Chinese Journal of Aeronautics 25 (2012) 839-845

Contents lists available at ScienceDirect

Chinese Journal of Aeronautics

journal homepage: www.elsevier.com/locate/cja

Time-triggered State-machine Reliable Software Architecture for Micro Turbine Engine Control

ZHANG Qi[a,*], XU Guoqiang[b], DING Shuiting[b]

[a]*School of Transportation Science and Technology, Beihang University, Beijing 100191, China*

[b]*National Key Laboratory of Science and Technology on Aero-Engine Aero-thermodynamics, Beihang University, Beijing 100191 , China*

Received 22 August 2011; revised 5 September 2011; accepted 23 September 2011

Abstract

Time-triggered (TT) embedded software pattern is well accepted in aerospace industry for its high reliability. Finite-state-machine (FSM) design method is widely used for its high efficiency and predictable behavior. In this paper, the time-triggered and state-machine combination software architecture is implemented for a 25 kg thrust micro turbine engine (MTE) used for unmanned aerial vehicle (UAV) system; also model-based-design development workflow for airworthiness software directive DO-178B is utilized. Experimental results show that time-triggered state-machine software architecture and development method could shorten the system development time, reduce the system test cost and make the turbine engine easily comply with the airworthiness rules.

Keywords: airworthiness; time-triggered; finite-state-machine; model-based-design; turbine engine control

1. Introduction

Micro turbine engine (MTE) can be used as an effective high speed propulsion system for miniature unmanned aerial vehicles (UAVs) and missiles, but nowadays the MTEs used are mostly developed by hobby model engine manufacturers. The control system development and design within them are less strictly tested according to airworthiness rules such as hardware design command DO-254 and software design command DO-178B [1-3], especially the control software which was mostly developed by amateur hobbyists. In order to reach higher reliability and fulfill airworthiness command rules, DO-178B software certification should be planned and implemented, and the total control software design should be adapted, re-organized and re-evaluated [3-4].

In modern embedded system software development, two main architectures are commonly used, "event-triggered (ET)" or "time-triggered (TT)". Event-triggered designs handle multiple interrupts. The developer may write interrupt service routine (ISR) code to handle the various events directly or employ a conventional real-time operating system (RTOS) to support the event handling. Whether an RTOS is used or not, the following result is the same: the system must be designed in such a way that events—which may occur at "random" points in time and in various combinations—can always be handled correctly. These "random" interrupts result in the unpredictability, and the complete system configurations and tests are difficult to implement [4]. However, time-triggered designs only have one interrupt enabled, and all other inputs are polled. This single interrupt is usually linked to a timer "tick", which might occur (for example) every millisecond. Among the timer "tick", the system tasks are scheduled and triggered, so time-triggered architecture has good predictability and the traceability [2].

Finite-state-machines (FSMs) have been a great progress for the design of common digital circuit. However, they can be very useful also for software devel-

*Corresponding author. Tel.: +86-10-82339079.
E-mail address: zhangqi01@buaa.edu.cn

1000-9361 © 2012 Elsevier Ltd. Open access under CC BY-NC-ND license.
doi: 10.1016/S1000-9361(11)60453-6

oper. Actually nowadays desktop operating systems and application software are mostly event-based and these fields can be easily handled with software based on finite-state-machines which are simpler and easier to understand, debug and modify [5-6]. Embedded software can also benefit from state-machines because of their efficient way to use limited resources of system. Software finite-state-machines are a simple and elegant solution for system that must deal with timings and decisions.

Software in airborne systems in the early 1980s resulted in a need for industry-accepted guidelines for satisfying airworthiness requirements. DO-178, "software considerations in airborne systems and equipment certification," in its revised version—DO-178B—became the defining standard for aerospace systems and software. DO-178B is primarily a process-oriented document in which objectives are defined and a means of satisfying these objectives is described. Failure conditions associated with the system and its software components undergo system safety assessment according to the famous A-E categories, which determine the level of effort required to show compliance with certification requirements. DO-178B has clearly defined objectives for key software lifecycle process activities including software requirements, design, coding, integration, verification and configuration management. For each objective, outputs are specified that need to be created, verified and ultimately, used for certification [7-8].

