

ReaGOS

real-time kernel

By Ari Okkonen, OBP Research Oy

ReaGOS, when you need

- **an embedded real-time system**
- **fast responses during long computations**
- **exact timing**
- **portability**
- **quick applicability**

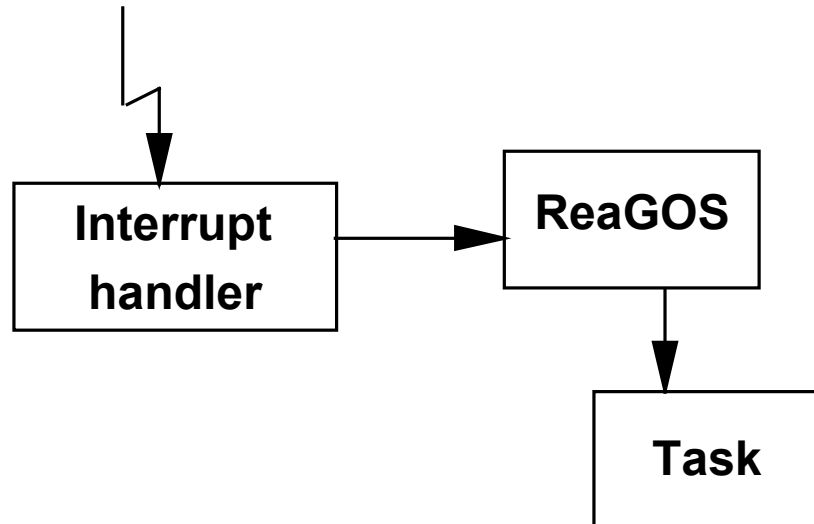
ReaGOS takes care of

- **data and event transfer**
 - all C data types, including structured ones
 - between tasks
 - between tasks and device drivers
 - buffering when needed
- **timing**
- **preemptive priority scheduling**

ReaGOS techniques

- **written using pure ANSI/ISO-C**
- **event driver architecture single stack**
- **a high priority task preempts scheduling of low priority tasks**
- **directly from the hardware to an interrupt handler**
 - **ReaGOS participates to handling of an interrupt only when an interrupt handler sends information to a task.**
- **is generated specifically to the application**
- **seamless connection to ReaGeniX generated code modules**
- **patented: FI-91026**

Interrupt handling



- **An interrupt handler performs needed hardware operations first fast response**
- **When an interrupt handler sends information to tasks, it calls finally a ReaGOS funktion.**

Single stack

- **portability - no machine dependent code for stack switch**
- **memory saving - space for interrupt handlers is not needed in several stacks**
- **only the state of an interrupted computation is saved in the stack**
- **the state of a task resides in the statically reserved area**

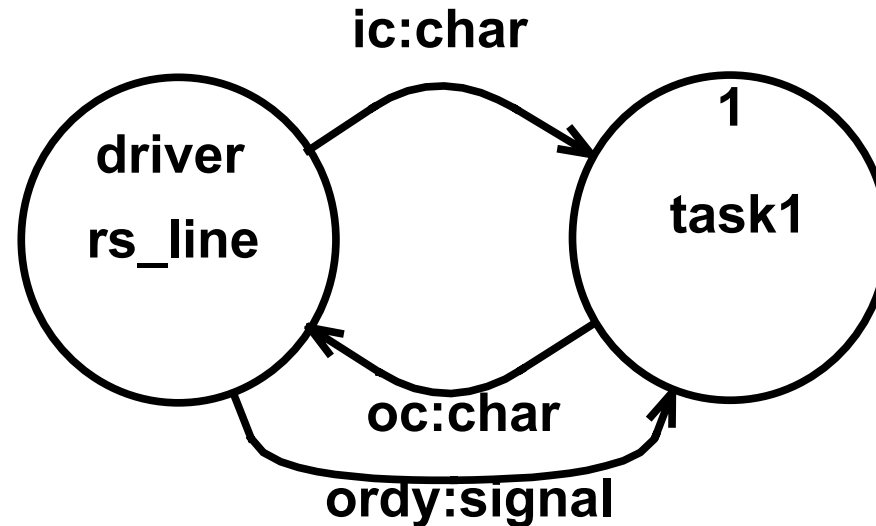
A task diagram

- **ReaGOS is generated from a task diagram by the ReaGeniX Priorizer**
- **a data flow diagram in principle**
- **annotations to specify**
 - **task priorities and device drivers**
 - **buffer capacities to data flows**

Device drivers

- **a device driver consists of interrupt handlers and flow handlers**
 - **an interrupt handler is called directly by the hardware**
 - **ReaGOS calls a flow handler when information is sent from a task to a device driver**
- **device drivers are executed interrupts disabled**
- **lowest level device drivers are written by hand for maximum performance**
- **minimum amount of hand written code is needed to interface an interrupt handler to the rest of the system**

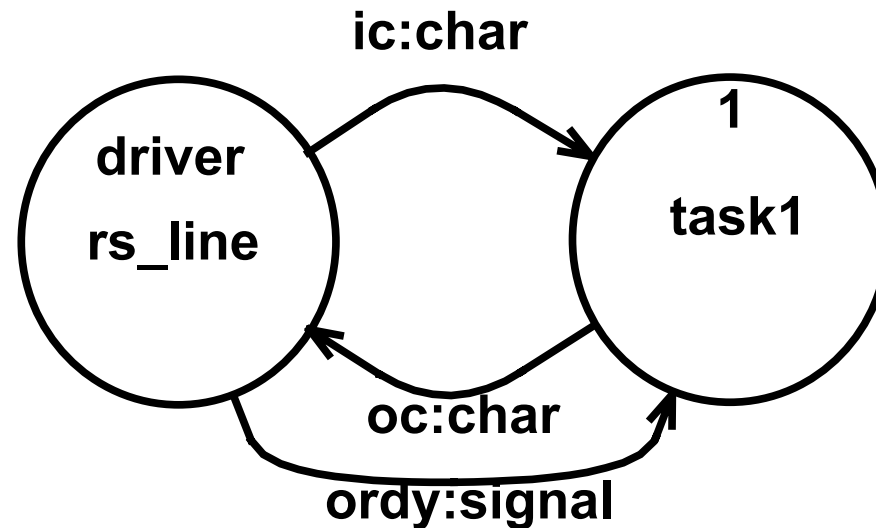
A simple interrupt handler



- **simple but complete input driver**

```
void interrupt rx_int(void){
    *p_rs_line_ic = inp(rx_data);
    rs_line_ic_schedule();
}
```

A simple output flow handler



- **a simple but complete output driver**

```
void o_rs_line__oc(void) {  
    outp(tx_data,v_rs_line__oc);  
}  
void interrupt tx_int(void) {  
    rs_line_ordy_schedule();  
}
```

Adapting ReaGOS to hardware

- a main program to set up stack and hardware state
- real-time clock driver
- other low level device drivers
- macros `ENABLE_INTERRUPTS()` and `DISABLE_INTERRUPTS()`

Summary

ReaGOS

- **is a preemptive priority scheduler for embedded systems**
- **is generated specifically to the application**
- **handles timing and inter-task communication**
- **is easy to port to different hardware platforms**
- **connects seamlessly to tasks generated by ReaGeniX**
- **allows fast responses to interrupts**