TinyRealTime – An EDF Kernel for the Atmel ATmega8L AVR

Anton Cervin and Dan Henriksson

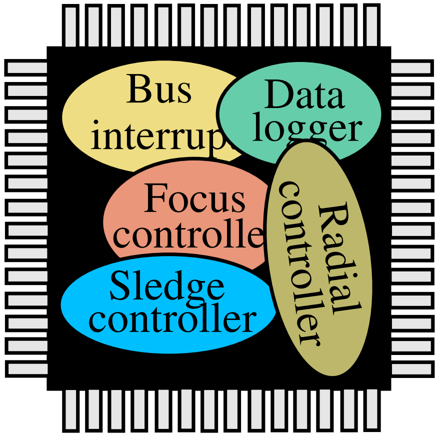
January 2004

Outline

- Motivation
- Kernel implementation
  - Memory layout
  - Kernel data structures
  - Timing
  - Kernel internal workings
- API
- Application
- Conclusions

Motivation

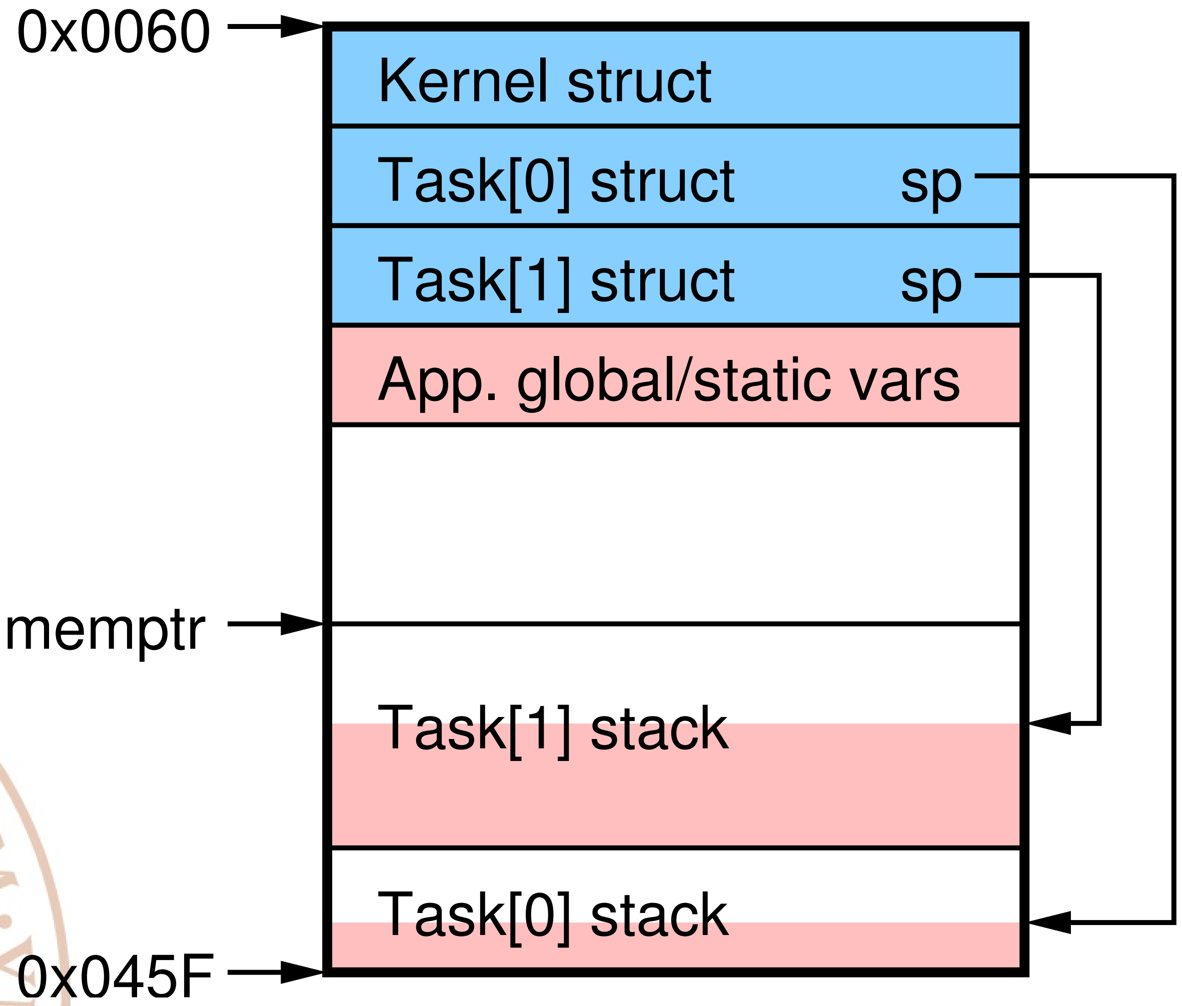
- Embedded control systems are becoming increasingly complex
- Microcontrollers with very limited resources
- Many parallel activities competing for the CPU



- Need for optimal deadline-based scheduling also in embedded control systems!

