TOPPERS

Introductory implementation seminar

(JSP-1.4|AKI-H8/3069F | instant noodle timer session)

DAY 1

TOPPERS Project

Educational Working Group

About this document

- **Document Usage Conditions**

1. Copyrights

2. For opinions, proposals, and questions concerning this document, send it by e-mail to the TOPPERS Project secretariat.

   This document is subject to change, for content improvement and otherwise, without prior notification.

3.

Seminar Goal

- Acquire implementing technique of RTOS (Real-Time Operating System) by applying application programs to TOPPERS/JSP kernel.
- Increase understanding of basic hardware interface through on hand training.
- By reviewing the SESSAME embedded software programmer educational seminar, broaden knowledge of analyzing, planning, and implementing processes.

The computer board used for implementing the application is equipped with Renesas Technology Corp. CPU H8/3069F. Compile and link can be done on the Cygwin development tool. First, familiarize yourself to this environment for developing. Confirm action by downloading created execution file to target board RAM. For porting application to TOPPERS/JSP kernel, learn the TOPPERS/JSP kernel's construction method on the development tool by referencing the sample program. Later, expand the sample program to create the desired system. The RTOS (Real Time Operating System) functions used for expansion are task create and start, task communication using shared memory, and task communication utilizing eventflags.

The hardware interface is controlled by a device interface contained in the source file named *device.c*. There is no need to comprehend the contents entirely, but direct access to the hardware port is included in this file. This driver description is written accordingly to the ITRON device driver design guideline.

In the SESSAME introductory embedded software seminar, utilizing structure designing, requirement analysis, designing, implementation, and testing was studied through the development process. Let us revise this process using the study material.

Seminar Schedule

Day 1

1. Checking the development tool
   0.5 hr
2. What is embedded?
   Revising the introductory SESSAM seminar 1.5 hrs
3. Confirming operation of non-OS teaching material 1 hr
4. What is a Real-Time OS? 1.5 hrs
   RTOS conditional transfer, confirmation by utilizing HW I/F education material
5. Construction of instant noodle timer
   Constructing the timer 1.5 hrs

Day2

1. Implementing the instant noodle timer
   Implementing the timer 1.5 hrs
2. Review 0.5 hr
   Evaluation of construction and application
3. Synchronization and communication
   1 hr
   Correspondence between tasks
   Synchronizing and exclusive control
4. Reconstruction using eventflags
   1.5 hrs
5. Summary 1 hr

Checking the development tool

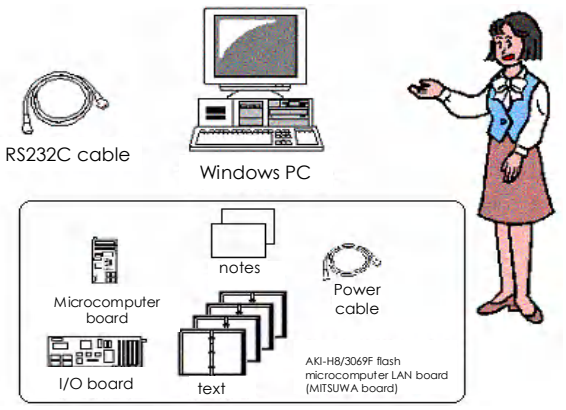
1. Checking the development tool
2. Embedded? Revising the SESSAME introductory seminar
3. Confirming operation of non-OS teaching material
4. What is a Real-Time OS?
5. Constructing an instant noodle timer

TOPPERS

Let us learn about the AKI-H8/3069F development tool. Check the material provided and the four tools of the development tool, understand the TOPPERS development tool, see if the source can be referred by the editor, and if building is possible using the development tool.

Checking your training environment
Checking materials to be used

- Computer
- AKI-H8/3069F Set
  - ・microcomputer board and mother board (ready made)
  - ・textbooks(4), notes (2)
  - ・CD-ROM (1)
  - ・power cable
- RS232C cable



TOPPERS

・Computer　the following tools are needed
　Cygwin
　Editor
　communication software
・microcomputer board AKI-H8/3069F and I/O board
　ready built
・RS232C cable communication link between computer and board
　use for display to monitor and user program download
・text　seminar textbook and μITRON4.0 specification booklet
・power cable　provide power to boards
・CD-ROM　explanatory note of boards and development tool included

Checking the development tool

Checking the Windows development tool

- Cygwin, TeraTerm icons are on the desktop
- GNU development tool, h8write are executed from the command line

Cygwin	Emulates a Linux-like environment on Windows
GNU development tool	Cross development tool BINUTILS,GCC （C compiler, Assembler, Linker）
h8write	Write simplified monitor to Flash ROM
TeraTerm	Terminal software (connects board and computer)
Simplified monitor	Download program to RAM

The development tool is composed from five tools.

• Cygwin emulates a Linux-like environment on Windows.

• GNU development tool consists C-complier, Assembler, and Linker.

• h8write is used when writing simplified monitor to Flash ROM. It is used from Cygwin command line. Cygwin, GNU development tool, and h8write are included in the CD-ROM attached to the board.

• TeraTerm is a terminal software used to connect board and computer.

• Simplified monitor is used when downloading user program to the target.

Explanation of the development tool

Constructing the development tool

- Development tools can be installed to your own computer

  GNU development tool can be found on the CD-ROM attached to AKI-H8/3069F. Using this, build the sample program.
- For reference to specific use of the tool see attached material.
- A sample program for use in this seminar is included in the CD-R.

Setting up AKI-H8/3069F development tool

1. Construction of development tool
   Install Cygwin from the attached CD-ROM
   (H8 cross-compiler will be installed as well)

   Under a work directory of your choosing, create a source archive of the training material.
   tar zxvf toppers-beginner-h8-2005-03-22.tar.gz

   Under the *toppers-beginner-h8* directory, execute "make" command.
   This will create the following:
   ・configurator
   ・kernel library
   (this may take some time)

   At the end of the training session, to delete unnecessary files, execute
   "make realclean" command under *toppers-beginner-h8* directory.

2. Build user application
   ・Non-OS kitchen timer
   Under non_os directory, execute "make" command

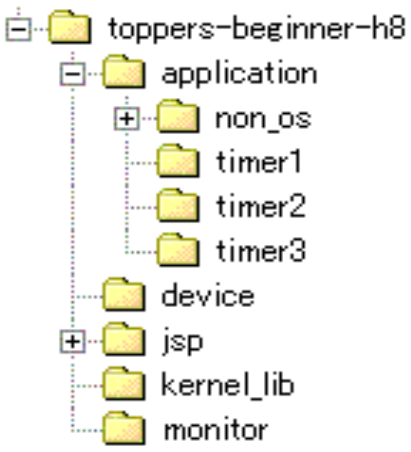
   non_os.srec file will be created; download this to target

   ・RTOS kitchen timer
   Under directories timer1, timer2, timer3, execute "make depend"
   and "make" commands.

   jsp.srec file will be created; download this to target

Confirming development directories
Development programs and set-up directories

- Confirm development directories through Windows explorer
- Start up Cygwin and access source code within the non_os directory
- Check so programs can be built
- Check that the editor is in working condition



Confirm source code tree within you computer.

Directory composition of the source code tree

toppers-beginner-h8

Makefile	environment construction Makefile
monitor	simplified monitor
jsp	same as jsp1.4 distributed package
kernel_lib	compiled kernel; contains library “libkernel.a”
device	device driver
	access routine to the on-board device
readme.txt	simple manual for the device driver
application	application used for this training seminar
non_os	kitchen timer not using an OS
timer1	model kitchen timer（used for conparison only）
timer2	model kitchen timer（used for reconstruction during session）
timer3	furnished kitchen timer example（solution）
eventflag	furnished kitchen timer example using an event flag

Construction of hardware for the microcomputer board

- For details, see appendix at end
- Hardware dependence is concealed by the device driver, therefore there is no need for the application programmer to be conscious of this matter

A simple manual is included in toppers-beginner-h8/dvice/readme.txt.

What is embedded?
Revising the SESSAME introductory seminar

1. Checking the development tool
2. **Embedded? Revising the SESSAME introductory seminar**
3. Confirming operation of non-OS teaching material
4. What is Real-Time OS?
5. Constructing an instant noodle timer

Here, we will study about embedded systems in general. Also, we will confirm the specifications of the instant noodle timer that we will build in this seminar.

Using the structural design covered in the SESSAME introductory level seminar, analyze the requirement and construct a instant noodle timer. Download the non_os.srec to the microcomputer board and conduct various tests.

What is an embedded system?

Confirming the targeted system

- Buried into various machinery, is a computer which controls the specific machinery

  "embedded" is a loose term, the range where "embedded system" may vary depending on the interpreting person(s)

  ex) plant control, PDA, game machine

- Here, "embedded system" will be used as a broad term

  In exaggeration, anything excluding personal computers and general-purpose computers, such as mainframes, are considered embedded systems

  It can also be said that it is a system made for specific purposes

In this seminar, anything excluding general-purpose computers are classified as embedded systems.

There are times where a narrow definition of embedded system (where appearance is not a computer) is called a deeply embedded system.

The following are examples of embedded systems:

・electric household appliances（microwave oven, washing machine, dryer, air conditioner)

・audio visual appliances(television, video machine, digital camera, audio devices)

・amusement/educational appliances(game machines, electronic musical instruments, Karaoke, slot machines)

・personal schedule appliances(PDA, electronic organizer, car navigational system)

・computer peripheral equipment(printers, scanners, disks, CD-ROM drives)

・office appliances(copy machines, fax machines)

・communication appliances/terminals(telephones, answering machines, mobile phones)

network equipment(exchanger, PBX, network router, HUB)

・transportation machinery(automobiles, traffic signals, trains, aircrafts, ships)

・industrial controls／factory automation appliances(plant controls, machinery, industrial robots)

・building appliances(building illumination, air conditioners, electrical systems, elevators)

・medical appliances／welfare appliances(sphygmomanometer , electrocardiogram, x-ray machines, CT scanners)

・Space／military(rockets, satellites, missiles)

・other business appliances(data terminals, POS terminals, vending machines)

・other processing appliances(syncroscope, IC testers, electrical meter readers)

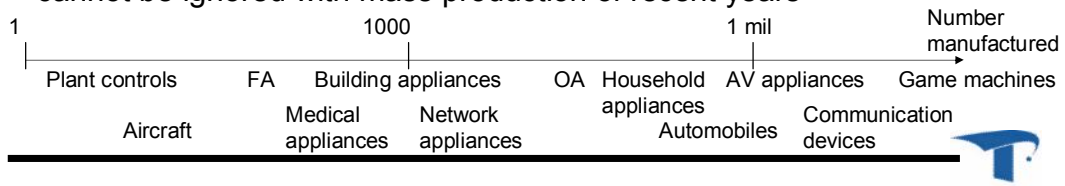
Diversity of embedded systems

Manifold and there is no clear technical education

- Embedded systems are manifold from many aspects, such as in system size and property
- Currently, there is no effective classification index proposed
- Equation that affects embedded system development:

Total cost = development cost + manufacturing cost x number of manufactured product

Also, time-to-market factor needs to be added. Development costs cannot be ignored with mass production of recent years



TOPPERS

Characteristics of embedded software development are as follows:

・Programming closely related to hardware

Because hardware construction and peripheral devices differ in every system, programming that handles hardware directly is needed. Development can be difficult because technical know-how is greatly needed.

