To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

Application Note

16

H8S/2655 Series
On-Chip Supporting Modules
Application Note
Renesas Microcomputer

Renesas Electronics
www.renesas.com

Rev.1.0 1997.09

Notice

When using this document, keep the following in mind:

1. This document may, wholly or partially, be subject to change without notice.
2. All rights are reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without Hitachi's permission.
3. Hitachi will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's unit according to this document.
4. Circuitry and other examples described herein are meant merely to indicate the characteristics and performance of Hitachi's semiconductor products. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.
5. No license is granted by implication or otherwise under any patents or other rights of any third party or Hitachi, Ltd.
6. MEDICAL APPLICATIONS: Hitachi's products are not authorized for use in MEDICAL APPLICATIONS without the written consent of the appropriate officer of Hitachi's sales company. Such use includes, but is not limited to, use in life support systems. Buyers of Hitachi's products are requested to notify the relevant Hitachi sales offices when planning to use the products in MEDICAL APPLICATIONS.

Preface

The H8S/2655 Series comprises high-performance microcomputers with a 32-bit H8S/2600 CPU core and a comprehensive set of peripheral functions.

On-chip peripherals include RAM, a 16-bit timer-pulse unit (TPU), programmable pulse generator (PPG), serial communication interface (SCI), data transfer controller (DTC), and DMA controller (DMAC), enabling this series to be used for compact, high-performance system applications.

This Application Note consists of an introductory section containing operating examples that use the on-chip peripheral functions independently, and an application section in which on-chip peripheral functions are used in combination.

The operation of the programs and circuits shown in this Application Note has been checked, but correct operation should be reconfirmed before any of these examples are actually used.

Contents

Section 1 Using the H8S/2655 Series Application Note

This Application Note is divided into two parts as shown in figure 1-1.

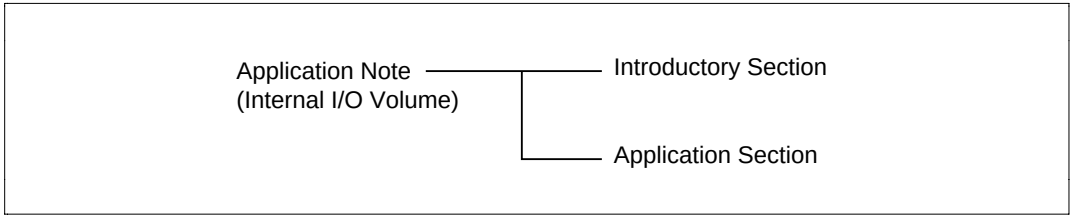


Figure 1-1 Organization of Application Note

Introductory Section

This section describes the operation of the H8S/2655 Series peripheral functions, based on examples of individual tasks.

Application Section

This section describes the operation of combinations of H8S/2655 Series peripheral functions, based on examples of combined tasks.

1.1 Introductory Section

The introductory section uses the layout shown in figure 1-2 to describe the individual use of the peripheral functions.

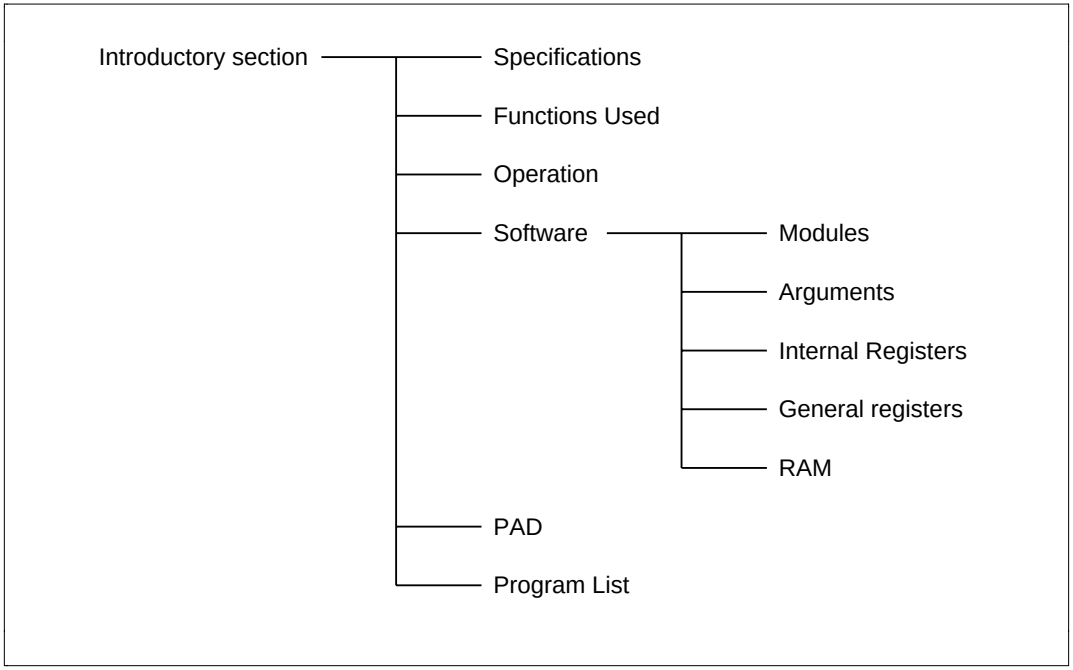


Figure 1-2 Organization of Introductory Section

1. Specifications

   Describes the system specifications for each task.

2. Functions Used

   Describes the features of the peripheral function(s) used in the sample task, and peripheral function assignments.

3. Operation

   Describes the operation of each task, using timing charts.

4. Software

   a. Modules

      Describes the software modules used in the operation of the sample task.

   b. Arguments

      Describes the input arguments needed to execute the modules, and the output arguments after execution.

   c. Internal Registers

      Describes the peripheral function internal registers (timer control registers, serial mode registers, etc.) set by the modules.

   d. RAM

      Describes the labels and functions of the RAM used by the modules.

5. PAD

   Describes the software that executes the sample task, using a PAD.

6. Program List

   Shows a program list of the software that executes the sample task.

1.2 Application Section

The application section uses the layout shown in figure 1-3 to describe the combined use of peripheral functions.

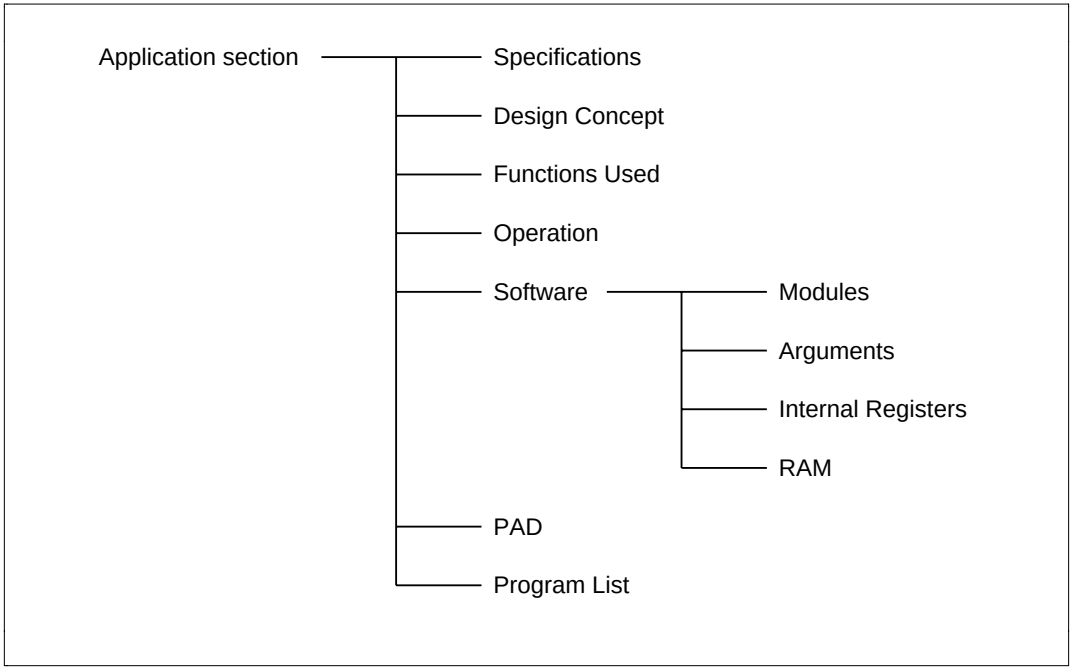


Figure 1-3 Organization of Application Section

1. Specifications

   Describes the system specifications for each task.

2. Design Concept

   Describes the method used to implement the sample task system.

3. Functions Used

   Describes the features of the peripheral functions used in the sample task, and peripheral function assignments.

4. Operation

   Describes the operation of each task, using timing charts.

5. Software

   a. Modules

      Describes the software modules used in the operation of the sample task.

   b. Arguments

      Describes the input arguments needed to execute the modules, and the output arguments after execution.

   c. Internal Registers

      Describes the peripheral function internal registers (timer control registers, serial mode registers, etc.) set by the modules.

   d. RAM

      Describes the labels and functions of the RAM used by the modules.

6. PAD

   Describes the software that executes the sample task, using a PAD.

7. Program List

   Shows a program list of the software that executes the sample task.

Section 2 Common Files Used by Tasks

2.1 Vector Table Definition File

Figure 2-1 shows a vector table definition file that uses the C language. As shown in the figure, a file is created that secures the interrupt handling start addresses. When an interrupt is used, the start label of the interrupt handling routine is written in the vector location corresponding to the interrupt. In the example shown here, the TPU channel 0 compare-match A interrupt is used. The start address (PWHL1) is an external reference (see A). The label for the location of TGI0A is PWHL1 (see B).

```
/*************************************/
/*                                   */
/*      H8S/2600 VECTOR TABLE        */
/*                                   */
/*************************************/
#pragma section VECT

extern void INIT( void );
extern void PWHL1 (void);
const void (*const vect_tbl[])(void) =
{
    INIT,                           /* H'000000   POWER-ON Reset     */
    INIT,                           /* H'000004   Manual Reset       */
    INIT,                           /* H'000008   (System Reserve)   */
    INIT,                           /* H'00000C   (System Reserve)   */
    INIT,                           /* H'000010   (System Reserve)   */
    INIT,                           /* H'000014   Trace              */
    INIT,                           /* H'000018   (System Reserve)   */
    INIT,                           /* H'00001C   NMI                */
    INIT,                           /* H'000020   TRAPA1             */
    INIT,                           /* H'000024   TRAPA2             */
    INIT,                           /* H'000028   TRAPA3             */
    INIT,                           /* H'00002C   TRAPA4             */
    INIT,                           /* H'000030   (System Reserve)   */
    INIT,                           /* H'000034   (System Reserve)   */
    INIT,                           /* H'000038   (System Reserve)   */
    INIT,                           /* H'00003C   (System Reserve)   */
    INIT,                           /* H'000040   IRQ0               */
    INIT,                           /* H'000044   IRQ1               */
    INIT,                           /* H'000048   IRQ2               */
    INIT,                           /* H'00004C   IRQ3               */
    INIT,                           /* H'000050   IRQ4               */
    INIT,                           /* H'000054   IRQ5               */
    INIT,                           /* H'000058   IRQ6               */
    INIT,                           /* H'00005C   IRQ7               */
    INIT,                           /* H'000060   SWDTEND            */
    INIT,                           /* H'000064   WOVI               */
    INIT,                           /* H'000068   CMI                */
    INIT,                           /* H'00006C   (System Reserve)   */
    INIT,                           /* H'000070   A/D                */
    INIT,                           /* H'000074   (System Reserve)   */
    INIT,                           /* H'000078   (System Reserve)   */
    INIT,                           /* H'00007C   (System Reserve)   */
    PWHL1,                          /* H'000080   TGI0A              */
    INIT,                           /* H'000084   TGI0B              */
    INIT,                           /* H'000088   TGI0C              */
    INIT,                           /* H'00008C   TGI0D              */
    INIT,                           /* H'000090   TCI0V              */
    INIT,                           /* H'000094   (System Reserve)   */
    INIT,                           /* H'000098   (System Reserve)   */
    INIT,                           /* H'00009C   (System Reserve)   */
```

A: External reference to label PWHL1

B: Label PWHL1 description

Figure 2.1 Vector Table Definition File (1)

```
    INIT,                               /* H'0000A0   TGI1A             */
    INIT,                               /* H'0000A4   TGI1B             */
    INIT,                               /* H'0000A8   TCI1V             */
    INIT,                               /* H'0000AC   TCI1U             */
    INIT,                               /* H'0000B0   TGI2A             */
    INIT,                               /* H'0000B4   TGI2B             */
    INIT,                               /* H'0000B8   TCI2V             */
    INIT,                               /* H'0000BC   TCI2U             */
    INIT,                               /* H'0000C0   TGI3A             */
    INIT,                               /* H'0000C4   TGI3B             */
    INIT,                               /* H'0000C8   TGI3C             */
    INIT,                               /* H'0000CC   TGI3D             */
    INIT,                               /* H'0000D0   TCI3V             */
    INIT,                               /* H'0000D4   (System Reserve)  */
    INIT,                               /* H'0000D8   (System Reserve)  */
    INIT,                               /* H'0000DC   (System Reserve)  */
    INIT,                               /* H'0000E0   TGI4A             */
    INIT,                               /* H'0000E4   TGI4B             */
    INIT,                               /* H'0000E8   TCI4V             */
    INIT,                               /* H'0000EC   TCI4U             */
    INIT,                               /* H'0000F0   TGI5A             */
    INIT,                               /* H'0000F4   TGI5B             */
    INIT,                               /* H'0000F8   TCI5V             */
    INIT,                               /* H'0000FC   TCI5U             */
    INIT,                               /* H'000100   CMIA0             */
    INIT,                               /* H'000104   CMIB0             */
    INIT,                               /* H'000108   OVI0              */
    INIT,                               /* H'00010C   (System Reserve)  */
    INIT,                               /* H'000110   CMIA1             */
    INIT,                               /* H'000114   CMIB1             */
    INIT,                               /* H'000118   OVI1              */
    INIT,                               /* H'00011C   (System Reserve)  */
    INIT,                               /* H'000120   DEND0A            */
    INIT,                               /* H'000124   DEND0B            */
    INIT,                               /* H'000128   DEND1A            */
    INIT,                               /* H'00012C   DEND1B            */
    INIT,                               /* H'000130   (System Reserve)  */
    INIT,                               /* H'000134   (System Reserve)  */
    INIT,                               /* H'000138   (System Reserve)  */
    INIT,                               /* H'00013C   (System Reserve)  */
    INIT,                               /* H'000140   ERI0              */
    INIT,                               /* H'000144   RXI0              */
    INIT,                               /* H'000148   TXI0              */
    INIT,                               /* H'00014C   TEI0              */
    INIT,                               /* H'000150   ERI1              */
    INIT,                               /* H'000154   RXI1              */
    INIT,                               /* H'000158   TXI1              */
    INIT,                               /* H'00015C   TEI1              */
    INIT,                               /* H'000160   ERI2              */
    INIT,                               /* H'000164   RXI2              */
    INIT,                               /* H'000168   TXI2              */
    INIT,                               /* H'00016C   TEI2              */
};
#pragma section
```

Figure 2.1 Vector Table Definition File (2)

2.2 Register Definition File

The register definition file is shown in the appendix. As shown there, a bit field declaration is made for the bits in each register, allowing bit access.

2.3 Stack Initialization File

Figure 2-2 shows the stack initialization file. As C cannot be used for the stack initialization description, assembly language is embedded in C, and individual tasks are called after stack initialization.

The C compiler (CH38.EXE) cannot create an object file directly when assembler is embedded. It is therefore necessary to create an assembler expansion file “subfile_name.SRC” as a code option, and assemble this file with the assembler (ASM38.EXE) to create the object file.

The code option specification for creating the CH38.EXE assembler expansion file is -c=a. See the compiler manual for details.

```
#include <machine.h>
#pragma noregsave(INIT)

void PWHLMN(void);
void _INITSCT(void);

void INIT(void)
#pragma asm
       mov.l #h'fffc00,er7
#pragma endasm
{
       _INITSCT();
       PWHLMN();
       sleep();
}
```

Figure 2-2 Stack Initialization File

2.4 File Linkage

Figure 2-3 shows the submit file used in file linkage. The vector definition file, register definition file, stack initialization file, and individual tasks are linked in accordance with the submit file shown here.

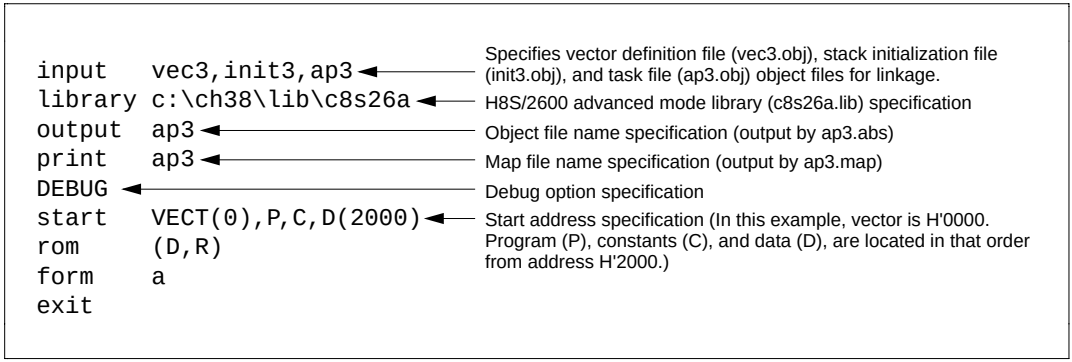


Figure 2.3 Submit File

3.1 Pulse Output 16-Bit Timer-Pulse Unit (TPU)

Specifications

1. A 50% duty pulse is output based on the cycle data set in RAM is output as shown in figure 1.
2. At 20 MHz operation, any output pulse cycle from 100 ns to 3.27 ms can be set.

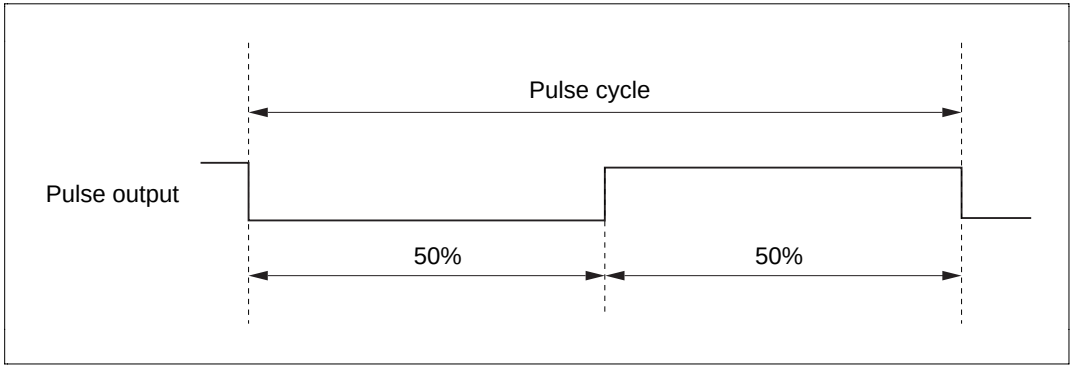


Figure 1 Example of Pulse Output

Functions Used

1. In this sample task, a pulse with a 50% duty cycle is output using TPU0.

   a. Figure 2 shows the TPU0 block diagram for this sample task. The following functions are used by TPU0:

   - A function that automatically outputs pulses by hardware without software intervention (output compare)
   - A function that clears the timer counter in the event of a compare-match (counter clear)
   - A function that inverts the output each time a compare-match occurs (toggle output)

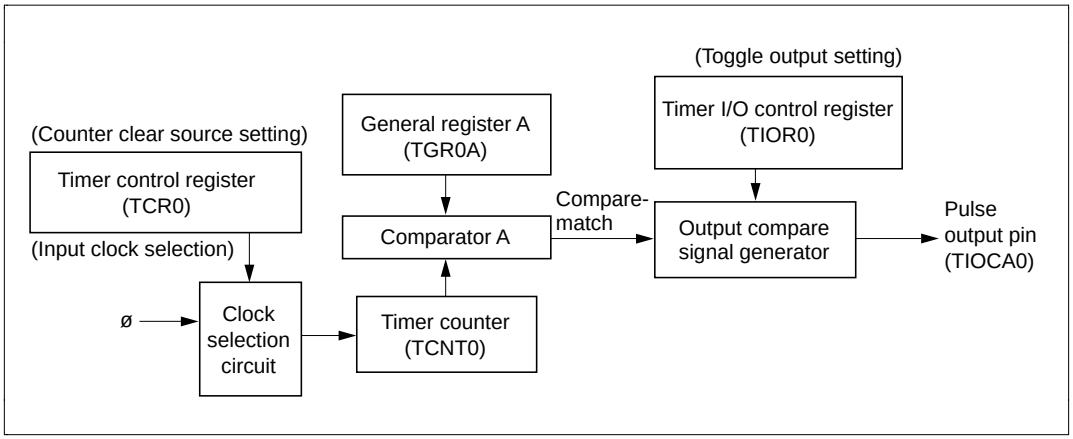


Figure 2 Pulse Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform pulse output.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TCR0	Selects TCNT0 input clock and counter clear source
TIOCA0	Pulse output
TIOR0	Pulse output level setting
TGR0A	1/2 pulse cycle setting

Operation

Figure 3 shows the principles of the operation. Pulses are output by means of H8S/2655 hardware and software processing as shown in the figure.