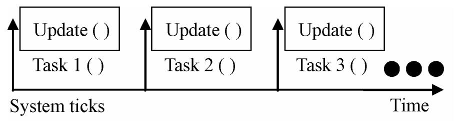
Model-based-design method was in its infancy when DO-178B was introduced, but with the increasing growth of software complexity, more and more companies and organizations have adopted model-based-design [9-10]. Model-based-design emerges as a means of addressing the difficulties and complexities inherent in the designs of control systems. Developers recognized that software design needs to start before physical prototypes and systems are available. Traditional design processes results in the delayed discovery of design and requirement errors later in the design cycle, which leads to expensive delays and missed windows of opportunities. Model-based-design provides a single design environment that enables developers to use a single model of their entire system for data analysis, model visualization, testing and validation, and ultimately product deployment, with or without automatic code generation. Once the model is built and completely tested, accurate real-time software for the production embedded design is automatically generated, thus saving time and reducing costs compared to traditional manual coding. Model-based-design with automatic code generation can also be used in rapid prototyping, enabling sub-system designs to be tested and optimized [11-12]. Furthermore, model-based-design creates a structure for software reuse that permits established designs to be effectively and reliably upgraded in a more simplistic manner [12-15].

In this paper, the compound time-triggered and finite-state-machine software architecture is implemented for a 25 kg thrust MTE used for micro high speed UAV. The model-based-design development method using MATLAB/Simulink software tools is adopted according to DO-178B requirements. Nowadays there are no mature airworthiness command rules for UAVs. Britain is the first to propose the draft airworthiness command rules CAP722-UAV operations in UK airspace-guidance [16], and the design obeys the rules in it. Because of the complexity of control system software development, some simplifications are introduced. Although the time-triggered architecture is simplified according to system requirements, test result is satisfying. The finite-state-machine software structure provides a fast evaluation and test method for system's performance and failsafe mode according to DO-178B. The total development experience has shown that the model-based-design method and hybrid reliable software architecture are more efficient and economical.

2. Software Architecture and Design Basics

2.1. Time-triggered mode basics

Time-triggered architecture means that we can determine in advance before the system begins executing—exactly what it will do at every moment of time in which it is running. This level of predictability has a number of significant benefits, for both developers and eventual system users. The simplified time-triggered software architecture is illustrated in Fig. 1. The figure assumes that a system timer has been confirmed to overflow periodically (in this case every millisecond). This "system tick" is used in turn to trigger an ISR, represented in the figure as a function called Update ( ). The Update ( ) function schedules the system tasks and sets the corresponding flag bits. As a result, the system main program is just a while loop which dispatches tasks following execute sequences [4-5]. This software architecture provides reliability insurance because of its precise predictability. At the moment what the control system is doing is obviously.



```
While (1)                    Update ( ) //1ms interrupt
{                            {
Sch_Dispatch_Tasks ( );      Schedule ( );
}                            }
```

Fig. 1 Simplified time-triggered software architecture.

The time-triggered software architecture can be viewed as a single timer interrupt service routine that is shared between many different tasks. As a result, only one timer needs to be initialized, and any changes to the timing generally require only one function to be altered. Furthermore, we can generally use the same scheduler whether we need to execute one, ten or one hundred different tasks. This "shared ISR" method is very similar to the shared printing facilities (for example) provided by a desktop operating system (OS) [2] such as Windows.

2.2. Finite-state-machine basics

Finite-state-machine method describes system as combinations of different states. System reacts to external inputs and changes current state to next state. To describe a finite-state-machine, normally four elements are required: states, inputs, transitions and actions. Two states, two inputs, four transitions and four actions finite-state-machine can be described using state diagram as in Fig. 2.

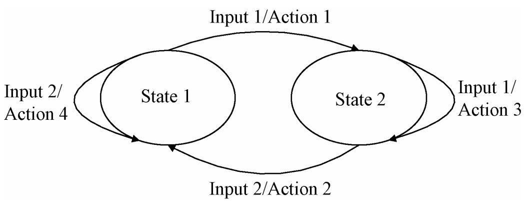


Fig. 2 State diagram of a simple finite-state-machine.

