - Inputs and outputs may have different sampling rates

- Multi-level systems
  - Low-level control, cinematic (fast rate)
  - Medium level control, dynamic (medium rate)
  - High level control, planning

- Multi-modal systems
  - Multiple modes of functioning
    - Example of an airplane: takeoff, cruise, landing
  - What happens when we switch mode?
Real Time Systems
What is a real-time system

• A system that must respond to external events within a bounded time
• Real-Time does not mean “fast”, but on time
• Example:
  – Temperature control system
  – Regulates the temperature in a closed environment
  – Response time characteristics: several seconds (or minutes)
  – A system that adjusts the temperature once every 10 seconds is more than enough!
  – It is a real-time system
Real-Time systems

• Problem: not all systems are so simple like the temperature regulator

• Example: inverted pendulum with vision
  - Two control loops: angle and position
  - Video camera for sampling position
  - Potentiometer for sampling angle
Inverted Pendulum Example

Inverted Pendulum Example

Matrix gain

force → Pendulum

θ' → d1 → θ_f → f1 → θ

x' → d2 → x_f → f2 → x

Data: 23/02/2004

Ingegneria dell'Automazione: Sistemi in Tempo Reale

Giuseppe Lipari
Software structure - static

- How to write the software?
  - First approach: single loop
  - Every sampling period
    - Read sensors (x,theta)
    - Compute all the functions
    - Send output data (force)
    - Wait for next activation
Single-loop approach - problems

- Image grabber's max sampling rate = 25 fps
  - Sampling period = 40 msec
- Algorithm for computing x
  - Complex image recognition algorithm: takes 20 msec
- Sampling period cannot be less than 22 msec
  - At this rate, the system is unstable!
- Observation:
  - Keeping angle = 0 is more important than keeping position at 0
  - Use two rates, one for position control, the other one for angle control
Two rates

Matrix gain is executed at the faster rate
- It takes the old values if d2 results are not ready yet
SW structure – static scheduling

- Theta sampling period: 4 msec
- X sampling period = 40 msec
- Observation: suppose f2 takes 20 msec to execute
  - Divide f2 into sub functions f2_1, f2_2, ..., f2_10, each one of 2 msec length
Problems with static partitioning

● Althought the system is now stable, the solution is not flexible
  – Manual partition of f2 is not easy
  – We must put care on how to divide f2
    ● If we update some variable y in f2_i and f2_j, no other function should update y between f2_i and f2_j, otherwise it could use an inconsistent value!
  – Suppose that we want to change the sampling period for the angle to 5 msec
  – We have to “partition” f2 into a different number of functions!
  – The period for the angle loop must be a divisor of the period for the position loop!
Using Concurrency

• Since the two loops are almost independent, we can use multi-programming
  – In modern operating systems, programs can be:
    • SEQUENTIAL
    • CONCURRENT
  – Using multi-programming, a user program consists of one or more flows of executions (threads)
• Thread = function that execute *concurrently* with the other threads
Concurrency

• Concurrency
  – If number of processors = number of threads -> threads execute in parallel
  – If number of processors = 1 -> threads execute in time-sharing
  – If number of processors < number of threads -> combination of parallelism and time-sharing

• Every OS provides automatic support for concurrency:
  – It provides an “abstract” machine with an infinite number of processors
  – The abstract machine is then “simulated” through time-sharing
Threads

• The concepts of task, concurrency, scheduler, etc. will be presented in more details during the course
• By now, it is important to have an idea of what is a thread
• We use the name thread or task with the same meaning

```c
int main()
{
    // main thread,
    // creates all others
    task_create(..., myTask1, ...);
    task_create(..., myTask2, ...);
    // from now on, three tasks
    // are running, main, myTask1,
    // and myTask2
}
```

```c
int mytask1()
{
    f1();
    d1();
    matrix();
    actuate();
}

int myTask2()
{
    f2();
    d2();
}
```
Tasks

• Solutions with timers
  – Define two concurrent tasks
  – myTask1 computes the angle loop plus the matrix gain
  – myTask2 computes the position loop
• Define one periodic timer tm1 for myTask1
  – Every 4 msec, tm1 activates myTask1
  – Once every 10 times, tm1 activates also myTask2
• Define 1 additional timer tm2
  – It is a timeout timer equal to 2 msec
  – It is set every time myTask2 starts executing
  – When the timer expires, myTask2 is suspended by the RTOS
Static scheduling with tasks

- We need the support of the RTOS
  - For activating and suspending the tasks

![Diagram of static scheduling with tasks](image)
Table driven scheduling

• This approach is called table-driven scheduling
  – The sequence of tasks to be executed is stored in a table
  – For each table entry
    • Task id
    • Time to execute
  – After the last table entry, the system starts from the beginning
  – In the previous example,
    • Task 1, 1.5 msec
    • Task 2, 2 msec
    • IDLE, 0.5 msec
Dynamic scheduling

- Another approach is to use a system wide scheduler
  - SCHEDULER = algorithm to choose which task is to be executed next
- We can model the scheduler as a queue
  - The queue contains the active tasks, ordered by some rule
Scheduling

• Examples of non real-time algorithms:
  - Round Robin (RR)
  - Shortest Job First (SJF)
  - FCFS (First Come First Served)

• Examples of real-time algorithms:
  - Fixed priority
  - Earliest Deadline First (EDF)
  - Least Laxity First (LLF)
  - etc.
Example

• In the inverted pendulum example
  – myTask1 has higher priority than myTask2
Schedulability

- How can we guarantee that all tasks are actually completed “in time”?
- Each task is assigned a “deadline”
  - The time by which it has to complete
  - Usually, a periodic task must complete before the next period
    - Deadline = Period
- For each task, we measure the computation time
  - Worst Case Computation Time (WCET) = maximum possible execution time for a task
- Given these information, we can perform a “schedulability analysis”
Example of schedulability analysis

• In the inverted pendulum example
  – myTask 1   WCET = 1.5   period = 4
  – myTask 2   WCET = 20    period = 40

  – Total load = 1.5 / 4 + 20 / 40 = 3.5 / 4 = 87.5 %

• The system **IS SCHEDULABLE**
  – Each task will *always* complete before its next period

• In general, the analysis may depend on
  – The scheduling algorithm
  – The task constraints
Design decisions

• When implementing the system, the designer has to take several decisions
  – How many task
  – Period and duration of each task
  – Which function is mapped on each task

• These choices are often arbitrary!
  – Most of the times design is based on the experience of the programmers
Other issues

• We have not considered yet input/output
  – Input = sensors
  – Output = actuators
  – Many different sensor types
  – Each board is equipped with HW devices
    • A device is an interface between the computer and the external world

• Typical INPUT structure
  – Data is collected by the device and stored in an internal HW buffer
  – A routine reads the data from the HW buffer into memory
    • Polling or interrupt based
  – A task reads the data from the memory
Impact of Input/Output

• When analysing our system we have to take into account input/output routines
  – In the case of polling, how often we have to inspect the HW buffer
  – In the case of interrupt, how often the interrupt is raised and how long it takes

• Other issues
  – Communication between tasks
  – Interactions and blocking
  – User input
  – Supervisor and mode change
  – Fault Tolerance