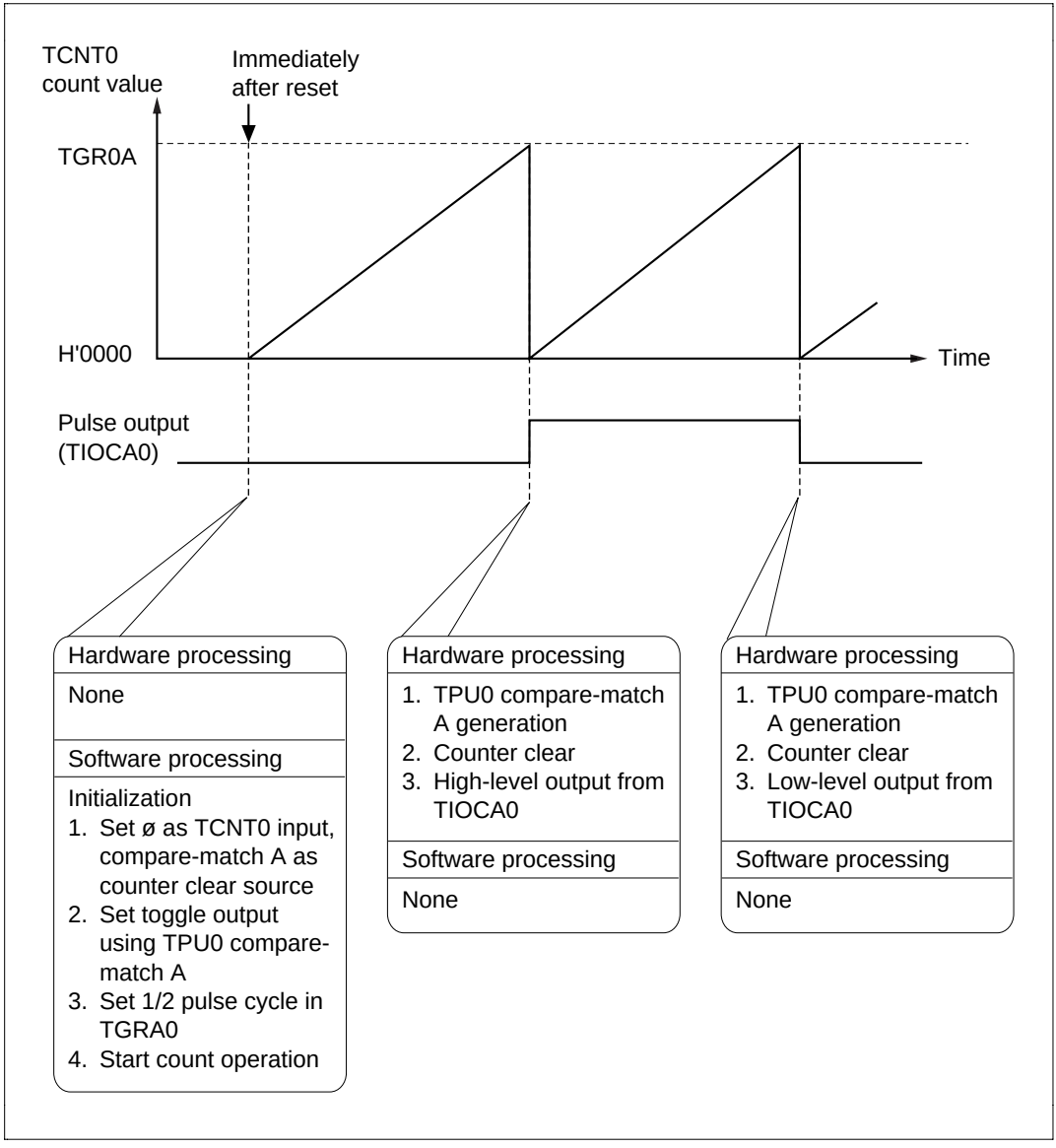


Figure 3 Principles of Pulse Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	poutmn	TPU and RAM initialization, and pulse output

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
pul_cyc	Setting of timer value corresponding to pulse cycle Pulse cycle is found from the following formula: Pulse cycle (ns) = Timer value × ø cycle (50 ns at 20 MHz operation)	Unsigned short	Main routine	Input

3. Internal Registers Used

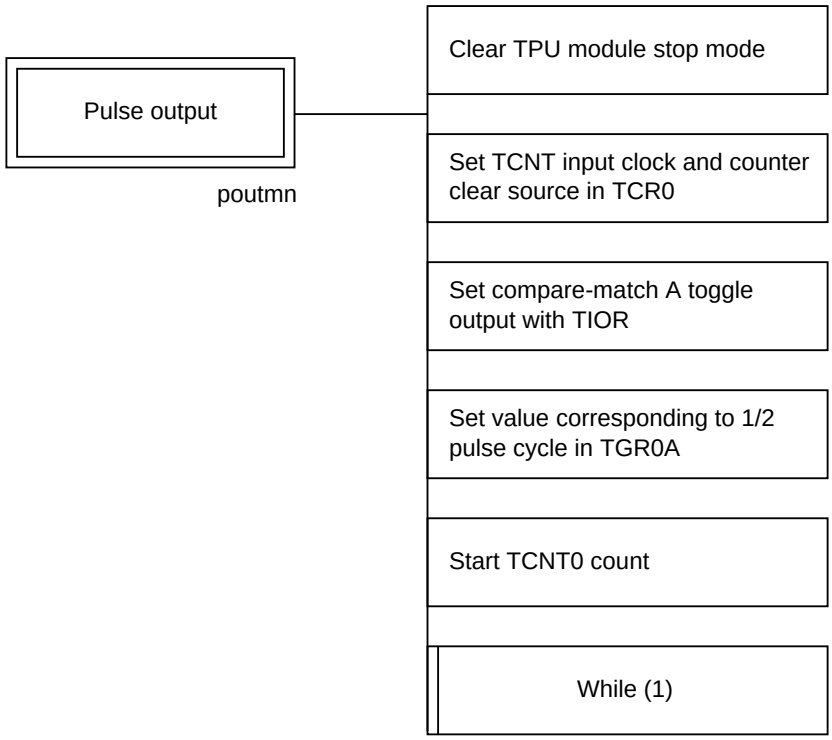
Register Name	Function	Module
TSTR	Sets timer counter operation/disabling	Main routine
TCR0	Sets TCNT input clock and counter clear source	Main routine
TIOR0	Sets output pulse level on compare-match A	Main routine
TGR0A	1/2 output pulse cycle setting	Main routine
MSTPCR	Clears TPU module stop mode	Main routine

4. RAM Used

This application example does not use any RAM apart from the arguments.

PAD

1. Main routine



Program List

```
#include <machine.h>
#include <h8s.h>

/*************************************/
/*      PROTCOL                      */
/*************************************/
void poutmn(void);

/*************************************/
/*      RAM ALLOCATION               */
/*************************************/
#define pul_cyc (*(unsigned short *)0xffec00)

/*************************************/
/*      MAIN PROGRAM : poutmn        */
/*************************************/
void poutmn(void)
{
        MSTPCR = 0x1fff;
        TPU_TCR0 = 0x20;         /* initialize TCR0 */
        TIOR0H = 0x03;           /* initialize TIOR0 */
        TGR0A = pul_cyc/2;       /* set data to TGR0A */
        TSTR = 0x01;             /* TCNT0 start */
        while(1);                /* loop */
}
```

Specifications

The phase difference between two-phase encoder pulses input to external clock pins TCLKA and TCLKB is detected, and the number of incrementations/decrementations within the measurement period is stored in RAM.

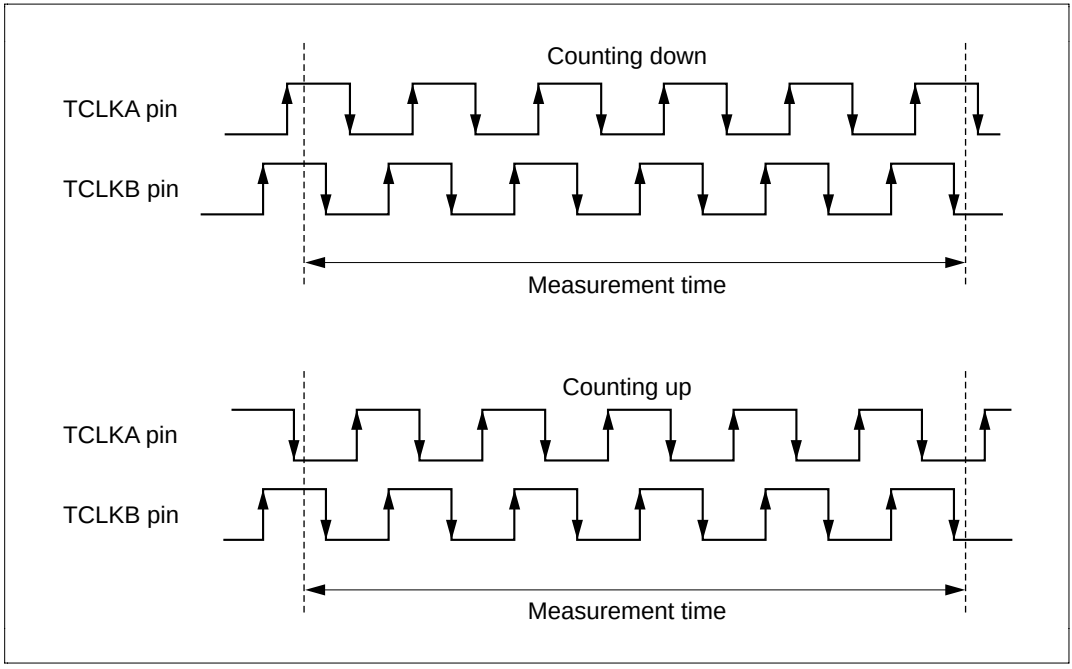


Figure 1 Two-Phase Encoder Count

Functions Used

1. In this sample task, two-phase encoder counting is performed using TPU1. The two-phase encoder count block diagram is shown in figure 2. This task uses the following functions.

   a. A function that detects the phase difference between two-phase encoder pulses input to external clock pins TCLKA and TCLKB is detected, and increments or decrements the TPU1 counter (phase counting mode)

   b. A function that transfers the value of the counter operating on the external clock to a general register on compare-match with the other channel

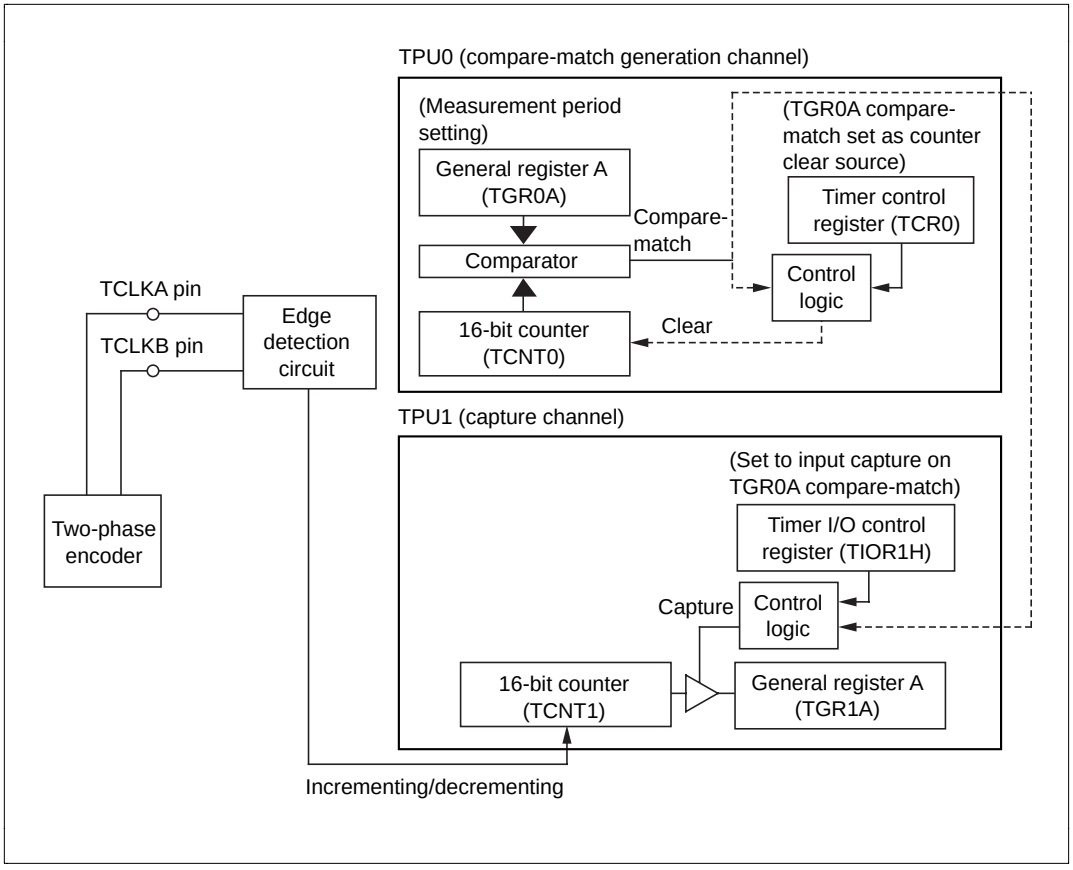


Figure 2 Two-Phase Encoder Count Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform two-phase encoder counting.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TCLKA, B	Two-phase encoder pulse input pins
TCR0	Sets TGR0A compare-match as counter clear source
TGR0A	Measurement period setting
TIOR1	Sets input capture on TGR0A compare-match
TGR1A	Holds input capture A count result

Operation

Figure 3 shows the principles of the operation. Two-phase encoder counting (incrementing) is performed by means of H8S/2655 hardware and software processing as shown in the figure.

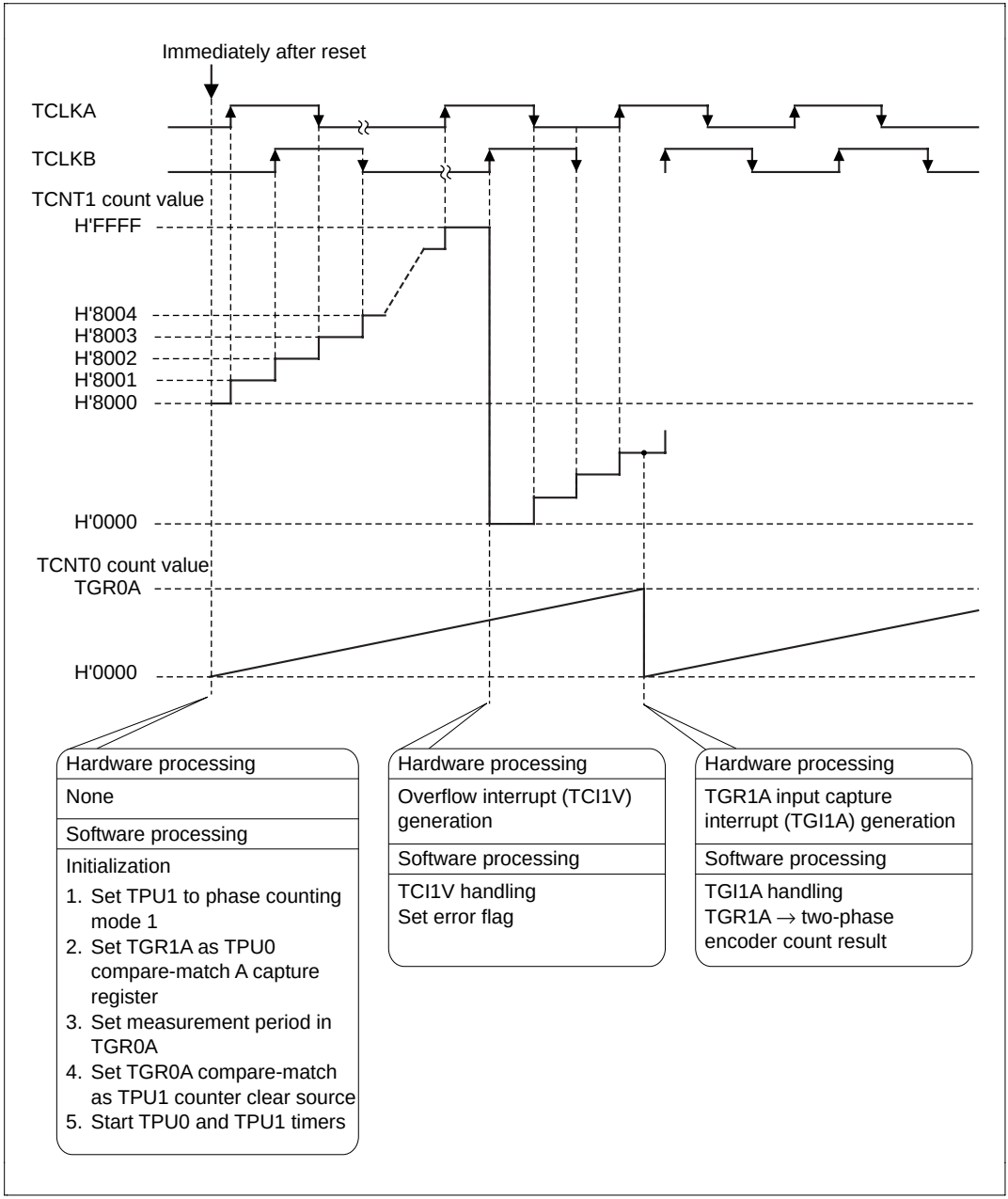


Figure 3 Principles of Two-Phase Encoder Count (Increment) Operation

Figure 4 shows the principles of the operation. Two-phase encoder counting (decrementing) is performed by means of H8S/2655 hardware and software processing as shown in the figure.

