Feedback Control Scheduling for Crane Control System

Oumair Naseer

School of Engineering, University of Warwick

Coventry, United Kingdom

THE UNIVERSITY OF WARWICK

Outline

- Goal of This Research Work
- Control Scheduling Co-Design
- Feedback Control Scheduling (FCS)
- Control Scheduling Co-Design Tools
- Trade-offs
- FCS Updated Architecture
- FCS Integration with Hardware/Software Co-Design Level
- Practical Implementation: Crane Control System
- FCS Integration with Crane Control System
- Experiments
- Results
- Conclusion
- Tool Demonstration

Goal of This Research Work

- The goal of this research work is to provide:
  - Direct practical application: Over the past few years, there is no or a very small amount of work is done on the practical side. That's why only a few real time systems, having feedback based control scheduling implemented, are actually deployed.
  - Software level implementation and realization of FCS: Control scheduling co-design tools have their own limitations. e.g. Truetime is used for simulation based analysis. Jitterbug is used to evaluate the effects of latencies, lost samples, and jitter on control performance.
  - Close the gape between control theory and computing techniques.

Control Scheduling Co-design

- Control scheduling co-design takes into account both the control techniques and the real time computing aspects simultaneously at design level, Fig1.1.
- Real time computing systems integrated with feedback control theory are more robust against internal and external disturbances.
- All classical real time scheduling algorithms are open loop [1] as such they don't have any feedback from CPU or task controller.

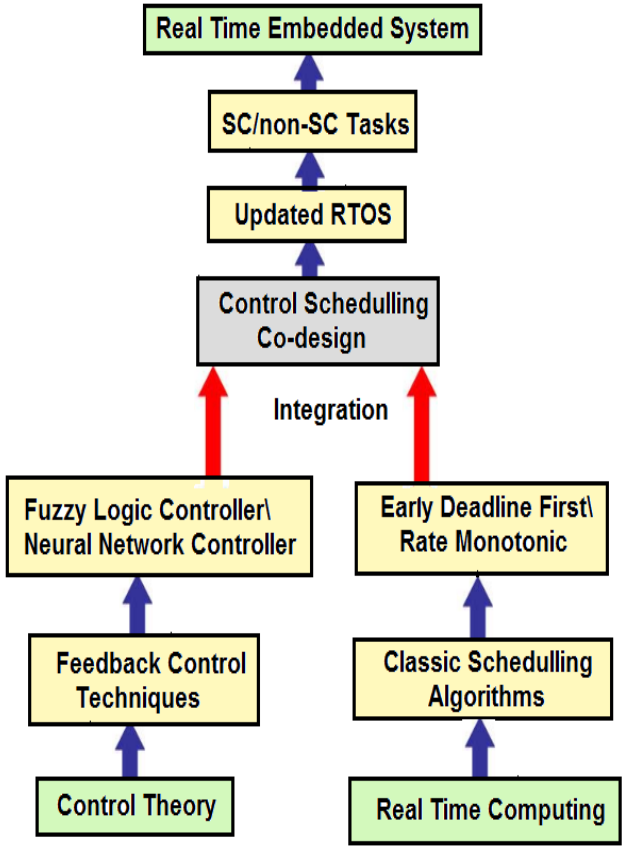


Figure 1.1 Basic architecture of Feedback based Control Scheduling co-design.

[1] C. Lu, J.A. Stankovic, G. Tao, S.H. Son, "Feedback control real-time scheduling: framework, modeling, and algorithms", Real-time Systems, Vol.23, No.1/2, pp. 85-126, 2002.

Feedback Control Scheduling

- Given: Limited resources and task set. FCS: To distribute resources among tasks based on the feedback from task and CPU to provide quality of service in terms of CPU utilization and resource management, Fig1.2.
- Inputs: Time periods of the tasks.
- Output: CPU utilization.
- Classical scheduling algorithms especially: Rate Monotonic and Earliest Deadline First cannot achieve the optimal possible quality of service level that features real time constraints and requirements.

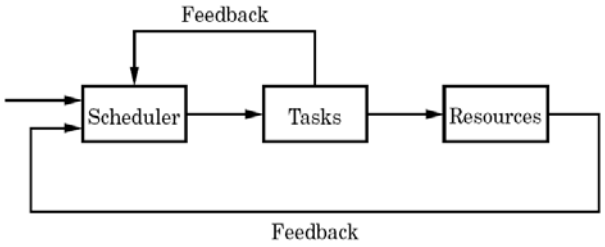
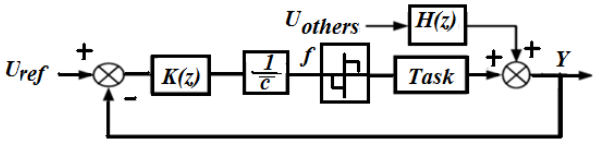


Figure 1.2 A general feedback scheduling structure. The resources are distributed among the tasks based on feedback from the actual resource use.



$$\widehat{U}_{(kh_s)} = \frac{(1-\lambda)}{z-\lambda} \sum_{i=1}^{n} \bar{c}_i(kh_s) f_i(kh_s)$$

Where h is the sampling frequency currently assigned to the task (i.e. at each sampling instant $(k+1)h_s$) and $\bar{c}$ is the mean of its measured job execution-time. λ is a forgetting factor used to smooth the measure. $\widehat{U}_{kh_s}$ is the processor load induced by task at time k and period h_s .