・Separation of development tool and target environment

It is not always possible to program on the target system. Cross development tool, remote debugging and simulation may need to be considered upon development. Also, there are systems where system verification is required without suspending the current system.

・Harmonious and parallel development with hardware (may be needed)

Verification and debugging before the target system is ready is becoming possible.

・Various platforms (hardware, operating systems)

In many cases, the system is optimized to the hardware and/or operating system.

・Cost required for verification is considerable

Verification for high reliance, indecisiveness due to timing, and checking target appliance by running the appliance, costs required can become considerable.

・Software construction for exceptional treatment may become necessary.

・A prerequisite that the software within the system can be trusted

Current trend of embedded systems

Transitions within embedded system developments

- Broaden range in use of embedded systems
  - Digital consumer (mobile phones, digital appliances, ITS, etc.)
- Expansion and complexity of traditional embedded systems
  - Complex appliances, digitalization, network use
  - Advanced user interfaces
  - Better functionality and value added due to computer control
- Reduction of development period (time-to-market) and cost down
- Fluidity of borderline between software and hardware
  - Securing quality and reliability of system becomes an issue

TOPPERS

With the current trends, reutilization of design assets has become a point to be taken of notice.

・Reutilization of design assets

As a common subject between software and hardware design, effective use of design assets has become indispensable. There are two directions to this inclination.

One is to reuse company assets constructed in the past. Inefficient development can turn into a legacy system. To avoid such turnout, acknowledging and re-factoring assets are necessary.

Another is the case where purchasing software parts/IPs into the constructed asset is utilized. Since developing the software in-house cannot correspond to time reduction demands and diversity of technical systems, investment becomes extremely costly. To solve these points, demand for open source embedding systems is desired.

・Obstruction factors to reutilizing design assets

One obstruction factor is the extreme diversity of hardware, processor, operating systems, middleware, and network forms. These need to be selected by enclosure. Also, high cost needs for testing is another factor.

Characteristics of embedded systems

4 Pillars

- Purpose specified system
  the whole system is composed for a specified purpose
- Resource limitations
  Cost down, lowered electricity consumption, lighter weight appliances
- High reliability is demanded
  Malfunction of system directly ties in to malfunction of appliance
  System repair requires high costs
- Real-time is essential
  Fast operation is required, but it is necessary that the system runs in compliance with what the appliance is bound to do

➡ Most embedded appliances are real-time systems

TOPPERS

With diversification of embedded systems, there are several distinctive features.

•Exclusive system

The whole system is designed exclusively for one objective.

•Strict resource limitation

Rigorous costdown is desired, especially with mass production

Reduced electricity consumption, running environment factor (temperature, executing environment), and downsizing are some special restrictions.

•High reliability

Because system errors, in many instances, may directly affect mechanical error, it is impossible not to offer guarantees at all. It may come in conflict with PL laws.

System redressing may become costly

User expectations exist for mechanical reliance

•Real time

Operation in guidance with the time requirement of the control target machine is desired

This does not mean that speed covers the requirement

A system that fulfills the requirements is a Real-time system

Revision of SESSAME introductory seminar

Revision using teaching materials

- SESSAME embedded software engineer seminar

  For embedded software development, using *Electrical water boiler* and *Shishiodoshi* as examples, structural designing techniques for embedded programming

requirement assessment → construction → implementing → testing

In this seminar, let us explain by utilizing teaching materials, about embedded structural designing.

TOPPERS

About the SESSAME embedded software engineer seminar

The Society of Embedded Software Skill Acquisition for Managers and Engineers (SESSAME) is an organization committed to training embedded software engineers and managers through designed curriculums and research for the basis of the training and tool development for training.

In the introductory seminar, analysis, design, programming, and testing is the focal point. Beginning with the analysis and designing of the embedded system, it goes on to introduce the techniques for analyzing, designing and testing.

What to make with the microcomputer board?

A timer that notifies the best moment for eating

- Create an “Instant noodle timer” using a microcomputer board
- What are the specifications?

After turning switch on, to confirm execution blink LED at one second interval.

By turning timer startup switch on, the timer starts; when off, the timer stops. Initial timeout is one minute.

During execution, turning the timer extension switch on, extends timeout by one minute. Turning it on twice extends timeout to three minutes.

Blink timer LED every 10 seconds to confirm timer execution, at timeout, blink for 30 seconds.

Before developing. . .

TOPPERS

A regular timer is composed by a LCD displaying the time and a buzzer. With the AKI-H8/3069F board, because the only devices are a LED and a switch, time and timeout notification is done by the LED.

For time notification, the power (confirm) LED blinks at one second intervals. Timer notification is done through the timer display LED. When the timer is started, it will blink at 10 second intervals. For timeout notification, the timer display LED blinks at 30 second intervals for notification.

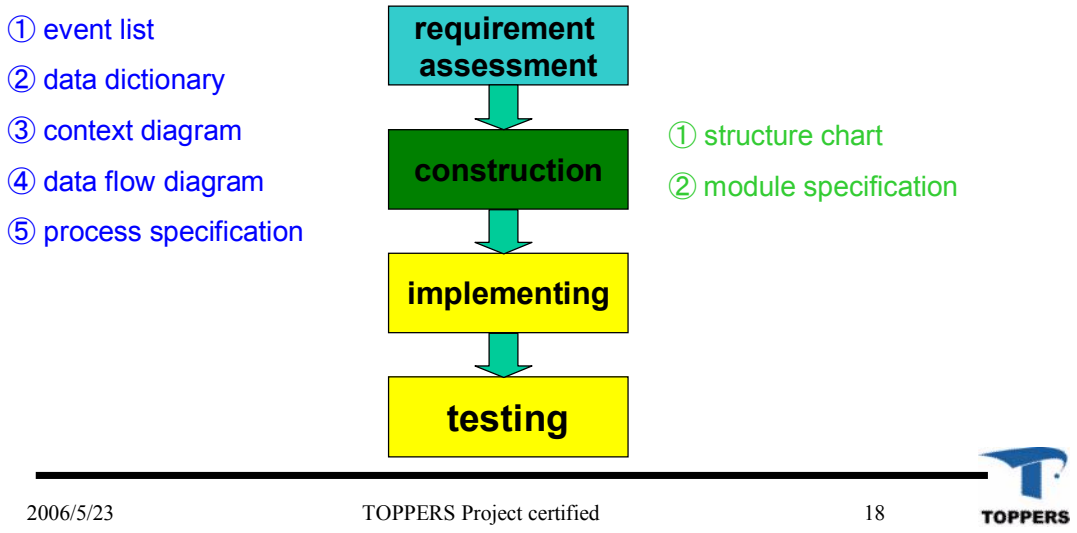
Starting the timer is done by turning the timer switch ON, and stopping the timer is accomplished by turning the timer switch OFF. Initially the timeout setting is one minute. Turning the timer extension switch on extends the timeout time by one minute increments.

Corresponding relation with actual device

analysis level name	actual name	H8 port number
•Timer start switch	DIPSW1-1	P50
•Timer extend switch	DIPSW1-2	P51
•Power (confirming) LED	LED1 (red)	P46
•Timer display LED	LED2 (green)	P47

Revision of SESSAME introductory seminar

- Embedded development begins with requirement assessment



TOPPERS

As mentioned before, an embedded system is a system where high reliance and real-time nature is required. On the contrary, due to the expanding scale of systems in recent years, decline in quality and securing maintenance has become a problem. Aiming to solve these problems and making the system more visual, let us begin by analyzing the requirements to construct the system.

Requirement analysis is where comprehension and interpretation of the development target system is accomplished. First, analyze by considering and arranging what the required system is, not how to satisfy the specifications. (WHAT point of view)

The SESSAME requirement analysis begins by assessing the requirements; structural design comes in at the designing stage. With the structural design, the focal point is how to build based on the results of the requirement analysis. (HOW point of view)

Structural design is accomplished by the following development procedures

1. Event list
2. Constant diagram
3. Flow diagram
4. Process specification

If necessary, a data dictionary is made.

Requirement assessment 1

Event list, data dictionary

- Event list

Event: occurrences that happen outside the system which affects the system

Stimulus: a way to send notice of the event

Action: function which the system executes when an event occurs

Response: reply from the system to the outside

- Data dictionary

event	stimulus	action	response
Power ON	Start action	Show current time	Power indicator LED to blink at one sec. interval
Start timer	Timer starter switch ON	Start timer Notify start to user Notify user of elapsed time (timeout)	Blink timer LED every 10 secs. at 0.25 interval twice Blink timer LED for 30 sec. at 0.25 sec. intervals
Add timer clock	Time extender switch ON	Extend timeout clock for one minute	None
Timer OFF	Timer starter switch OFF	Stop timer	Reset timer
Power OFF	End action		

Initial setting	By making the power indicator LED blink at one second intervals, notify the user of its current status
Timeout clock	Initial timeout is one minute, add one minute for each extension
Timeout display	While timer is running, blink timer LED every 10 seconds, when timeout occurs, blink timer LED for 30 seconds

"Blink" means that the light alternating state of on and off

TOPPERS

An event list is a list which clarifies how the system deals with external occurrences. The list is composed by four parts. By creating an event list, the action of the instant noodle timer will become clear.

