TaltOS

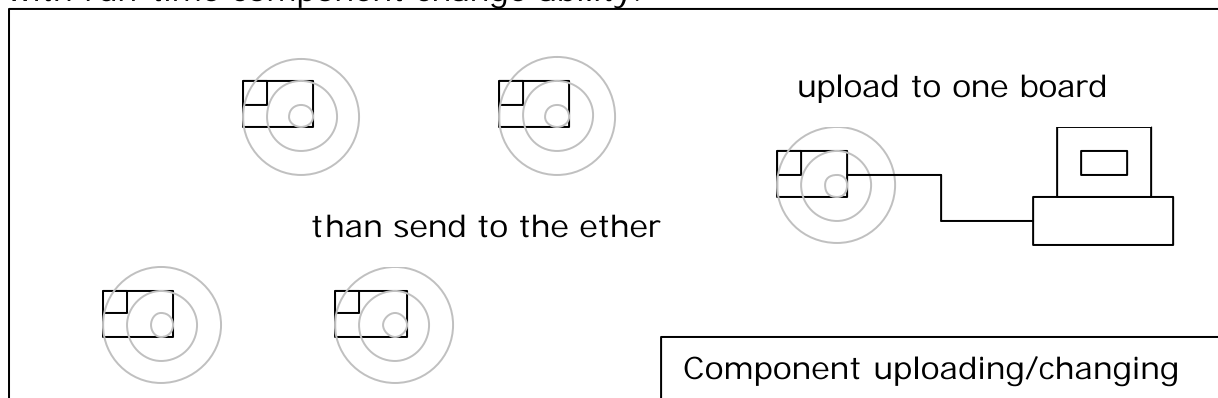
A component-based hard real-time embedded operating system with run-time component change ability.

Pre-planning

I. The problem situation

Programming embedded systems is a quite important work these days. Every high-tech product contains some printed-circuit board with a processor and memory. But only a few are capable to change his on-chip software, and can do this only with special cables in offline mode, writing the whole software again on the non-volatile memory.

If we imagine a sensor network in a development stage, the reconfiguration of fifty boards and chips can take a while with one computer and a serial cable. It would be much easier to upload a small base software, after that we can runtime change the components with radio communication, so we can program hundreds of boards in no time. The component ability changes our boards to be a multi-functioned hardware. What do we need to do that? A component-based hard real-time embedded operating system with run-time component change ability.



Component uploading/changing

II. Explication

Component-based: The object-oriented aspect is well utilized in PC software development, but not in embedded systems. This software would like to change this. Just look at how children build magnificent things from small building blocks ☺.

Hard real-time: Is there something which is more important than this in embedded systems?

Operating system: The base software, not much, just a couple of functions, can be called by applications, and of course a task manager. What other? We have components…

Run-time component change ability: Well, this is it. We don't want to gather every sensor; we don't want to reset them to update the software on it. Just send them the patches and updates through some communication. Let the operating system do the rest.

III. Resources for the development

To develop this software I'm using a Philips chip called LPC2106. This is an ARM-based microcontroller with 128 Kb non-volatile flash-memory and 64 Kb SRAM on a simple evaluation board built at my university (University of Technology and Economy of Budapest). We plan to build a better board with radio-communication, but it is not finished yet (maybe it won't be ☹). The chip contains a small built-in software I can upload a binary to the flash memory with, and another for the in-application programming of the flash. To compile the arm-assembly, C and C++ programs, I'm using the GNU gcc cross compiler (release: 3.4.1).