Control Scheduling Co-design Tools

- Jitterbug:
  - Commonly used for statistical analysis i.e. how timing affects the control performance
  - Useful to evaluate the effects of latencies, lost samples, and jitter on control performance.
  - Jitterbug guaranties stability of controller, if cost is finite and statistical analysis is only limited for linear systems and only quadratic performances are measured.
  - It also uses various assumptions such as; time delays are assumed to be independent from period to period and delay characterctics are constant over time.
- Truetime:
  - Commonly used for simulation based analysis i.e. how timing affects the control performance.
  - Useful to investigate the un-deterministic timing affects e.g. control task preemption or transmission delays and to adjust the controller dynamics dynamically.
  - Provides computer and network blocks for detailed analysis of control system against samples, jitters and delays. Computer blocks are event driven and the selection of the scheduling algorithm for individual computer block is user dependent.
- AIDA & XILO:
  - Support fault injection as well.

Hw/Sw Co-design Tools

- Industrial Approaches:
  - Sandars co-design methodology
  - Lockheed Martin ATl co-design methodology
- Academics Research:
  - Chinook Hw/Sw co-design methodology, **University of Washington** - Chou, Ortega, Borriello
  - Siera Hw/Sw co-design methodologies, **Grenoble University** - Ismail, Jerraya
  - Cosmos, **University of Braunschweig** - Ernst, Henkel, Benner
  - Cosyma**, U. C. Berkeley** - Chiodo, Giusto, Jurecska, Hsieh, Sangiovanni-Vincentelli
  - Polis, **U. C. Berkeley** - Kalavade, Lee
  - Ptolmey**, U. C. Berkeley** - Srivastava, Broderson
- Recently emerging control integrated tools:
  - PtolmeyII, extended
  - Metropolis, extended
  - MetaH, Vestal, Steve.
  - RTSIM, JIN, Guo-zhe, and Shu-yu
  - SIMICS, Magnusson, Peter S., Magnus Christensson, Jesper Eskilson, Daniel.

Trade-offs

- Unfortunately, the design of embedded control systems is often based on the principle of separation of concerns [2] of control and computing. Because of that, the tools which are built on the basis of control theory don't have fault tolerance at software level. Similarly, Hw/Sw co-design tools don't integrate feedback control theory.
- We will explore these challenges from practical perspective:
  - How feedback controller is integrated with the overall HW/SW co-design cycle?
  - What kind of software modifications are required to facilitate dynamic task periods allocation? What kind of hardware modifications should be made to enable continuous CPU utilization monitoring?
  - How feedback controller should be implemented with the current of-the-shelf RTOS scheduler?
  - In the presence of faults, how execution time of control tasks varies? Will feedback scheduler be able to compensate the execution time variation in user-defined tasks while keeping the scheduling intact?
  - What scheduling parameters should be measured offline (prior to scheduling)? What scheduling parameters should be measured online (dynamically while scheduling)?

[2] K.E. Årzén, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha, Integrated control and scheduling. Technical Report ISRN LUTFD2/TFRT7586SE. Lund Institute of Technology, Sweden, 1999.

FCS Updated Architecture

- Tasks are first submitted to the submission task queue by task controller. Fig 3.1.
- Based on the quality of control level, tasks are de-queued from submission task queue and en-queued to accepted task queue.
- Tasks are divided into two categories: Safety Critical tasks (SC) and non- Safety Critical tasks (non-SC).
- For safety requirements, Fault tolerant scheme: replication with check-pointing is integrated with FCS. Fig3.2

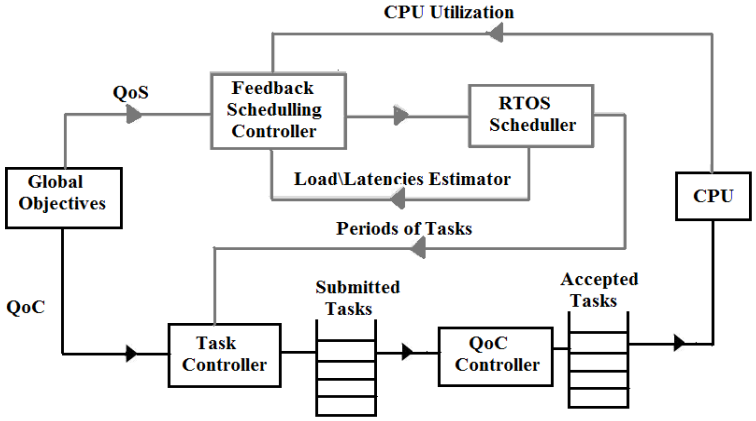


Figure 3.1 Modified FCS architecture.

