A microkernel for multitasking operation mode on the base of Motorola 68hc11 microcontroller

Eng. Boyko Baev Petrov - Assistant Professor
Eng. Iordanka Vassileva Rouskova - Assistant Professor
Department of Electronics and Electronic Technologies
Technical University - Plovdiv Branch
E-mail address: bpetrov@tu-plovdiv.bg jvr@tu-plovdiv.bg

Introduction

The effective trade-off of a good hardware solution and a multifunctional software is of great importance to the creation of microconroller systems.

Practice shows that microprocessor systems development time is determined mainly by the process creation and testing for specific software requirements of speed and costs for the systems as a whole.

In order to shorten the development time of a given application, specific or standard program modules are most frequently used, such as: MCX, Buffalo arithmetic software libraries etc. The system engineer should do the particular work - to determine the conditions and the order for different application activations.

Now the method of embedded kernel is widely used. The specific applications features are provided as different user tasks. Their activation, the data exchange and the synchronization between tasks are provided by the operating system kernel. This article proposes:

- to divide user tasks into main and background tasks;
- to structure main user tasks with minimized execution time;
- to structure a microkernel with embedded sub-system activating main tasks of fixed priority.

Description of the proposed user tasks' division and of the main task structure

There are three types of user tasks structures [1] - linear, cyclic and combined(fig. 1):

- linear - a task with one input and one output, without internal branches;
- cyclic - a task with one input and one output, with a waiting cycle for incoming activating condition .
- combined - a task with one input and one output, in which a waiting cycle is included in the linear structure.

The use of cyclic and combined structures user tasks activated in real-time operating system seems to demand:

- multitask mode for each task in activated, blocked (wait-state) or passive state.

- information flags - to indicate the current state of each task.

The structures of linear and combined type aren't stated clearly enough to define those moments of time when the program modules activated by special interrupt functional blocks are executed. It is not easy to give a parametric description of the current state , which makes the decision making process for the next activation also very difficult.

In order to overcome these disadvantages a division of user tasks to "main" tasks and "background" tasks is proposed. The main tasks are program modules activated by the operating system. An instruction "return to the operating system" is used as an end of the program module. Background tasks are program modules, activated only if a specified event occurs. This event should be identified by a special integrated circuit or by an element of the microcontroller architecture (i.e. interface interrupts, analog-digital converters, keyboard etc.). A return from interrupt instruction is used for the end of the program module.

The linear, cyclic and combined structures can be used for description of each main tasks along with the current state description. The event control structure (fig. 2) will be more suitable for this purpose. The structure proposed here:

- can work with its own user stack detached from the system one, also detached from all the rest of the tasks;

- includes an instruction for system interrupts authorization;

- executes the main operation of the user task after doing a current check-up - otherwise it is transferred back to the operating system;

- uses an introduction for program interrupt as an end of task.

It should be noted that the description of the principal activity in a user task include a branched or cyclic algorithm without limitation of the number of conditions or cycles included. The only constraint concerns the requirement that for each task no block for check-up of an activating or deactivating condition of a separate program module should be included. If it is really necessary to include such a block - then it should be detached as a separated main task. The advantages of the proposed structure compared to the linear, cyclic and combined structures are:

- the opportunity to work with a task-owned stack and the use of temporary data storage instructions;

- the formation of a free execution zone created by the instruction for system interrupt enable and by the instruction of program interrupts(SWI). When the task execution is located in this free zone all the background for the operating system tasks can be executed;

- in case of a non-executed activating condition, the control is given back to the operating system for activation of the next main task, which conditions the minimal execution time for the task(the cycles for waiting for a condition are absent);

- only one status bit is needed for the current state task description.

Structure of a micro-kernel with built in sub-system for task activation.

A structure of a microkernel for control of user task executions is shown in fig. 3. In order to provide a periodical transfer of the control to other user tasks this part of the operating system is designed to work with a real-time interrupt sub-system(RTIS). This interrupt is an element of the architecture or Motorola's single chip microcontroller M68HC11c.

The methods for obtaining and transferring back the control from and to main tasks envisaged here are - main task activation from the beginning and reactivation of a main task from the point of its interrupt request coming from the RTI sub-system. Under the terms of these methods for activation the return back to the operating system will be respectively- return after the end of a main task and going out from the main task after an interrupt request execution, coming from RTI sub-system.

It is obvious that a main task should be interrupted by a background task. During the execution of a background task, other interrupts may not be executed. After the end of the background task the control is transferred back to the location of the interrupt in accordance with built-in automation of every microcontroller. In this manner the kernel needs only two points for input and only two points for output.