The finite-state-machine starts from a default state which is always the power on startup program, and then responds to external inputs and decides what should do next. Software finite-state-machine implementation is a straightforward process and can be performed in any programming language, ranging from the low level assembly language to high level object-oriented programming language such as C++. In this system design, C language is used.

The main advantage of using a state-machine in embedded design consists in its flexibility to add, delete or change the flow of the program without impacting the overall system code structure. The finite-state-machine software architecture offers an convenient way to system test at different levels, which is vital to DO-178B A-E level categories [6,12]. Also, another benefit from finite-state-machine method is that finite-state-machine implementation for microcontroller only requires limited memory and registers.

2.3. Model-based-design method for DO-178B

Model-based-design re-engineers the traditional development process from one which is paper-based to one that uses an executable model that is repository for all information about the concept, design and implementation [8]. The model is used throughout the four stages of development: research, design, implementation, and verification & validation. At each stage, the model is updated and elaborated ensuring continuity and traceability throughout the evolution of the design. Model-based-design is the use of models to describe the specifications, operation and performance of a component or a system of components [7].

Modern software tools such as MATLAB/Simulink have been widely used for model simulation, and in an endeavor to realize one environment for total stage, various additional tools were developed [9-10, 12]. Now MATLAB has been released with the DO qualification kit (for DO-178B), which allows selected verification tools to be qualified for a DO-178B project. As shown in Fig. 3, a highly automated verification workflow with qualified tools is now available. The modeling tools provide engineers with high degrees of flexibility for expressing designs. The code generation tools produce efficient results and provide many code optimization options [9].

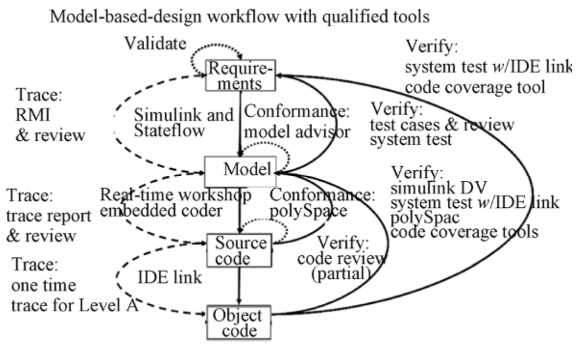

Model-based-design workflow with qualified tools
Validate
Requirements
Trace: RMI & review
Simulink and Stateflow
Conformance: model advisor
Verify: system test w/IDE link code coverage tool
Verify: test cases & review system test
Model
Trace: trace report & review
Real-time workshop embedded coder
Conformance: polySpace
Verify: simulink DV system test w/IDE link polySpac code coverage tools
Source code
Verify: code review (partial)
Trace: one time trace for Level A
IDE link
Object code

Fig. 3 Model-based-design method for DO-178B.

3. Software Architecture Implementation and Development in MTE Control

3.1. 25 kg MTE overview

The 25 kg MTE is a single radial compressor and an axial flow turbine. The combustion chamber is annular type fitted with a unique "low pressure" fuel system. Both the front and the rear hybrid bearings are lubricated and cooled by the fuel system. The engine control unit (ECU) software protects the turbine engine from misuse and accidental damage, and also implements failsafe protection airworthiness rules. The complete assembled 25 kg MTE is shown in Fig. 4.

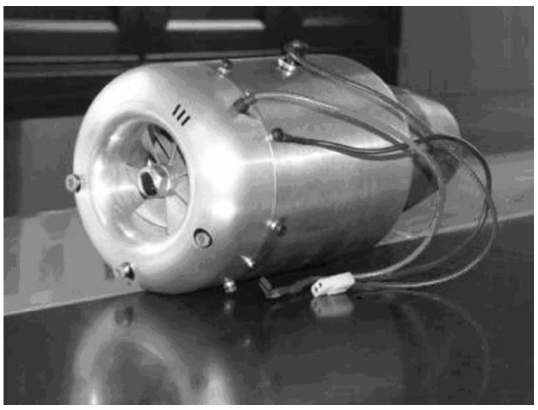


Fig. 4 Assembled 25 kg MTE.

The 25 kg MTE control system hardware diagram is shown in Fig. 5. In Fig. 5, RPM stands for revolutions per minute. The control system employs Freescale's 16 bit microcontroller MC9S12DG128 as its digital control core. MC9s12DG128 has 128 KB Flash ROM and 8 KB RAM which are enough for the control program.

