A "Co-operative First" Approach to Software Development for Reliable Embedded Systems

Michael J. Pont
Embedded Systems Laboratory
University of Leicester

Presentation at Embedded Systems Show
Birmingham, UK, 13 October 2004

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Overview

- In this talk, I will explore what it means to develop systems using a simple scheduler (a "time-triggered co-operative scheduler") rather than using a "real" real-time operating system
- I'll explain what a TTC scheduler is, and examine some of its strengths and weaknesses
- My focus will be on "deeply embedded" systems (e.g. automotive systems) rather than "embedded desktop" designs (e.g. PDAs)
- I will look at both single-processor designs and multi-processor (distributed) designs

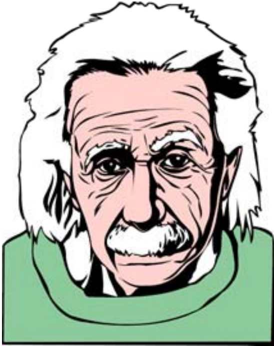
Overview

- My aim will be to suggest that - for many projects - use of a TTC software architecture is likely to result in a simple, cheap and reliable system
- I will not argue that a TTC architecture is appropriate for all systems
- I will argue that – for the class of (deeply) embedded systems considered in this talk – it makes sense to adopt a "co-operative first" approach to system development.

What does "co-operative first" mean?

1. If your system is a "good match" for a Super Loop architecture, use this. If a Super Loop is not a good match …
2. Consider using a T-ISR scheduler …
3. Consider using a full co-operative scheduler ...
4. Consider using a T-T hybrid scheduler (or a multi-processor design) …
5. Consider using a fully pre-emptive approach.

Underlying philosophy



"Everything should be made as simple as possible, but not simpler."
Albert Einstein
Quoted in *Reader's Digest,* October 1977.

Structure

- *Introduction*
- Some loose definitions
- Explain what a TTC architecture is
- Look at (and compare) some of the different ways in which single-processor TTC designs can be implemented:
  - Super Loops
  - T-ISRs
  - Complete TTC schedulers
- Look at simple ways of dealing with "the long task problem"
- Look at hybrid designs
- Look at multi-processor designs
- Conclusions

Some loose definitions

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Loose definitions [1]

- For our purposes:
  - A **periodic task** will be implemented as a C function which is called – for example – every millisecond or every 100 milliseconds during some or all of the time that the system is active.
  - A **one-shot task** will be implemented as a C function which may be called if a particular event takes place. For example, a one-shot task might be called when a switch is pressed, or a character is received over a serial connection.

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Loose definitions [2]

- Timing constraints:
  - A system is considered to have **soft timing (ST) constraints** if execution of any task more than 1 second late (or early) can cause a significant change in behaviour.
  - A system is considered to have **hard timing (HT) constraints** if execution of any task more than 1 millisecond late (or early) can cause a significant change in behaviour.
  - A system is considered to have **very hard timing (VHT) constraints** if execution of any task more than 1μs late (or early) can cause a significant change in behaviour.

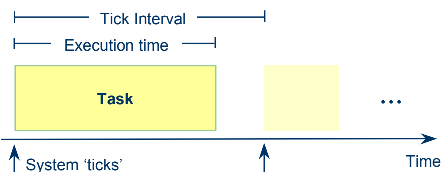
What is a TTC architecture (and how can we build one)**?**

What is a TTCS architecture?

- Basic behaviour
  - We execute one or more periodic tasks (e.g. run Task A every 1 ms)
  - All tasks "run to completion"
- Options
  - We may support one-shot tasks (e.g. run Task B after 30 ms)
- Advantages
  - No complex context switching, locks, semaphores
  - Easy to understand, cheap to build
  - Highly predictable behaviour
- Key disadvantage
  - Problems with "long tasks"

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

What's the simplest implementation?




```
while(1)
      {
      Task();                // Run task

      Delay_One_Second();    // Wait for 1 second
      }
```

"Super Loop"

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Should you use a "Super Loop"?