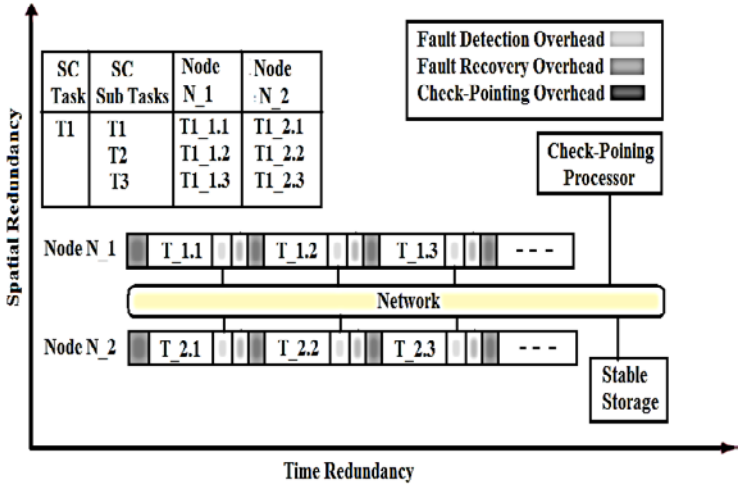


Figure 3.2 SC and non-SC task Scheduling.

FCS Integration with HW/SW Co-Design

- Feedback control theory is analysed before the functional description of the overall system in the design cycle, Fig 4.1.

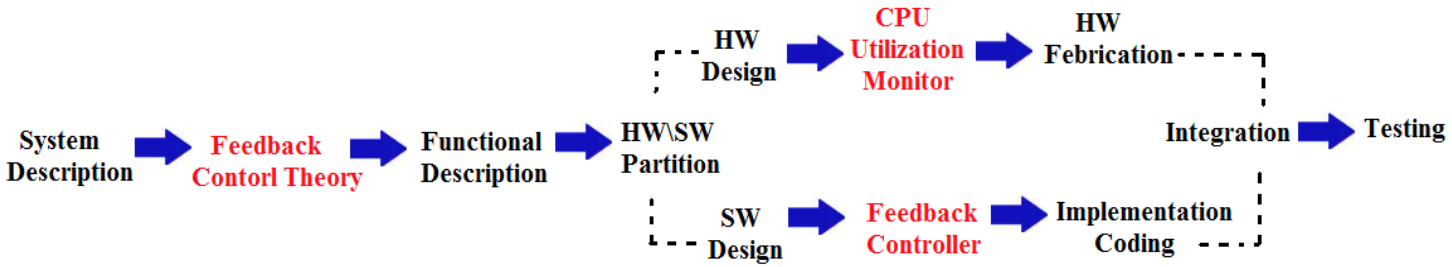


Figure 4.1 Feedback Control Scheduling integration with Hw/Sw Co-design cycle.

- During feedback control theory integration, the most important step is the selection of the feedback controller and scheduling parameters.
- On software side FCS is implemented on the top of Real Time Operating System (RTOS).
- On hardware side a continuous CPU utilization monitoring is provided.

Crane Control System

- Crane control system consists of two major parts: Operator Control Unit (OCU) and Machine Control Unit (MCU), Fig1.2.
- In control society, crane control system is designed based on the principles of separation of concern of control and computing in the form of control tasks only.
- In computing society crane control system is designed as a hard real time embedded system using earliest deadline first scheduler in the form of safety critical tasks.