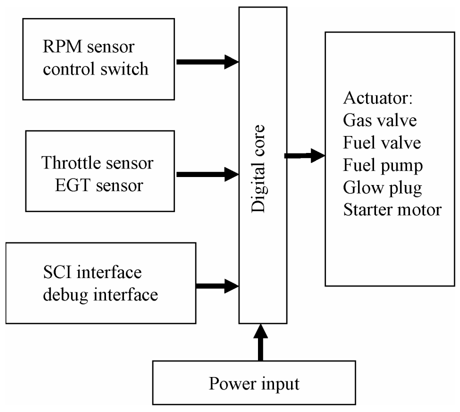


Fig. 5 Control system hardware overview.

The speed sensor is KMZ10A which is an extremely sensitive magnetic field sensor, employing the magneto resistive effect of thin-film perm alloy. It is actually a resistor bridge and the output differential signal is amplified and converted to square wave.

Control switch is just on-off switch interface and pulled up by a resistor to avoid electronic interference.

Throttle sensor is a linear potentiometer which is sampled by microcontroller AD converter. Throttle signal means user's thrust requirement.

Exhaust gas temperature (EGT) sensor is K type thermocouple and can measure up to 1 000 °C.

Serial communication interface (SCI) interface is used to interface to PC and monitor the control program running status and gather experimental data for analysis.

Background debug model (BDM) debug interface is used to download and reprogram the microcontroller.

Gas valve and fuel valve are solenoid valve which control the propane gas and fuel flow. Propane gas is only needed on engine startup.

Fuel pump is a gear pump driven by brush motor which provides fuel to engine after successful startup to idle run status.

Glow plug is used for starting propane gas ignition and driven by a few burst pulses whose duty cycle is restricted to avoid the burnout.

Starter motor is used at startup for speeding up the engine to startup speed. Also the motor will be used at emergency shutoff and normal shutdown to blow the fuel out of engine and cool down the engine, which is vital to safety requirement.

3.2. Time-triggered state-machine software architecture implementation

As expressed in Section 2.2, the basic finite-state-machine model is timeless; it is not possible to model within the basic finite-state-machine framework system properties that are dependent on the progression of real time, such as the duration of computations or the limited temporal validity of real-time data. To overcome these limitations, efforts have been made to modify the finite-state-machine model to include some notion of time. The objective of this paper is to expand the existing work on basic finite-state-machines and timed automata to include the concept of a sparse global time base as a central element of the model. Such an extended finite-state-machine model can be called a periodic finite-state-machine (PFSM) model. The PFSM model incorporates the notions of state variables, global time, periodic clock constraints and time-triggered activities. Thereby, PFSMs enable a concise and intuitive representation of distributed control systems and reduce the gap between a modeled system and its implementation.

Although finite-state-machine method is event-based in essence, it does not conflict with the time-triggered development method. In this control system design, the periodic timer interrupt overflow acts as trigger event, and the system finite-state-machine is evaluated in each event. As a result, this software architecture realizes time-triggered and finite-state-machine integration and precise time stamp information is included. This novel hybrid software architecture serves as a foundation for a model-based development process and formal analysis for distributed embedded real-time systems.

Basic finite-state-machine model data structure should be modified to include the time stamp, and this is straightforward and can be easily done [4]. In this system design, the main program firstly initializes the real time interrupt (RTI) and completes system control initialization, then adds Engine_Control_Update ( ) task, and lastly starts the task scheduler. In the while loop program, there only exists scheduler dispatch task [4]. In the RTI routine, all tasks' status is updated. If scheduled, the task will be invoked by the task dispatch function SCH_Dispatch_Tasks ( ) and executed. The SCH_Dispatch_Tasks ( ) function checks all tasks' status and determines which task will be executed.

Up to the implementation of time-triggered schedule software architecture, finite-state-machine based engine control task is the only task and the function is Engine_Control_Update ( ). In this function, engine control operation is divided into several states according to the engine operation procedure. The normal complete operation procedure is shown in Fig. 6.