- Super Loops have the following advantages:
  - VERY simple to implement in any language, using any compiler
  - Easy to understand
  - Minimal resource requirements
  - Appropriate for executing one periodic task with ST constraints
- Super Loops have the following disadvantages:
  - Obtaining precise timing is often tricky
    (if you need precise timing, there are better solutions)
  - Running multiple tasks is often tricky
    (if you need to run multiple tasks, there are better solutions)

ASIDE: A better Super Loop

```
while(1)
  {
  Set_Timer_To_Overflow_After_One_Second();

  Task();

  Wait_For_Timer_To_Overflow();
  }
```

"Super Loop
with
Sandwich Delay"

Creating a "Timer ISR" (T-ISR) scheduler

Update() Update() Update() ...

System 'ticks' — Time

BACKGROUND PROCESSING

```
while(1)
   {
   ; // Do "nothing"
   }
```

FOREGROUND PROCESSING

```
Update();
```

One-second timer

Embedded Systems Laboratory http://www.le.ac.uk/eg/embedded

Creating a T-ISR scheduler

- Many effective embedded systems are created in this way – simply using a timer ISR (often with a long `switch` statement).
- Any embedded processor (8-bit, 16-bit, 32-bit, …) with an on-chip timer can be used
- Many C compilers provide support for interrupts – at worst, a small assembly-language "harness" may be required
- Further details (including comprehensive notes and code examples) are available here:

`http://www.le.ac.uk/engineering/mjp9/pttesguide.htm`

Embedded Systems Laboratory http://www.le.ac.uk/eg/embedded

Should you use a T-ISR scheduler?

- T-ISR schedulers have the following advantages:
  - Simple to implement in most languages, using a wide range of compilers
  - Easy to understand
  - Low resource requirements
  - Very appropriate for executing one periodic task with HT or VHT constraints

- T-ISR schedulers have the following disadvantages:
  - Running multiple tasks can get tricky
    (once a `switch` statement gets beyond 10 pages, it can get very confusing – this can prove to be a maintenance headache)

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

A complete TTC scheduler implementation

```
void main(void)
   {
   SCH_Init_T2(); // Set up the scheduler

   PID_MOTOR_Init();
   PC_LINK_O_Init_Internal(9600);

   // Add tasks
   SCH_Add_Task(PID_MOTOR_Poll_Speed_Pulse, 1, 1);
   SCH_Add_Task(PID_MOTOR_Control_Motor, 300, 1000);
   SCH_Add_Task(PC_LINK_O_Update, 3, 1);

   SCH_Start();

   while(1)
      {
      SCH_Dispatch_Tasks();
      }
   }
```


Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

A complete TTC scheduler implementation

- You'll find a complete description of a flexible TTC scheduler – include full code listings – here:

  `http://www.le.ac.uk/engineering/mjp9/patterns.html`

  This version will run on PC hardware, and is written entirely in C (Open Watcom compiler).

- The above site also includes details – including all code listings - of a complete TTC scheduler in assembly language (written by Simon Key). This scheduler has extremely limited resource requirements.

Should you use a complete TTC scheduler?

- TTC schedulers have the following advantages:
  - Simple to implement in most languages, using a wide range of compilers
  - Complete implementations (in C) readily available for numerous platforms (ARM, C16x, 8051, PIC, AVR …)
  - Easy to understand and port
  - Easy to use
  - Low resource requirements
  - An effective way of executing multiple tasks (periodic or one-shot) with ST, HT or VHT constraints
  - Architecture readily extends to support multi-processor designs (more later)

What is the “long task” problem (and how can we solve it)?

What are the issues with TTC designs?

- TTC schedulers have many attractive features
- TTC schedulers are easy to build

HOWEVER:

- TTC schedulers - like all co-operative designs - are susceptible to the “long task” problem ...

