HBRP PUBLICATION

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

Micro Operation System for Microprocessor Applications

Giriraj Soni[1]*, K. B. Ramesh[2]
[1]UG student, Department of Electronics & Instrumentation Engineering,
R V College of Engineering, Bengaluru, India
[2]Associate Professor, Department of Electronics & Instrumentation,
R V College of Engineering, Bengaluru, India
Corresponding Author
E-Mail Id: girirajsoni.ei20@rvce.edu.in

ABSTRACT

The microprocessor is an extremely helpful device for our modern communication. A chip is a computer processor where the data processing logic and control is remembered for a solitary integrated circuit, or few integrated circuits. The performance of any computer is vastly dependent on them. In this paper Real-time applications for embedded systems often use microprocessor systems. Especially when using single-chip microprocessors, there are limitations with the size of the operation and program memory and those are disadvantages to using conventional RTOS, which occupy a superfluous measure of memory, and the majority of their administrations will stay unused. The microprocessor contains the arithmetic, logic, and control circuitry expected to fill the roles of a central processing unit. The incorporated circuit is equipped for deciphering and executing program guidelines and performing arithmetic operations. The presented system includes subsystems:

(a) The cooperative management as many as eight to sixteen tasks for a time-independent role
(b) Pre-emptive multitasking for time role management. In this mode are tackled objective undertakings of numerical control and execution of PID regulators.

The proposed solution will bring the simplified design of digital control applications when the commercially delivered applications are unnecessary robust and solve tasks like file management etc.

Keywords: *Microprocessor, CPU, IC*

INTRODUCTION

The expression "micro" signifies extremely small and "processor" signifies what speeds up assignments. So in the overall sense, the term, microprocessor‟ implies something small that can speed up various assignments as requested.

A microprocessor is a computer processor where the data processing logic and control is included on a single integrated circuit, or a small number of integrated circuits. But the actual definition of a microprocessor is slightly different than this. A microprocessor is a tiny electronic chip containing transistors found inside a computer’s central processing unit and other electronic devices. Its essential capacity is to take input, process it, and afterward give suitable result.[1,2]

A microprocessor is a computer processor that incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits. The

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

microprocessor is a multipurpose, programmable device that acknowledges digital data as input, processes it as per guidelines put away in its memory, and gives results as result. Microprocessors contain both combinational rationale and successive digital logic.[3]

Microchips work on numbers and images addressed in the binary numeral system. The combination of an entire CPU onto a single chip or on a couple of chips significantly decreased the expense of handling power. Integrated circuit processors are delivered in enormous numbers by exceptionally automated processes bringing about a low for each unit cost. Single-chip processors increment unwavering quality as there are numerous less electrical associations with come up short. It is shown, that the fitting methodology is to isolate handling of input signals, custom application and result signals into CPU peripherals. To synchronize the cycles in implanted applications is adequate to execute the wait states, the sleeps state, and some synchronization implies. Introduced system incorporates subsystems: (a) the helpful administration upwards of eight to sixteen errands for time free role and (b) Pre-emptive performing various tasks for time job the board. In this mode are tackled objective undertakings of numerical control and execution of PID regulators.[4,5]

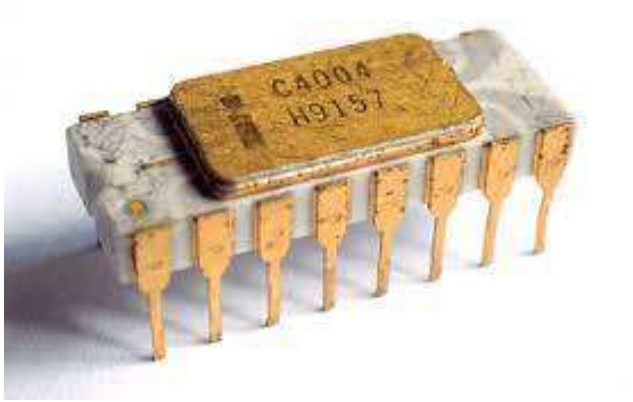


Fig. 1: *INTEL4004.*

The proposed solution will bring simplified design of digital control applications, when the commercially delivered applications are unnecessary robust and solve task like file management etc.

Proposed arrangements will diminish undertakings the executives the base, so that limits memory requests of the microprocessor units with giving essential administration errands.

Microchips are typically made of silicon and frequently called "Logic chips" or as it were "Processors". The first-generation microprocessor family was of 8-cycle. In any case, today we utilize both 32-cycle and 64-bit microchips.

WHAT IS A RTOS?

Give a free item that outperforms the quality and administration requested by clients of business choices

Basic Functions of a RTOS

There are commonly features implemented into RTOS systems. A number of objects (such as tasks and queues) are dynamically created and destroyed at run-time, so most of RTOS provides a scheme to ensure correct allocation of memory and typically

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

need a mechanism to manage memory. One of the utmost basic elements of an RTOS is a task. Depending upon the RTOS the code may be written to run once through to completion or it may enter the typical while loop. Assignments can be dynamic made and erased or they should be in every way made at assemblage time. To choose the following undertaking to run each errand has appointing a need and the RTOS chooses the most noteworthy one.