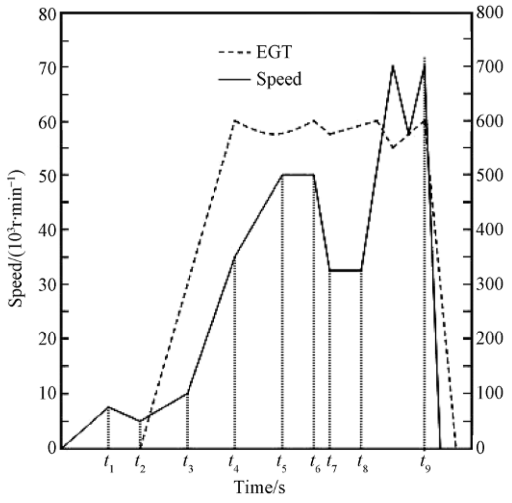


Fig. 6 Engine normal complete operation procedure.

0-t_1, the start switch is on, the start motor is power on and the engine reaches 7 500 r/min startup speed.

t_1-t_2, the start motor stops, the glow plug is on, and the propane gas valve is open (pulsing). Until t_2, the EGT begins to increase.

t_2-t_3, the start motor restarts, the glow plug is off, the propane gas valve is normally open. Until t_3, the engine speed reaches 10 000 r/min which is the suitable time to offer fuel to engine and the pump motor is power on.

t_3-t_4, the start motor, the propane gas valve and the fuel pump are all on. Until t_4, the engine speed reaches 36 000 r/min; the gas valve and the start motor are shut off.

t_4-t_5, the engine is powered only by fuel. Until t_5, the engine speed reaches 50 000 r/min and the engine starts to calibration.

t_5-t_6, the engine keeps the calibration speed until t_6.

t_6-t_7, the engine decreases to idle speed 36 000 r/min.

t_7-t_8, the engine works at idle speed.

t_8-t_9, the engine works normally according to user's throttle input.

t_9-, the engine stops if the start switch is pushed off. The fuel pump is power off and the start motor is power on to cool down the engine.

0-t_8, time fragments all belong to startup process and can be integrated into a startup state, so the engine control finite-state-machine is divided into different states according to the normal system operation procedure.

According to the engine operation procedure, the corresponding control finite-state-machine chart is shown in Fig. 7.

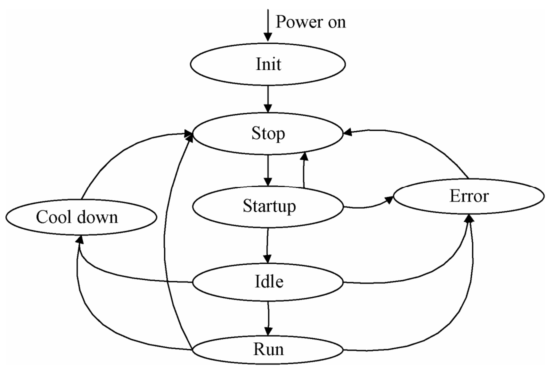


Fig. 7 Engine control finite-state-machine chart.

Once the system state-machine chart division is completed, implementing a state-machine in software is a straightforward process and can be easily performed in any programming language. In this system design, C language is used for its portability and flexibility. The implementation code uses "switch-case" structure to describe the system state-machine.

3.3. Model-based-design for DO-178B

Manned aerial vehicle airworthiness DO-178B command rule classifies system potential failure conditions into Level A to Level E, as shown in Table 1.

Table 1 DO-178B software failure level

Level	Failure condition	Description
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No effect	Failure has no impact on safety, aircraft operation, or crew workload

UAV airworthiness is still under development, and the draft document CAP722 is a good reference. CAP722 has proposed which command rules should be used according to different UAV conditions [10, 12, 16].

In this paper, the DO-178B failure level is simplified, and only Levels A, B are included because the MTE targets UAV applications. According to system operation procedure, Level A failure cases include fuel pump error, power error; Level B failure cases include over temperature, over speed, loss of communication and control switch error.

After DO-178B software failure levels are determined, model-based-design with MathWorks is implemented [12, 14]. According to Fig. 3, the following software development workflow is implemented.

1) Requirement process

For requirement validation, use traditional peer reviews of the requirements. For requirement linking, use the requirements management interface to establish links between the model and high level requirements in textual or third-party tool form.