1.Event external occurrences which affect the system

It can be considered as an occurrence that the system cannot control

2.Stimulus information input of event occurrence

3.Action system function(s) for when an event occurs

4.Response external response when an event occurs

With the instant noodle timer, an event list is made in regard to the four events.

Collection of terms, along with comments, used for analysis and designing are entered into the data dictionary. The terms should be in layman's terms, and when used during development, should be uniform.

Caution!: during requirement analysis, the names used for stimulus are logic names; implementation dependent names should not be used.

- Timer start switch x SW1
- Timer extend switch x SW2

Requirement assessment 2

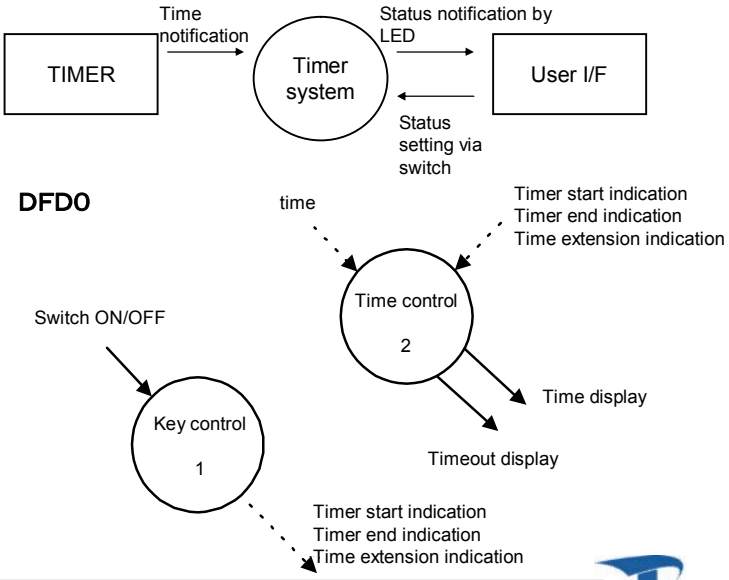
Context diagram, data flow diagram

- Context diagram

Chart showing relation between system and external terminator

Top of the data flow diagram

- Data Flow Diagram



TOPPERS

A context diagram is a notation utilized also with UML. It is a diagram which shows the relation between the system and external terminator(s). User I/F (user) and timer is designated to be the terminator with the instant noodle timer system, but if the timer is thought to be part of the system, then it is not necessary to designate it as a terminator.

As a description for the flow diagram a data flow diagram (DFD) is used. Data flow diagram is composed by the circle (process convert) and arrows that lead to (input data) and lead out (output data) of the circle. The circular portion performs conversion to the input data. With the instant noodle timer system there are two processes:

1. Process 1: key management process. By turning the switch on/off, data is converted to three output data: timer start notification, timer stop notification, and time extend notification.

2. Process 2: time control process. Timer time and request notification from the three switches is converted to output data of time display and timeout display.

These data flow diagrams are the first flow diagrams extracted from the system and is therefore called DFD0.

Requirement assessment 3

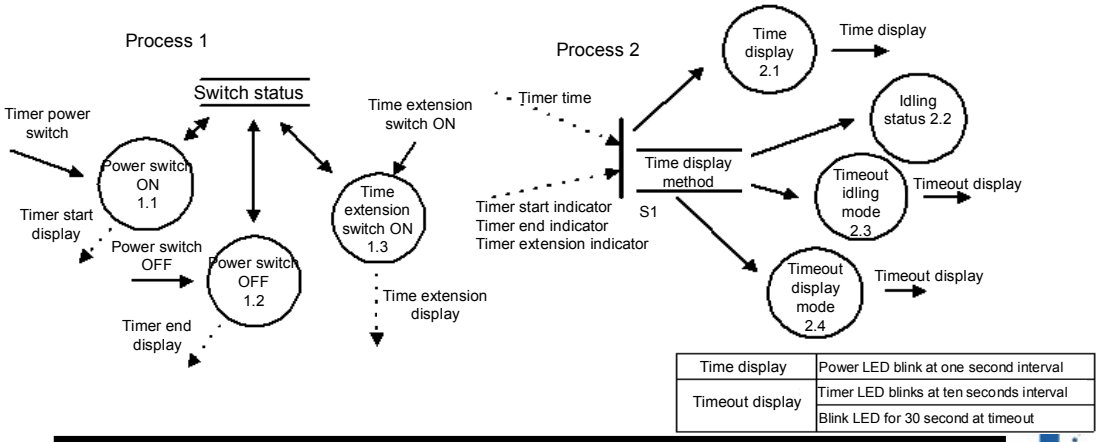
Detailed data flow diagram

- Process 1 detailed

  Key control functions detailed by diagram

- Process 2 detailed

  Time display functions detailed by diagram



Let us analyze the two processes of DFD0 in a detailed data flow diagram.

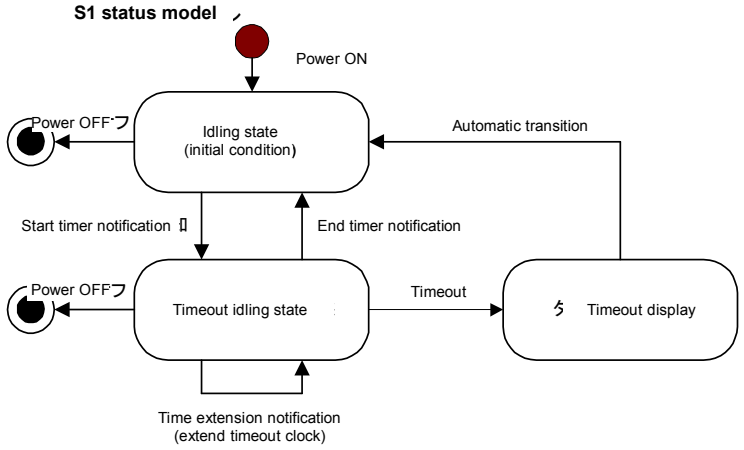
The switch status and time display method holds a condition. For detailing the condition, use the status model. Process 1 is broken down to three processes: timer switch ON 1.1, timer switch OFF 1.2, and timer extension switch ON 1.3. Process 2 is broken down into four processes: time display 2.1, idling status 2.2, timeout wait mode 2.3, and timeout display mode 2.4.

There are two displays for timeout, time lapsed display every 10 seconds and timeout display.

Requirement assessment 4

Time display form condition model（S1）

- S1 holds three conditions



TOPPERS

Here, the time display method has been modelized. Time display 2.1 is always displayed without being affected by the time display method and therefore is not noted in the status model. As timer functions, changes are made to three status conditions: idling status, timeout wait mode, and timeout display.

Requirement assessment 5

Process specification (2.2-2.4)

MT is maximum time, CT is current time

- 2.2 initial condition

1. Reset timeout time

   MT = 0、CT= 0

- 2.3 Timeout idling mode

First time round:

1. set timeout time

   MT = default timeout

For every extended time notification

1. extend timeout

   MT += 1 minute

For every control cycle

1. Idling timeout display check

   if CT is a multiple of 10 seconds:

   request 2 seconds of timeout display

2. Renew current time

   CT += Δ

- 2.4 Timeout display mode

1. Request 30 seconds of timeout display

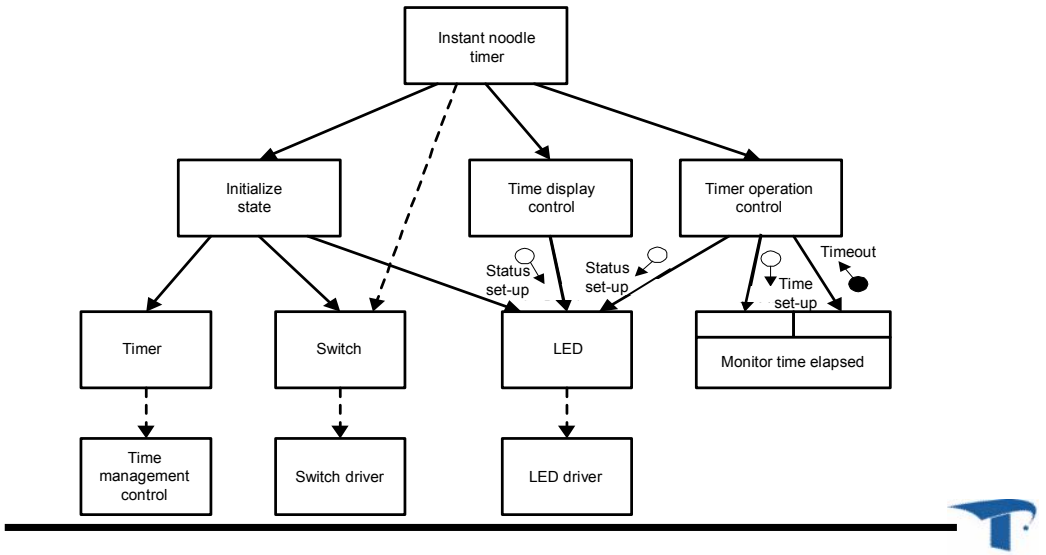
TOPPERS

Write a process specification document about the analyzed process. The document describes the changes made from input to output. So long as the specification is adequately described, there are no restrictions to how it is presented. In this example, a written specification was made, but it can be a diagram or a condition model as from the previous pages.

Construction 1

Hierarchical structure diagram

- Hierarchical structure diagram based on assessment of data flow analysis



TOPPERS

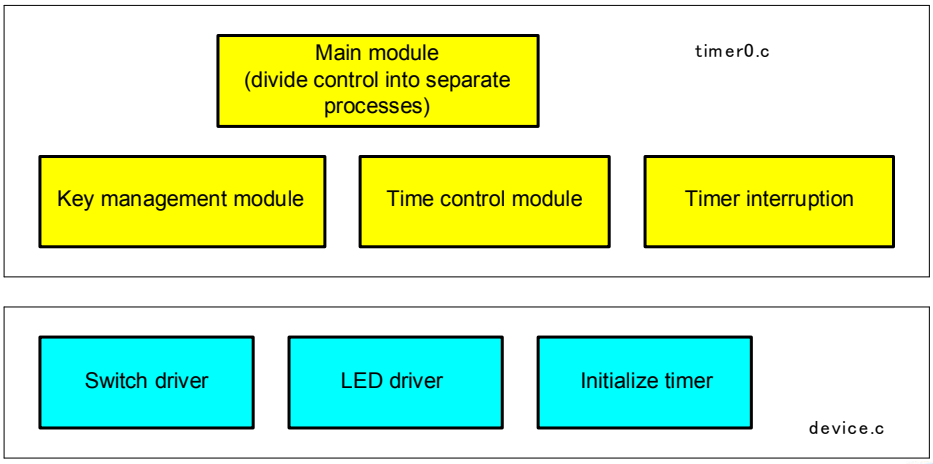
Create a hierarchical structure diagram from the structure analysis data flow diagram.