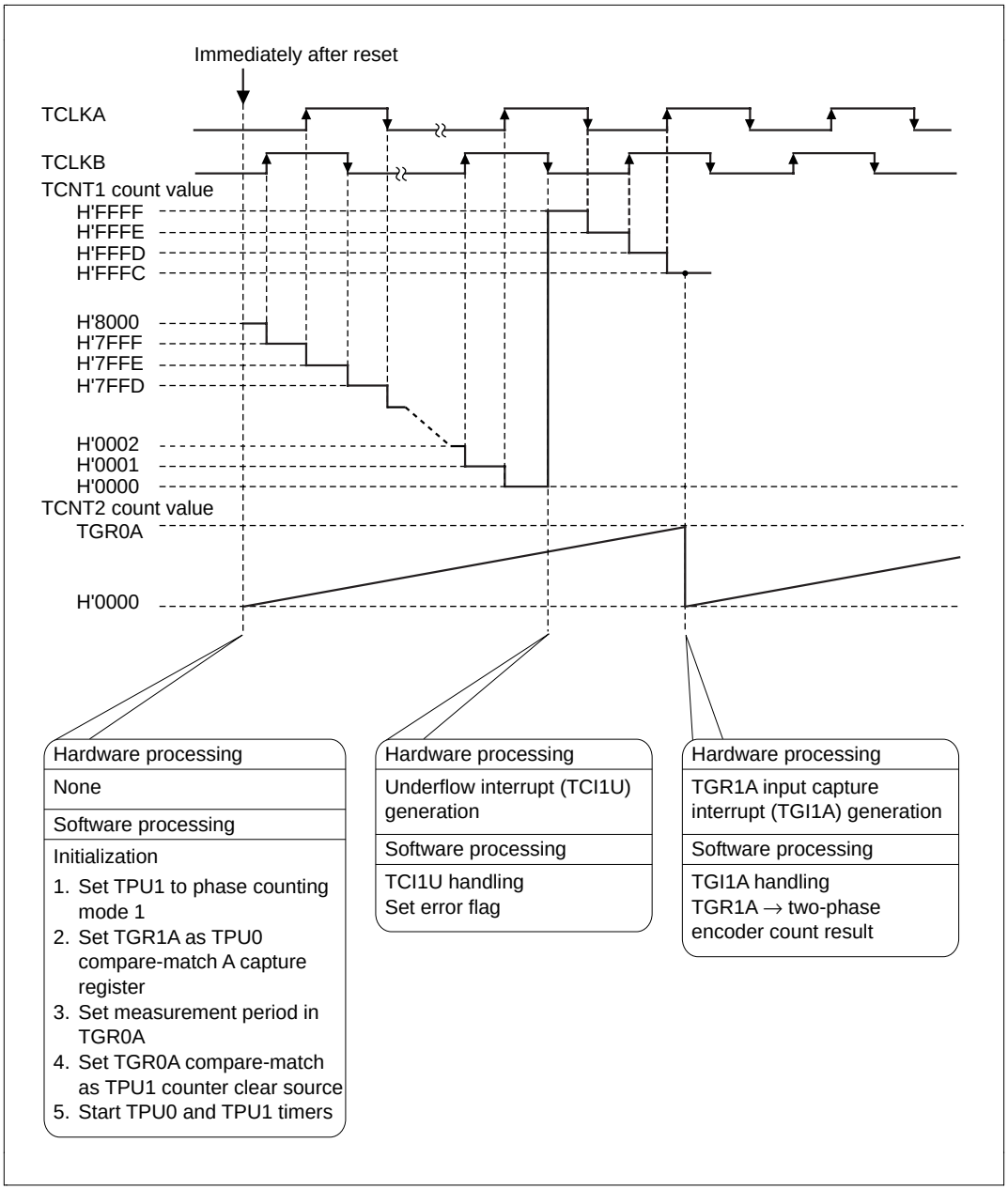


Figure 4 Principles of Two-Phase Encoder Count (Decrement) Operation

Software

1. Modules

Module Name	Label	Function
Main routine	cntmn	Two-phase encoder count initialization
Capture interrupt	ramset	Stores count result in RAM
Overflow detection	error1	Sets overflow generation flag
Underflow detection	error2	Sets underflow generation flag

2. Arguments

Label	Function	Data Length	Module	Input/ Output
count	Setting of count result within measurement period	Unsigned short	Capture interrupt	Output
err_over	Indicates whether overflow has been generated 1: Overflow 0: No overflow	Unsigned char	Overflow detection	Output
err_under	Indicates whether underflow has been generated 1: Underflow 0: No underflow	Unsigned char	Underflow detection	Output
cnttim	Sets measurement period	Unsigned short	Main routine	Input

3. Internal Registers Used

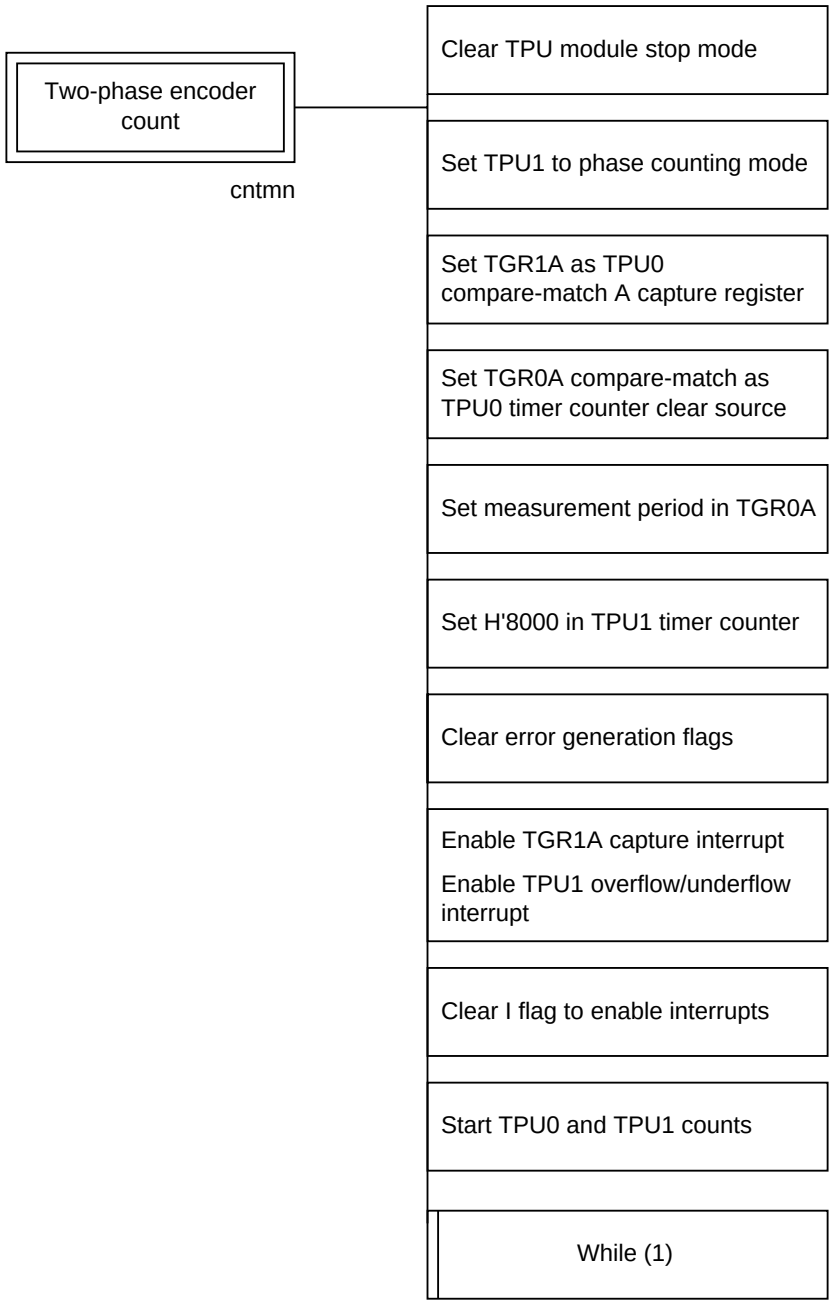
Register Name	Function	Module
TSTR	Controls TPU0 and TPU1 timer counter count start/stop	Main routine
TCR0	Sets TGR0A compare-match as counter clear source	Main routine
TIOR0	Sets TGR0A as output compare register	Main routine
TMDR1	Sets TPU1 to phase counting mode 1	Main routine
TCR1	Sets TGR0A compare-match as counter clear source	Main routine
TIOR1	Sets TGR1A as TGR0A compare-match capture register	Main routine
TCNT1	Initialized to H'8000	Main routine
TIER1	Enables interrupts by bits TGFA, TCFU, TCFV	Main routine
TSR1	Enables input capture and overflow/underflow interrupts	Main routine Capture interrupt
MSTPCR	Clears TPU module stop mode	Main routine

4. RAM Used

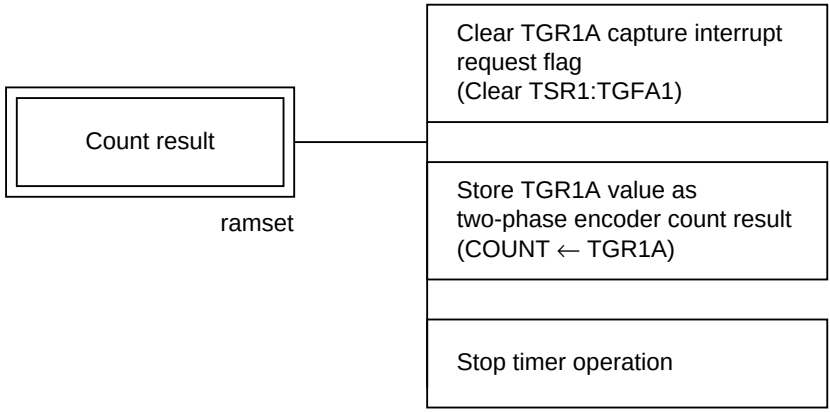
This sample task does not use any RAM apart from the arguments.

PAD

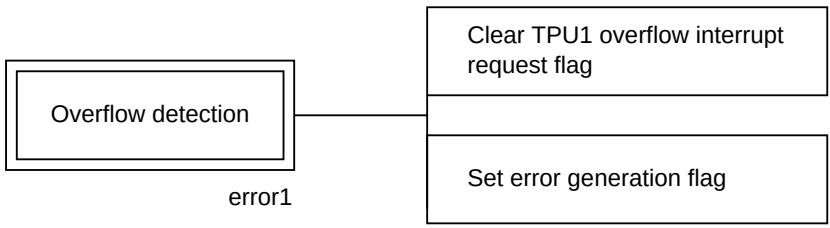
1. Main routine



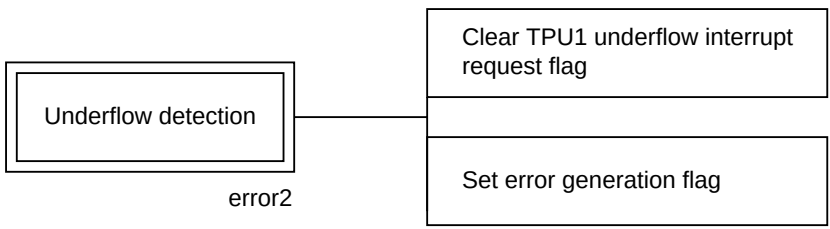
2. Capture interrupt



3. Overflow detection



4. Underflow detection



Program List

```
#include <machine.h>
#include "..\h8sapn\h8s.h"

/*****************************************/
/*            PROTCOL                    */
/*****************************************/
void cntmn(void);
/*****************************************/
/*         RAM ALLOCATION                */
/*****************************************/
#define count (*(unsigned short *)0xffec00)
#define cnttime (*(unsigned short *)0xffec02)
volatile struct ERROR{
        char    over;
        char    under;
};
#define err (*(struct ERROR *)0xffec04)

/*****************************************/
/*         MAIN PROGRAM : cntmnd         */
/*****************************************/
void cntmn(void)
{
        MSTPCR = 0x1fff;        /* Disable module(TPU) stop mode*/
        TMDR1 = 0xc4;           /* Initialize TMDR1 ch1:phase counting mode1 */
        TIOR0H = 0;             /* Initialize TIOR0H */
        TIOR1H = 0x0c;          /* Initialize TIOR1H *
        TPU_TCR0 = 0x20;        /* Initialize TCR0 */
        TPU_TCR1 = 0x20;        /* Initialize TCR1 */
        TGR0A = cnttime;        /* Set counting time */
        TPU_TCNT1 = 0x8000;     /* Counter start H'8000 */
        err.over = 0x00;        /* Over flow flag clear */
        err.under = 0x00;       /* Under flow flag clear */
        TIER1 = 0x71;           /* Enable timer interrupt */
        set_imask_ccr(0);       /* CCR Ibit clear */
        TSTR = 0x03;            /* Start TCNT0,1 */
        while(1);
}
/*****************************************/
/*            NAME : ramset              */
/*****************************************/
#pragma interrupt (ramset)
void ramset(void)
{
        TSR1_BP.TGFA1 = 0;      /* Clear TGFA1 request */
        count = TGR1A;          /* Store count data */
        TSTR = 0x00;            /* Stop counter */
}
/*****************************************/
/*            NAME : error1              */
/*****************************************/
#pragma interrupt (error1)
void error1(void)
{
```

```
        TSR1_BP.TCFV1 = 0;      /* Clear TCFV1 request */
        err.over = 0x01;
}
/*****************************************/
/*            NAME : error2                */
/*****************************************/
#pragma interrupt (error2)
void error2(void)
{
        TSR1_BP.TCFU1 = 0;      /* Clear TCFU1 request */
        err.under = 0x01;
}
```

Specifications

1. Pulse high and low widths are measured as shown in figure 1, and the results are stored in RAM.

2. At 20 MHz operation, pulse high and low widths of 1.9 µs to 3.27 ms can be measured, in 50 ns units.

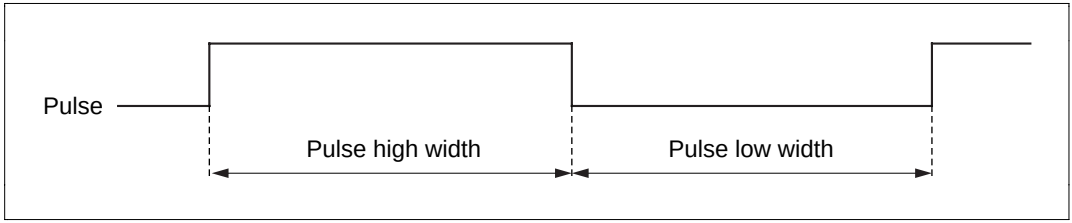


Figure 1 Pulse Width Measurement Timing

Functions Used

1. In this sample task, pulse high and low widths are measured using TPU0.

   a. Figure 2 shows the TPU0 block diagram. This task uses the following functions:

   - A function that performs detection of the rising and falling edges of a pulse, and sets the timer value at those points in an internal register (input capture)
   - A function that clears the timer counter when input capture occurs
   - A function that initiates interrupt handling on detection of the rising and falling edges of a pulse

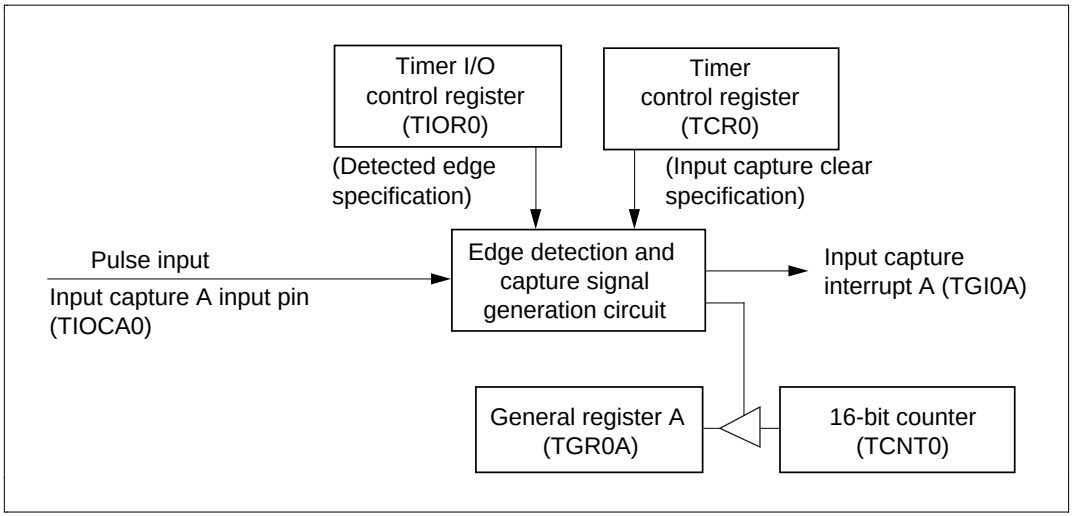


Figure 2 Pulse High and Low Width Measurement Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to measure pulse high and low widths.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TCR0	Selects counter clear source
TIOR0	Selects input capture signal input edge
TIOCA0	Inputs pulse to be measured
TGR0A	Detects counter value at rise and fall of pulse
TGI0A	Initiates pulse high and low width measurement on rise and fall of pulse

Operation

Figure 3 shows the principles of the operation. Pulse high and low widths are measured by means of H8S/2655 hardware and software processing as shown in the figure.

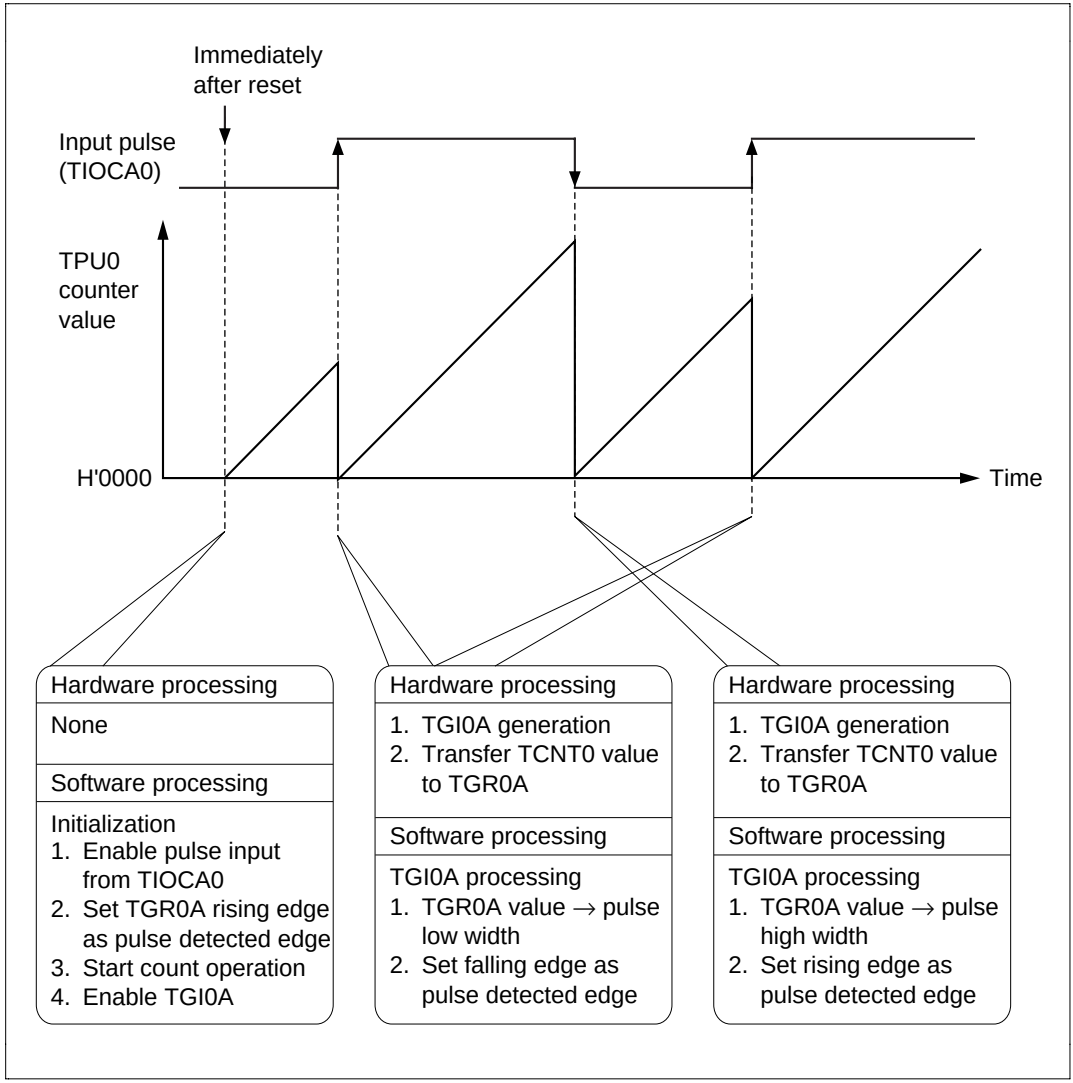


Figure 3 Principles of Pulse Width Measurement Operation

Software

1. Modules

Module Name	Label	Function
Main routine	PWHLMN	TPU and RAM initialization.
Pulse high and low width measurement	PWHL1	Initiated by TGI0A; measures pulse high and low widths according to TGR0A value and sets result in RAM

2. Arguments

Label	Function	Data Length	Module	Input/ Output
pwh_hdata	Setting of timer value corresponding to pulse high width Pulse high width is found from the following formula: Pulse high width (ns) = Timer value × ø cycle (50 ns at 20 MHz operation)	Unsigned short	Pulse high and low width measurement	Output
pwh_ldata	Setting of timer value corresponding to pulse low width Pulse low width is found from the following formula: Pulse low width (ns) = Timer value × ø cycle (50 ns at 20 MHz operation)	Unsigned short		