2) Modeling process

For the low level requirements, Simulink and Stateflow are used for modeling. And this paper uses Stateflow to model the control software finite-state-machine, which is more efficient and convenient. It must be noted that in model process, the model means not only the control algorithms and safety boundaries, but also the test information and comments which are much more important for later test and review. It can also be said that only the model containing all information can provide a single "truth" in the design.

3) Coding process

Auto code generation can be realized with the combination of Real-time Workshop Embedded Coder and Embedded IDE Link™ product. In this design, Metrowerks Codewarrior for HCS12 V4.7 IDE development environment is used. The control model algorithm can be converted to source code automatically, and the structure of files is standard [15-16], as shown in Fig. 8. The verification of the source code, whether it is automatically generated or not, requires a code review as part of the DO-178B process. PolySpace products are used to automate portions of the code review by using the DO qualification kit.

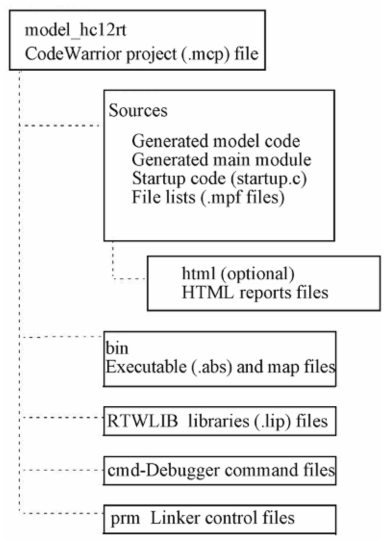


Fig. 8　Auto code generation files structure.

4) Object code testing process

DO-178B requires that equivalence class tests be performed on input data ranges. The model coverage tool helps in this area because it can record signal ranges during testing and show the minimum and maximum values achieved. Additionally, the Simulink design verifier can be set up to automatically generate equivalence class test cases using the test objective block on input signals. In this case the user must specify the test values based on the data range and desired equivalence classes.

4. Experiment and Test Results

The software development experiments follow three stages:

1) The time-triggered finite-state-machine control system architecture is implemented. The time-triggered mode is realized through the microcontroller's timer overflow ISR which appears every millisecond. The control finite-state-machine structure is implemented in the Engine_Control_Update ( ) function.

2) The total system control model is realized and simulated. MATLAB, Simulink and Stateflow software tools are used to model the system architecture. The DO qualification kit is used to meet the requirement of DO-178B airworthiness command rule. Various components and system failure modes are modeled and evaluated through simulation, and the simulation results are documented in the system model.

3) The total actual system is tested. At this stage, various practical abnormal work situation and failure mode according to DO-178B are evaluated. The control model is revised and information is documented.

At the first development stage, three software architectures are evaluated. Table 2 shows their memory space requirement in practice. Time-triggered finite-state-machine method requires the same memory spaces as event triggered manual code method, and much less than event-triggered with RTOS support. Event-triggered mode method offers good real time performance, but the manual code method introduces latent uncertainty and error because of human work. Also the system reliability test cannot be implemented easily, and careful test draft should be planed. Event-triggered architecture with commercial RTOS can be used to reduce the human error and enforce the robustness. In this system design, μC/OS is ported and evaluated. Because the RTOS must deal with complicated task management, content switch and priority determination, this method requires large memory spaces. Combining the memory space requirement, real time performance, predictability and test cost, the time-triggered finite-state-machine method provides an appropriate method for MTE control.

Table 2　Requirement of different memory spaces

Byte

Mode	Memory		
	Data	ROM	RAM
ET-manual code	28	2 243	180
ET-RTOS(μC/OS)	58	4 786	1 184
TT-FSM	25	1 846	168

Although the time-triggered finite-state-machine in this system is simplified to one task Engine_Control_Update ( ), the system shows satisfying predictability and traceability. Predictability is obvious because the timer overflow occurs every millisecond, and the only task is Engine_Control_Update ( ). The traceability can be easily realized through monitoring the control state-machine implemented in Engine_Control_Update ( ) function in which time stamp is added. The control software algorithm can be easily modified with MATLAB Stateflow tool. The modification is graphical and the source code generation is automatic. Adding and deleting states do not affect the main control system software structure at all. This software architecture improves efficiency greatly compared with event-triggered structure.