Functions will be extracted naturally by the DFD process. Next, the control modules will be extracted. In the above example, lapsed time monitor is the designated module to monitor time. Last, external input/output module will be added. In this diagram the timer, switch, and LED is the input/output module.

Construction 2

Structural diagram broken down into module units

- Divide units into process and device components



TOPPERS

Divide the structure diagram into module units. Here, they are separated into two parts, modules in relation with the DFD process and input/output modules.

Close examination is needed for module division. Details will be noted at time of review.

Construction 3

Module specification design

NAME	OBJECTIVE	parameter	OUT PUT	SOURCE
Main	Manage key and time control within a given period	none	none	timer0.c
Key control	Control key status and relay condition to time control	none	none	
Time control	Display LED for each time	none	none	
Initialize timer	Initialize timer hardware	none	none	device.c
Initialize switch	Initialize switch hardware	none	none	
Initialize LED	Initialize LED hardware	none	none	
Switch read-in	Read in switch condition	**number**	**condition**	
LED setup	Setup LED for lights on / off	**number condition**	none	
Timer interruption	Control current time through interruption of fixed time	none	none	timer0.c

Based on module division, create a module specification document. The document can be separated into interface specification and function specification. The specification of this slide is the interface specification. Since the instant noodle timer itself is not a large system and the specifications can easily be inferred from the objective, the function specification is omitted. The following is a brief explanation for the two specifications.

1.Interface specification

Description should be made based on what the module is assigned to do. Clarifying the input/output is important. As shown in the slide example, name, objective, parameter, and output should be noted as separate items.

2.Function specification

Description should be made based on how to achieve the module function.

e.g. – key control

(1) If timer start switch is on, then notify timer start

(2) If timer start switch is off, then notify timer stop

(3) if timer extension switch is on, then notify time extension

Construction 4
Making nonfunctional requisites clear

- Time interim
  handle through timer interim function
- Device interface
  At least two switches are outfitted, sensed through polling
  At least two LED ports are outfitted; lights turn on/off by setting
- Product requirements
  ROM less than 512KB, external RAM less than 16MB
- Error management mechanism
  No specific requirements

Functions that are not shown need to be clarified as well. With the instant noodle timer, hardware dependent function factors are part of the design. By separating factors which are implementation dependent for analysis and designing, a more comprehensive analysis and design can be done for the factors that are implementation independent. It is necessary to clarify non-function factors so that implementation designing will become easier.

The following are some examples of non-function factors:

1. Time restrictions
   Response time
   Within system classification : hard real-time, soft real-time
2. Resource restrictions
   ROM/RAM size
   CPU power
3. System restrictions
   development tool, factory inspection, maintenance in market
4. System security level
   Fault tolerance, overload, fail safe

Implementation

Refer to implemented object

- Previously mentioned outfitted object *non_os* is provided
- Details will be explained in the next section, "Movement n-OS learning material"

start.S	Initialization of hardware when power is turned on
sys_support.S	Vector table
device.c	Switch and LED control interface
timer0.c	Source file outfitted with main routine, two processes, and interim.

1. timer0.c

void main(void);

function to be executed after hardware startup, after device initialization, calls process 1 (switch_process) process 2 (timer_process) every 250ms.

void swicth_process(void);

timer extention switch; when the timer start switch changes, communicate information is written into the shared memory area

void timer_process(void);

if the shared memory is set, the timer state transition is executed. Without relation to transition state, LED blinks every second.

void timer_handler_entry(void);

timer interrupt program; interrupt occurs every 1ms, and base_time is incremented

2. device.c

void initial_key(void);

void initial_led(void);

initialize each device

Int get_key(int sw);

read-in key state corresponding to switch, ON/OFF

void set_led(int led, int req);

LED to reg state; ON turns light on, OFF turns light off

Test 1

What is a test?

- Objective of testing
  - Expand the range of quality guarantee as much as possible
  - Effectively detect malfunctions
- How to proceed with test
  - Comprehensive testing

    Overall testing

    control pass test, function inclusion test, state inclusion test
  - Pinpoint testing

    Target testing to points where there may be malfunctions

    borderline test, stress test

From past experience…

analyze the trouble and grasp the weak points

TOPPERS

The effect of trouble is enormous

•Because of one excessive line, out of 3 million, AT&T suffered damages of 1.1 billion dollars.

Long distance calls were turned busy for nine hours because an error recovery code was bypassed.

•General Motors had to make recalls due to a brake system bug

The stoppage distance was extended by 15-20 meters. 3.5 million cars were recalled, several million dollars were sustained in damages.

•ARIANE5 rocket exploded

400 million dollars were sustained in damages due to overflow.

Test 2

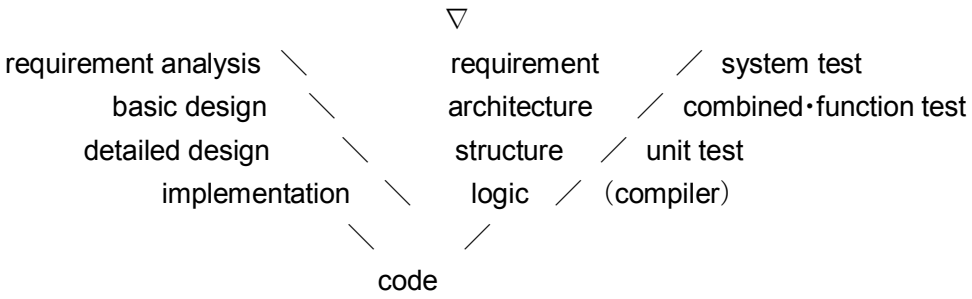
Test awareness

- Consideration on how to test
  （comprehensive, pinpoint）
- A test is…
  - ･planned sensibly
  - ･successful when you find a malfunction
  - ･diverse in method, and needs to be chosen to suit objective
- When you get better at testing, development with less malfunctions become possible
- Be conscious of testing so that you can create a malfunction-less software

TOPPERS

How to test

Embedded testing phase : V model



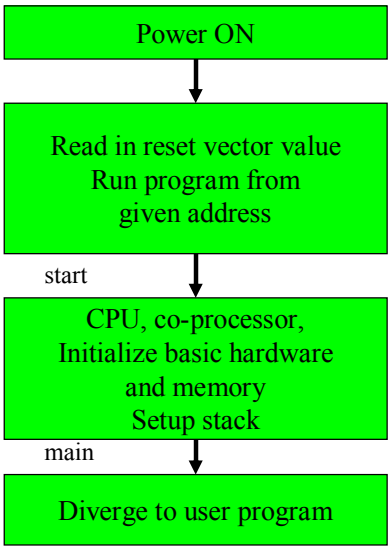
Confirming operation of non-OS teaching material

1. Checking the development tool
2. Embedded? Revising the SESSAME introductory seminar
3. **Confirming operation of non-OS teaching material**
4. What is Real-Time OS?
5. Constructing an instant noodle timer

Using the implemented training material, let us confirm, build, and run a simple test.

Implementing knowledge 1
Startup

- In general operating systems, the OS gives command to main()
- When there is no operating system, you must create a program that gives command to main()
- This process is called startup



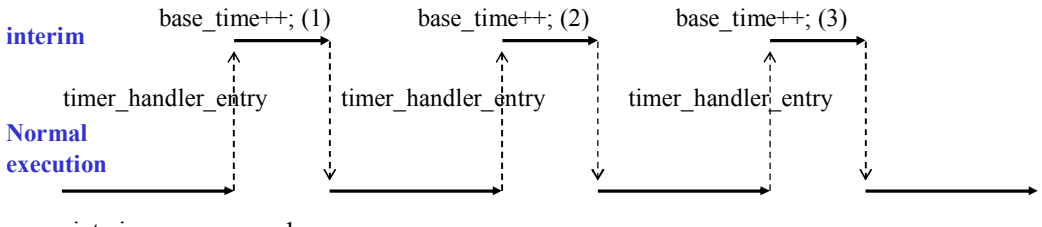
TOPPERS

```
/*
 *  instant noodle timer main function
 *  after initializing hardware and data, start switch_process and timer_process every 250ms.
 */
void
main(void)
{
        unsigned long timer250;

        initial_key();                  /* initialize switch */
        initial_led();                  /* initialize LED */
        initial_timer();                /* initialize timer */
        base_time = 0;
        event = 0;
        timer250 = 250;
        enaint();                               /* permit interrupt  */
        up_sw = get_key(UP_SW);                 /* read-in current timer extension sw */
        start_sw = get_key(START_SW);           /* read-in current timer start sw */
        timer_state = STATE_STOP_TIMER;  /* initial timer_process state */
        sec = 0;
        alaem = 0;
        timer_led = OFF;
        pow_led = OFF;

        for(;;){                                /* endless loop */
                if(timer250 <= base_time){
                        switch_process();
                        timer_process();
                        timer250 += 250;
                }
        }
}
```

Implementing knowledge 2

Interim action

- Interruption can be made when the hardware wishes to insert a particular action to the CPU (timer_handler_entry)
- Set the timer_handler_entry to the interrupt vector table and setup the interrupt to the hardware



TOPPERS

```
/*
 *  Switch process
 */
void
switch_process(void)
{
	unsigned char sw;

	sw = get_key(START_SW);
	if(start_sw != sw){
		if(sw == ON)
			event |= EVT_TIMER_START;
		else
			event |= EVT_TIMER_STOP;
		start_sw = sw;
	}
	sw = get_key(UP_SW);
	if(up_sw != sw){
		if(sw == ON)
			event |= EVT_TIMER_COUNT;
		up_sw = sw;
	}
}
/*
 *  Timer interrupt function
 *    start every 1ms.  Start from base_time has millisecond counts
 */
void
timer_handler(void)
{
	base_time++;
}
```