3. Internal Registers Used

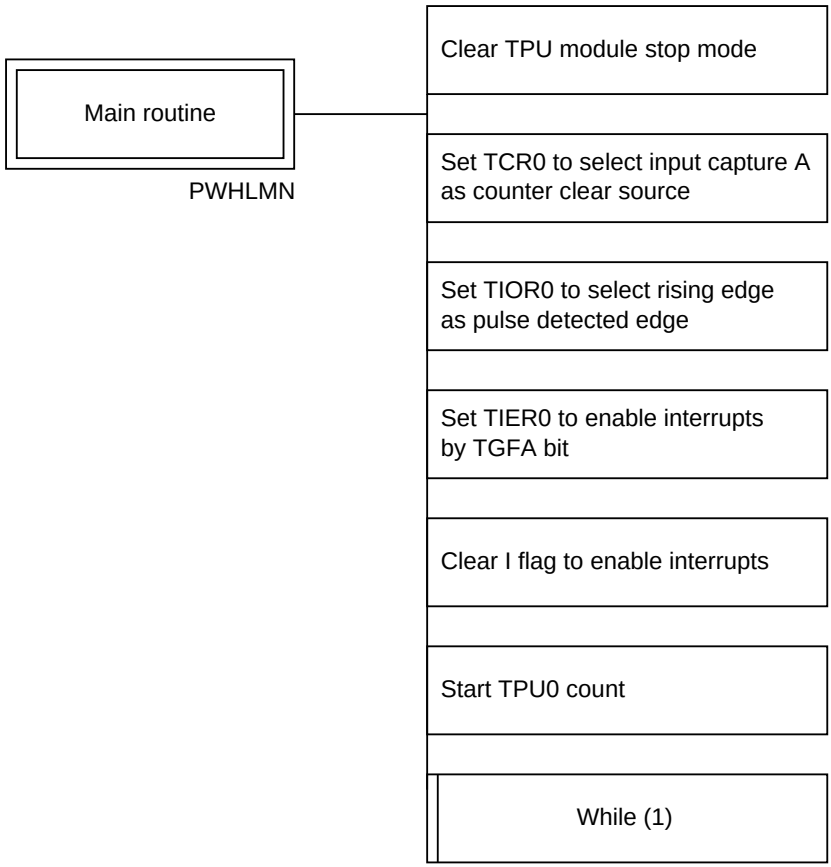
Register Name	Function	Module
TSTR	Specifies timer counter operation/disabling	Main routine
TCR0	Selects TCNT counter clock, specifies input capture A as counter clear source	Main routine, pulse high and low width measurement
TIOR0	Set so that transfer is performed from TCNT to TGR0A on detection of pulse rise or fall	Main routine
TIER0	Enables interrupts by TGI0A	Main routine
TGR0A	TCNT value at rise or fall of pulse is set in this register and used to calculate pulse cycle	Pulse high and low width measurement
TSR0	Indicates input capture A generation	Pulse high and low width measurement
MSTPCR	Clears TPU module stop mode	Main routine

4. RAM Used

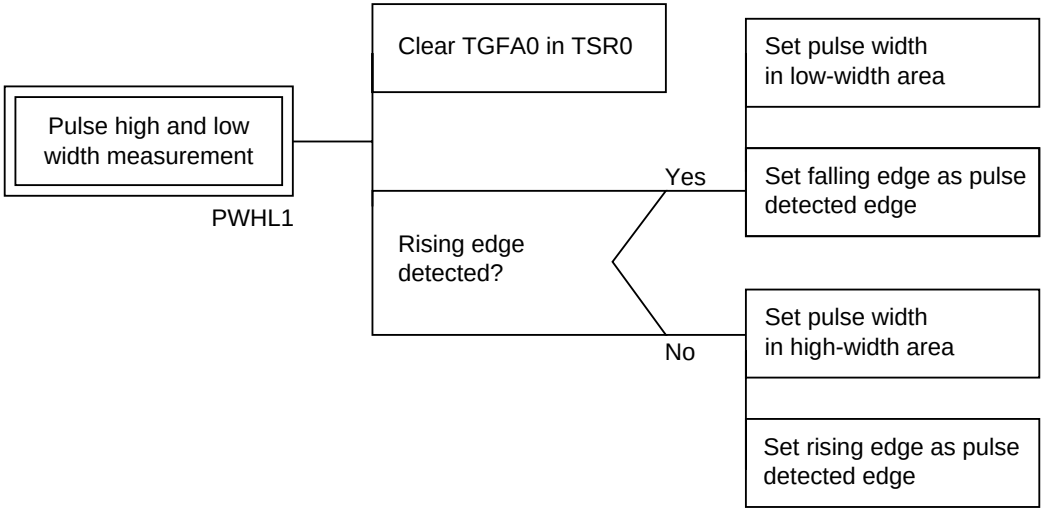
This sample task does not use any RAM apart from the arguments.

PAD

1. Main routine



2. Pulse high and low width measurement



Program List

```
#include <machine.h>
#include <H8S.H>
/*********************************/
/*           PROTOCOL            */
/*********************************/
void PWHLMN(void);
#pragma interrupt (PWHL1)
/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/

# define pwh_ldata   (*(unsigned short * )0xffec00)   /* Pulse high width time
*/
# define pwh_hdata   (*(unsigned short * )0xffec02)   /* Pulse low width time */

/*********************************/
/*     MAIN PROGRAM: PWHLMN      */
/*********************************/
void PWHLMN(void)
{
        MSTPCR = 0xdfff;                /* Disable module(TPU) stop mode*/
        TPU_TCR0 = 0x20;                /* Initialize TCR0 */
        TIOR0H = 0x08;                  /* Initialize TIOR0H */
        TIER0 = 0x41;                   /* Initialize TIER0 */
        set_imask_ccr(0);               /* Enable interrupt */
        TSTR  = 0x01;                   /* Start TCNT0 */
        while(1);                       /* Loop */
}

/*********************************/
/*   INTERRUPT PROGRAM: PWHL1    */
/*********************************/
void PWHL1(void)
{
        TSR0_BP.TGFA0 = 0;              /* Clear TGFA0 request */
        if(TIOR0H == 0x08)              /* Edge is "high"? */
                {                       /* Yes */
                pwh_ldata = TGR0A;
                TIOR0H = 0x09;    /* Set TGR0A captures falling edge of input */
                }
        else{                           /* No */
                pwh_hdata = TGR0A;
                TIOR0H = 0x08;          /* Set TGR0A captures rising edge of
input */
                }
}
```

Specifications

1. 32-bit counter operations are performed to vary the pulse high width and output a variable-duty long-cycle pulse.
2. A duty cycle of 0% to 100% can be set, with a resolution of 1/65,535.
3. At 20 MHz operation, a pulse cycle of 6.55 ms to 214.7 s can be set in 3.27 ms units.

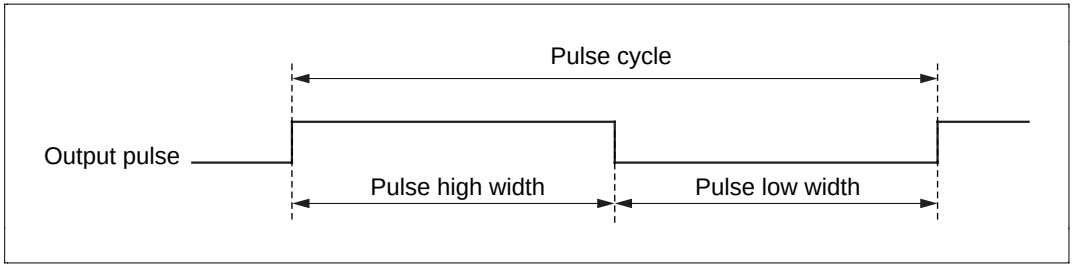


Figure 1 Example of Long-Cycle Pulse Output

Functions Used

1. In this sample task, the TPU1 and TPU2 16-bit counters are connected to operate as a 32-bit counter, to output a long-cycle pulse from TPU1. Figure 2 shows the TPU block diagram for this sample task. This sample task uses the following functions:

    - A function that connects two 16-bit counter channels for operation as a 32-bit counter (cascaded operation)
    - A function that automatically outputs pulses by hardware without software intervention (output compare)
    - PWM output generation using TGR1A and TGR1B as a pair (PWM mode 1)

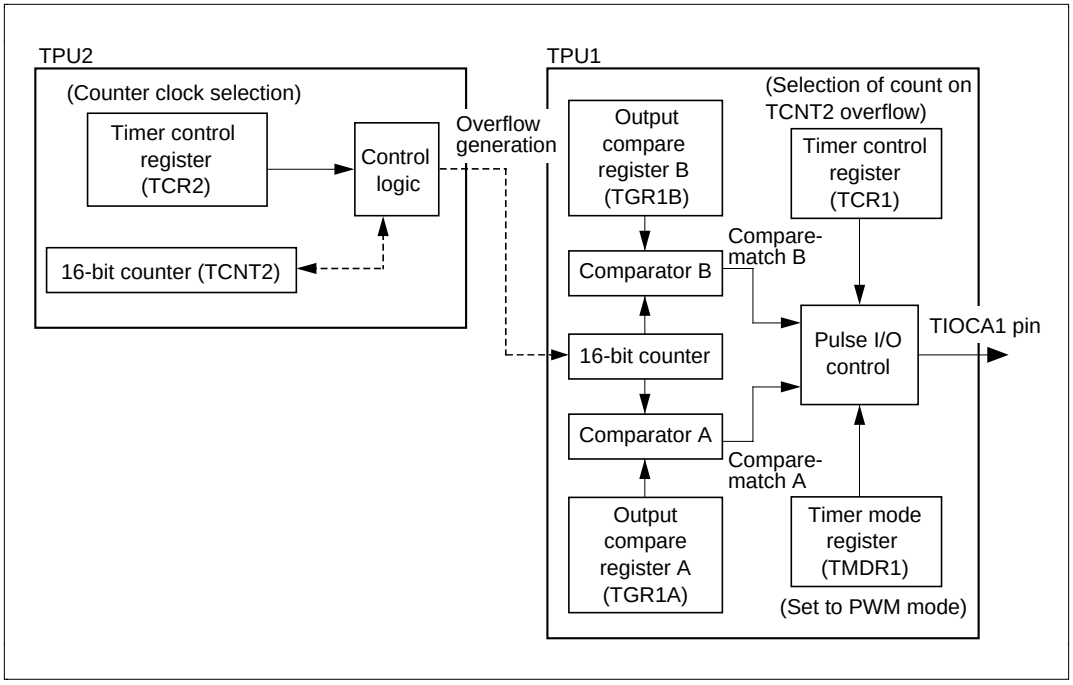


Figure 2 Long-Cycle Pulse Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to create a timer counter that performs 32-bit operation.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
TPU1	TMDR1	Selects PWM mode 1
	TCR1	Selects TCNT1 input clock and counter clear source
	TCNT1	Upper 16 bits of 32-bit counter
	TGR1A	Pulse low width setting
	TGR1B	Pulse high width setting
	TIOCA1	Pulse output
TPU2	TCR2	Selects TCNT2 input clock
	TCNT2	Lower 16 bits of 32-bit counter

Operation

Figure 3 shows the principles of the operation. Long-cycle pulses are output by means of H8S/2655 hardware and software processing as shown in the figure.

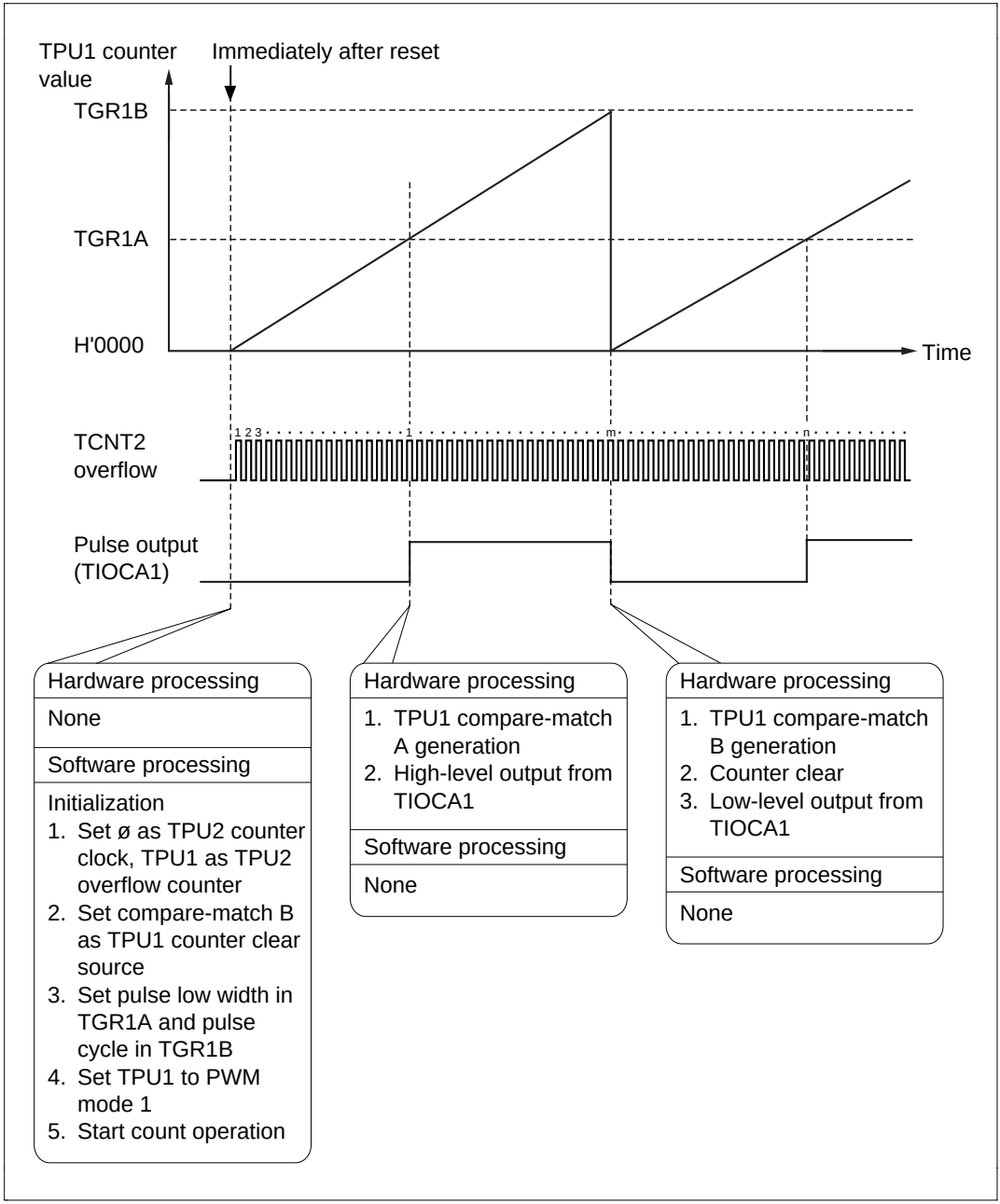


Figure 3 Principles of Long-Cycle Pulse Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	LPULMN	Performs 32-bit counter operation using TPU1 and TPU2 counters, and outputs long-cycle pulses

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
lpul_wid	Setting of timer value corresponding to output pulse low width Pulse low width is found from the following formula: Low width (ms) = Timer value × external clock (3.27 ms at 20 MHz operation)	Unsigned short	Main routine	Input
lpul_cyc	Setting of timer value corresponding to output pulse cycle Cycle is found from the following formula: Cycle (ms) = Timer value × external clock* (3.27 ms at 20 MHz operation)	Unsigned short	Main routine	Input

Note: * External clock: TCNT2 overflow output

3. Internal Registers Used

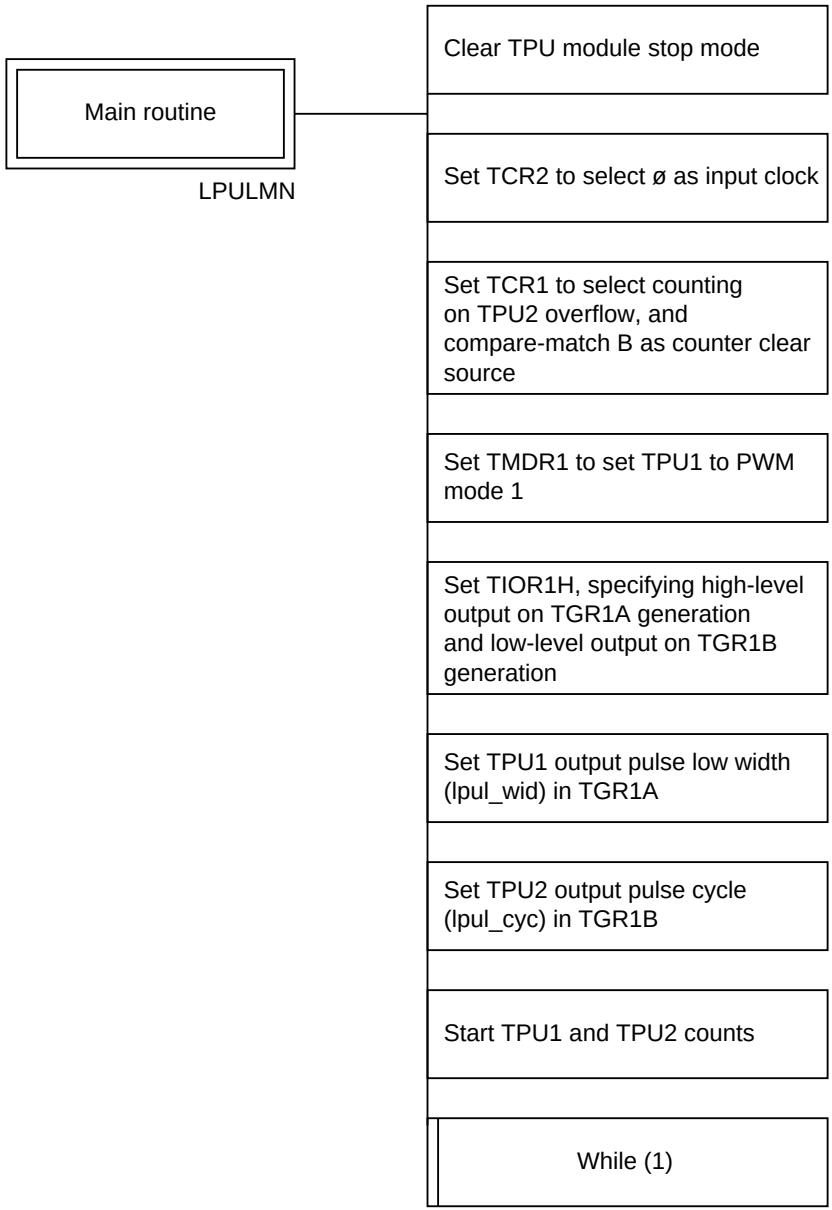
Register Name		Function	Module
TPU1	TSTR	Sets timer counter operation/disabling	Main routine
	TMDR1	Selects PWM mode 1	Main routine
	TCR1	Sets TCNT1 input clock and counter clear source	Main routine
	TCNT1	Counts TCNT2 overflows to perform 32-bit counter operation	Main routine
	TGR1A	Pulse low width setting	Main routine
	TGR1B	Pulse high width setting	Main routine
	TIOCA1	Pulse output	Main routine
TPU2	TCR2	Selects TCNT2 input clock	Main routine
	TCNT2	16-bit free-running counter	Main routine

4. RAM Used

This sample task does not use any RAM apart from the arguments.