Mostly the user must ensure that each task has a different priority. The simple tasks which are variously called 'co-routines' typically share a single stack and may be restricted in the functions. RTOSs use different scheduling systems and algorithms such as fully pre-emptive RTOS or a cooperative RTOS.

In case of a co-operative RTOS the user must write software so that the tasks execute and then yield control to other tasks at suitable points. A RTOS where many assignments can have a similar need will commonly utilize a 'round-robin' planning algorithm to guarantee that everyone gets a portion of the processors assets.

Rather than directly editing the RTOS source code many RTOSs provide the ability to attach functions or code into parts of the kernel itself. A RTOS with a large memory footprint may appear low performance. On the other hand it can provide a large range of functions that reduce programming and such solution can be better solution than a small fast kernel is used.

A modern RTOS provides a large number of functions that can be used to create a user program, but the final application may spend more time executing the kernel rather than the user's code. It is important that the RTOS is fast and any functions it contains are efficient.

A RTOS can disable interrupts while it switches between executing tasks. If a high priority interrupt arrives it may not get serviced until the RTOS re-enables interrupts again. This introduces unwanted jitter in the application response time.

Fig. 2: *State Machine of machine task and structure of a subtask.*

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

RTOS Implementation
The basic idea of suggested system is to implement system of tasks scheduling using in PLC machines. Such system consists of tree type of tasks:

- Main task, which is over and over executed in typical need level, this assignment can be hindered by a higher need quick errand and by a hinder undertakings;
- Fast task, has higher priority than main task and it can be interrupted only by interrupt tasks;
- Interrupt task has the highest priority, it cannot be interrupted by any other tasks;
- Auxiliary tasks have lower priority than main and interrupt tasks.

State Machines
The state machine represents a system of separate states and transition between these states. The transition is provided when fulfilling a corresponding condition. There are two basic models of state machines.

Moore Machine
The FSM uses only entry actions, i.e., output depends only on the state. The benefit of the Moore model is an improvement of the conduct. Consider a parking gate. The FSM has four states-gate" and "closed-gate". The state machine recognizes four commands: "open-gate", "close-gate", "is-opened" and "is-closed" which trigger state changes.

The passage activity in state "Opening" turns over an engine opening the entryway, the section activity in state "closing" turns over an engine in the other bearing shutting the door. The command "is opened" turn FSM from "opening gate" state to "opened gate" and so on. States "Opened" and "Closed" stop the engine when completely opened or closed. They signal to the rest of the world (e.g., to other state machines) the circumstance: "door is open" or "door is closed"

Mealy Machine
The FSM utilizes just information activities, i.e., yield relies upon info and state. The utilization of a Mealy FSM drives regularly to a decrease of the quantity of states. By and by blended models are regularly utilized.

The Main Task Implementation
The principle errand can be carried out as a composite state machine. It is mean that the whole primary undertaking is made as a set out of composites states.

Every one of this composite state addresses one basic subtask of the primary assignment.

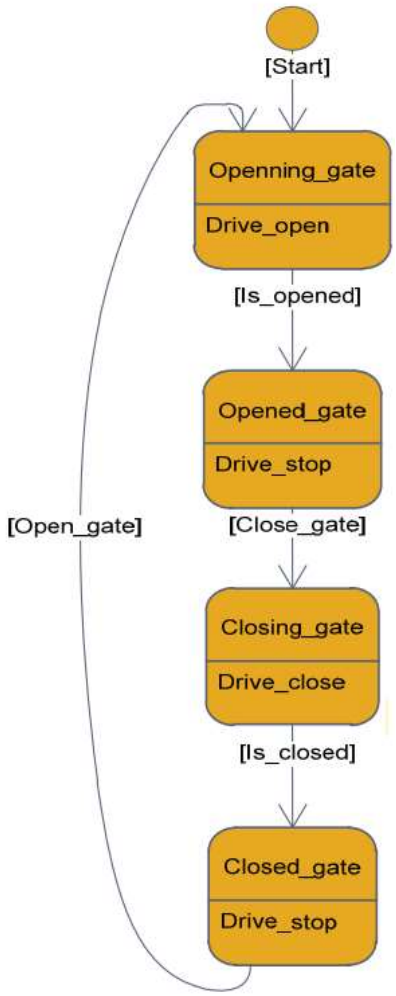
Utilizing clear programming method each undertaking can be executed into a while loop:

```
Main task ()
{ while (true)
{ call substate_1();
 call substate_2();
…
call substate_n(0);
 }
}
```

The substate may have following structure:

```
subtate_n()
{
   switch (state)
{
 case state_1: provide_state1();
 break;
 case state_2: provide_state2();
 break;
 …
case state_n: provide_staten();
}
 }
```

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]



Moore Machine of a Gate Automata

The state variable has been a worldwide variable. This program design might be addressing as a state machine. Design of a created state is given on figure. Transitions S1, S2, S3 and S4 chose the real state, for instance S1 progress select the State1, etc. Advances T1, T2, T3 and T4 are changes to the following stage

```
structure main_task_descriptor
{
 int number_of_tasks;
 pointer entry_task[n];
pointer actual_entry_task[n];
int mask_task;
 int actual_task;
}
```

The value of the number_of_tasks variable is the number of all task, entry_task[n] contains an entry point of echo of the total number n tasks, where the n is the constant of maximum implemented tasks.