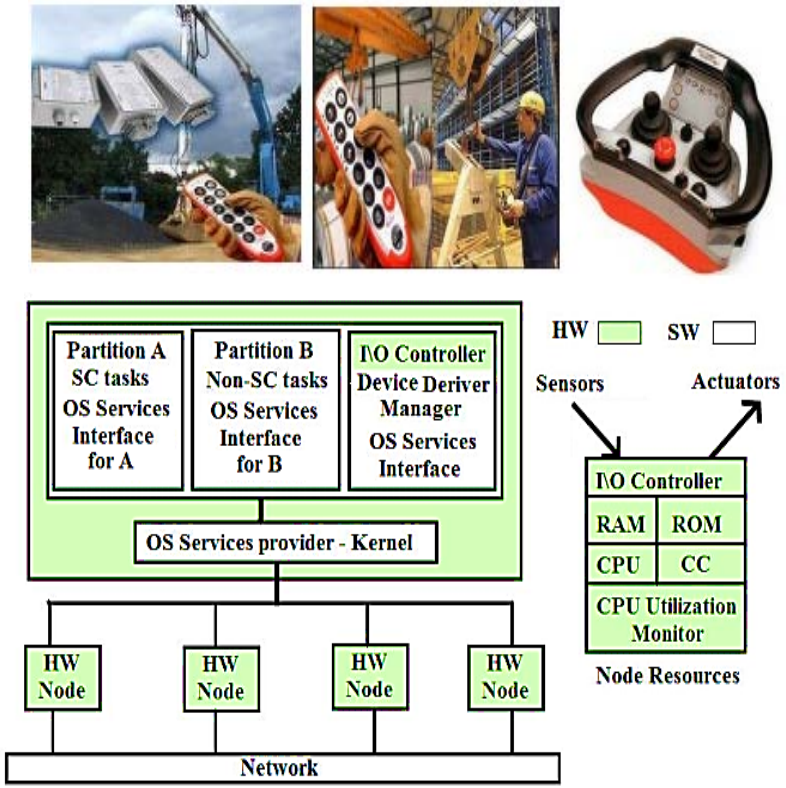


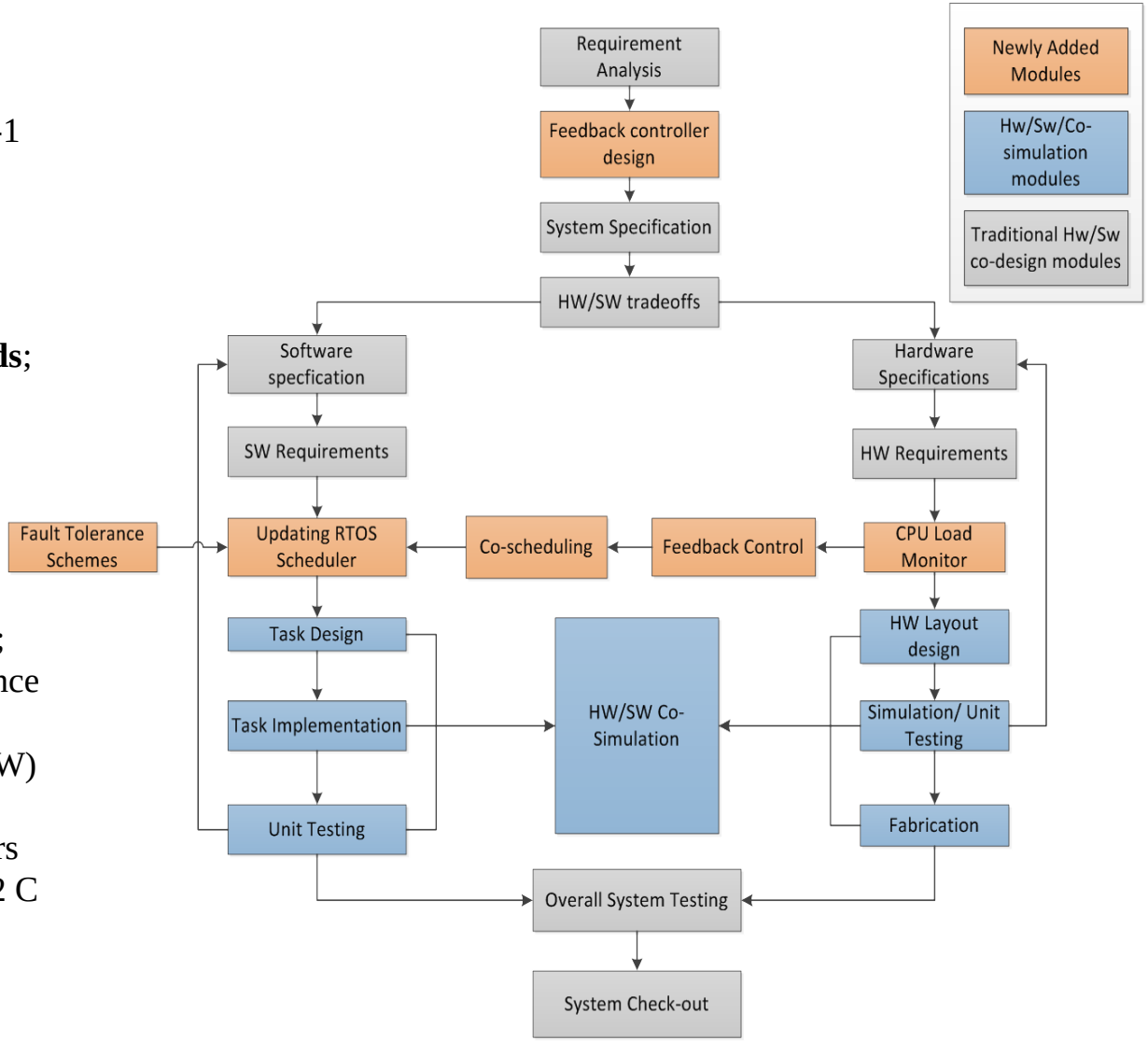
Figure 2.1 System architecture model follows the integrated approach where each core is divided into two parts. Part A executes SC tasks and Part B executes non-SC tasks. SC and non-SC tasks can execute at any hardware node.