PAD

1. Main routine



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void LPULMN(void);

/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/

# define lpul_wid  (*(unsigned short * )0xffec00) /* Pulse width */
# define lpul_cyc  (*(unsigned short * )0xffec02) /* Pulse cycle */

/*********************************/
/*     MAIN PROGRAM: LPULMN      */
/*********************************/
void LPULMN(void)
{
        MSTPCR = 0xdfff;         /* Disable module(TPU) stop mode*/
        TCR2 = 0;                /* Initialize TCR2 */
        TPU_TCR1 = 0x47;         /* Initialize TCR1 */
        TMDR1 = 0xC2;            /* Initialize TMDR1 ch1:PWM mode */
        TIOR1H = 0x52;           /* Initialize TIOR1H */
        TGR1A = lpul_wid;        /* Set pulse low period time */
        TGR1B = lpul_cyc;        /* Set pulse cycle time */
        TSTR = 0x06;             /* Start TCNT1,2 */
        while(1);                /* Loop */
}
```

Specifications

1. The pulse high width is varied to produce variable-duty 15-phase PWM waveform output, as shown in figure 1.

2. At 20 MHz operation, any output PWM cycle from 100 ns to 3.27 ms can be set.

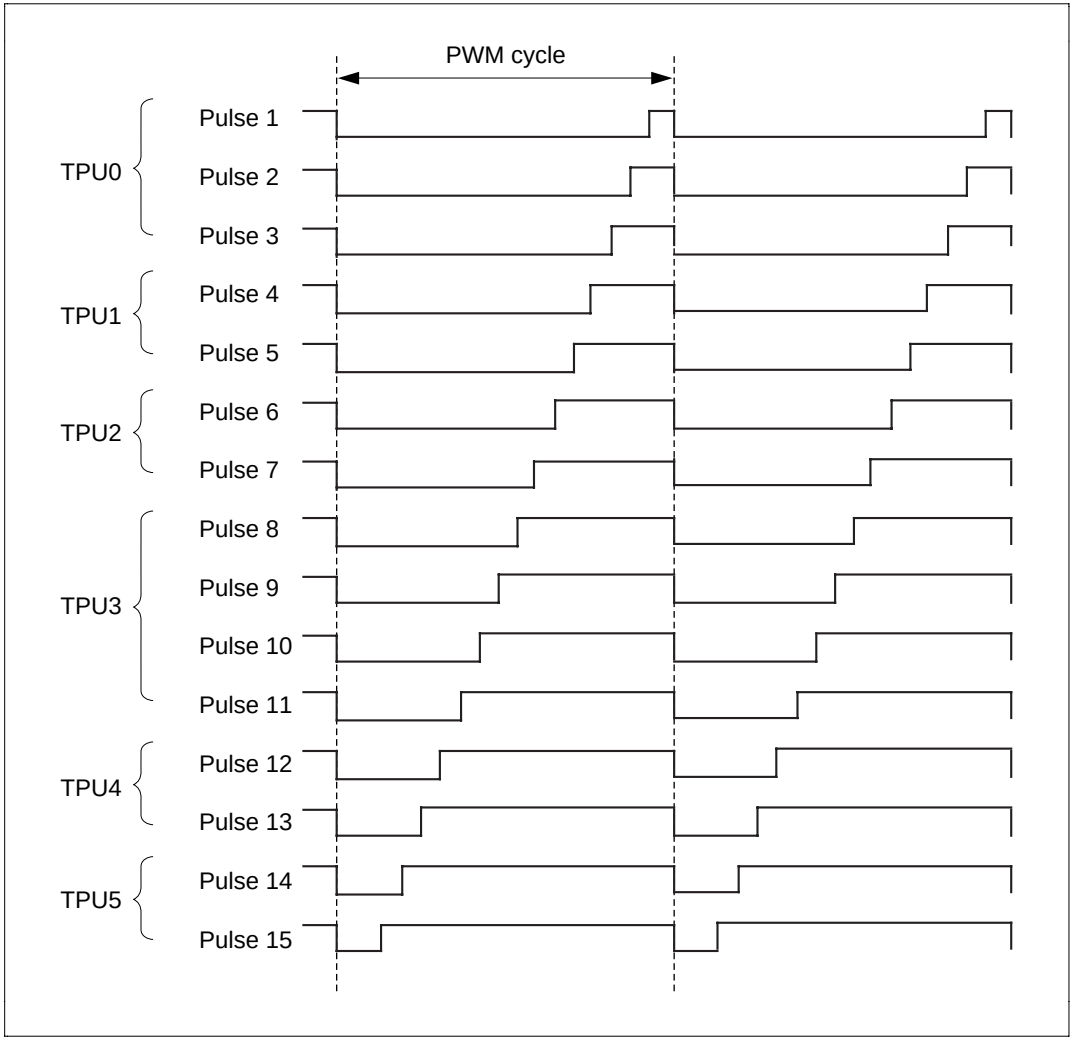


Figure 1 Example of PWM Waveform Output

Functions Used

1. In this sample task, TPU0 to TPU5 are operated synchronously to produce 15-phase PWM waveform output.

   a. Figure 2 shows the TPU block diagram for this sample task.

      This sample task uses the following function:

      - The ability to generate a maximum of 15-phase PWM output through a combination of synchronous operations (PWM mode 2)

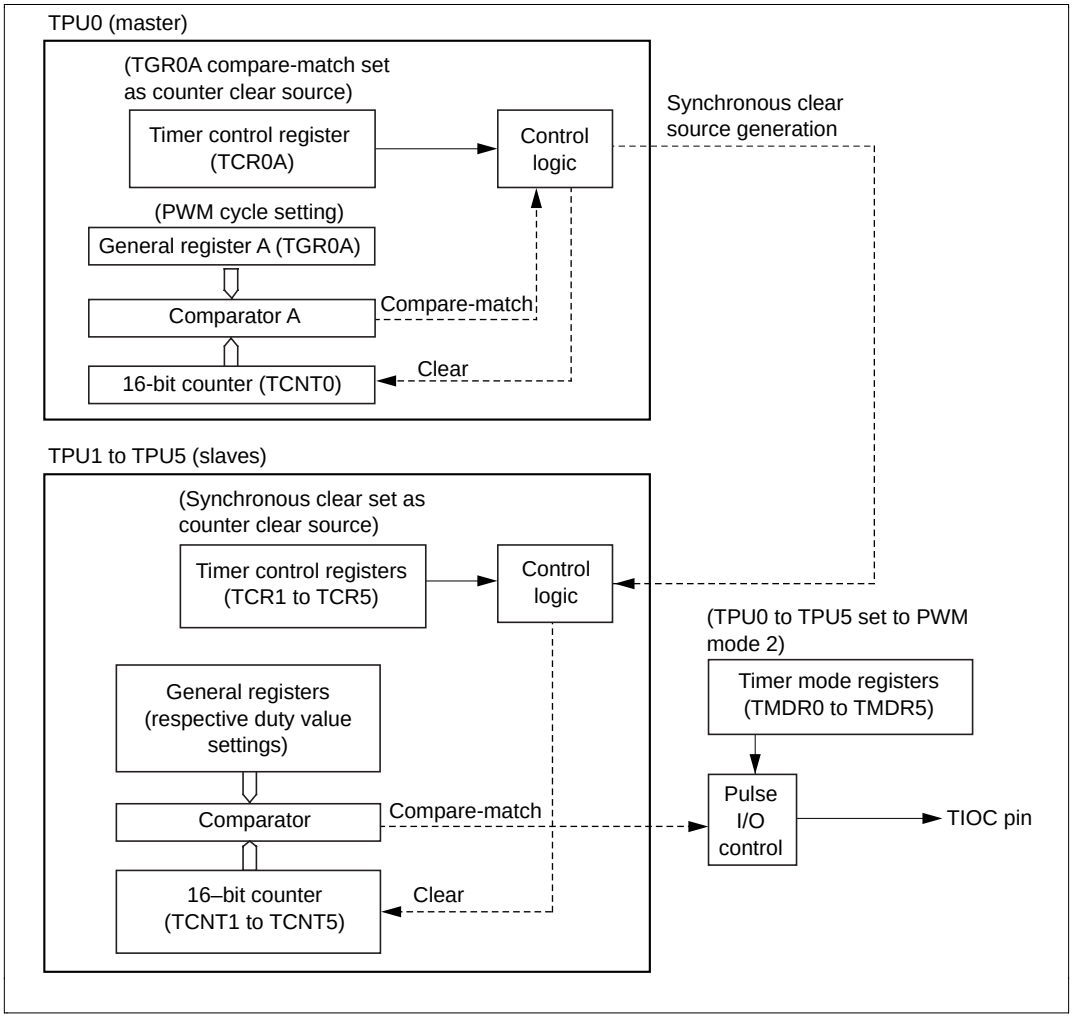


Figure 2 PWM 15-Phase Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform PWM pulse output.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TIOCA1 to TIOCA5 TIOCB0 to TIOCB5, TIOCC0, TIOCC3, TIOCD1, TIOCD3	PWM pulse output pins
TCR0 to TCR5	Select TPU0 to TPU5 timer counter clear sources
TMDR0 to TMDR5	Specify operation of TPU0 to TPU5 in PWM mode 2
TGR0A	PWM cycle setting
TGR0B to TGR5B	Duty value settings

Operation

Figure 3 shows the principles of the PWM 15-phase output operation. Pulses are output from the TPU0 to TPU5 PWM output pins by means of H8S/2655 hardware and software processing as shown in the figure.

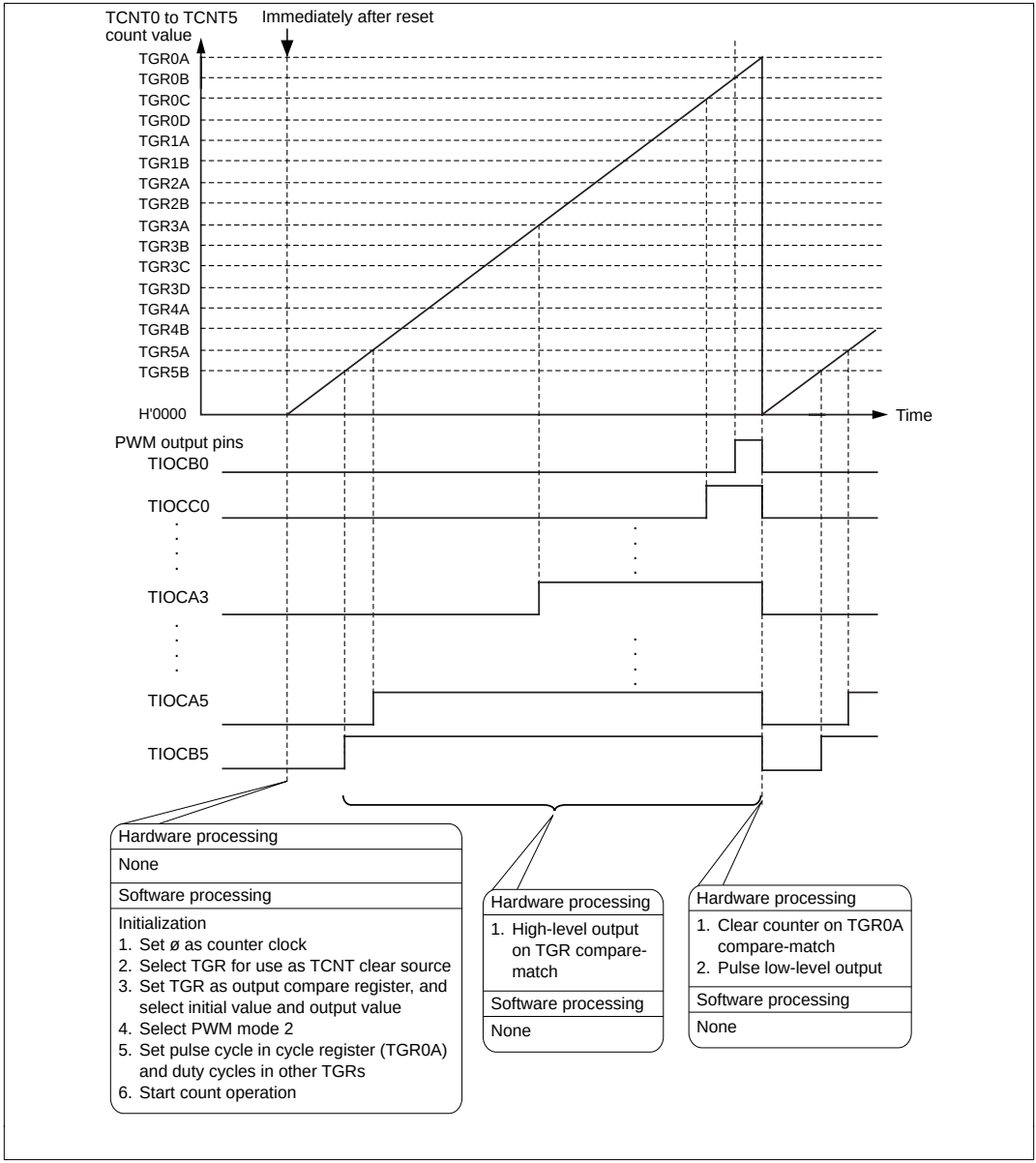


Figure 3 Principles of PWM 15-Phase Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	pwm15mn	TPU0 to TPU5 synchronous clear and PWM output setting

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
pwm[0] to pwm[14]	Setting of timer counter value corresponding to pulse high width Pulse high width is found from the following formula: Pulse high width (ns) = Timer counter value × ø cycle (50 ns at 20 MHz operation) × individual channel input clock division ratio	Unsigned short	Main routine	Input
pwm_cyc	Setting of timer counter value corresponding to PWM cycle PWM cycle is found from the following formula: PWM cycle (ns) = Timer counter value × ø cycle (50 ns at 20 MHz operation) × individual channel input clock division ratio	Unsigned short	Main routine	Input

3. Internal Registers Used

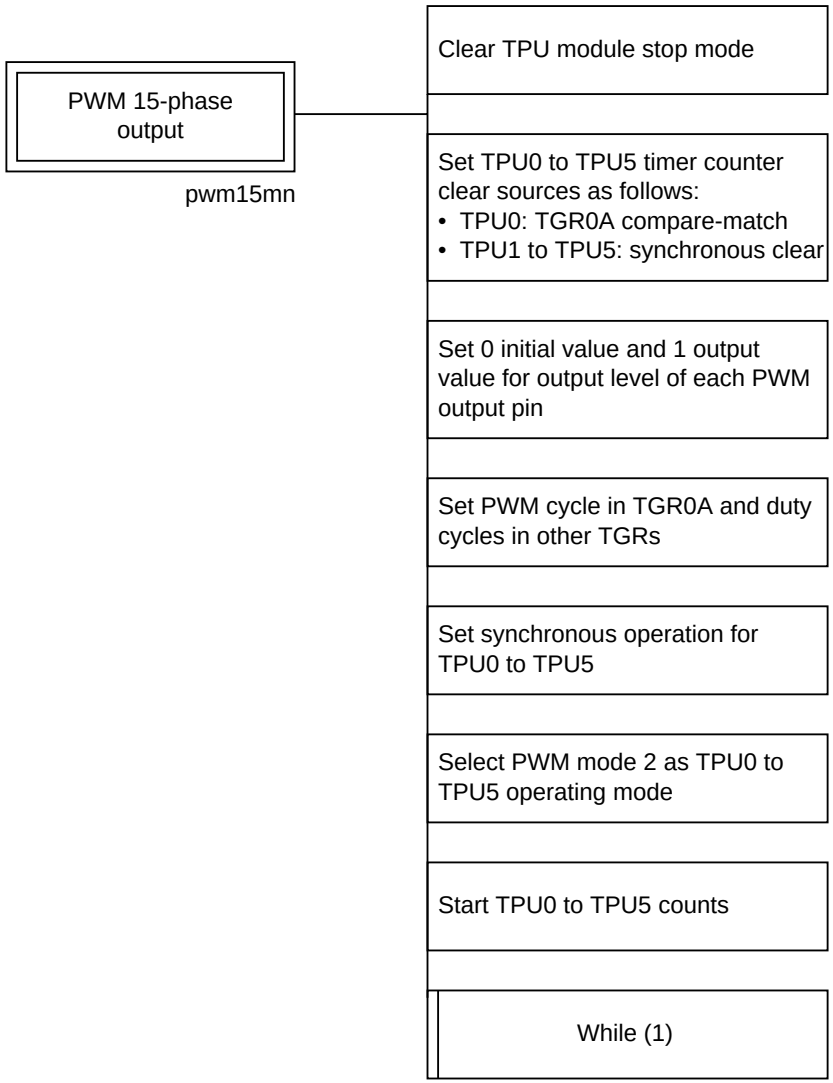
Register Name	Function	Module
TSTR	Performs count starting and stopping for TPU0 to TPU5 timer counters	Main routine
TSYR	Selects synchronous operation of TPU0 to TPU5 TCNT counters	Main routine
TCR0	Sets TGR0A compare-match as timer counter clear source	Main routine
TCR1 to TCR5	Set synchronous clear as timer counter clear source	Main routine
TIOR0 to TIOR5	Set output level of each PWM output pin	Main routine
TMDR0 to TMDR5	Select PWM mode 2	Main routine
TGR0A	Sets PWM cycle	Main routine
TGR0B to TGR5B	Set timer counter value at which high level is output from PWM output pin	Main routine
MSTPCR	Clears TPU module stop mode	Main routine

4. RAM Used

This sample task does not use any RAM apart from the arguments.

PAD

1. Main routine



Program List

```
#include <machine.h>
#include <h8s.h>

/*****************************************/
/*           PROTOCOL                    */
/*****************************************/
void pwm15mn(void);

/*****************************************/
/*        RAM ALLOCATION                 */
/*****************************************/
# define pwm      ((unsigned short * ) 0xffec00)     /* Pulse high width */
# define pwm_cyc  (*(unsigned short * )0xffec1e)     /* Pulse cycle */

/*****************************************/
/*     MAIN PROGRAM : pwm15mn            */
/*****************************************/
void pwm15mn(void)
{
        MSTPCR = 0x1fff;         /* Disable module stop mode*/

        TPU_TCR0 = 0x20;         /* Initialize TCR0 */
        TPU_TCR1 = 0x60;         /* Initialize TCR1 */
        TCR2 = 0x60;             /* Initialize TCR2 */
        TCR3 = 0x60;             /* Initialize TCR3 */
        TCR4 = 0x60;             /* Initialize TCR4 */
        TCR5 = 0x60;             /* Initialize TCR5 */

        TIOR0H = 0x20;           /* Initialize TIOR0H */
        TIOR0L = 0x22;           /* Initialize TIOR0L */
        TIOR1H = 0x22;           /* Initialize TIOR1H */
        TIOR2H = 0x22;           /* Initialize TIOR2H */
        TIOR3H = 0x22;           /* Initialize TIOR3H */
        TIOR3L = 0x22;           /* Initialize TIOR3L */
        TIOR4H = 0x22;           /* Initialize TIOR4H */
        TIOR5H = 0x22;           /* Initialize TIOR5H */

        TGR0A = pwm_cyc;         /* Set PWM cycle */
        TGR0B = pwm[0];          /* Set duty */
        TGR0C = pwm[1];
        TGR0D = pwm[2];
        TGR1A = pwm[3];
        TGR1B = pwm[4];
        TGR2A = pwm[5];
        TGR2B = pwm[6];
        TGR3A = pwm[7];
        TGR3B = pwm[8];
        TGR3C = pwm[9];
        TGR3D = pwm[10];
        TGR4A = pwm[11];
        TGR4B = pwm[12];
        TGR5A = pwm[13];
        TGR5B = pwm[14];
```

```
    TSYR = 0x3f;              /* Set synchronization mode ch0-5 */

    TMDR0 = 0xc3;             /* Set PWM mode2 */
    TMDR1 = 0xc3;             /* Set PWM mode2 */
    TMDR2 = 0xc3;             /* Set PWM mode2 */
    TMDR3 = 0xc3;             /* Set PWM mode2 */
    TMDR4 = 0xc3;             /* Set PWM mode2 */
    TMDR5 = 0xc3;             /* Set PWM mode2 */