Implementing knowledge 3

Verifying the source code

- Outfit process1 (key control) as switch_process() and process2 (timer control) as timer_process()
- The two processes are called by main() every 250ms and is executed
- base_time is incremented at one minute intervals by interim function. main() monitors base_time through polling


```
/*
 *  Timer process
 */
void
timer_process(void)
{
        if(event != 0){                                 /* notification from switch_process */
                if(event & EVT_TIMER_START){            /* start */
                        timer_state = STATE_ACT_TIMER;
                        current_time = 0;
                        max_time = 60*1000;
                }
                if(event & EVT_TIMER_STOP){             /* stop */
                        timer_state = STATE_STOP_TIMER;
                        alaem = 0;
                }
                if(event & EVT_TIMER_COUNT){            /* timeup */
                        if(timer_state == STATE_ACT_TIMER){
                                max_time += 60*1000;
                        }
                }
                event = 0;
        }
```

Study
Implementation with interrupt and scheduling

- If it is a minimal development, the outcome is clear-cut

  Because the example is simple, it is easily implemented. But with complex systems, implementation of modules are difficult to create due to hardware dependency which cannot be clearly defined

- Actual system load is unknown

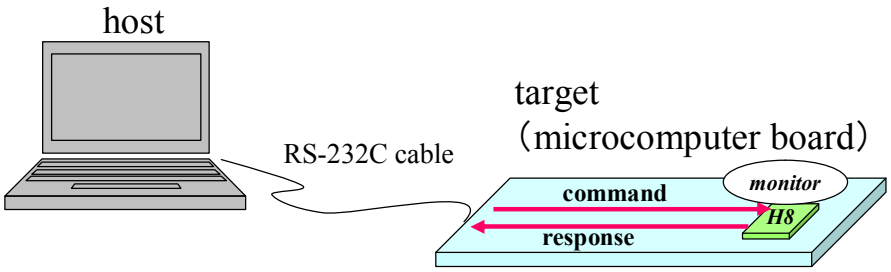
  It is unknown how much load the program is putting onto the system

  → assurance of real-time


```
/* continued from previous page */

        switch(timer_state){                /* judge timer state */
        case STATE_ACT_TIMER:               /* timer executing state */
                if(current_time >= max_time){
                        alaem = 15*4;
                        timer_state = STATE_TIMEOUT;
                }
                else if((current_time % (10*1000)) == 0){
                        alaem = 1*4;
                }
                current_time += 250;        /* timeup */
                break;
        case STATE_TIMEOUT:                 /* timeout state */
                timer_state = STATE_STOP_TIMER;
                break;
        default:                            /* timer stopped state */
                break;
        }
        if(++sec > 3){                      /* display time:1sec past */
                pow_led ^= ON;              /* power check LED ON/OFF */
                sec = 0;
        }
        if(alaem > 0){                      /* request warning:250ms lapsed */
                timer_led ^= ON;            /* timer display LED ON/OFF */
                alaem--;
        }
        else            /* if there is no warning request and timer display LED is ON, then turn timer LED off */
                timer_led = OFF;

        set_led(TIMER_LED, timer_led);  /* timer display LED setting */
        set_led(POW_LED, pow_led);      /* power check LED setting */
}
```

Test environment 1

Role of the monitor



- The monitor is a program on the target
- It receives commands from the host and responds by supplying debugging functions
  - User program download
  - Execute user program

Test environment 2
Merits of using a monitor

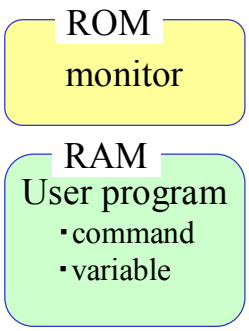
Compared to writing onto a ROM

1. Saves time
   - Writing time, ROM exchange time
2. If the ROM is a flashROM, there is no need to worry about rewrite restrictions

Test environment 3

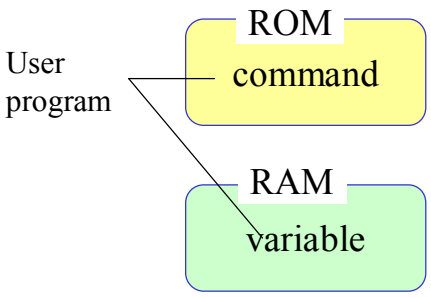
Memory mapping

1. Debugging（monitor）



・debugging through a monitor
User programs are downloaded all to RAM

2. ROM written（no monitor）



・ROM written
Command is allocated to ROM

Execution procedure

Refer to attached material for actual execution procedure

For detailed operational procedures, see the attached <Tools use direction> booklet.
(BeginnerTranningSeminarH8Tool0010102.pdf)

Test/confirm operation

Conduct a pinpoint type test

1. Turn power ON and check that power LED is blinking
2. Turn timer ON/OFF switch, confirm that timer starts up
3. Confirm that timeout is notified in one minute
4. Turn time interim ON/OFF switch, confirm that timeout is extended
5. If there are no problems, conduct the test

If there is a problem, speak up!!

TOPPERS

Turning the power ON starts the instant noodle timer.

Using the two switches, check if the timer activates as specified.

What is Real-Time operating system?

1. Checking the development tool
2. Embedded? Revising the SESSAME introductory seminar
3. Confirming operation of non-OS teaching material

Here, significance for using RTOS, such as the TOPPERS/JSP kernel, to design systems is explained. In succession, a quick note on the μ ITRON4.0 specification and description of task and task scheduling will continue.

Afterwards, using the board, execution of the sample program timer1 on the TOPPERS/JSP kernel will be done.

μ ITRON4.0 specification Ver.4.00.00 can be downloaded from the following site:

http://www.assoc.tron.org/eng/document.html

Role of the Operating System (OS)

There is no clear definition of an operating system

- Abstraction, virtualization, and diversification of computer property
  - Make property accessible from application
  - Improve application portability
- Collectively manage computer property efficiently and securely
  - Efficiently utilize property – i.e. make access possible by multiple applications
  - Elevate security by controlling accessibility to property

The main objective of an operating system is to manage the computer system efficiently. It has evolved by pursuing this objective. To cope with the rapid progress of hardware, means for software that does not depend on hardware and improving user convenience plays a great role.

As part of an operating system, a file system virtualizes and multiplex hardware resources and can be said to be a safe and efficient managing system.

What is a Real-Time OS（RTOS)?

Operating system to construct a real-time system

Standpoint of requirements for construction of a real-time system differs depending on the operating system

- Holds forecast possibility
  each service time of OS is priorly established
- Has capacity for real-time system
  supports priority integration and precedence upper limit protocol
- Control time restriction（handled partially by some study base）
  The OS takes into account the time restriction of each process and determines the scheduling
- Speedy response

There are many definitions to a real time system. A basic definition is “a system dependent to result output time with correct process result added to the result value”. A real-time system is not only a system where a fast response is demanded. Many embedded systems are real-time systems. Within these, there some where real-time form is needed. A system which demands a critical real-time form is called a hard real-time system.

In the above factors, many commercial real-time operating systems have prediction possibilities, but the levels differ depending on the operating system. Also, functions to constructing real-time operating systems are supported. Time restriction control is still in research.

During synchronization processing, a task with a low priority may hinder a task with a high priority. When the task with a high priority needs to process a critical execution, this becomes a problem. There are several ways to deal with the problem.

•Basic Priority Inheritance Protocol

Gives the low priority task hindering the high priority task a higher priority

•Priority Ceiling Protocol

When entering a critical session, a blocking mechanism is invoked to prevent chain blocking

Composition of Real-Time Operating System
Real-time kernel

- Core module of RTOS
- Mainly handles property that is common to most systems

  processor, memory, timer, etc.
- Many do not have protective functions
- May be called “Real-time monitor” or “Real-time executive”

  when employing real-time kernel exclusively,

  RTOS = Real-time kernel

Two characteristics of embedded system seen from operating system point of view

• Protective functions are not essential

The sole design purpose of an embedded software is to control the target embedding device. This means that the software itself is fixed to the device. Accordingly when debugging and testing is through, the premise that the application software is reliable can be made. Therefore, functions for protection is not necessary needed.

• I/O in need of absolute support does not exist

There is an opinion that resources which operation takes time should be materialized on the kernel. The I/O device is slow compared to processors and memory, and is placed on the kernel minimally. Also, I/O devices common to many embedded devices are nonexistent and it is rare for the same I/O device to be shared by multiple applications. However, embedded systems with characteristics similar to general systems has become popular in recent years (e.g.- PDA, mobile phones)

Main functions of the Real-Time OS

- Function of real-time kernel
  - multitask mechanism → accession and virtualization of processors
  - linkage between tasks, synchronization mechanism
  - interim control/processing, anomaly control/processing
  - time synchronization/control → accession and virtualization of timers
  - memory control →accession and virtualization of memory
  - system control

  etc.
- Functions outside of real-time kernel
  - input/output control function
  - communication/network function
  - user interface function (e.g. GUI)
  - program module control function etc., etc.

TOPPERS

Multitask structure

A task is a unit of parallel executed programs. A program that is processing a task is executed in order. Also a program with different task is executed in parallel. Therefore it can be said that the processor has been virtualized/multitasked. A multitask structure is where multiple tasks are executed in a pseudo parallel fashion. With a single processor, only one task can be executed at a time. In reality, it is the illusion that multiple tasks are executed simultaneously.

Time synchronization/control

ITRON4.0 controls the system time and event occurrences by relative time. Each task can be run by separate relative time occurrence event, and therefore can be said that the timer is virtualized.

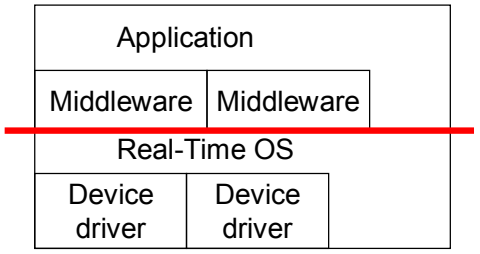
Memory control

General operating systems such as UNIX and Windows utilizes virtual memory for running user programs in separate memory spaces. ITRON, which does not have a memory preservation function, utilizes a memory pooling function and controls memory areas dynamically.