OCU/MCU Design Diagram

Safety Standards;
- ✓ ISO EN 13849-1
- ✓ EN 954-1

Motor Industrial Software Standards;
- ✓ MISRA C
- ✓ Hungarian Notation
- ✓ CCCC
- ✓ SQMlint

Development Tool;
- ✓ High Performance Embedded Workshop (HEW)
- ✓ Renesas Microcontrollers M16C and M32 C family



Development Tool;
- ✓ Auto CAD

Hardware Modules;
- ✓ RF module
- ✓ Battery
- ✓ CAN Open
- ✓ Controller Board

Testing;
- ✓ Unit Testing
- ✓ Functional Testing
- ✓ Field Testing

FCS Integration with OCU at Software Level

- OCU is a hand held unit with two microcontrollers connected through an I2C bus network for inter-processor communication. Fig 5.1
- OCU has eight three level push button keypad unit. Each button has three levels to control the speed of MCU (Crane). There is an emergency stop button to stop the MCU in case of any emergency.

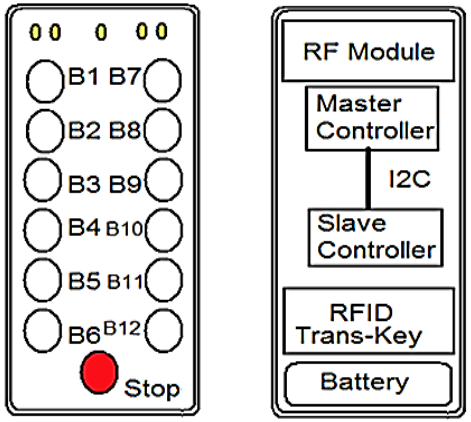


Figure 5.1 Operator Control Unit (OCU) architecture.

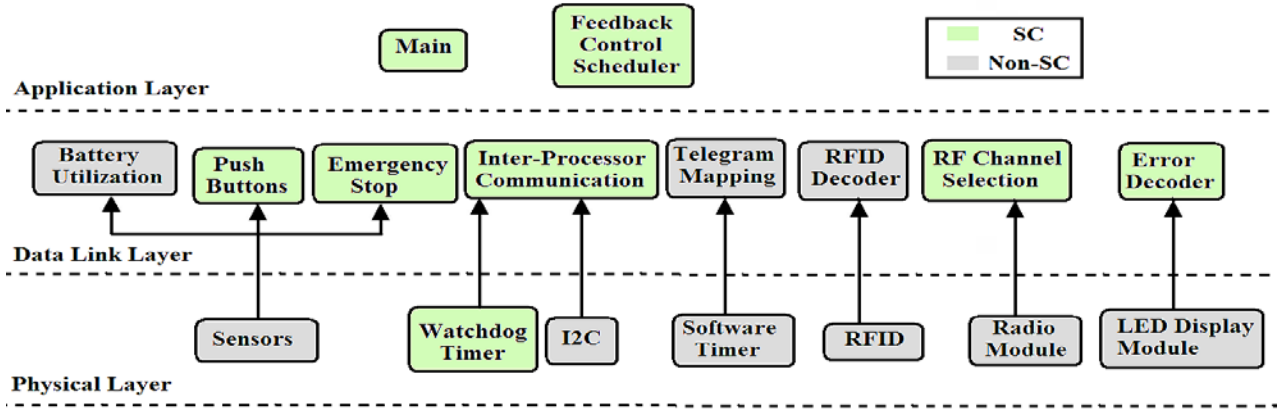


Figure 5.2 Three layed architecture of OCU.

- The application layer contains Feedback Scheduler Controller. FCS is implemented as a part of pre-emptive EDF scheduler. Fig 5.2.

FCS Integration with MCU at Software Level

- MCU mainly consists of relays. There is a main relay associated to each MCU, which activates when emergency stop button is pressed. Fig 6.1.

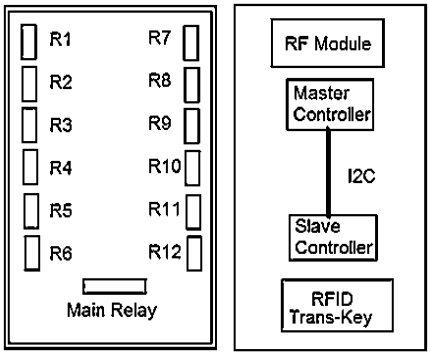


Figure 6.1 Machine Control Unit (MCU) architecture.

