DOI: 10.35598//mcfpga.2019.005

Organization Features of Parallel Processes in Programs for Microcontrollers with a Small Amount of Program Memory

Andrij Verygha
Department of the Radio Engineering and Information Security,
Yuriy Fedkovych Chernivtsi National University
Chernivtsi, Ukraine,
veriga@ukr.net

***Abstract*—A way of organizing parallel processes without the use of real-time operating systems is described, which is convenient for writing programs for microcontrollers with a small size of program memory.**

Keywords—microcontroller, program, parallel process

I. INTRODUCTION

Microcontrollers have found wide application in various kinds of radio equipment. Depending on the design task, microcontrollers can be used from the simplest models to the edvanced, which have a significant number of peripheral modules and a large amount of program memory.

The most popular programming languages for microcontrollers is the assembler and C. The assembler language's uncomfortabling is the compiler's dependence on the kernel being used, among which the most popular are PIC, AVR, 8051, and Cortex [1, 2, 3].

In terms of program portability, C / C ++ compilers are more convenient. Using this language, various types of real-time operating systems are written specifically for microcontrollers (RTOS) [4, 5]. They allow you to implement flexible interfaces with different peripherals (displays, keyboards, memory cards, data ports, etc.) and perform parallel processes.

Application of operations system requires a sufficient amount of program memory (> 16 kilowords), which can only be used in complex applications.

In simple applications, the use of expensive microcontrollers is not appropriate. The volume of program memory in this case may not be enough to accommodate the operating system.

II. BRANCHED PROCESSES

Microcontrollers are used not as a separate unit, but as the main controller of the device. It must interact with the entire periphery connected to it. The knowledge and skills of a programmer who writes programs for a computer differ for the programmer of microcontrollers. The style of writing programs is made with experience and years.

For beginners it is typical to use standard software constructions. When organizing a branching of the program, the nested (multi-level) condition statements (if-else, switch-case) are used and often the operators of the unconditional goto transition. Embedded cycles may also be present in conditions and in other cycles. Here is an example of a possible variant of the code of the program.

```
if (Menu)
 {
 if (ff_kl_buton_star) goto endMenu;
 if (menu==1)
 {
 ...
 if (ff_kl_buton_star) goto endMenu;
...
//- - - - - - - - - - - - - - - - - - -
  if (submenu==1)
   {
   …
   do
   {
   …
   if (!ff_kl_buton_star)
    {
    …
     if (Condition_1)
      {
      while (1)
       {
       ...
       if (Condition_2)
        {
        if (Condition_3)
         {
         ...
         }
wait:
        if (Condition_3) break;
        if (!ff_KeyRead) goto wait;
        if (Condition_4)
         {
         ...
         }
         else
         {
         ...
         }
        …
        }
```

```
      else
      {
      ...
      break;
      }
     }
    }
    else
    {
    ...
    break;
    }
   }
  }
  while (!ff_kl_buton_star);
  break;
  }
//- - - - - - - - - - - - - - - - - - -
 if (submenu==2)
  {
  …
  }
//- - - - - - - - - - - - - - - - - - -
 if (menu==2)
 {
 …
 }
}
…
endMenu:
…
```

This design can be observed when organizing complex multi-level menus. To exit the menu item or the menu at all, you need to insert it in the checking program for a certain condition (highlighted in bold). The output is complicated by the presence of cyclic constructions.

When you exit the submenus of the menu sometimes there is a need to restore certain settings. If the transition after the recovery is always at the same point of the program, then you can use the goto unconditional transfer operator to part of the recovery code. From this place go to the given point. If at different points - it will have to organize a separate subroutine.

This approach leads to an increase in the number of transfer marks, the allocation of the program code block with curly braces ({}). The number of errors during the writing of the program increases. This increases the time it is written. It also complicates the process paralleling and interrupting, the program's response to interrupt from the peripheral modules of the microcontroller. For such an approach is inherent in the redundancy of the code, correspondingly, the volume of the program increases.

III. Processing Action Processor

It is suggested to apply a slightly different approach to building branched and parallel processes as a simplified RTOS replacement.

In the program, enter a block (or procedure) that will track signals from peripheral modules, interrupts, internal program processes, signs of execution of program blocks, prioritize, assign the required code of action. Let's name the block of program PROCESSING ACTION PROCESSOR (PAP). An example of building a program is given below.

```
while (1)
{
//PAP
…
//Actions
switch (Action)
      {
case 1:// Action 1.1
      {
      …
      break;
      }
case 2:// Action 1.2
      {
      …
      break;
      }
case 3:// Action 2
      {
      …
      break;
      }
…
default: break;
}
…
}
```

Actions are divided into more elementary. They should be through-cross, it is desirable to avoid cyclic structures inside the action. The PAP block and actions are performed in an infinite loop. You can delay one action and give permission to others, repeat the same action the required number of times.

IV. Conclusion

Using the processor to process operations and split actions into elementary operations simplifies the writing of programs and reduces the program code volume after compiling approximately 1.1-1.5 times (depending on the optimization of the software code).

References

[1] "8-Bit MCUs | Microchip Technology", *Microchip.com*, 2019. [Online]. Available: https://www.microchip.com/design-centers/8-bit. [Accessed: 18- Jun- 2019].

[2] "8051 Family - Nuvoton Direct", *Nuvoton Direct*, 2019. [Online]. Available: https://direct.nuvoton.com/en/8051-family/. [Accessed: 18- Jun- 2019].

[3] "Microcontrollers - STM32 Arm Cortex MCUs - STMicroelectronics", *St.com*, 2019. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. [Accessed: 18- Jun- 2019].

[4] "Free RTOS for ST STM32 ARM Cortex-M3 microcontroller", *Freertos.org*, 2019. [Online]. Available: https://www.freertos.org/portstm32iar.html. [Accessed: 18- Jun- 2019].

[5] "FreeRTOS Kernel - The FreeRTOS kernel is an open source real time operating system and the de-facto standard solution for microcontrollers and small microprocessors - STMicroelectronics", *St.com*, 2019. [Online]. Available: https://www.st.com/en/embedded-software/freertos-kernel.html. [Accessed: 18- Jun- 2019].