The “long task” problem

- If the system is executing a task, it cannot react to events.
- For example:
  if one or more of the tasks takes 100 ms to execute, you cannot react to events more rapidly than this.
- If a task “hangs” we may lose control completely …

```
void Task(void)
   {
   while(1)
      {
      // Crash the plane …
      }
   }
```

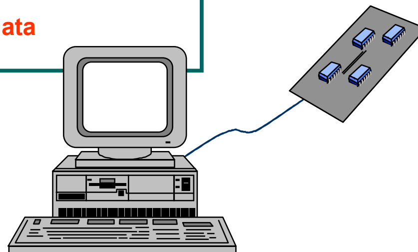

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

The “long task” problem (Example)

```
printf(“Current core temperature is 36.678 degrees”);
```

Assume:
Using RS-232 to send data to a PC (or similar)
9600 baud

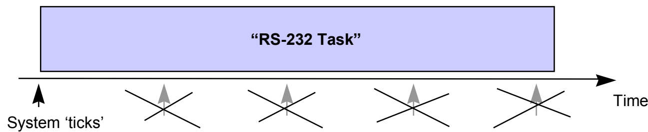
Takes ~ 42 ms to send these data



Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

The “long task” problem (Example)

For example, assume 10 ms ticks:



As a <u>general</u> rule, we require ALL tasks to complete within the tick interval

Dealing with long tasks (Patterns)

- To support the development of TTC designs, we have assembled a collection of more than 70 “design patterns”
- These patterns cover all aspects of the system development – from schedulers and task design through to reset circuits.
- More information (including numerous examples) here:

`http://www.le.ac.uk/engineering/mjp9/patterns.html`

Dealing with long tasks (Patterns)

- LOOP TIMEOUT
- HARDWARE TIMEOUT
- MULTI-STAGE TASK (example shortly)
- HYBRID SCHEDULER (more later ...)
- WATCHDOG TIMER
- SHARED-CLOCK SCHEDULER (more later ...)

`http://www.le.ac.uk/engineering/mjp9/patterns.html`

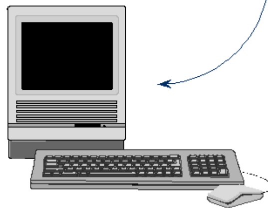
Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Dealing with long tasks (Patterns)

```
Current core temperature
is 36.678 degrees
```

All characters written immediately to buffer (very fast operation)

Buffer



Scheduler sends one character to PC every 10 ms (for example)

PATTERN: "Multi-Stage Task"

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Dealing with long tasks (Patterns)

- During this conference, Chisanga Mwelwa (ESL) will describe some of his work with patterns, including techniques for auto code generation for TTC designs:
  - In the PhD forum (today)
  - In the BAE Systems "auto code" session on Thursday

Mwelwa, C., Pont, M.J. and Ward, D. "Code generation supported by a pattern-based design methodology" (PhD forum)

Dealing with long tasks (Task guardians)

- Even if you have created a "perfect" TTC design, there may still be the risk that a task will "hang" and never return control to the scheduler

```
void Task(void)
   {
   while(1)
      {
      // Crash the car …
      }
   }
```

Dealing with long tasks (Task guardians)

- In the PhD forum, Zemian Hughes will describe some simple – but effective – "task guardians" that can help safeguard the behaviour of your TTC system in these circumstances

Hughes, Z.H. and Pont, M.J. "Design and test of a task guardian for use in TTCS embedded systems" (PhD forum)

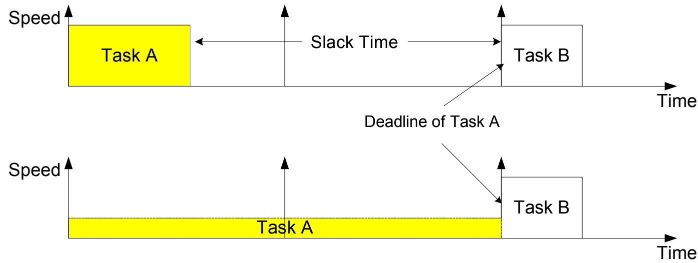
Dealing with long tasks (Brute force!)