Categorizing Real-Time OS by architecture

General OS type and real-time kernel type

- General OS type



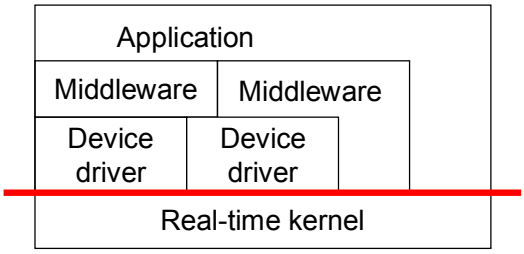
OS functions are abundant

Size is relatively large

Response time is generally slow

Device driver is generated with a separate API

- Real-time kernel type



Kernel functions are limited

Kernel size is small

Response time is generally fast

Device driver and application are the same API

TOPPERS

Operating system role

•Abstract, virtualize, and multiply computer resources

Access to resources from application is made easier and portability from application is improved

•Collectively control resources safely and efficiently

Simultaneous access to resources by multiple applications is made possible. Access rights to resources is controlled, improving security

There is no clear definition for what range of software is to be used for an operating system. For example, a broad software such as the Windows system can be considered to be a part of an operating system for its role. However, compared to real-time kernel of embedded systems, the size and functions differ greatly.

Advantages of utilizing Real-Time OS

Structuring of software becomes easier

- Structuring of software becomes easier

  Enhance productivity, maintenance, and reliability through structuring

  (approach differs from modulation by structural design technique)

- Construction of time restricted multiple processes made easier

  ➡ **Software development base using software parts**

- Conceal difference between hardware (processor)

  Capable of developing software without detailed knowledge of hardware

TOPPERS

Merits for utilizing real-time operating system

•Task utilizing software structure

By assigning independent processing to tasks, multiple subsystems can be processed on the same processor, separate I/O devices and event processing becomes possible. Separation of logical processing order and time based processing order can be achieved. The program accounts the logical processing order by task unit therefore making time based processing order execution possible.

•Construction of multi-processing system with time restriction is made easy

By separating logical processing order and time based processing order, maintenance and recycling is improved. This becomes the base for software development (component base development) utilizing software parts.

•Conceal hardware (especially processors) differences

It is possible to develop applications without knowledge of hardware specifics. This does not mean that it is not necessary to the project. It is said that “A real-time kernel is the processor’s device driver”

Disadvantage of using Real-Time OS

Engineer with knowledge of the OS is needed

- Decline of performance due to OS overhead
- Memory consumption by OS or executing tasks

  TOPPERS/JSP kernel is a Real-Time OS which memory consumption is low

- During obstacle analysis, it may become necessary to analyze within the OS

An engineer who has knowledge of the operating system is needed within the production

Demerit of utilizing real-time operating systems and avoidance scheme

•Operating system overhead affecting execution performance

•Memory consumption by the operating system

In exchange for merits utilizing operating systems, memory efficiency declines. However, by the improvement of implementation techniques, memory size has become small.

•Debugging and testing becomes difficult

Well conducted synchronization and communication between tasks are necessary. Techniques based on model base designing and simulations to make the software more visible is also effective.

•Analysis becomes difficult due to operating system black box manifestation

Comprehend the internal works of the real-time operating system. Various development tools can be used, such as OS monitoring, for improvement.

μITRON specification kernel functions
(μITRON4.0 specifications)

■ Kernel functions

- Task management function
- Task dependent synchronization function
- Task exception handling function
- Synchronization and Communication functions
- Extended Synchronization and Communication functions
- Memory pool management function
- Time management function
- System state management function
- Interrupt management function
- Service call management function
- System configuration management function

■ Service call number

- Full set
  Service call:166(13*) staticAPI:21
- Standard Profile
  Service call:70(13*) staticAPI:11
- Automotive Control Profile
  Service call:43(11*) staticAPI:8
- Minimum set

* Number in brackets are non-task context service calls

TOPPERS

Service call is an interface used when tasks, such as application programs, send requests to the kernel. General kernel functions are as follows:

•Task management function

Function used to handle/refer task condition directly. Some of the functions included are task create and delete, start and stop, cancel start request to task, and task precedence alteration and reference function.

•Task dependent synchronization function

Synchronization function for directly handling the task condition. Task wakeup wait function and wakeup function, cancel wakeup request function, force cancel task waiting function, transition to force wait state function, and delay task execution function are some examples.

•Task exception handling function

Function which handles task exceptions within the task' s context. Functions included are the ability to define a task exception routine, to request a task exception handling, to enable and disable task exception handling, and to reference the state of a task exception handling.

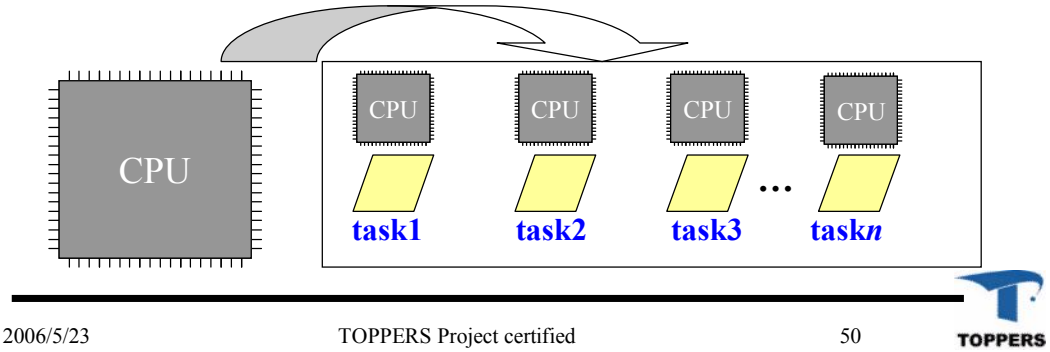
•Synchronization and communication functions

A task is a function used for synchronization/communication between tasks by an independent object. Functions include semaphore, eventflags, data queue, and mailbox.

What is a task?

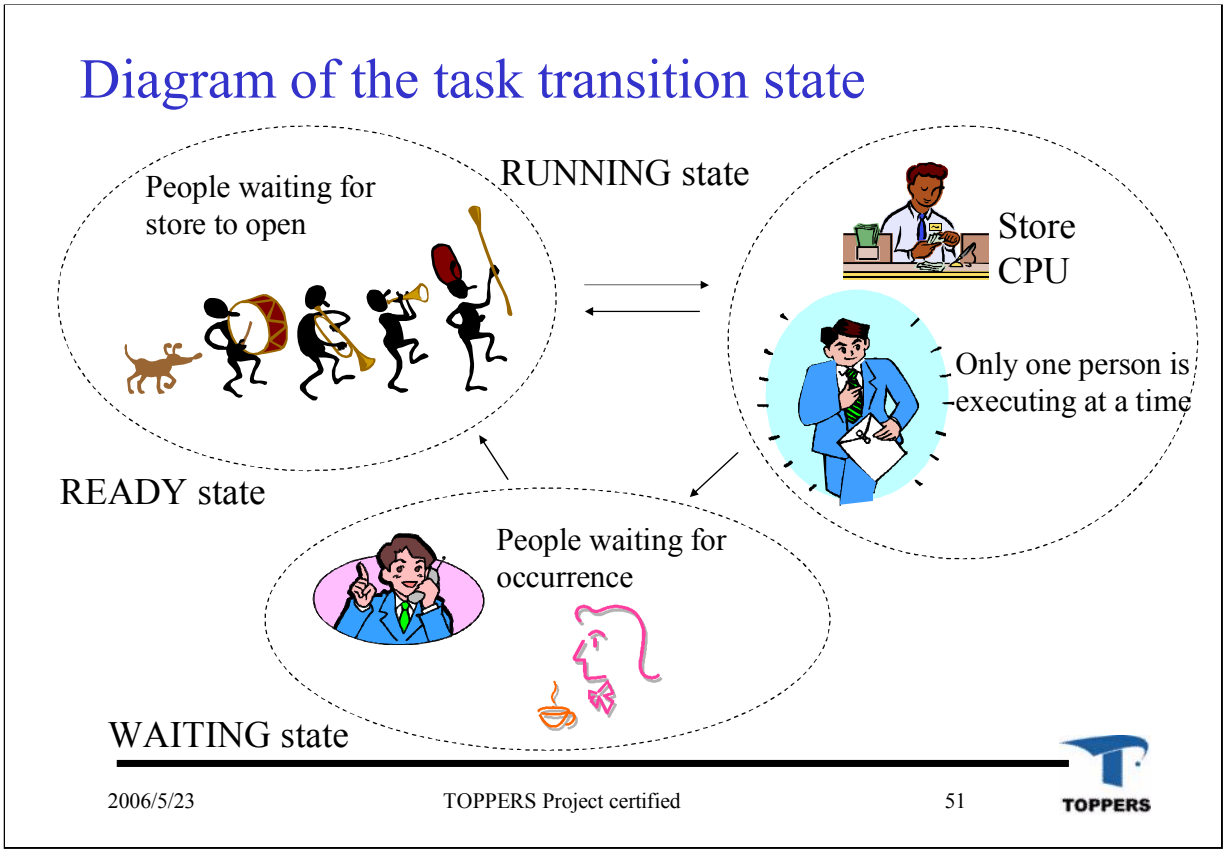
Think on a small scale using tasks

- A task is an execution path through address space
- Accession and virtualization of processors
- Benefits of tasks
  - seems as if each task has its own CPU
  - breaks down a large problem for analysis



TOPPERS

A task is a program execution unit with a purpose. A program within a task is executed in order. Differing task of a program is executed in parallel in a pseudo manner. Real-time operating system supports the multitask structure. A multitask structure are multiple tasks executed with the illusion of parallelism. With a single processor, only one task can be executed at a time. It is the illusion that multiple tasks are being executed that is created.



