Embedded systems engineering
Distributed real-time systems

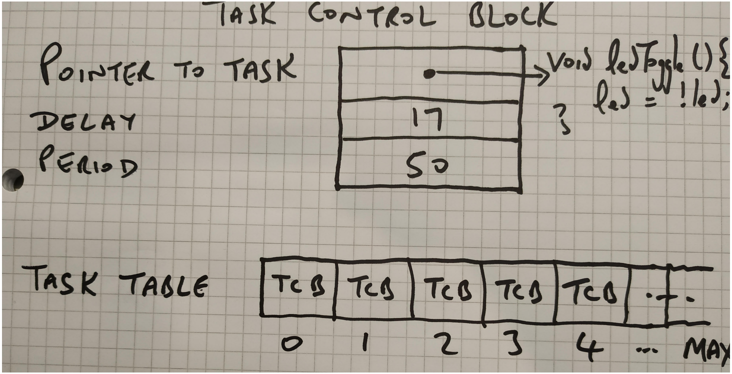
David Kendall

Implementing a simple time-triggered scheduler

- Scheduler task table
- Scheduler data declarations
- Initialising the scheduler
- Adding tasks to the task table
- Starting the scheduler
- Dispatching tasks
- Sleeping when there is no work to do

Based on Pont but for ARM target and with some restructuring of code.

Scheduler Task Table



A Task Control Block (TCB) has

- a pointer to a C function
- a delay (in ticks)
- a period (in ticks)

The task table is an array of TCBs indexed from 0 .. MAX - 1

Scheduler Data Declarations

Pointer to task is a *void function pointer*

```
typedef void (*pVoidFunc_t)(void);
```

Task Control Block is a *struct*

```
typedef struct schTCB {
  pVoidFunc_t task;
  uint32_t delay;
  uint32_t period;
} schTCB_t;
```

The task scheduling table is an array of TCBs

```
static schTCB_t schTasks[TT_SCHED_MAX_TASKS];
```

Scheduler Data Declarations (ctd)

We also need a *tick count* that is incremented by the timer interrupt handler and decremented by the dispatcher

- Should be declared as *volatile*, since it can be updated by the interrupt handler at any time
  `static volatile uint32_t tickCount = 0`

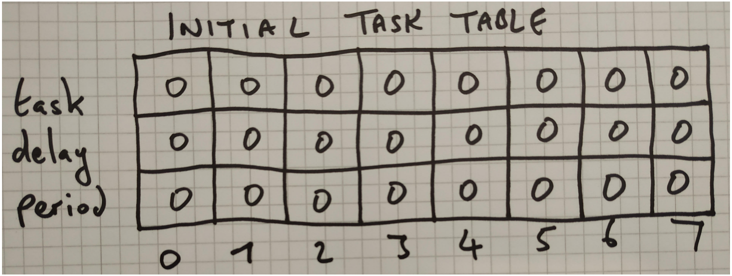
Configuration parameters to be configured by the application developer

- The maximum number of tasks in the scheduling table
  `TT_SCHED_MAX_TASKS`
- The frequency of the timer interrupt in Hertz (Hz), i.e. number of times per second that the timer interrupts
  `TT_SCHED_TICK_HZ`

Initialise the task table

```
void schInit(void);
```

The job of the initialisation function is very simple: for every TCB in the scheduling table, set each field in the TCB to 0



Scheduler Add Tasks

```
void schAddTask(         // add a task to the task set
  pVoidFunc_t task,         // the task to add
  uint32_t delay,           // the delay in ms
  uint32_t period);         // the period
```

Example: `schAddTask(ledToggle, 0, 50);`

The job the `schAddTask` function is to find the first empty slot in the task table array and initialise the fields of the TCB there with the parameters passed to the function

An empty entry in the task table can be identified easily because its `task` field is a `NULL` pointer (0)

Scheduler Start

Configure and start a timer to interrupt periodically

```
void schInit(void);
```

- The ARM Cortex M3 / M4 has a simple timer called the `SysTick` timer that has been introduced to simplify the implementation of a timing source for a scheduler / operating system
- There is a CMSIS function `SysTick_Config` that can be used to configure the timer to raise an interrupt periodically after a count
- The CMSIS constant `SystemCoreClock` gives the frequency of the system core clock – use this with `TT_SCHED_TICK_HZ` to get the value to use with `SysTick_Config`

You also need to declare a handler for the `SysTick` timer interrupt

```
void SysTick_Handler(void);
```

The `SysTick` handler just needs to increment `tickCount`

Scheduler Dispatch

```
void schDispatch(void);
```

Protecting data shared with the interrupt handler

- the dispatch function shares the `tickCount` variable with the timer interrupt handler
- Protect this variable by disabling interrupts while it is used in the dispatch function
- Use `__disable_irq()` and `__enable_irq()` to disable and enable interrupts
- Interrupts should be disabled for the shortest possible time to ensure that the system remains responsive

Updating the task table

- if `tickCount > 0` then the task table must be updated by decrementing the `delay` field in every non-empty TCB

Scheduler Dispatch (ctd)

Running tasks that are ready to run and reinitialising the `delay`

- If the `delay` field in any TCB has reached 0, the function for the task must be called, e.g. `(*(schTasks[i].task))();`
- When the task function returns, reinitialise the `delay` field using the `period` value

Now `tickCount` should be decremented to show that this tick has been serviced

If all task functions have not completed their execution before the next `SysTick` tick then `tickCount` may not be 0, so check it again and run the dispatcher again, if necessary

Failing to execute all task functions within a single frame may cause problems

Scheduler Sleep

It is a good idea to allow the micro-controller to go into a low-power state (sleep) when there is no work for it to do currently
`schSleep(void);`

The best way to do this with the ARM Cortex M3 / M4 micro-controllers is to use the `__WFI()` instruction which is a hint to the micro-controller to go to sleep and *Wait For Interrupt*

The micro-controller may or may not take notice of this hint

Time-triggered scheduler: header file

```
#ifndef __SCHEDULER_H
#define __SCHEDULER_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <ttSchedConfig.h>

typedef void (*pVoidFunc_t)(void);

/* Task Control Block structure */
typedef struct schTCB {
  pVoidFunc_t task;
  uint32_t delay;
  uint32_t period;
} schTCB_t;

void SysTick_Handler(void); // install own handler for SysTick
void schInit(void);         // initialise the scheduler
void schAddTask(            // add a task to the task set
  pVoidFunc_t,                // the task to add
  uint32_t,                   // the delay in ms
  uint32_t);                  // the period
void schStart(void);        // start ticking
void schDispatch(void);     // run the next task
void schSleep(void);        // go to sleep to save power

#ifdef __cplusplus
}
#endif
#endif
```

Time-triggered scheduler: use in practice

```
int main() {
  /* Initialise devices if necessary */

  schInit();           /* initialise the task table           */

  schAddTask(f, 0, 5);  /* prepare to run task f every 5 ticks  */
  schAddTask(g, 1, 10); /* prepare to run task g every 10 ticks */
  schAddTask(h, 3, 15); /* prepare to run task h every 15 ticks */

  schStart();          /* start ticking ...                   */

  while (true) {
    schDispatch();     /* if a task is ready to run, run it */
  }
}
```

Acknowledgements

- Pont, M., Patterns for Time-triggered embedded systems, TTE Systems, 2010