    TSTR = 0x3f;              /* Start TCNT0-5 */
    while(1);                 /* Loop */
}
```

Specifications

1. 7-phase pulse output is performed in synchronization with the falling edge of an external signal, as shown in figure 1.

2. The delay time from the external signal falling edge and the pulse width can be varied within the following ranges:

   200 ns ≤ delay time < external signal cycle—pulse width
   50 ns ≤ pulse width < external signal cycle—delay time

3. At 20 MHz operation, any external signal pulse width from 250 ns to 3.27 ms can be set.

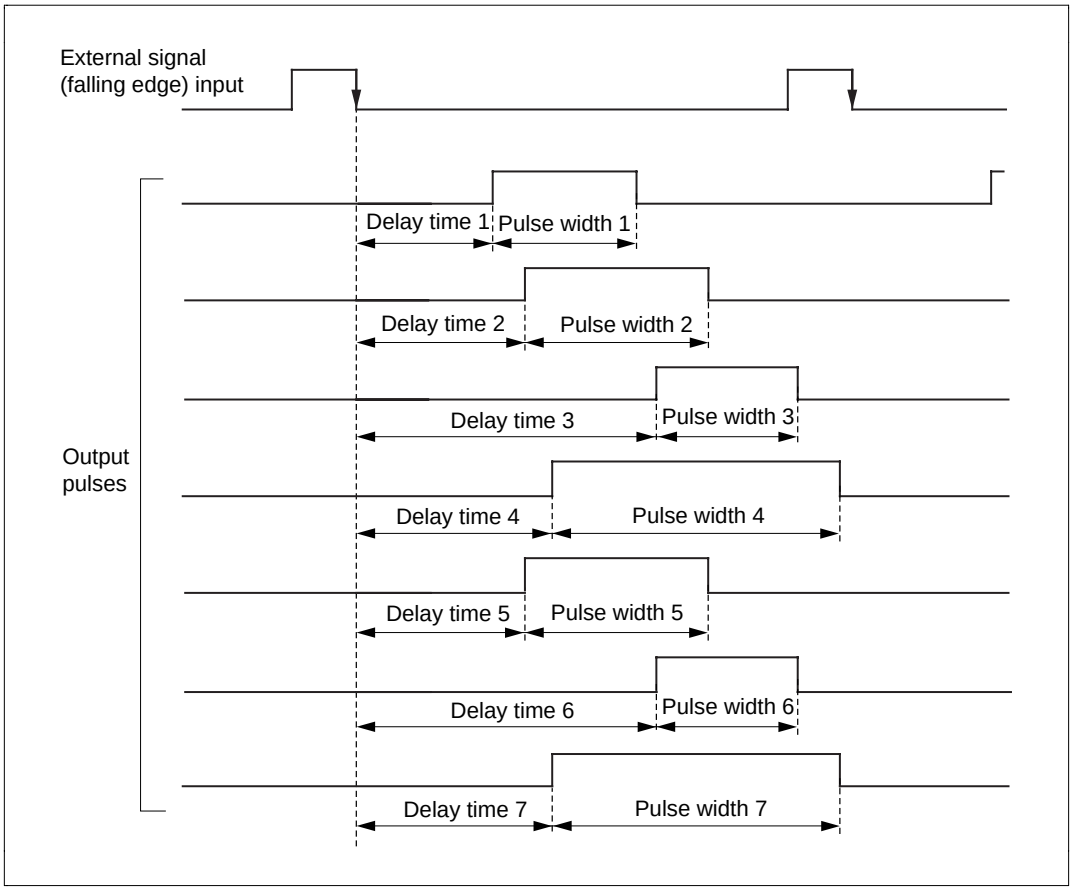


Figure 1 Example of Synchronous Pulse Output

Functions Used

1. In this sample task, simultaneous resetting of multiple timer counters is performed using an external signal, to generate 7-phase pulse output.

   a. Figure 2 shows the TPU block diagram for this sample task. The following TPU functions are used to generate 7-phase pulse output synchronized with an external signal:

   - A function that clears the timer counter when a pulse falling edge is detected
   - A function that simultaneously clears multiple timer counters (synchronous operation)
   - Generation of PWM output using TGRA and TGRB as a pair, and TGRC and TGRD as a pair (PWM mode 1)

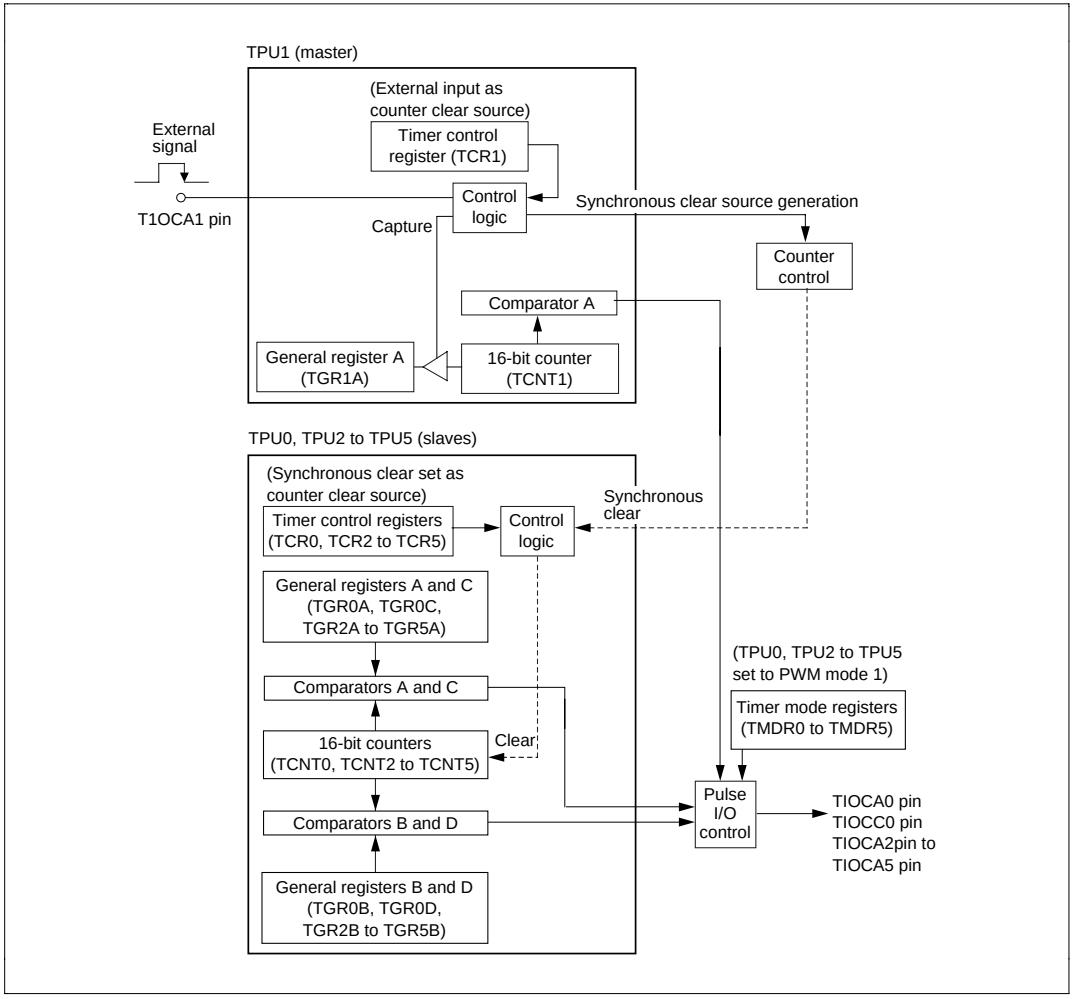


Figure 2 Externally Triggered 7-Phase Pulse Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform externally triggered 7-phase pulse output.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TMDR0 to TMDR5	Select PWM mode 1 as operating mode for TPU0 and TPU2 to TPU5
TCR0 to TCR5	Select timer counter clear sources
TIOCA1	Trigger signal input
TIOCA0, TIOCC0, TIOCA2 to TIOCA5	PWM pulse output
TGR0A, TGR0C, TGR2A to TGR5A	Set high pulse output level (delay time)
TGR0B, TGR0D, TGR2B to TGR5B	Set low pulse output level (pulse width)

Operation

Figure 3 shows the principles of 7-phase pulse output synchronized with an external signal. PWM pulses are output by means of H8S/2655 hardware and software processing as shown in the figure.

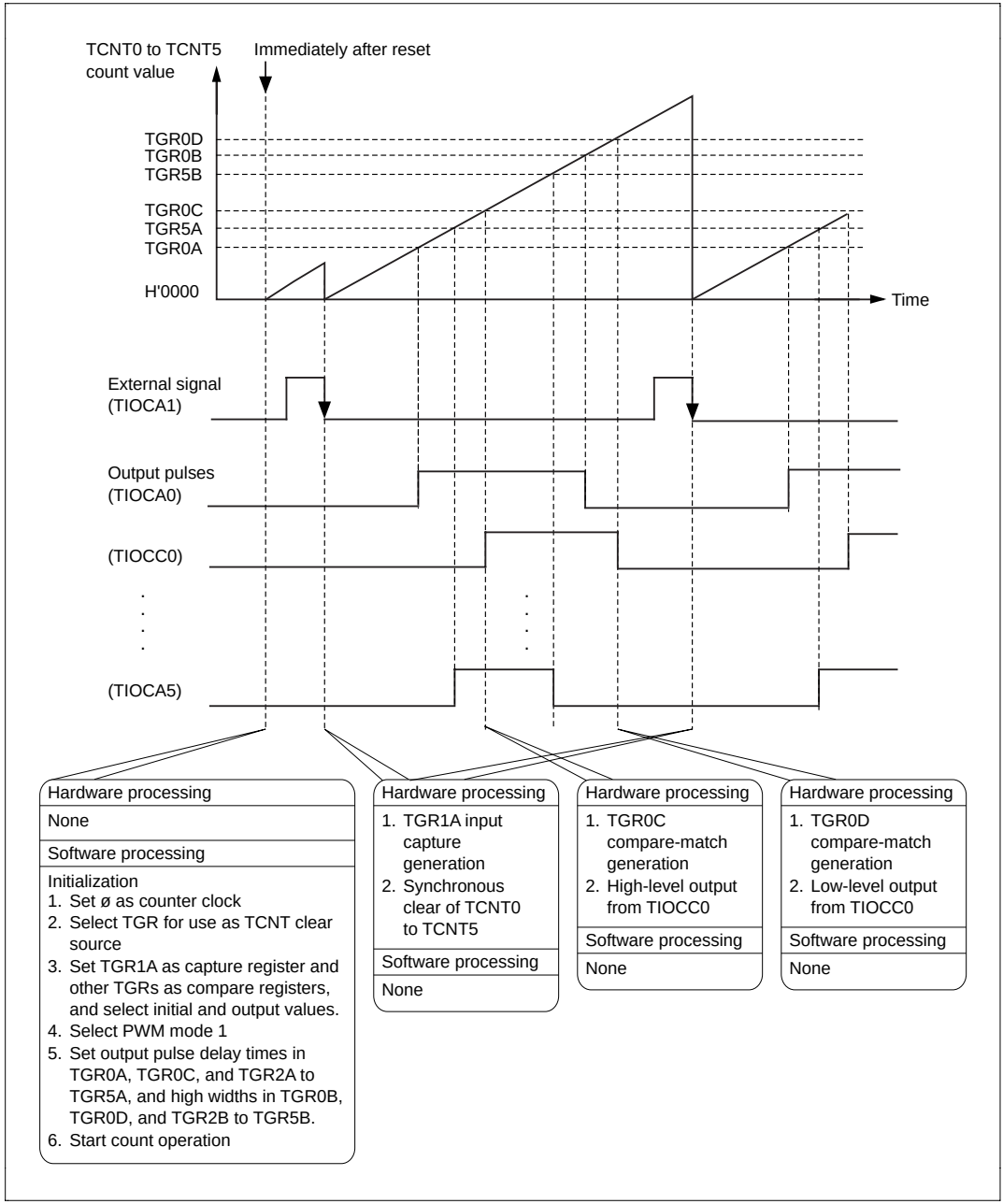


Figure 3 Principles of Pulse Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	cntrsmn	TPU0 to TPU5 synchronous clear and PWM output setting

2. Arguments

Label	Function	Data Length	Module	Input/ Output
set_wid[0] to set_wid[6]	Setting of timer counter value corresponding to pulse high width Pulse high width is found from the following formula: Pulse high width (ns) = Timer counter value × ø cycle (50 ns at 20 MHz operation) × individual channel input clock division ratio	Unsigned short	Main routine	Input
set_dly[0] to set_dly[6]	Setting of timer counter value corresponding to delay time from fall of external input pulse Delay time is found from the following formula: Delay time (ns) = Timer counter value × ø cycle (50 ns at 20 MHz operation) × individual channel input clock division ratio	Unsigned short	Main routine	Input

3. Internal Registers Used

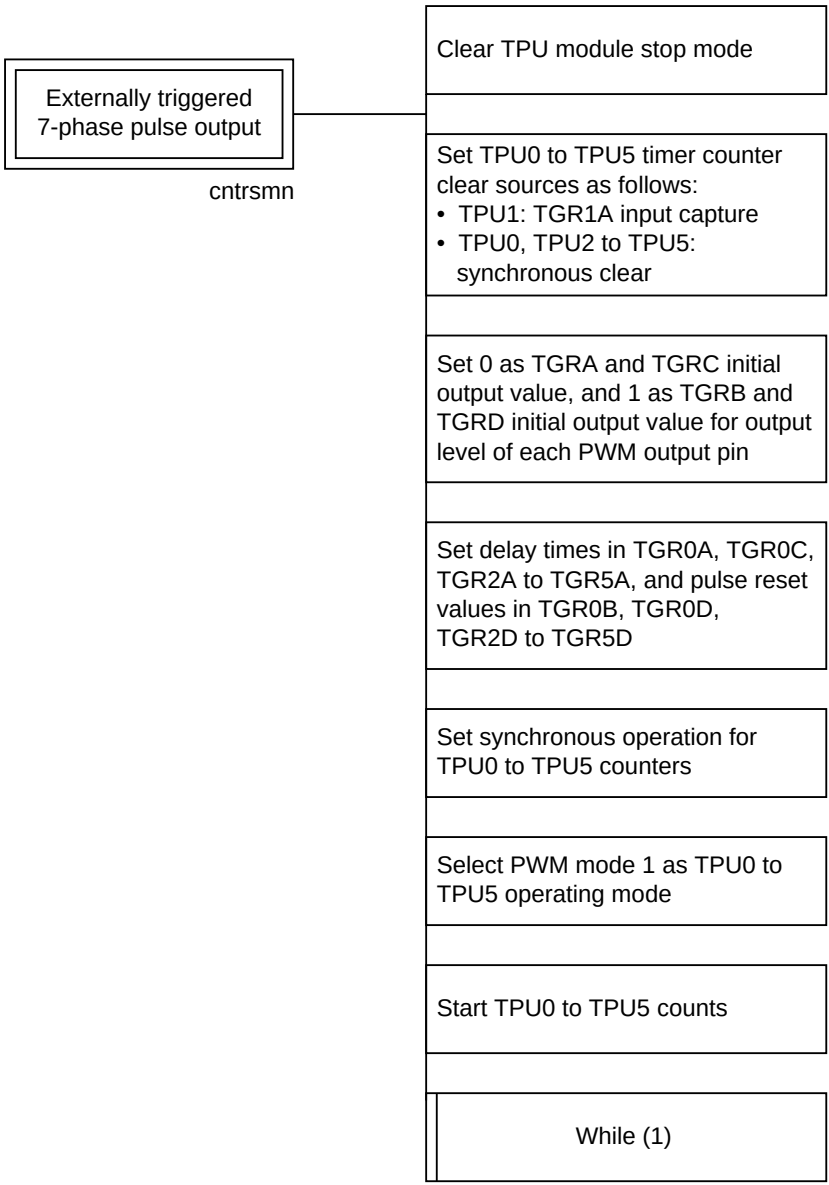
Register Name	Function	Module
TSTR	Performs count starting and stopping for TPU0 to TPU5 timer counters	Main routine
TSYR	Selects synchronous operation of TPU0 to TPU5 TCNT counters	Main routine
TCR0	Sets TGR0A input capture as timer counter clear source	Main routine
TCR1 to TCR5	Set synchronous clear as timer counter clear source	Main routine
TIOR0 to TIOR5	Set output level of each PWM output pin TGRA, TGRC: 0 initial output value TGRB, TGRD: 1 initial output value	Main routine
TMDR0 to TMDR5	Select PWM mode 1	Main routine
TGR0A, TGR0C TGR2A to 5A	Set timer counter values corresponding to delay times from fall of external input pulse	Main routine
TGR0A, TGR0C TGR2A to 5A	Set timer counter values at which low level is output from PWM output pins	Main routine
MSTPCR	Clears TPU module stop mode	Main routine

4. RAM Used

This sample task does not use any RAM apart from the arguments.

PAD

1. Main routine



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTOCOL                   */
/****************************************/
void cntrsmn(void);

/****************************************/
/*      RAM ALLOCATION                  */
/****************************************/
# define set_wid  ((unsigned short * ) 0xffec00)        /* Pulse width time*/
# define set_dly  ((unsigned short * ) 0xffec0e)        /* Delay time */
# define point    ((unsigned short * ) 0xffec1c)        /* Work */

/****************************************/
/*     MAIN PROGRAM : cntrsmn           */
/****************************************/
void cntrsmn(void)
{
        MSTPCR = 0x1fff;         /* Disable module(TPU) stop mode*/
        TPU_TCR0 = 0x60;         /* Initialize TCR0 */
        TPU_TCR1 = 0x20;         /* Initialize TCR1 */
        TCR2 = 0x60;             /* Initialize TCR2 */
        TCR3 = 0x60;             /* Initialize TCR3 */
        TCR4 = 0x60;             /* Initialize TCR4 */
        TCR5 = 0x60;             /* Initialize TCR5 */
        TIOR0H = 0x52;           /* Set TGR0A capture falling edge of input */
        TIOR0L = 0x52;           /* Initialize TIOR0L */
        TIOR1H = 0x09;           /* Initialize TIOR1H */
        TIOR2H = 0x52;           /* Initialize TIOR2H */
        TIOR3H = 0x52;           /* Initialize TIOR3H */
        TIOR3L = 0x52;           /* Initialize TIOR3L */
        TIOR4H = 0x52;           /* Initialize TIOR4H */
        TIOR5H = 0x52;           /* Initialize TIOR5H */

        TGR0A = set_dly[0];      /* Set delay time */
        TGR0C = set_dly[1];
        TGR2A = set_dly[2];
        TGR3A = set_dly[3];
        TGR3C = set_dly[4];
        TGR4A = set_dly[5];
        TGR5A = set_dly[6];

        TGR0B = set_wid[0];      /* Set reset timing */
        TGR0D = set_wid[1];
        TGR2B = set_wid[2];
        TGR3B = set_wid[3];
        TGR3D = set_wid[4];
        TGR4B = set_wid[5];
        TGR5B = set_wid[6];

        TSYR = 0x3f;             /* Set synchronization mode ch0-5 */
```

```
        TMDR0 = 0xc2;           /* Set PWM mode1 */
        TMDR1 = 0xc0;           /* Set usual mode */
        TMDR2 = 0xc2;           /* Set PWM mode1  */
        TMDR3 = 0xc2;           /* Set PWM mode1 */
        TMDR4 = 0xc2;           /* Set PWM mode1 */
        TMDR5 = 0xc2;           /* Set PWM mode1 */

        TSTR = 0x3f;            /* Start TCNT0-5 */
        while(1);               /* Loop */
}
```

Specifications

1. A one-shot pulse is output in synchronization with the falling edge of an external signal, as shown in figure 1.
2. The delay time from the external signal falling edge and the pulse width can be varied within the following ranges:

   1 µs ≤ delay time < reference pulse cycle—pulse width

   50 ns ≤ pulse width < reference pulse cycle—delay time
3. A reference pulse frequency of ..... Hz or higher can be input.

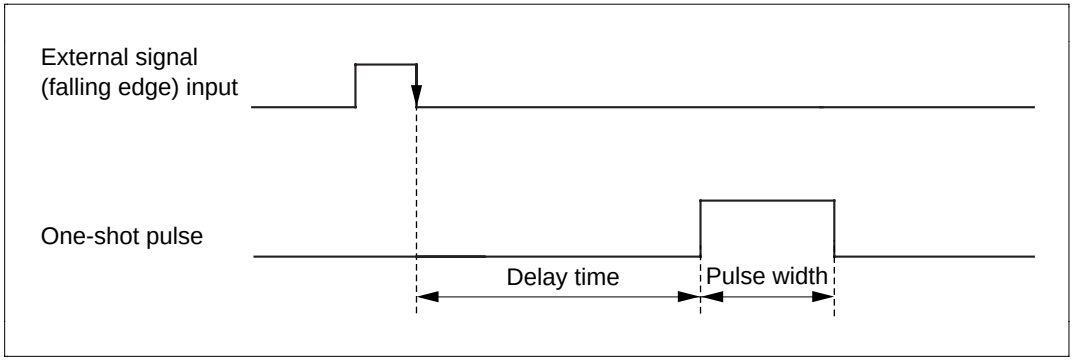


Figure 1 Example of One-Shot Pulse Output

Functions Used

1. In this sample task, a one-shot pulse is output using DMAC0A, DMAC0B, and TPU0.

   a. Figure 2 shows the on-chip function block diagram for this sample task. The following TPU and DMAC functions are used to output a one-shot pulse:

   TPU

   - A function that transfers the buffer register contents to a general register when a compare-match occurs (buffered operation)
   - Output/input capture register setting for each register
   - Counter clearing by input capture

   DMAC

   - A function that activates the DMAC when TPU input capture occurs

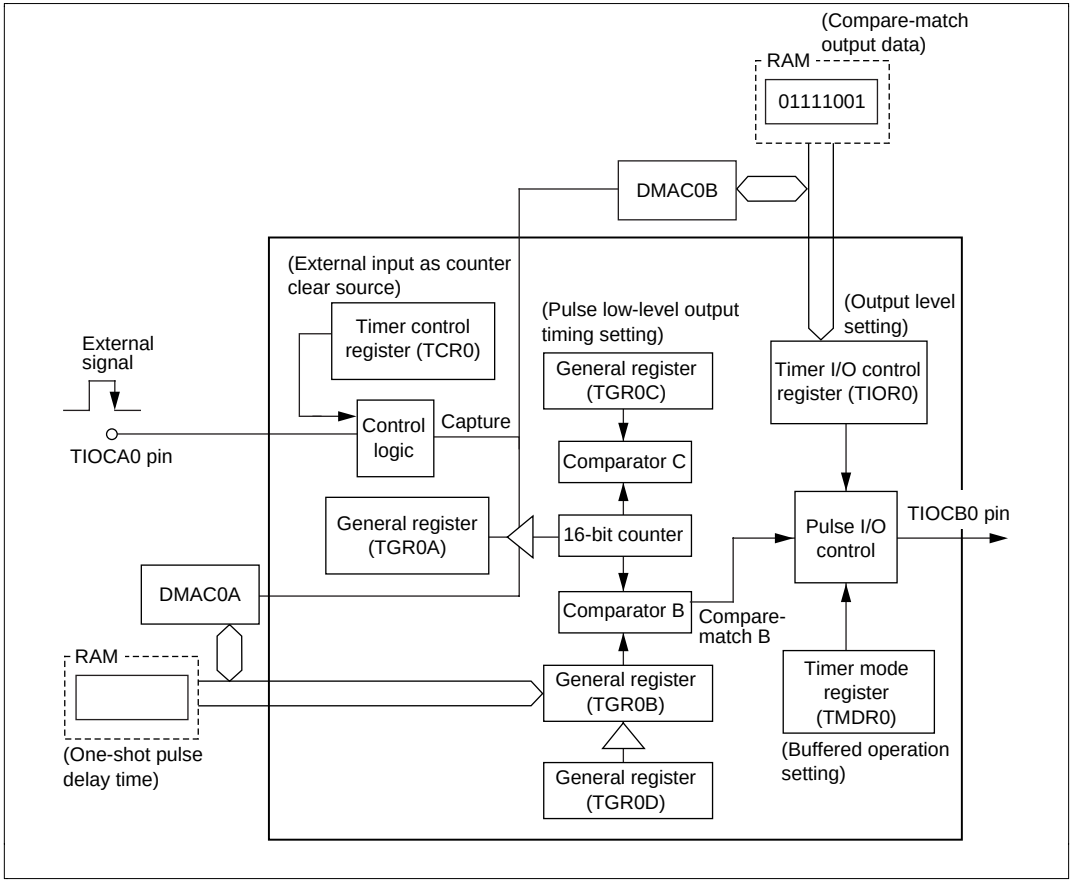


Figure 2 One-Shot Pulse Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform one-shot pulse output.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
TPU0	TCR0	Sets counter clear source
	TIER0	Enables TGI0C interrupt
	TIOR0	Sets TGR0A as capture register, TGR0B and TGR0C as compare-match registers
	TMDR0	Sets buffered operation
	TGR0B	One-shot pulse delay time setting
	TGR0C	One-shot pulse output disabled timing value setting
	TGR0D	One-shot pulse reset timing value setting
	TIOCA0	External signal input
	TIOCB0	One-shot pulse output
DMAC	DMABCRH, DMABCRL	Controls operation of each DMAC channel
	DMACR0A, DMACR0B	Sets transfer size, mode, and activation source for each channel
	MAR0A,B	Transfer source address setting
	IORA0A,B	Transfer destination address setting
	ETCR0A,B	Transfer number setting

Operation

Figure 3 shows the principles of the operation. A one-shot pulse is output by means of H8S/2655 hardware and software processing as shown in the figure.