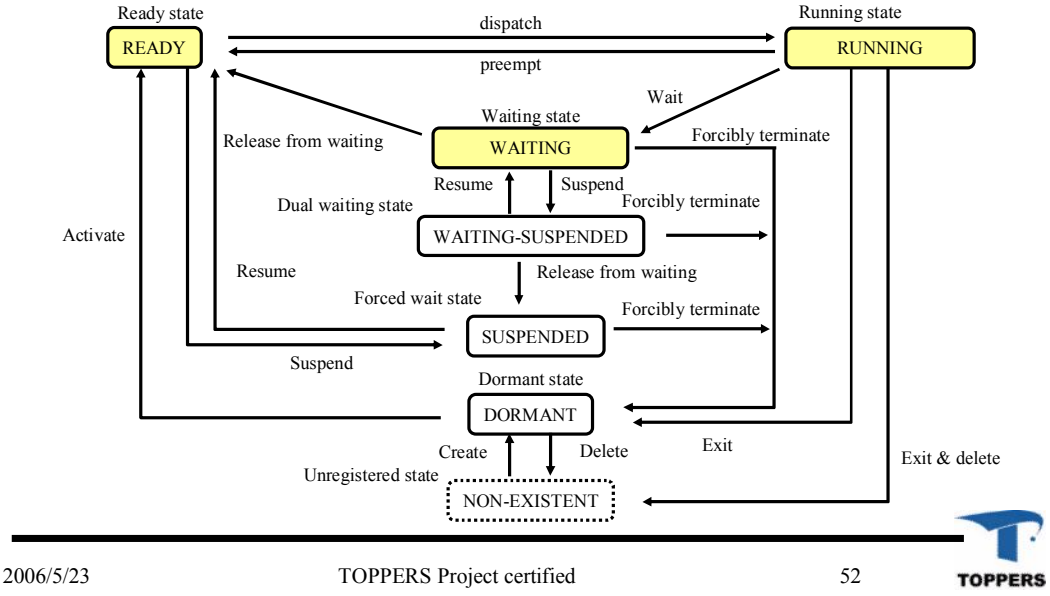
Why are multiple tasks necessary? In real-time operating systems, the motive differs from multitask (TSS) of general systems. With real-time operating systems, independent processes are assigned to independent tasks, therefore multiple subsystems are processed by a single processor and separate I/O devices and event processing is made possible.

How can this be achieved? It can be achieved by separating the system into a logical processing order and timely processing order. Logical processing order is defined in a program and is executed in the timely processing order. By this exchange, the program maintenance and reusability improves, and induction of software parts developed externally is made easier.

Transition of task condition

Singular task running at one time

- μITRON4.0 specification task condition transition



TOPPERS

Task conditions

(1) RUNNING

Task is currently running. However, if non-task context is running, the task holds its current RUNNING state unless otherwise specified.

(2) READY

Task is ready to execute but cannot do so due to a higher precedence task is executing.

(3) WAITING

Execution is blocked due to invocation of a service call specifying the condition that must be met before task execution.

(4) SUSPENDED

Execution is forcibly halted by another task.

(5) WAITING-SUSPENDED

State where the task is waiting for a condition to be met and is suspended.

(6) DORMANT

State where the task has not yet begun execution or has already finished. When in this state, the context information is not saved

(7) NON-EXISTENT

Task does not exist because it has not yet been created or has been deleted

Basic transitional service calls

Task management functions & task associated synchronize function

- Task management functions

CRE_TSK	create task
act_tsk,iact_tsk	activate task
can_act	cancel task activation request
ext_tsk	exit task
ter_tsk	terminate task
chg_pri	change task priority
get_pri	reference task priority

- Task associated synchronize functions

slp_tsk	put task to sleep
tslp_tsk	put task to sleep (with Timeout)
wup_tsk,iwup_tsk	wakeup task
can_wup	cancel task wakeup requests
rel_wai,irel_wai	release task from waiting
sus_tsk	suspend task
rsm_tsk	resume suspended task
frsm_tsk	forcibly resume suspended task
dly_tsk	delay task

TOPPERS

•Static API

With the μITRON4.0 specification, the method to statically create the object which composes the system has been standardized and static API was prescribed. In the above slide, the function written in capital are static APIs. The static API is defined in the configuration file. Using the configurator (TOPPERS/cfg/cfg_exe) create the include file (kernel_id.h) and C language file (kernel_cfg.c). Other service calls are defined in the C language interface. Here, CRE_TSK is used as an example.

```
CRE_TSK(ID tskid, {ATR tskatr, VP_INT exinf, FP task, PRI itskpri,
                                                SIZE stksz, VP stk} );
```

【Parameter】

ID tskid	: task ID number
ATR tskatr	: task attribute ((TA_HLNG || TA_ASM) | [TA_ACT])
VP_INT exinf	: task extension information
FP task	: task start address
PRI itskpri	: task initial priority
SIZE stksz	: task stack size (in bytes)
VP stk	: task stack space base address

Multitask structure organization

What makes multitasks possible

- Multitask structure is made for concurrent operation by one central processing unit of two or more processes

  Make it appear as multiple tasks are executed simultaneously, where in fact there is only one task executed at a time

- Dispatch

  (dispatcher : switching module)
- Scheduling

  (scheduler : task determination module)
- Scheduling algorithm

Multitask structure organization factor

•Dispatch (task dispatch, task switch)

Switching of currently executing task on a processor with another. The module that performs the dispatch is called a dispatcher.

•Scheduling (task scheduling)

Process which determines what task is to be executed at what time. In many real-time operating systems, it is the process that determines which task to be executed next. Module that executes the schedule is called a scheduler.

•Scheduling algorithm

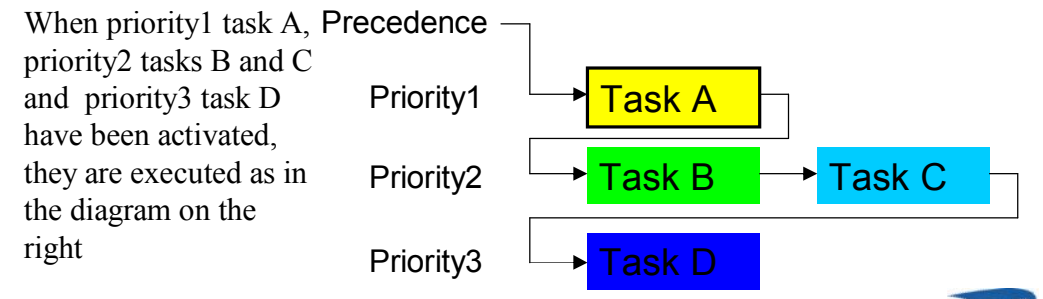
The method which determines the next executed task. In most real-time operating systems, including TOPPERS/JSP kernel, preemptive precedence based algorithm is supported. There are real-time operating systems which use other scheduling algorithms as well.

Scheduling algorithm of μITRON

Preemptive priority based scheduling

- Priority based scheduling
  - Task with the highest priority is executed
  - Lower priority tasks will not be executed until the high priority tasks are fully executed
  - When tasks are prioritized at the same level, it is executed on a FCFS (First Come First Served) basis

When priority1 task A, priority2 tasks B and C and priority3 task D have been activated, they are executed as in the diagram on the right



TOPPERS

•Priority based scheduling

Typical real-time scheduling mechanism of event driven scheduling. Task with the highest precedence is executed first. The task with the higher priority is executed. Until the task with high precedence is finished executing (or is terminated due to other reasons), tasks with lower precedence will not be executed. Preemptive and non-preemptive implementation is possible. For setting of precedence, static priority assignment, dynamic priority assignment, and dynamic priority changing can be set.

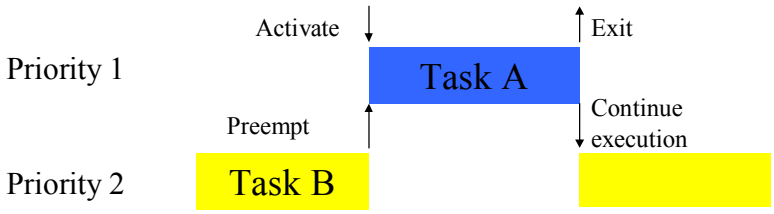
Task priority with μITRON is noted as integers larger than 1; the smaller the value, the higher the priority.

Scheduling algorithm of μITRON

Preemptive priority based scheduling

- Preemptive scheduling
  - when a high priority task turns to RUNNING state, the lower priority task will be moved to READY state even if the task was in execution
  - interruption is the key to changing of status switching

Task A has high priority and short execution time, Task B has low priority and long execution time



TOPPERS

• Preemptive and non-preemptive

Preempt means to appropriate, seize, or take for oneself before others

Preemptive scheduling : mechanism where a process is in execution, if the need arises to switch processes, the switch is made. There is a need to be a mechanism where a process can be halted and resumed. Normally, the operating system prepares such mechanism and the application program will not be aware of the switch. Be aware that when the interrupt occurs, resources apart from the processor is still secured.

Non-preemptive scheduling : when the process starts execution, no interruption is allowed until the execution is finished.

•Separation of logical process order and timely process order

In the above mentioned task A and task B, if execution of task A is attempted without a task mechanism, a periodical judgment procedure needs to be included into task B. By this, the maintenance and reusability of task B declines. Also, reply may decline and overhead due to checking may occur.

Constructing an instant noodle timer

1. Checking the development tool
2. Embedded? Revising the SESSAME introductory seminar
3. Confirming operation of non-OS teaching material
4. What is a Real-Time OS?
5. **Constructing an instant noodle timer**

Let us make an instant noodle timer using RTOS.

In this development tool, the debugger cannot be used, so utilize the system log display function for debugging.

Timer construction using JSP kernel 1

Introduction of sample program

- Composed from two tasks
- ENTRY_TASK : timer power switch status corresponds with timer LED
- TIMER_TASK : power LED to blink at one second interval
- When ENTRY_TASK is activated, TIMER_TASK is activatied by act_tsk
- Debugging is done by using the syslog_n function and showing current status on the log

The system log is not directly displayed by the task which requested the log. Since it is accumulated in the log table then displayed by LOGTASK, display through non-task context is possible. By lowering the LOGTASK priority, the log can be displayed without hindering the usual task execution. However, if execution rights is not given to LOGTASK over a long period, it is possible that the log message may be lost.