When an activation should come by the first input point *(the previous task being completed to the end)* the kernel has to do the following operations, till a decision to activate a new main task is made :

1. To define the number or the current task;

2. To mark the completed task with a "marker" validated for the following new activation - from the beginning.

When the activation is done by the second input point *(the main task is interrupted by the real-time interrupt system RTI)* the kernel has to complete the above-mentioned tasks, till a decision to activate a new main task is made:

1. To store the context (values of common-use registers) of the interrupted task by saving the content of task-owned stack.

2. To determine the number of the current interrupted task.

3. To mark the interrupted task with a marker valid for the next new activation - "by continuation".

After execution of these operations and for any kind of input point the kernel is ought to:

1. Determine the number of the next main task (in accordance with a fixed, dynamic or any other priority).

2. To check-up the status of a new-coming task - determining if it is from the beginning, or by continuation.

3. To activate the determined task according to its current state.

In case of activation "from the beginning" - the control is transferred to the start point of the determined main task. In case of activation "by continuation" the control is transferred to the point of interrupt, after restoring the main task context (the content of common-use registers at the moment of interrupt).

The described microkernel can support only two possible conditions for the user tasks - "completed" and "interrupted" and can work with its own stack. It has precisely two input and two output points. It requires two operating conditions - available RTI sub-system (depending on the single-chip microcontroller) and an instruction for program interrupt SWI (depending on the system of commands). If an instruction for software interrupt is not available in the chosen microcontroller chip(different from M68HC11c) it can be replaced by a similar one from the set of instructions.

The proposed structure of tasks allows a realization of multitask kernel with fixed task priority. A real microkernel program, written in Assembler language in a volume of 280 bytes has been used by the authors.

The experiments with this microkernel have been done with a program with nine linear and cyclic structure tasks executed for a period of time from 16 μsec to 18 msec. The interrupts used from the available architecture of the microprocessor M68HC11c are-software interrupt instruction with vector SWI and the real-time interrupt from a special timer (by a different vector) RTI.

Conclusion

The main advantage of this approach for interrupt organization is the avoiding of a possible " begin to cycle" and the definition of a time interval for background operation, thus the task can be activated by special interrupt requests, generated by external devices.

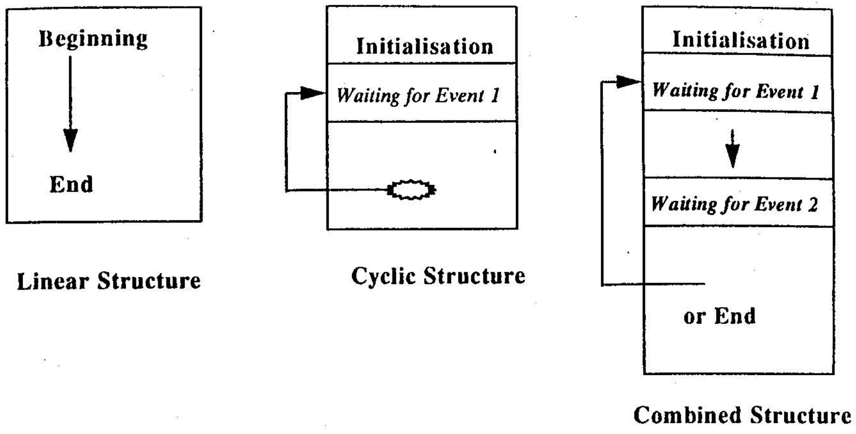
This solution can be implemented for any type of microcontroller, whose architecture is organized by similar interrupts and instructions.

References:

1.Луканчевски М. - "Системно програмиране за едночипови микрокомпютри"-Техника,София, 1993.

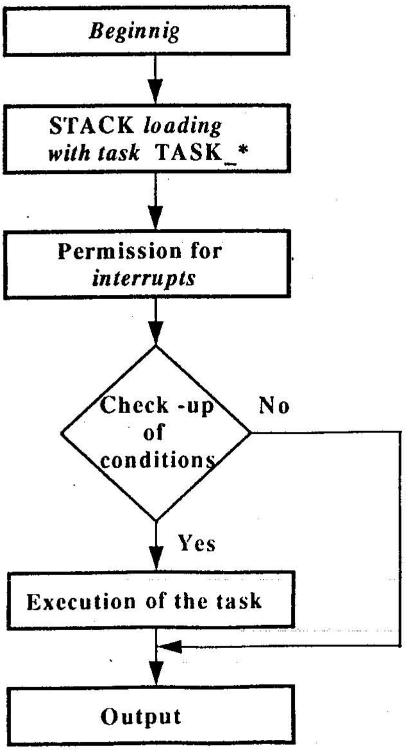
2.Anzinger G.The HP-RT Real-Time Operating System - HP-Journal, August 1993.

3.Motorola single chip microcontroller MC68HC11A8 User Manual.