- It is sometimes argued that a move to more powerful processors (e.g. moving from an 8051 platform to an ARM platform) means that it is "no longer necessary" to use TTC architectures and that, instead, the system now has enough CPU resources to support a "real" OS

- By contrast, TTC architectures are particularly appropriate on modern platforms, because problems caused by "long tasks" are greatly reduced

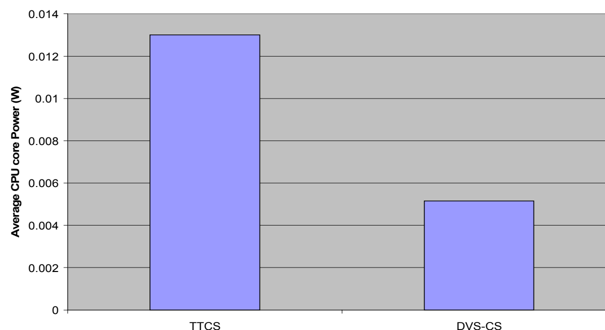
- More modern hardware also opens up other possibilities …

ASIDE: Applying DVS in TTC designs



Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

ASIDE: Applying DVS in TTC designs



Phatrapornnant, T. and Pont, M.J. "The application of dynamic voltage scaling in embedded systems employing a TTCS software architecture" (PhD forum)

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Bending the rules

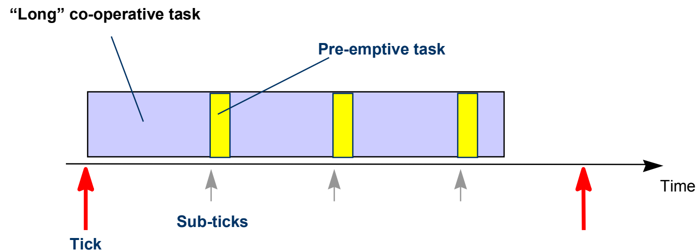
Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Bending the rules

- Sometimes you simply cannot meet the needs to run long tasks AND have a rapid response to events with a TTC design
- One solution is a multi-processor design (we will consider this approach shortly)
- Another solution is to build a “hybrid” scheduler
- A hybrid scheduler is a time-triggered scheduler, with support for a single pre-emptive task

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

A hybrid scheduler



NOTE:
- Time-triggered architecture
- One pre-emptive task
- Behaviour remains highly predictable
- Locking mechanisms required (but simple)

See "Patterns for Time-Triggered Embedded Systems" (Chapter 17)

Advantages of hybrid (T-T) architecture

```
void Co_operative_Task(void)
   {
   ...

   if (pre_emptive_task_is_due_to_execute)
      {
      // Do not access shared resource
      return;
      }
   else
      {
      // Access shared resource
      ...
      }
   }
```

The story so far ...

- I've argued that you should aim to build embedded systems using an architecture that is "as simple as possible but not simpler"
- I've looked at Super Loops
- I've looked at T-ISR schedulers
- I've looked at "complete" co-operative schedulers
- I've considered various ways of dealing with the "long task" problem - including the use of hybrid schedulers

- I will conclude by looking at multi-processor designs

Multi-processor TTC designs

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Multi-processor TTC designs

BACKGROUND PROCESSING

```
while(1)
   {
   ; // Do "nothing"
   }
```

FOREGROUND PROCESSING

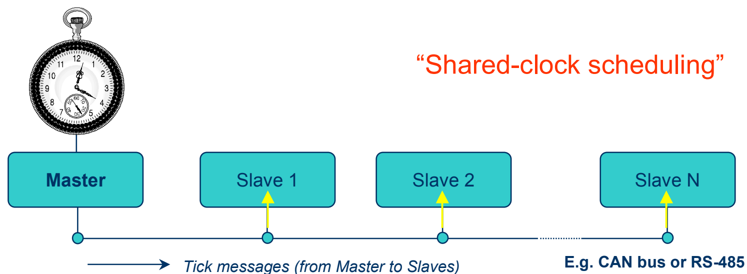
```
Update();
```

One-second timer

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Multi-processor TTC designs

"Shared-clock scheduling"



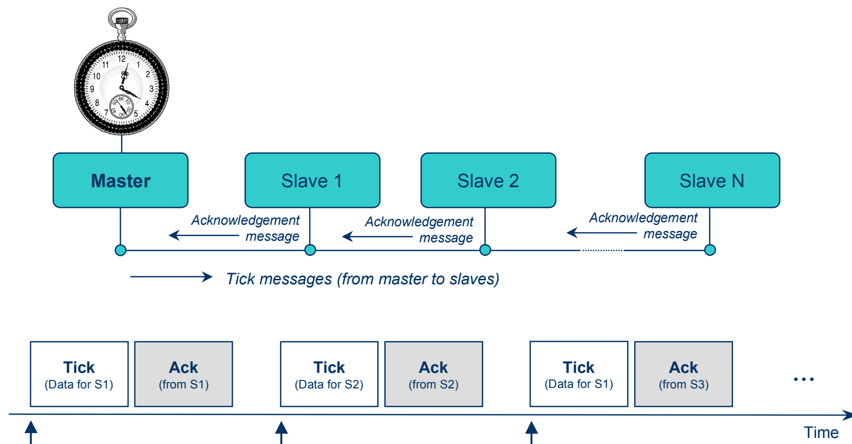
On the Master node, the TTC scheduler is (as before) driven by interrupts from an on-chip timer