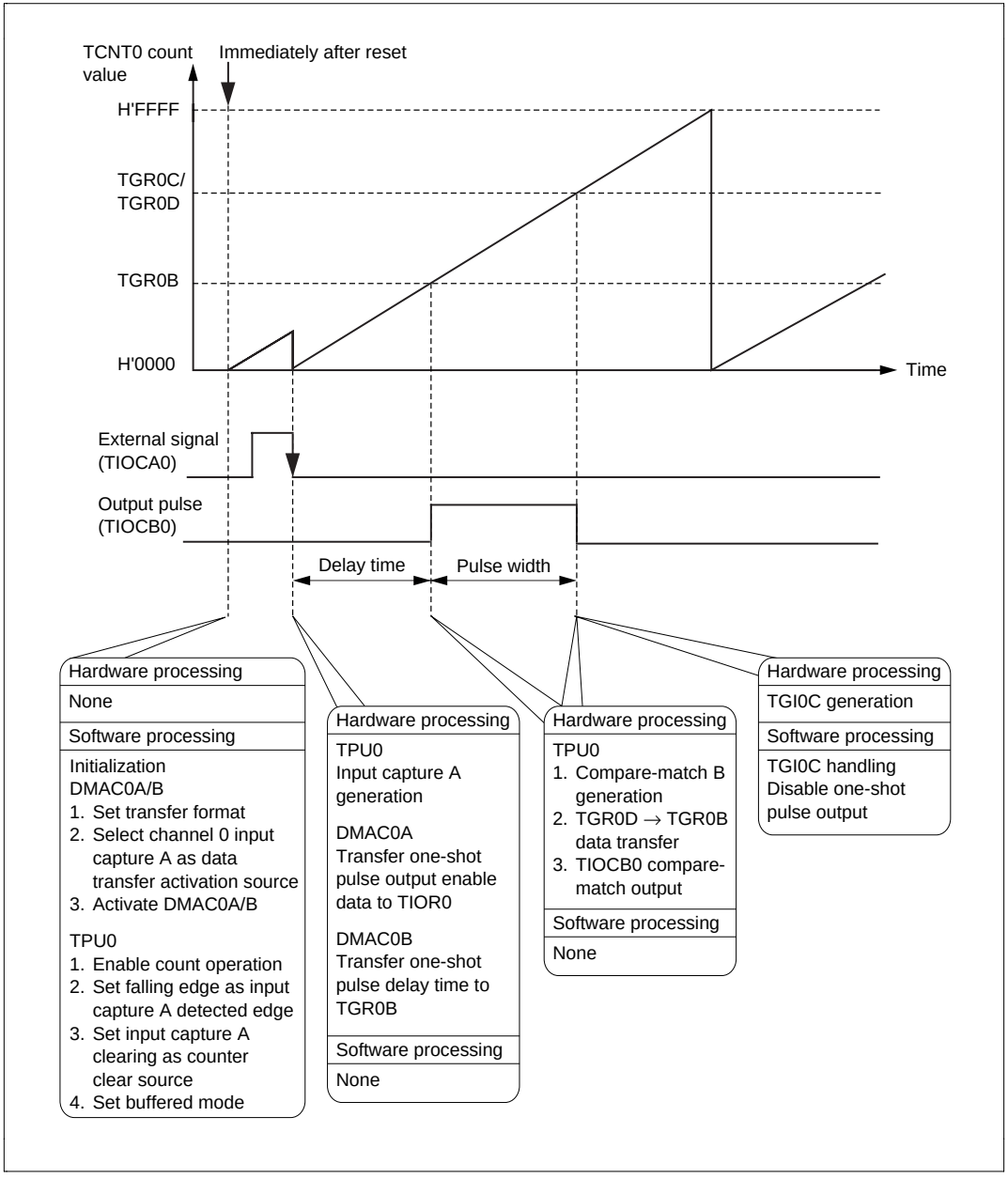


Figure 3 Principles of One-Shot Pulse Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	ONEMN	Sets delay time and pulse width in TGR0B and TGR0D, and one-shot pulse reset value in TGR0C, and outputs one-shot pulse
Pulse output disable	POUTDLE	Disables pulse output

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
set_dly	Setting of timer value corresponding to one-shot pulse delay time Delay time is found from the following formula: Delay time (ns) = Timer value × ø cycle (50 ns at 20 MHz operation)	Unsigned short	Main routine	Input
one_rst	Setting of timer value corresponding to one-shot pulse reset timing Pulse reset timing is found from the following formula: Pulse reset timing (ns) = Timer value × ø cycle (50 ns at 20 MHz operation)	Unsigned short	Main routine	Input
io_cntr	Setting of one-shot pulse output enable data (Input capture A: fall; compare-match B: toggle output)	Unsigned char	Main routine	Output

3. Internal Registers Used

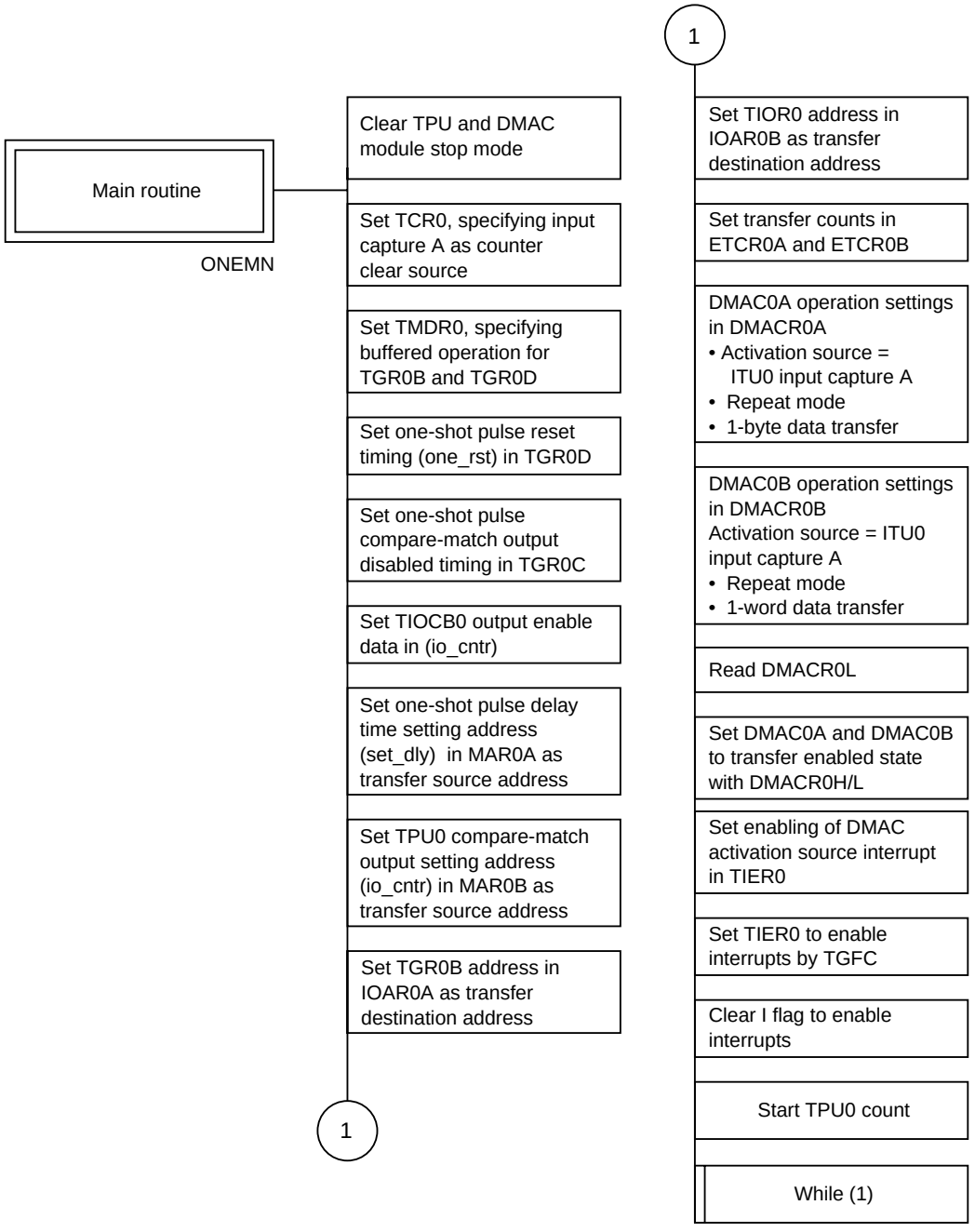
Register Name		Function	Module
TPU0	TSTR	Selects timer counter operation/stopping	Main routine
	TMDR	Sets TGR0B and TGR0D to buffered operation	Main routine
	TCR0	Sets TCNT input clock and counter clear source	Main routine
	TIOR0	Detects falling edge of input pulse	Main routine
		Sets level output from TIOCB0 on compare-match B	Pulse output disable
	TIER0	Enables TGI0C interrupt	Main routine/ pulse output disable
	TSR0	Indicates generation of compare-match by TGR0B	Main routine
	TGR0B	One-shot pulse delay time setting	Main routine
	TGR0C	One-shot pulse output disabled timing value setting	Main routine
	TGR0D	One-shot pulse reset timing value setting	Main routine
DMAC	DMABCR0 DMACR0A/B	Set operation of each DMAC channel	Main routine
	MAR0A/B	Set address of data to be transferred to each register	Main routine
	IOAR0A/B	Set address of transfer destination register of each channel	Main routine
	ETCR0A/B	Set transfer count of each channel	Main routine
MSTPCR		Clears TPU and DMAC module stop mode	Main routine

4. RAM Used

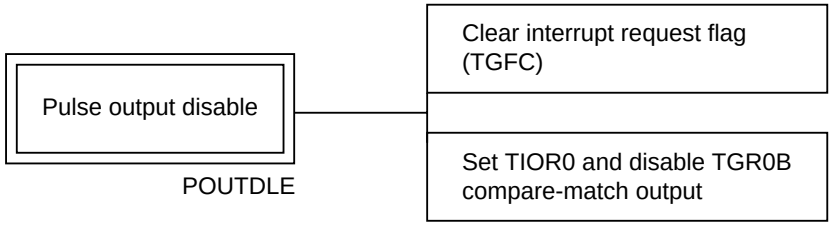
This sample task does not use any RAM apart from the arguments.

PAD

1. Main routine



2. Pulse output disable



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void ONEMN(void);
#pragma interrupt (POUTDLE)
/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/
# define set_dly  ((unsigned short * )0xffec00)  /* Pulse delay time */
# define one_rst  (*(unsigned short * )0xffec02) /* Pulse reset timing */
# define io_cntr  ((unsigned char * )0xffec04)   /* I/O control */

/*********************************/
/*     MAIN PROGRAM: ONEMN       */
/*********************************/
void ONEMN(void)
{
        MSTPCR = 0x5fff;         /* Disable module(DMA,TPU) stop mode*/
        TPU_TCR0 = 0x20;         /* Initialize TCR0 */
        TMDR0 = 0xE0;            /* TGR0B,D: buffer register */
        TGR0D = one_rst;
        TGR0C = one_rst;
        io_cntr[0] = 0x39;       /* Set output toggles at GRB compare match */
                                 /* Set TGRA0 captures falling edge of input */

        MAR0A_W = set_dly;       /* Set base address */
        MAR0B_B = io_cntr;       /* Set base address */
        IOAR0A = 0xffda;         /* Set excute (TGR0B) address */
        IOAR0B = 0xffd2;         /* Set excute (TIOR0H) address */
        ETCR0A = 0x0101;         /* Set excute count */
        ETCR0B = 0x0101;         /* Set excute count */
        DMACR0A = 0xA8;          /* Initialize DMACR0 */
        DMACR0B = 0x28;

        DMABCRH = 0x03;          /* Initialize DMABCR0 */
        DMABCRL |= 0x30;


        TIER0_BP.TGIEA0 = 1;     /* Enable DMAC */
        TIER0_BP.TGIEC0 = 1;     /* Enable TGI0C */
        set_imask_ccr(0);        /* Enable interrupt */
        TSTR = 0x01;             /* Start TCNT0 */
        while(1);                /* Loop */
}