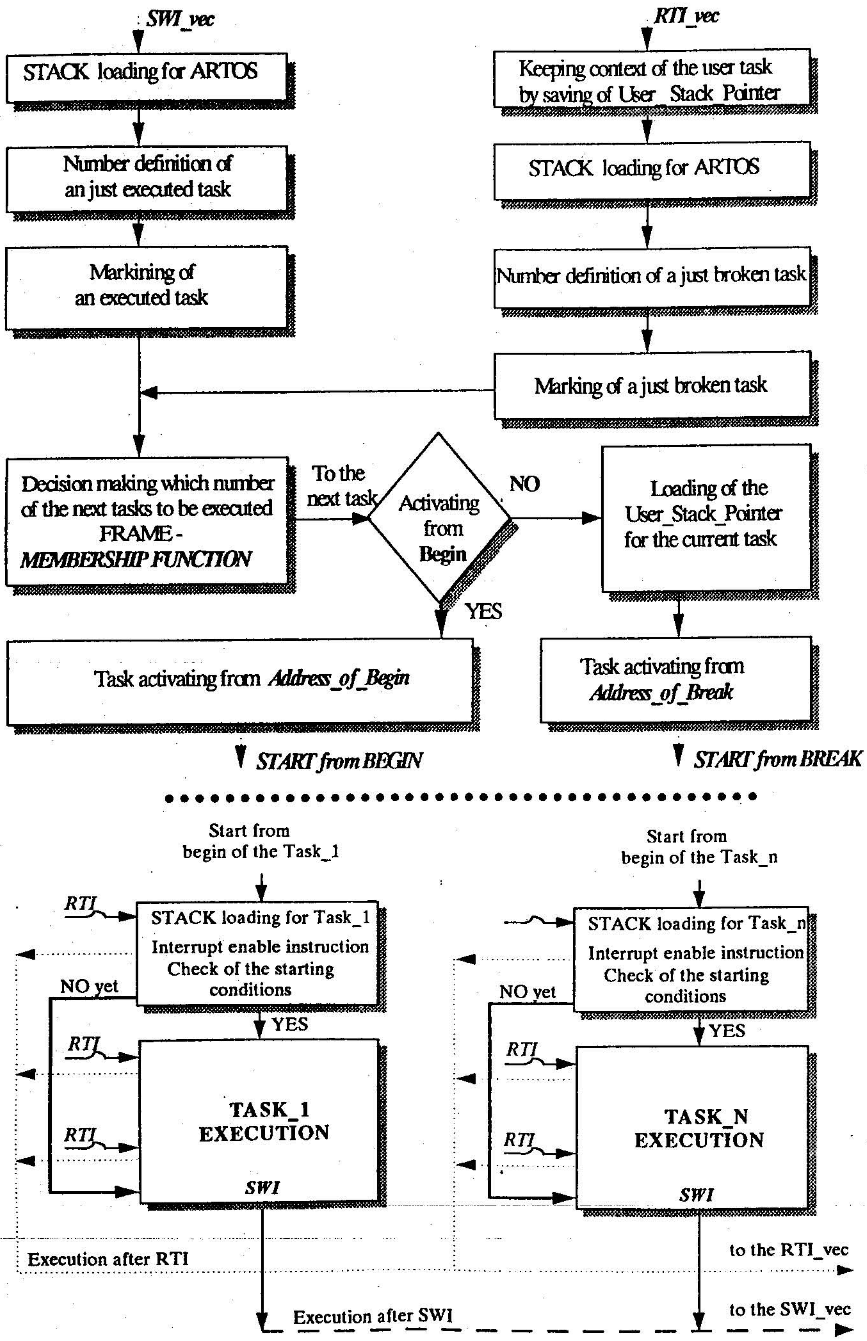
Beginning
End
Linear Structure
Initialisation
Waiting for Event 1
Cyclic Structure
Initialisation
Waiting for Event 1
Waiting for Event 2
or End
Combined Structure

Fig. 1



Beginnig
STACK loading with task TASK_*
Permission for interrupts
Check -up of conditions
No
Yes
Execution of the task
Output

Fig. 2



SWI_vec
STACK loading for ARTOS
Number definition of an just executed task
Markining of an executed task
RTI_vec
Keeping context of the user task by saving of User_Stack_Pointer
STACK loading for ARTOS
Number definition of a just broken task
Marking of a just broken task
Decision making which number of the next tasks to be executed FRAME - MEMBERSHIP FUNCTION
To the next task
Activating from Begin
NO
YES
Loading of the User_Stack_Pointer for the current task
Task activating from Address_of_Begin
Task activating from Address_of_Break
START from BEGIN
START from BREAK
Start from begin of the Task_1
Start from begin of the Task_n
RTI
STACK loading for Task_1
Interrupt enable instruction
Check of the starting conditions
NO yet
YES
TASK_1 EXECUTION
SWI
STACK loading for Task_n
Interrupt enable instruction
Check of the starting conditions
TASK_N EXECUTION
Execution after RTI
to the RTI_vec
Execution after SWI
to the SWI_vec

Fig. 3