Over-temperature is a common problem for MTE. According to DO-178B failure level definitions, this system assigns over temperature fault to Level B. In this situation, the control system should shut down the fuel pump and power on the start motor. The start motor works until the engine temperature falls into free air temperature scope. In the control state-machine, the over temperature error flag is recorded. The engine run data can then be downloaded to PC to be analyzed.

Through successful hardware in the loop (HIL) evaluation of the control software's correct performance, the actual engine test bench experiments are carried out. A successful engine run cycle is shown in Fig. 9. The EGT limit is 750 °C and the speed limit is 115 000 r/min. It can be seen that these two parameters are in good control.

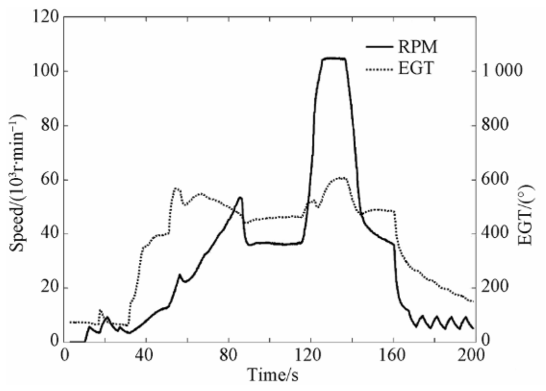


Fig. 9 Engine run cycle speed and EGT data.

5. Conclusions

The model-based-design method and time-triggered finite-state-machine hybrid software architecture implementation indicate that:

1) Time-triggered software pattern is simple and reliable.

2) Finite-state-machine method provides great traceability and efficiency for engine control task.

3) Model-based-design and DO-178 qualification kit provide complete uniform software environment for airworthiness command.

References

[1] Falla M. Advances in safety-critical systems. Lancaster: University of Lancaster Press, 1997.

[2] Hatton L. Safer C: developing software for high-integrity and safety-critical systems. London: McGraw-Hill, 1994.

[3] Hilderman V, Baghai T. Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware). Leesburg: Avionics Communications Inc., 2007.

[4] Pont M J, Parikh C R. Li Y, et al. The design of embedded systems using software patterns. Proceedings of Condition Monitoring, 1999; 221-236.

[5] Drumea A, Popescu C. Finite state machine and their application in software for industry control. 27th International Spring Seminar on Electronics Technology, 2004; 1: 25-29.

[6] MISRA. Guidelines for the use of the C language in vehicle-based software. London: Motor Industry Software Reliability Report, 1998.

[7] MathWorks. Model-based design for DO-178B. Massachusetts: Mathworks, 2010.

[8] RTCA/DO-178B. Software considerations in airborne systems and equipment certification. Washingtong D.C: RTCA Inc., 1992.

[9] Erkkinen T, Potter B. Model-based design for DO-178B with qualified tools. AIAA-2009-6233, 2009.

[10] Marian N, Guo Y. Model based design of embedded software. Research and Education in Mechatronics 2006; 156-163.

[11] Smith P F, Prabhu S M, Friedman J H. Best practices for establishing a model-based design culture. SAE Paper, 2007-01-0777, 2007.

[12] Anthony M, Behr M, Jardin M, et al. Model-based design for large high-integrity systems: a discussion on verification. and validation. Association for Unmanned Vehicle systems International, 2010; 180-195.

[13] Holliday N. Software development with real-time workshop embedded coder. Mathworks Aerospace and Defense Conference, 2008; 152-163.

[14] Potter B. Use of MathWorks tool suite to develop DO-178B certified code. ERAU/FAA Software Tool Forum, 2004; 152-159.

[15] Mathworks. Embedded target for Motorola HC12 for use with real-time workshop. Massachusetts: Mathworks, 2008.

[16] CAP722. Unmanned aerial vehicle operations in UK airspace—guidance. London: Directorate of Airspace Policy, 2002.

Biography:

ZHANG Qi worked as a lecturer in Beihang University from 2004 and now is a Ph.D. candidate there. His main research interests include aero-engine electronic fuel supply and control system design.
E-mail: zhangqi01@buaa.edu.cn