Variable actual_entry_task[n] include actual state address of each of all implemented tasks, the mask mask_task is a variable, which bits masks corresponding tasks, that set bit mean unmask, running task, zero bit mean mask task, which is exclude form executing. The actual_task variable consists of the number of the actually executed task.

The implementation of each state may be done using macros.
The first of them:

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

setStateMachine n
stores the actual program counter to the main_task_desriptor[n]. The second of macros,
returnStateMachine
jump to entry point of the task scheduler. The third
returnStateMachineAndContinue n
macro combines the actions on the two earlier macros. At first it stores the real address + length-macro to actual-entry-task[n]. Then jump to entry point of the task scheduler. Every task instigates with macro:

entryPointTaks n.
It increment number_of_tasks variable and store the actual address to actual_entry_task[n] and entry_taks[n].

This dual storing preserve the task reset. The main part of the task scheduler does:

```
while (true)
{
sheduller_entry_point:
    do
       {
       if (main_task_descriptor.
actual_task > main_task_descriptor.
number_of_tasks)
        {
             main_task_descriptor.
         actual_ task=0;
             break;
        }
       main_task_descriptor.
actual_task++;
    }
    while (mask_task &&
(main_task_descriptor. actual_
task>>1))
    jump main_task_descriptor.
actual_entry_task [actual_ task ];
}
```

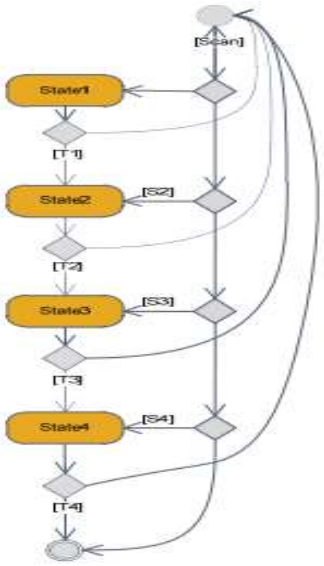


Fig. 3: *Implementation of one subtask as a state machine.*

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

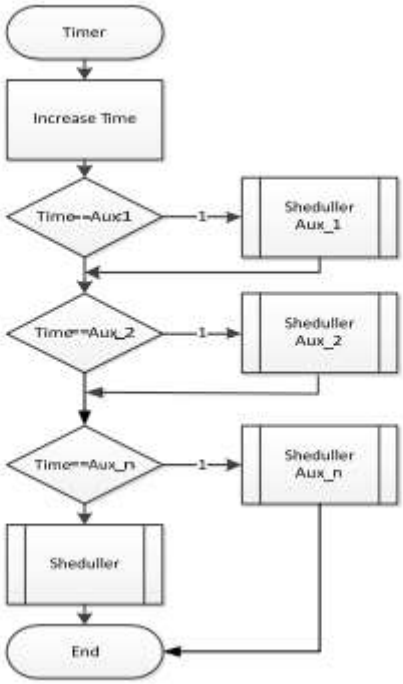


Fig. 4: *Structure of tasks and timing.*

In the main loop while jump to actual entry point of each of unmask tasks. The loop breaks when all task are executed. The mask task has set to zero corresponding bits in mask-task variable.

The periodically called auxiliary tasks have the same scheduler, which is call in predefined time period. Interrupt task are implemented as interrupt routine.

CONCLUSIONS

The proposed method minimizes the memory occupation of the controller and with the only minimum processor, the resource will implement a very light RTOS system. Only a minimum number of instructions is used for switching tasks. Then lead to some restrictions such as the routines must be re-entrant.

The implementation of periodic tasks is based on the timer interrupt and the principle of the scheduling is the same used in non-periodic task scheduler.

REFERENCE

1. Posadas, H., Adamez, J. A., Villar, E., Blasco, F., & Escuder, F. (2005). RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. *Design Automation for Embedded Systems, 10*(4), 209-227. https://www.researchgate.net/publication
2. Barkalov, A., Titarenko, L., & Hebda, O. (2010). Synthesis of Moore finite state machine with nonstandard presentation of state codes. *Przeglad Elektrotechniczny, 86*(9), 134-136.
3. Dziurzanski, P. (2003, June). Modelling of complex systems given as a mealy machine with linear decision diagrams. In *International Conference on Computational Science* (pp. 758-765). Springer, Berlin, Heidelberg.
4. Krejcar, O. (2009). Problem solving of low data throughput on mobile devices by artefacts prebuffering. *EURASIP Journal on Wireless Communications and Networking*, 2009, 1-8.

Journal of VLSI Design and its Advancement
Volume 4 Issue 3
DOI: [To be assigned]

5. Krejcar, O., & Frischer, R. (2010). 590. Detection of Internal Defects of Material on the Basis of Performance Spectral Density Analysis. Journal of Vibroengineering. 12(4), 541-551.