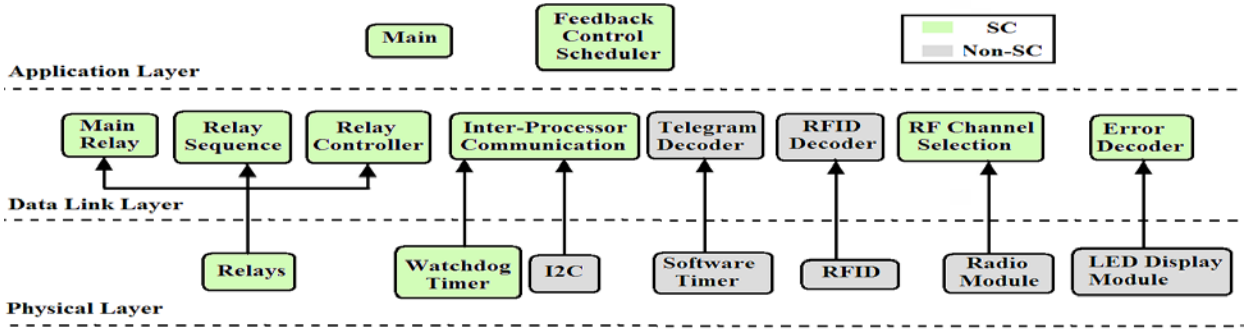


Figure 6.2 Three layerd software architecture of MCU.

- Application layer consists of the main module and the Feedback Control Scheduler. FCS is implemented as a part of Real Time Operating System. Fig 6.2.

FCS Integration with RTOS

- RTOS scheduler is updated with FT scheme integration and with a feedback loop.
- CPU utilization is continuously monitored by CPU utilization monitor.

Sr#	Modified Functionality	Library File Name	Description
1	Fault tolerance scheme (Check pointing based re-execution)	Libckpt	Compiler based check-pointing library.
		Libft	Programmer based check-pointing library.
2	Control Switching	Ctrl_Swth	Saves the content of task to a stable memory and transfer control to the other task.
3	Multitasking	Multi_Tasking	Uses jitter Controller services to perform multi-tasking.
4	Jitter Controller	Jter_Ctrl	Jitter controller avoids jitter by introducing a tiny delay between previous task execution end point and latest task start point.
5	Inter-processor Communication	InterPros_Com	Uses the services of Network Communication protocol such as I2C for inter-processor communication.
6	Stack Controller	Stack_Ctrl	Manages stack (Minimum stack size available before task can start execution).

FCS Implementation

- Safety Standards: ISO EN13849-1, EN954-1
- Embedded Software Coding Standards: Motor industry Software Reliability Association. C Programming (MISRA C).
- Code analysis and documentation standards: Hungarian notations and CCCC.

```
76  While(Accepted_Task_Queue.IsEmpty() != True)              // Checking if there are tasks-
77  {                                                         // avaliable in Accepted Task Queue.
78  ...
79
80      Current_Task = Submitted_Task_Queue.Dequeue();      // Getting the Current task status.
81
82      if (vCPU_ OverUtilized() == TRUE)
83      {
84          vFB_Get_Qos_Level();                    // Latest value of Quality of Services level.
85          vAssign_Level_Qos_Controller();
86          iTask_Period = unGet_Current_Task_Period();  // getting current task period.
87          if(fTask_Period == High)                      // Checking if the task period is high.
88          {
89              vUpdate_Task_Period_Low(Current_Task); // Updating the task period by assigning-
90          }                                           // new task period.
91          Accepted_Task_Queue.enqueue(Current_Task);  //inserting the updated task to Queue.
92          Schedule_EDF();     // Scheduling the task using Early Deadline First scheduling.
93      }
94      elseif(vCPU_UnderUtilized() == True)
95      {
96          vUpdate_Qos_Level();                    // Updating Quality of Services level value.
97          v_Update_Task_period(Current_Task);   //Assigning new task period to current task.
98          Update_scheduler_EDF();     // Updating the Scheduller.
99      }
100     else{
101         Schedule_EDF();   // Scheduling the task using Early Deadline First scheduling.
102     }
103 ...
104 }
105
```

Experiments

- Purpose: To investigate the trade-offs of using closed loop Feedback control based scheduler.
- The performance of the system is compared against open loop EDF scheduler.
- Four periodic SC tasks are considered.
- Task T1 scans the keypad associated to OCU and this task contains 18 lines of code and if executed independently would be completed in 29 clock cycles.
- Task T2 and T3 are associated to telegram building and contains 22 and 15 lines of code respectively and requires 28 and 19 Clock cycles respectively to execute.
- Task T4 is associated to inter-processor communication (I2C read/write) and requires 15 Clock cycles to complete under ideal circumstances.