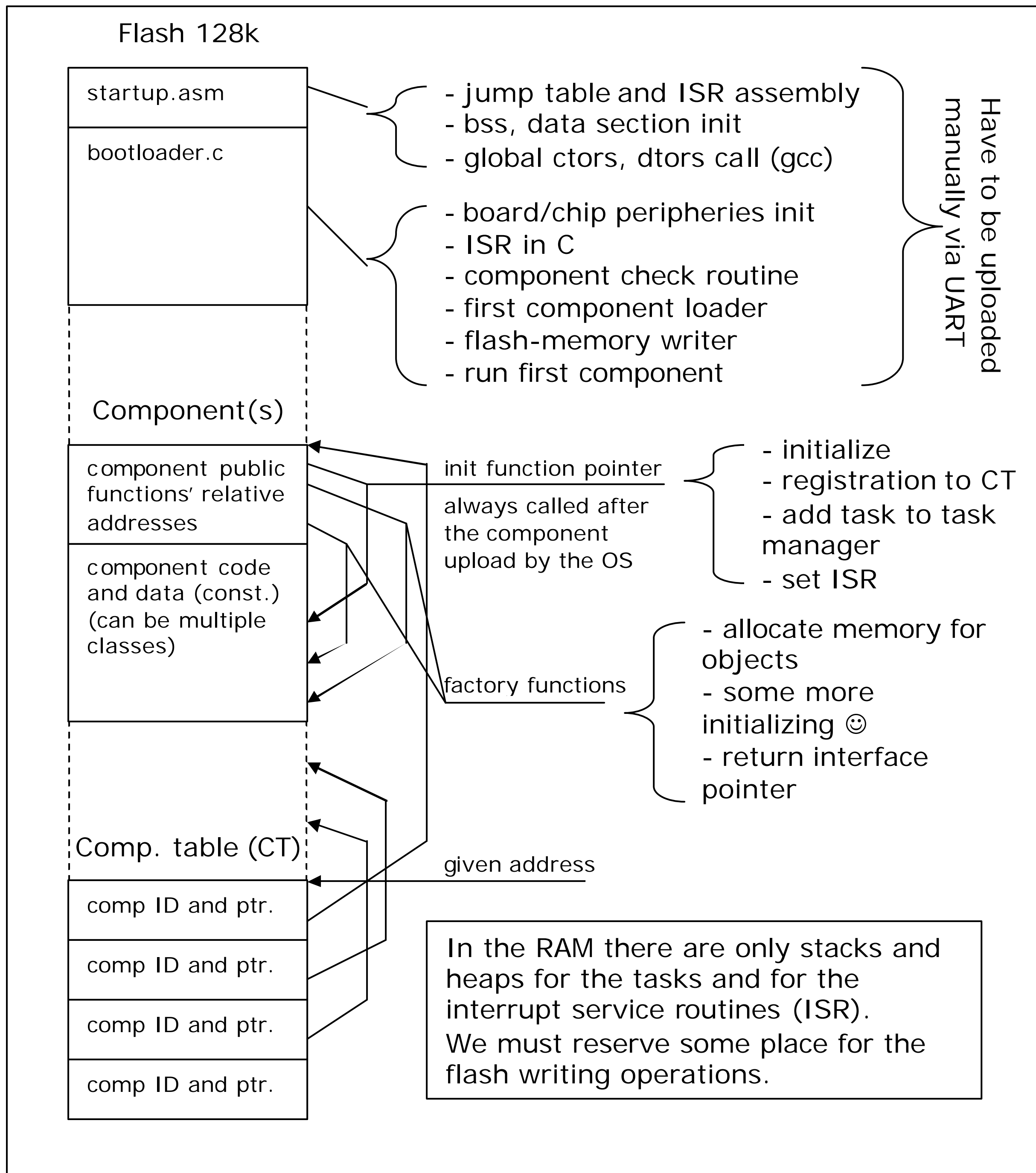
IV. Principles

1. This should be a portable operating system, not only designed for this Philips chip, so the assembly and chip/board specific code will be as minimal as possible.
2. The first small program we have to upload manually has to be as small as possible. Even the operating system should be a component too.
3. We only need the object-oriented aspect to use its component ability, not for the inheritance ability.
4. The software is to satisfy the hard real-time expects in serious situations, but the component changing, software uploading doesn't have to suit these exigencies. The reason is that the flash-writing is slow, and because of the small space on it, maybe we have to use a complex algorithm to place the components' code.

5. No static variable. We don't want to save data on the chip. (This can be changed.)

6. After a component change, we wouldn't like to reset the board. (This is optional; I hope we can do without reset.)

V. First ideas



About the boot-loader

The boot-loader contains only one assembly code for the startup for the chip specific vector table and the bss, data initialization of the second part, the real boot-loader. The startup must be in assembly, but the boot-loader's main part is in C.

This will initialize the needed peripheries for the upload (and maybe for the OS), like the UART (radio), the system clock, the vectored interrupt table and the timer. After this, the boot-loader will check if the flash contains any executable components (it can be a previously uploaded TaltOS too.). If there is something, it will run that, if not, he will expect a component from UART (radio), put it on the flash and run it. From this part, the TaltOS (the first uploaded component) will do the rest.

About the TaltOS

This operating system component should be small too. It contains a task manager (preemptive or not, not decided yet) which manages and schedules tasks. It also contains a component loader. The tasks can call some specific functions, which add or remove tasks, maybe the OS will handle the interrupt service routines too, and the tasks can only ask to change the interrupt vector table.

About the components

The components (also the TaltOS) should be written in C++. There are components which can act as a task, others will provide only functions. There is a function that every component must contain. The init function, what the TaltOS will execute after the upload. This can add tasks or initialize some memory and constant data for himself. There can be factory functions which create objects and return the dispointer. This part will be a bit like COM. The init and the factory functions should be C functions (not in-class methods). We have to find a solution to compile components. Because we won't know the memory address of the code before the uploading, so it has to contain relative addresses.

About the component changing

This will be a privileged state, so no interrupts during this operation. The TaltOS has to manage the flash writing; because of the small space there will be a complex algorithm to range the components.

About the component table

It is not decided yet if the component would contain its ID and other necessary information and after an unexpected reset the TaltOS would build up the component table in the SRAM or there would be an exact place for the component table in the flash memory for the information. The first solution is to see on the figure.

About the components' binary

This binary we upload will be a bit different. It is not specified yet what information would it contain. There will be a relative vector table for sure. During the project we have to specify this, too, some sort of file, which is more than just an executable.