The Master sends periodic "Tick" messages to the slaves

On the Slave nodes, the scheduler is driven by interrupts generated through the arrival of messages from the Master

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

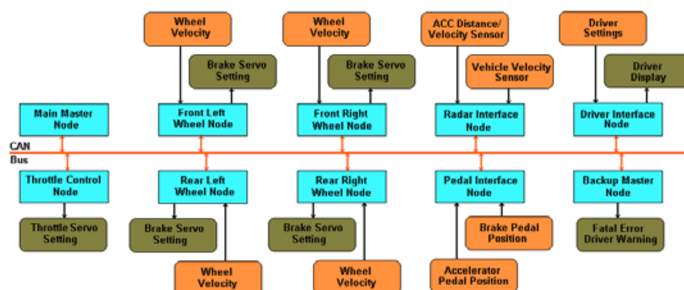
Message transfers (simple version)



Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Examples of distributed TTC designs [1]

- **An automotive testbed**
  (Dr. Michael J. Short, ESL, Leicester)



Track B, Thurs

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Examples of distributed TTC designs [2]

- Edwards, T., Pont, M.J., Scotson, P. and Crumpler, S. "A test-bed for evaluating and comparing designs for embedded control systems"
- Key, S. and Pont, M.J. "Implementing PID control systems using resource-limited embedded processors"
- Nahas, M. and Pont, M.J. "Reducing task jitter in shared-clock embedded systems using CAN"

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

How do I know my TTC design will work?

- If you want to determine how your TTC design will operate, simulation offers a good solution
- For example, in the PhD forum Dev Ayavoo will describe how to use an "Open Source" simulator with MATLAB to model the behaviour of distributed TTC designs.

Ayavoo, D., Pont, M.J. and Parker, S. "Using simulation to support the design of distributed embedded control systems: A case study" (PhD forum)

Embedded Systems Laboratory
http://www.le.ac.uk/eg/embedded

Summary

- I've looked – very briefly - at various ways in which TTC architectures can be implemented
- I've tried to demonstrate some of the attractive features of TTC designs (include the simple way in which we can port taskss between single- and multi-processor architectures)
- I've argued that the basic challenge with TTC designs – dealing with long tasks – is well understood, and (for many systems) easily solved
- I've given some suggestions for further reading in this area

Conclusions

1. If your system is a "good match" for a Super Loop architecture, use this. If a Super Loop is not a good match …
2. Consider using a T-ISR scheduler …
3. Consider using a full co-operative scheduler ...
4. Consider using a T-T hybrid scheduler (or a multi-processor design) …
5. Consider using a fully pre-emptive approach.

Keep it simple!

Further reading

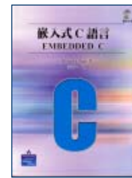
Pont, M.J. (2002) *"Embedded C"*,
Addison-Wesley. ISBN: 0-201-79523X.

`http://www.engg.le.ac.uk/books/pont/ec51.htm`

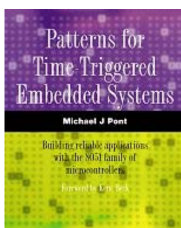
Translations (Chinese)

"How to create and use a simple 'TTC ISR' scheduler using C"

Single-processor designs only



Further reading



Pont, M.J. (2001) *"Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers"*, ACM Press / Addison-Wesley. ISBN: 0-201-331381.

`http://www.le.ac.uk/engineering/mjp9/pttes.html`

Translations (Chinese)

"How to create and use a complete TTC scheduler using C"

Single- and multi-processor designs
Co-operative and hybrid schedulers