Results

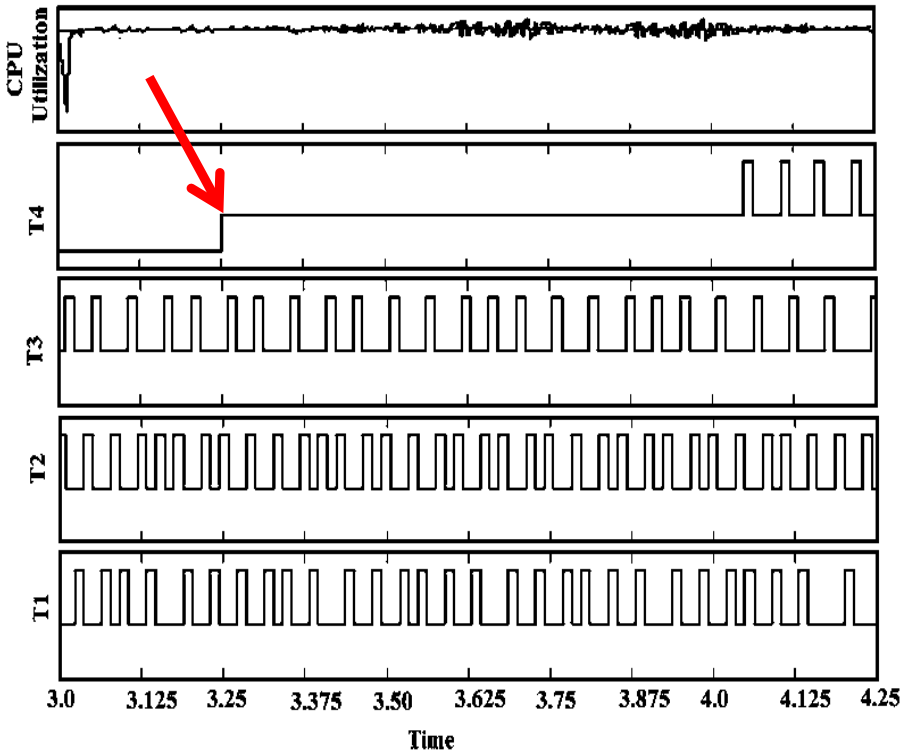


Figure 7.1 Scheduling of four tasks (T1-T4) using open loop EDF scheduler. A high level means task gets a chance to execute. A low level means task has completed its execution and is terminated. A middle level means, the execution of the task has been suspended.

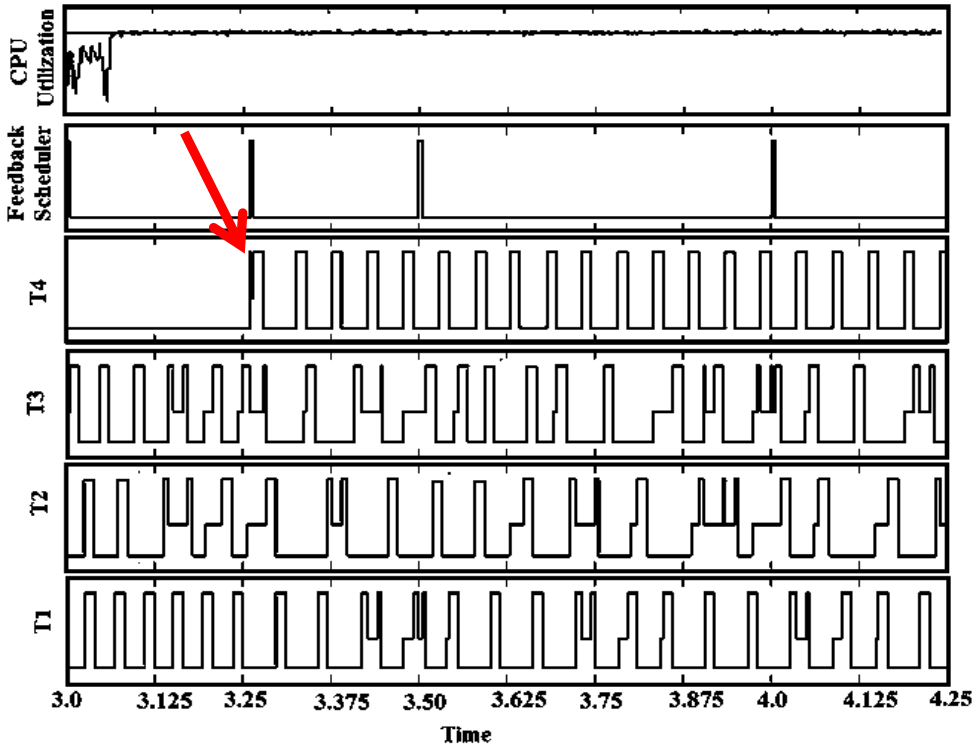


Figure 7.2 Scheduling of four tasks (T1-T4) using closed loop FCS. At time step 3.25 when Task T4 is introduced feedback based control scheduler activates and schedules task T4 immediately.