Timer construction using JSP kernel 2
Configuration file（timer1m.cfg）

- Define task by CRE_TSK

```
CRE_TSK(ENTRY_TASK, { TA_HLNG, 0,
entry_task,DEFAULT_MAIN_PRIORITY,STACK_SIZE, NULL});
CRE_TSK(TIMER_TASK, { TA_HLNG, (VP_INT) 0, timer_task,
TIMER_PRIORITY, STACK_SIZE, NULL });
```

- Establish serial driver and driver initialization and interrupt through serial.cfg
- Timer interrupt and time service setup through timer.cfg


```
#define _MACRO_ONLY
#include "./timer1.h"

INCLUDE("¥"timer1.h¥"");
CRE_TSK(ENTRY_TASK, { TA_HLNG | TA_ACT, 0, entry_task, DEFAULT_MAIN_PRIORITY,
                STACK_SIZE, NULL});
CRE_TSK(TIMER_TASK, { TA_HLNG, (VP_INT) 0, timer_task, TIMER_PRIORITY,
                STACK_SIZE, NULL });

#ifdef CPUEXC1
DEF_EXC(CPUEXC1, { TA_HLNG, cpuexc_handler} );
#endif /* CPUEXC1 */


#include "../../systask/timer.cfg"
#include "../../systask/serial.cfg"
#include "../../systask/logtask.cfg"
```

Timer construction using JSP kernel 3

Consideration of instant noodle timer（ENTRY_TASK）

- Initialize hardware (initial_key, initial_led)
- Activate task every 100ms. Waiting status at other times (WAIT state). Task status shift by tslp_tsk()
- Retrieve timer power switch status (get_key()), if there is a change, rewrite timer LED status (set_led())
- sw = get_key(START_SW);

  Retrieve timer switch status.

  Status is either ON or OFF.


```
/*
 *  main task
 *  (switch process)
 */
void
entry_task(VP_INT exinf)
{
        UB      start_sw, sw;
        UB      time_led = OFF;

        syslog_1(LOG_NOTICE, "Sample entry task starts (exinf = %d).", exinf);
        initial_key();                          /* initialize key */
        initial_led();                          /* initialize LED */
        start_sw = get_key(START_SW);     /* read-in current timer start sw */
        act_tsk(TIMER_TASK);                /* start timer task */

        for(;;){
                tslp_tsk(500);
                sw = get_key(START_SW);
                if(start_sw != sw){
                        syslog_1(LOG_NOTICE, "Change START_SW = 0x%x.", (int)sw);
                        if(sw == ON)
                                time_led = ON;
                        else
                                time_led = OFF;
                        start_sw = sw;
                }
                set_led(TIMER_LED, time_led);    /* timer LED setup */
        }
}
```

Timer construction using JSP kernel 4

Consideration of instant noodle timer（TIMER_TASK）

- Get current time (get_tim(&current_time);), activate timer at 250ms units (tslp_tsk(base_time - current_time));
- Should the base_time-current_time become a negative figure, what to do?
- Change power LED state every second
- set_led(POW_LED, pow_led);

  pow_led shows the state of the power LED, LED light on (power ON) and off (power OFF)


```
/*
 *  Timer task
 */
void
timer_task(VP_INT exinf)
{
        SYSTIM  base_time, current_time;
        TMO     tmout = 0;
        UB      pow_led = OFF;

        syslog_1(LOG_NOTICE, "Sample1 timer task starts (exinf = %d).", exinf);
        get_tim(&base_time);                    /* read current time */
        for(;;){
                tslp_tsk(tmout);                /* TICK wait */

                                                /* turn on LED at odd second */
                if ((base_time / T_1SEC) & 0x01) {
                        pow_led = ON;
                } else {
                        pow_led = OFF;
                }
                set_led(POW_LED, pow_led);

                base_time += T_TICK;
                get_tim(&current_time);
                tmout = base_time - current_time;
                if(tmout < 0)
                        tmout = 0;
        }
}
```

Timer construction using JSP kernel 5

Confirm development tool

- Development will be done in the timer2 directory
- The same programs contained in the timer1 directory exists in the timer2 directory
- Modify the source code for timer2, and actualize the functions
- After reprogramming, go back and check the alterations made

Before modifying

Check sample program content

1. Build sample program contained in the timer1 directory
   - Step 1 make depend
   - Step 2 make
2. timer1 is a project where two tasks executes; the LED blinks every second and displays switch state to LED
3. Using the simple monitor, download jsp.srec to target and execute

1.timer1.cfg

Create timer1 project configuration file, entry_task and timer_task, interrupt setting for serial driver, and set timer to be used by TOPPERS/JSP kernel

2. timer1.c

void timer_task(VP_INT exinf);

timer task : power LED blinks every 1 second

Void entry_task(VP_INT exinf);

Initializes key and LED device. There is no need for timer initialization because it is managed by the TOPPERS/JSP kernel. After initialization, start up at every 100 milliseconds; if timer start switch is on, then turns the timer display LED on, if switch is off, then turn timer display LED off

Appendix

The microcomputer board

AKI-H8/3069F

Outline and external specification

- RAM16KB, ROM512KB（external RAM16MB）
  4 switches, 2 LED can be controlled

OUTLINE

* Only DIPSW1-1 and DIPSW1-2 will be used in this seminar

External specification

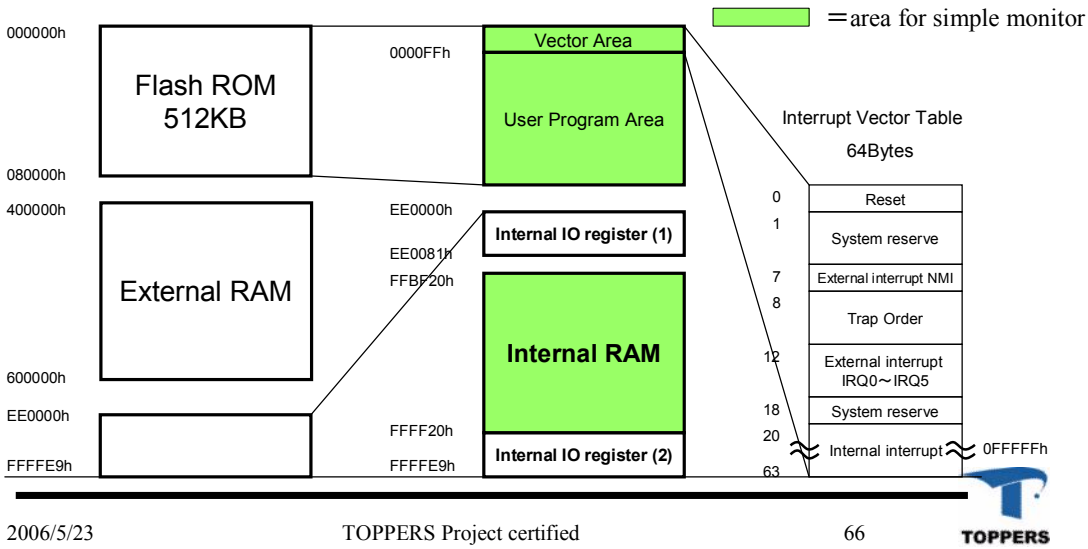
Item	Content
MCU	MCU:H8/3069F Action mode：expanded mode 5 Clock frequency：main clock 20MHz
Memory	Internal memory RAM :16KB Flash ROM :512KB External memory RAM :16MB

TOPPERS

AKI-H8/3069F

Memory map

- In this seminar the simple monitor will be used



2006/5/23 TOPPERS Project certified 66

AKI-H8/3069F memory map

Internal memory

RAM :16KB

Flash ROM :512KB

External memory

RAM :16MB

The simple monitor uses the Flash ROM and inner memory RAM

AKI-H8/3069F Circuit diagram

For the circuit diagram, see the diagram attached to AKI-H8/3069F

AKI-H8/3069F circuit diagram

Input/output port & connection correspondence

Logical name	Name	H8 port number
Timer start switch	DIPSW1-1	$P5_0$
Timer extend switch	DIPSW1-2	$P5_1$
(not in use)	DIPSW1-3	$P5_2$
(not in use)	DIPSW1-4	$P5_3$
Power LED	LED1 (red)	$P4_6$
Timer LED	LED2 (green)	$P4_7$

Since these hardware dependencies are hidden by the device driver, the application programmer does not need to be aware of them

A simple manual for the device driver is in toppers-beginner-h8/device/readme.txt

Mechanism for LED blinking

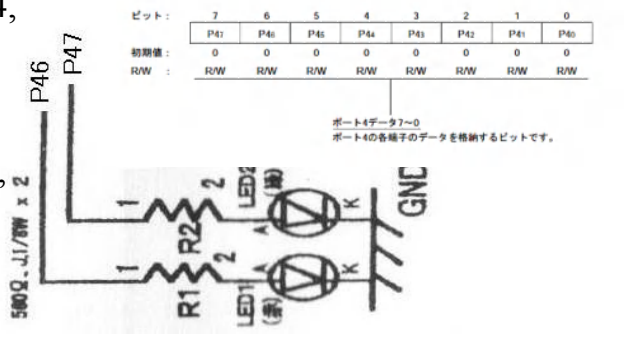
What type hardware controls it?

- LED is controlled by I/O port 4
- Port 4 is set for outputting
- LED 1 is connected to port 4, 6 bits and LED 2 is connected by 7 bits
- Cathode is molded to the LED. When it is set at 1 “H” level, an electrical current runs through and lights up

表 8.6　ポート 4 レジスタ構成

アドレス*	名　称	略　称	R/W	初期値
H'EE003	ポート 4 データディレクションレジスタ	P4DDR	W	H'00
H'FFFD3	ポート 4 データレジスタ	P4DR	R/W	H'00
H'EE03E	ポート 4 入力プルアップ MOS コントロールレジスタ	P4PCR	R/W	H'00

【注】　* アドバンストモード時のアドレス下位 20 ビットを示しています。



TOPPERS

Switch sensing mechanism

What type hardware controls it?

- Switch is connected to I/O port 5
- Port 5 is set for input
- DIPSW1-1 is connected to port 5, 0 bit, DIPSW1-2 is connected to port 5, 1 bit
- This switch is grand on negative logic, and therefore changes to 0 ”L” level when turned on

表 8.8　ポート 5 レジスタ構成

アドレス*	名　称	略　称	R/W	初期値	
				モード 1～4	モード 5、7
H'EE004	ポート 5 データディレクションレジスタ	P5DDR	W	H'FF	H'F0
H'FFFD4	ポート 5 データレジスタ	P5DR	R/W	H'F0	
H'EE03F	ポート 5 入力プルアップ MOS コントロールレジスタ	P5PCR	R/W	H'F0	

【注】　* アドバンストモード時のアドレス下位 20 ビットを示しています。