Memory layout

- 1024 bytes internal SRAM:



Kernel data structures

```
struct task {
  uint16_t sp;         // stack pointer
  uint32_t release;    // current/next release time
  uint32_t deadline; // absolute deadline
  uint8_t state;    // terminated=0, readyQ=1, timeQ=2, semQ[]=3..
};

struct kernel {
  uint8_t nbrOfTasks;
  uint8_t running;
  struct task tasks[MAXNBRTASKS+1];
  uint8_t semaphores[MAXNBRSEMAPHORES];
  uint8_t *memptr;    // pointer to free memory
  uint16_t cycles;    // nbr of major cycles
  uint32_t nextHit;   // next kernel wake-up time
};
```

Timing

- 16-bit Timer/Counter 1 used as clock
- 16 more clock bits stored in the kernel (major cycles)
- Trade-off between clock resolution and system life time

Prescaler	*Clock resolution*	*Life time*
1	68 ns	5 min
8	543 ns	39 min
64	4.3 μs	5 h
256	17.4 μs	21 h
1024	69.4 μs	83 h

- Output Compare interrupt used to run kernel at next event or at timer overflow

Kernel internal workings

```
SIGNAL(SIG_OUTPUT_COMPARE1A) {
  Store r0-r31,SREG on the stack;
  if (TIFR & 0x04) { ++kernel.cycles; TIFR |= 0x04; }
  now = (kernel.cycles << 16) + TCNT1;

  for (i=1; i <= kernel.nbrOfTasks; i++) {
    t = &kernel.tasks[i];
    if (t->state == TIMEQ) {
      if (t->release <= now)
        t->state = READYQ;
      else if (t->release < nextHit)
        nextHit = t->release;
    }
    if (t->state == READYQ)
      if (t->deadline < kernel.tasks[running].deadline)
        running = i;
  }
```

Kernel internal workings, cont’d

```
  if (running != oldrunning) {
    // store old context
    t = &kernel.tasks[oldrunning];
    t->sp = SP;
    // load new context
    t = &kernel.tasks[running];
    SP = t->sp;
    kernel.running = running;
  }

  Update Output Compare register;
  Restore r0-r31,SREG from the stack;
}
```

API

```
#define MAXNBRTASKS

#define MAXNBRSEMAPHORES

void trtInitKernel(uint16_t idletask_stacksize)

void trtCreateTask(void (*fun)(), uint16_t stacksize,
                   uint32_t release, uint32_t deadline)

void trtTerminate()

uint32_t trtCurrentTime()

uint32_t trtGetRelease()

uint32_t trtGetDeadline()

void trtSleepUntil(uint32_t release, uint32_t deadline)

void trtCreateSemaphore(uint8_t semnbr, uint8_t initVal)

void trtWait(uint8_t semnbr)

void trtSignal(uint8_t semnbr)
```

Example: trtSleepUntil

```
void trtSleepUntil(uint32_t release, uint32_t deadline) {
  struct task *t;
  t = &kernel.tasks[kernel.running];
  cli(); // turn off interrupts
  t->state = TIMEQ;
  t->release = release;
  t->deadline = deadline;
  SIG_OUTPUT_COMPARE1A(); // call interrupt handler to schedule
}
```

Balls and Beams Application

- Two multirate ball and beam controllers @ 25/50 Hz
- Two software PWM output tasks @ 1 kHz
- Seven tasks in total (including the idle task)
- Semaphores used to protect common variables (control signals and reference values)

Conclusions

- Feasible to use high-resolution event-based EDF scheduling on the ATmega8
- Kernel program memory size: $\approx$ 1200 bytes
- Kernel SRAM memory size (bytes):

  $$12 + 11 \cdot \text{MAXNBRTASKS} + \text{MAXNBRSEMAPHORES}$$
- Each task needs at least 35 bytes of stack memory
- Software PWM quite jitter sensitive (with large prescaler)