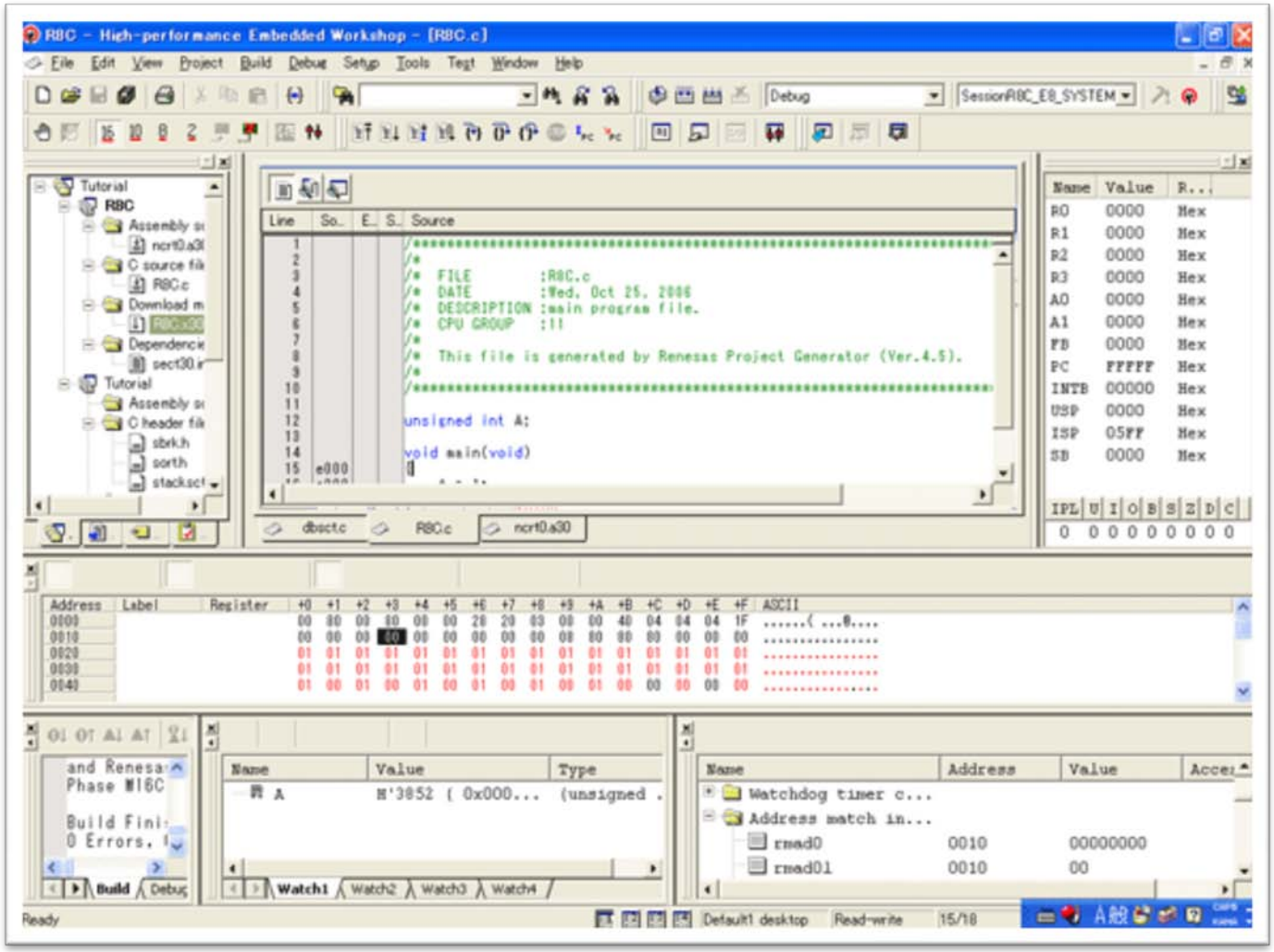
Conclusion

- In order to provide QoS (CPU utilization and resource management) under uncertain environment and internal/external execution variation, FCS is more efficient than classical open loop scheduling algorithms.
- FCS integrated crane control system is robust against the execution variation of tasks even in the presence of faults up till 200%. This integrated system provides the desired CPU utilization while making sure that all tasks will complete their deadlines at or before their WCET.
- However, the stability of the system depends on the integrated fault tolerance scheme.
- With replication and check pointing, system designer have to make trade-offs between the checkpoints assigned to Safety Critical tasks and the degree of replication. Exceeding execution time of the task beyond threshold (200%) system will no longer remain stable.
- Also introducing fewer checkpoints implies larger re-execution overhead in the presence of faults.

Future Consideration

- CPU utilization model used in this paper has some limitations.
- Initially, CPU model suffers under-utilization which clearly shows the CPU modelling error.
- FCS is implemented at the top of RTOS, so it is the responsibility of Operating System to provide scheduling services Application Program Interface (API).
- Crane control system uses check-pointing with replication as fault tolerant scheme, this scheme heavily depends on the bandwidth utilization of the network bus. With the increase in the bandwidth latency of the bus network, the SC tasks may miss their deadlines, so feedback control scheduler takes network stability into account as well.

Tool Demonstration



Thank You

Questions?