/*********************************/
/*   INTERRUPT PROGRAM: POUTDLE   */
/*********************************/
void POUTDLE(void)
{
        TSR0_BP.TGFC0 = 0;       /* Clear TGFC0 request */
        TIOR0H = 0x09;           /* Set disenable output data */
}
```

Specifications

1. Four sets of asynchronous 4-bit pulse outputs are generated using PPG output, as shown in figure 1.

2. A TPU compare-match is used as the PPG activation source.

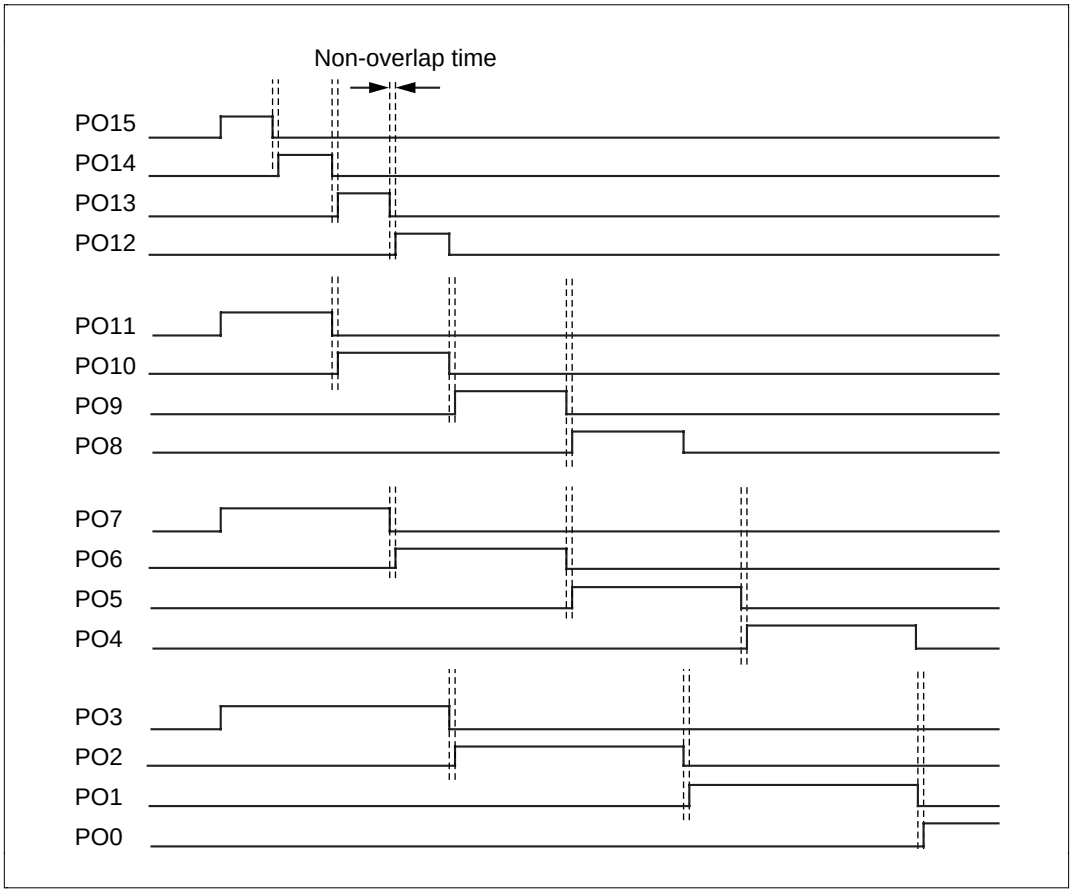


Figure 1 Example of Four 4-Bit Outputs

Functions Used

1. In this sample task, four sets of asynchronous 4-bit pulse outputs are generated using TPU0 to TPU3 and PPG output groups 3 to 0.

   a. Figure 2 shows the block diagram for this sample task, taking the example of TPU3/PPG group 3 pulse output. This task uses the following functions:

   - The ability to select output trigger signals in 4-bit groups, and to generate a maximum of four 4-bit outputs
   - The ability to select the output trigger signal for each group from among TPU 4-channel compare-match signals
   - The ability to set the non-overlap time between different pulse outputs

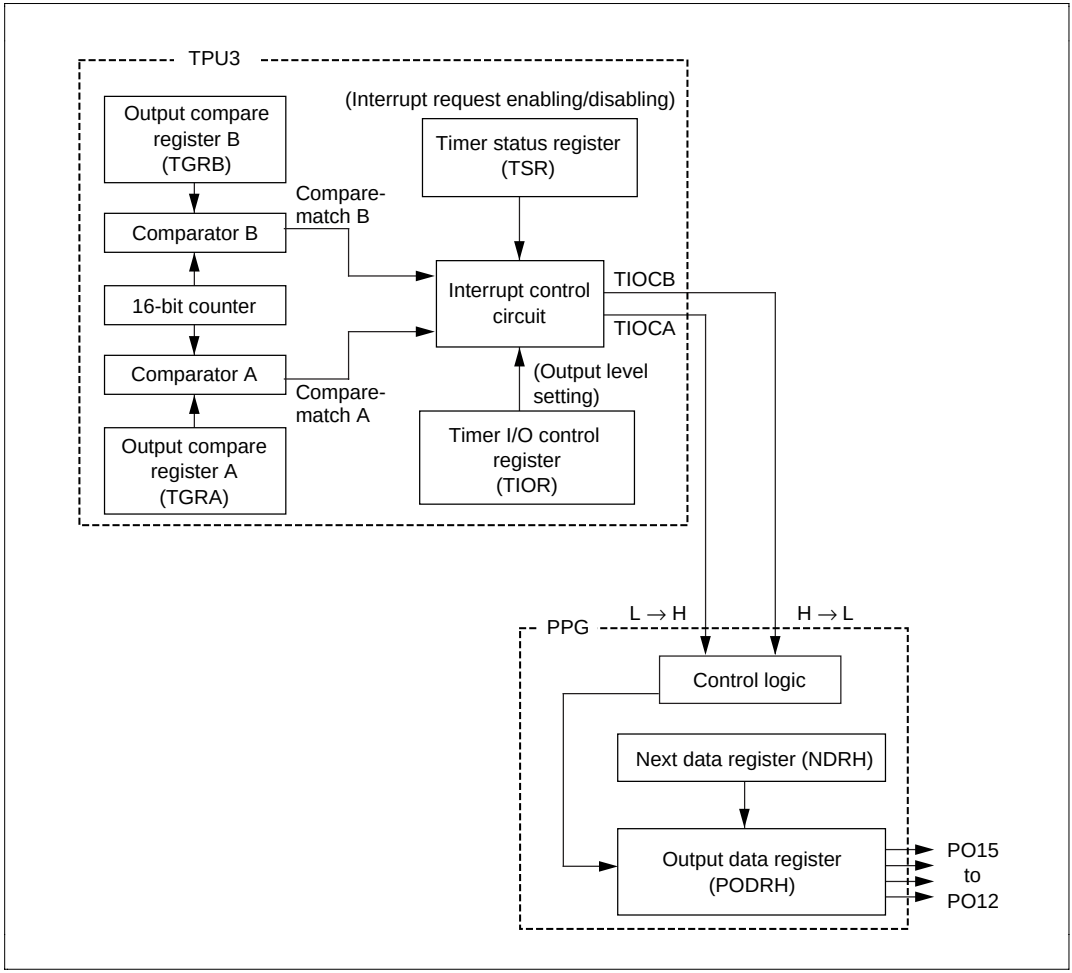


Figure 2 Four 4-Bit Output (Group 3) Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to generate four 4-bit outputs.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
PPG	PO15 to PO0	Four 4-bit outputs
	PMR	Non-overlap mode setting
	PCR	PPG output trigger signal setting
	NDERH	Enables PO15 to PO8 PPG output
	NDERL	Enables PO7 to PO0 PPG output
	NDRH	Stores PPG output data to be output next
	NDRL	Stores PPG output data to be output next
	P1DDR	PPG output pin setting
	P2DDR	PPG output pin setting
	PODRH	Stores PO15 to PO8 output data
	PODRL	Stores PO7 to PO0 output data
TPU	TGR0A to TGR3A	Non-overlap mode time setting
	TGR0B to TGR3B	PPG output trigger cycle setting
	TCR0 to TCR3	Counter clock and counter clear source setting
	TSR0 to TSR3	Indicate compare-match generation
	TIOR0 to TIOR3	TGR control
	TSTR	Selects timer counter operation/stopping
MSTPCR		Clears TPU and DMAC module stop mode

Operation

Figure 3 shows the principles of the data output operation using PPG output group 3. Four-phase non-overlap output is performed by means of H8S/2655 hardware and software processing.

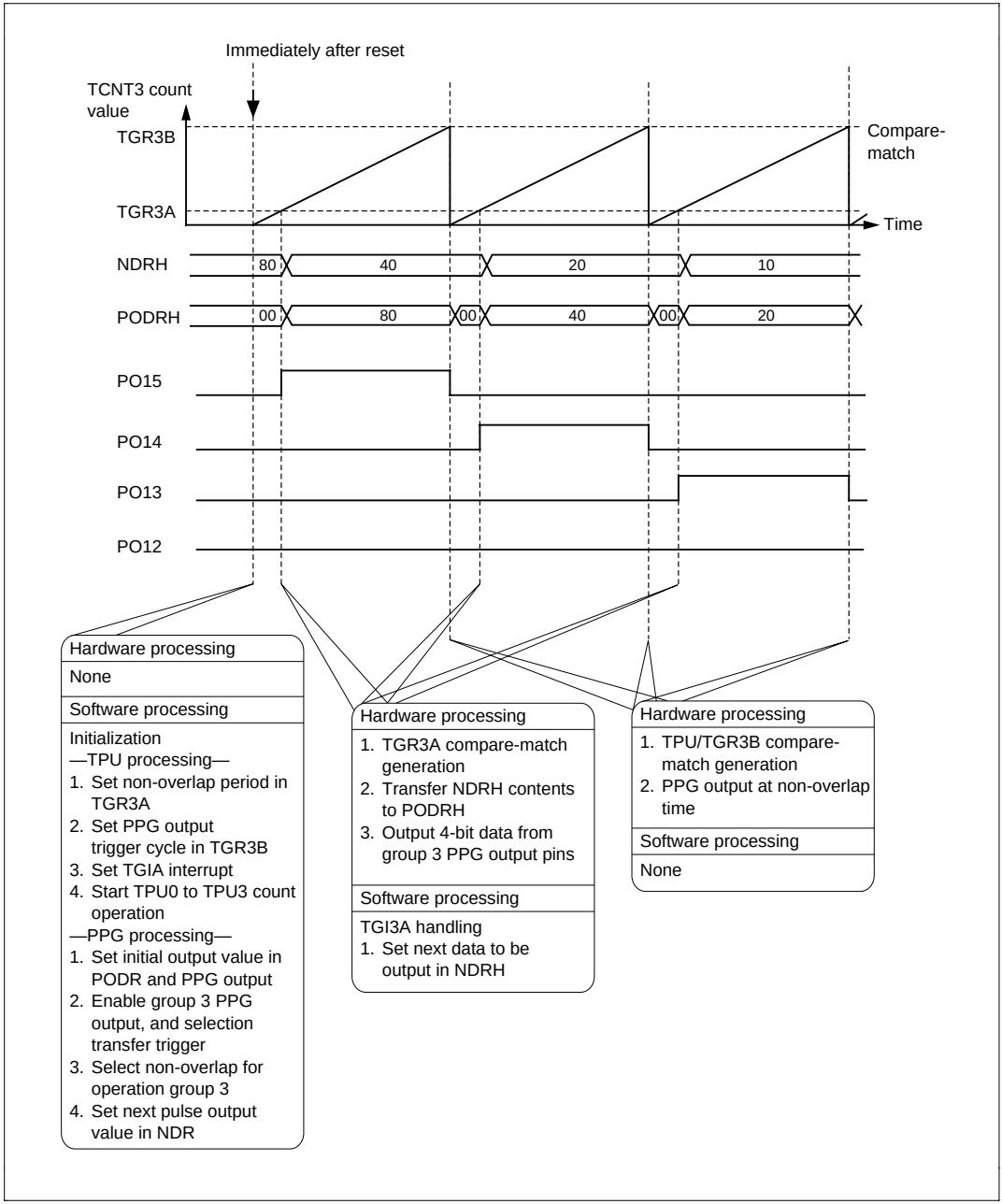


Figure 3 Principles of Four 4-Bit Output (Group 3) Operation

Software

1. Modules

Module Name	Label	Function
Main routine	ppg16mn	PPG and TPU initialization
Data setting 0	setdat0	Sets next data to be output in NDR0 (group 0)
Data setting 1	setdat1	Sets next data to be output in NDR1 (group 1)
Data setting 2	setdat2	Sets next data to be output in NDR2 (group 2)
Data setting 3	setdat3	Sets next data to be output in NDR3 (group 3)

2. Arguments

Label	Function	Data Length	Module	Input/Output
addcnt0	PPG group 0 output transfer counter	Unsigned char	Data setting 0	Output
addcnt1	PPG group 1 output transfer counter	Unsigned char	Data setting 1	Output
addcnt2	PPG group 2 output transfer counter	Unsigned char	Data setting 2	Output
addcnt3	PPG group 3 output transfer counter	Unsigned char	Data setting 3	Output

3. Internal Registers Used

Register Name		Function	Module
PPG	P1DDR	Enables PPG output for PO15 to PO8	Main routine
	P2DDR	Enables PPG output for PO7 to PO0	Main routine
	P1DR	Stores PO15 to PO8 output pattern data	Main routine
	P2DR	Stores PO7 to PO0 output pattern data	Main routine
	PMR	Sets PO15 to PO0 as non-overlap outputs	Main routine
	PCR	Selects pulse output trigger signal for each group Group 3: TPU3 compare-match Group 2: TPU2 compare-match Group 1: TPU1 compare-match Group 0: TPU0 compare-match	Main routine
	NDERL	Enables PPG outputs PO7 to PO0	Main routine
	NDERH	Enables PPG outputs PO15 to PO8	Main routine
	NDRL	Sets next output pattern for PO7 to PO0	Main routine Data setting 0, data setting 1
	NDRH	Sets next output pattern for PO15 to PO8	Main routine Data setting 2, data setting 3
TPU	TGR0A to TGR3A	Non-overlap time settings	Main routine
	TGR0B to TGR3B	PPG output trigger cycle settings	Main routine
	TCR0 to TCR3	Following TCR settings: • Counter clearing on TGRB compare-match • Counting on ø internal clock	Main routine
	TSR0 to TSR3	Indicate compare-match generation	Main routine
	TIER0 to TIER3	Enable TGIA interrupts	Main routine
	TSTR	Enables TCNT count operation	Main routine
MSTPCR		Clears TPU and PPG module stop mode	Main routine

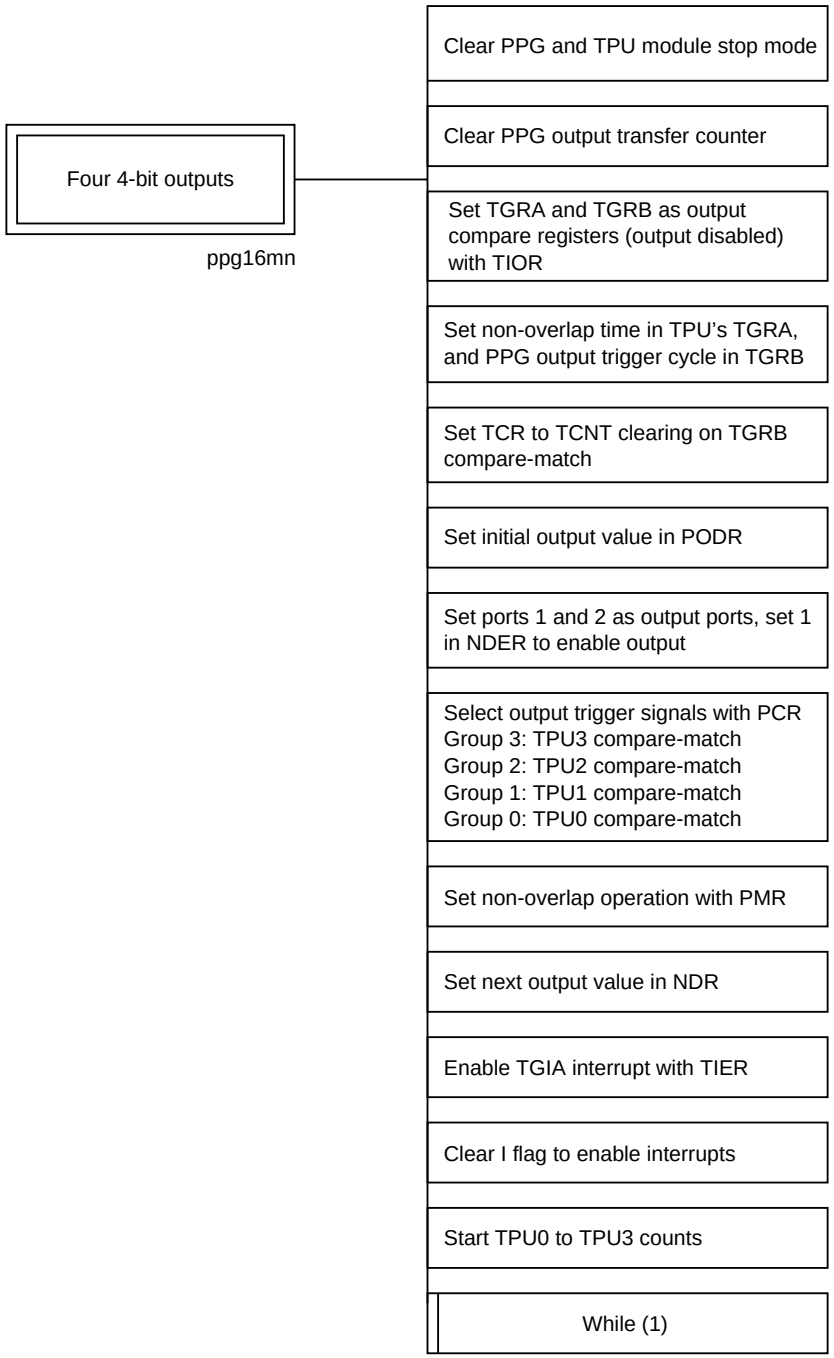
4. RAM Used

This sample task does not use any RAM apart from the arguments.

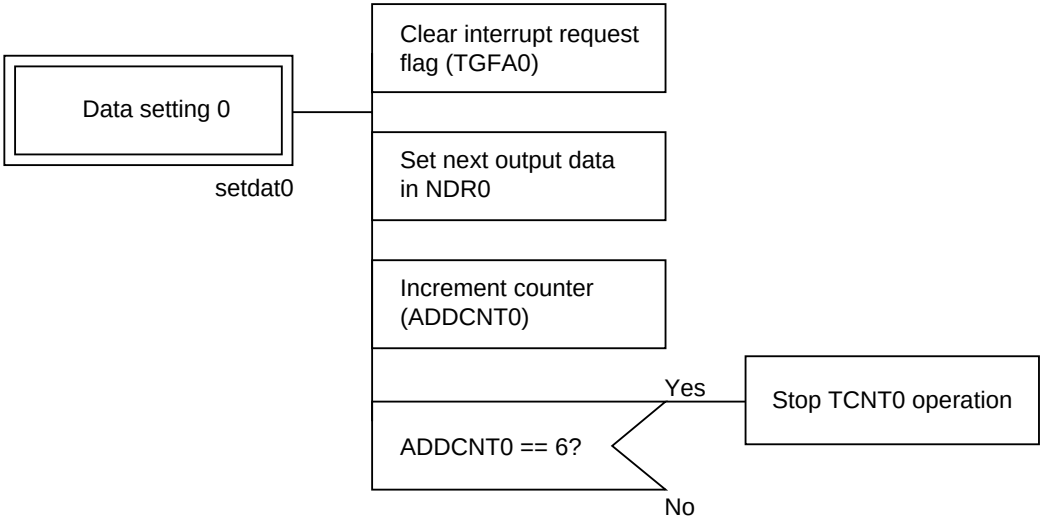
Table Name	Function	Data Length	Data Capacity
ndat_tab0	Stores data output from PPG group 0	Unsigned char	4 bytes
ndat_tab1	Stores data output from PPG group 1	Unsigned char	4 bytes
ndat_tab2	Stores data output from PPG group 2	Unsigned char	4 bytes
ndat_tab3	Stores data output from PPG group 3	Unsigned char	4 bytes

PAD

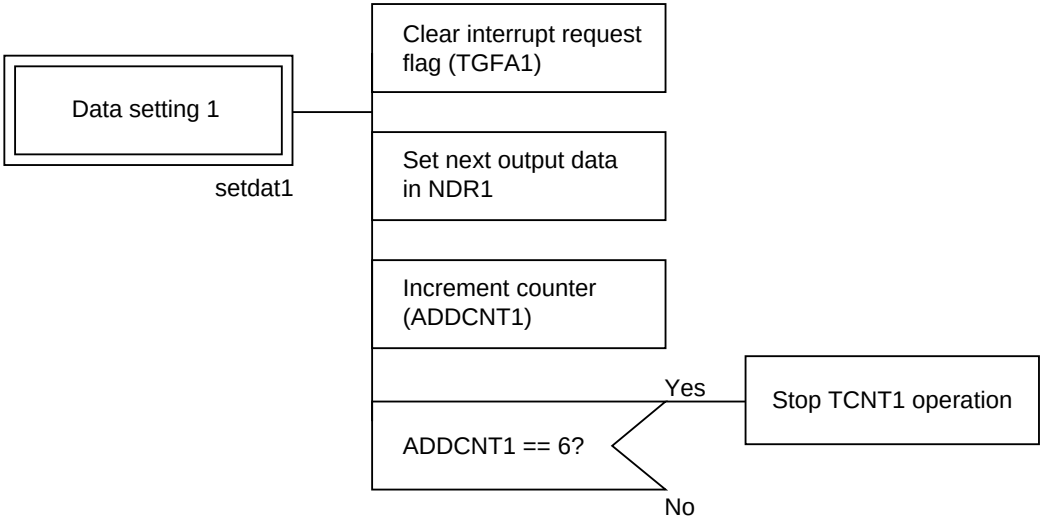
1. Main routine



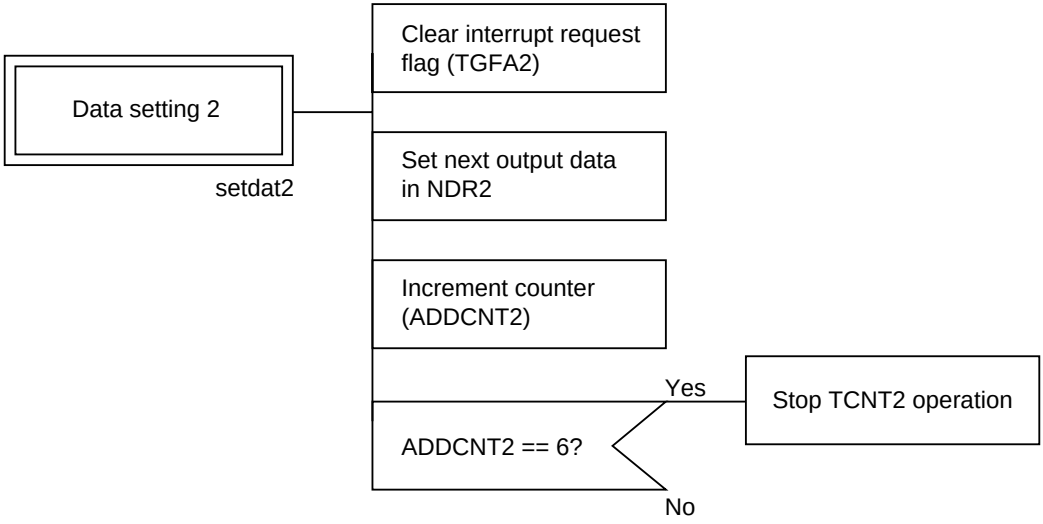
2. Data setting 0



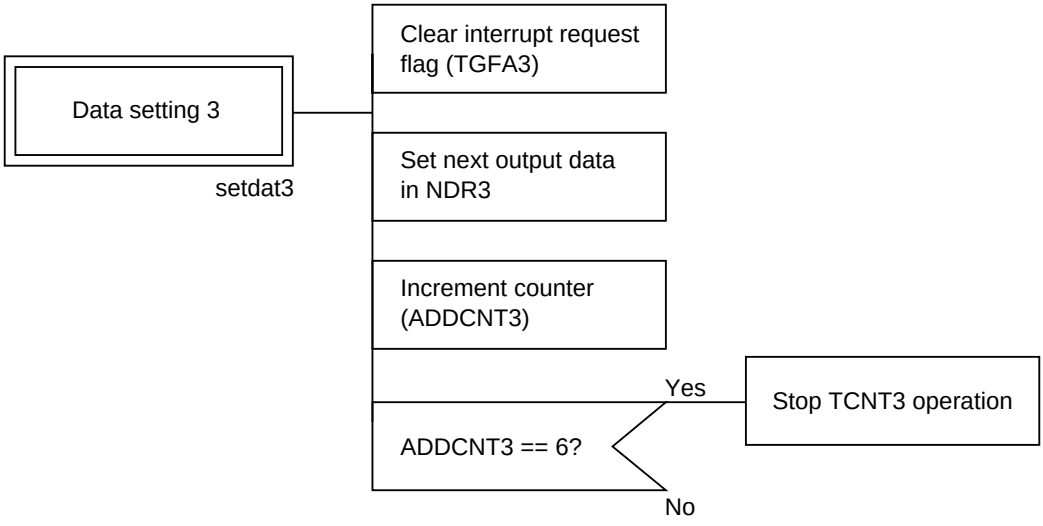
3. Data setting 1



4. Data setting 2



5. Data setting 3



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTOCOL                   */
/****************************************/
void ppg16mn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
# define ndat_tab0 ((unsigned char * ) 0xffec00)   /* Outout data table */
# define addcnt0  (*(unsigned char * ) 0xffec05)   /* Group0 transmit counter */
# define ndat_tab1 ((unsigned char * ) 0xffec06)   /* Outout data table */
# define addcnt1  (*(unsigned char * ) 0xffec0b)   /* Group1 transmit counter */
# define ndat_tab2 ((unsigned char * ) 0xffec0c)   /* Outout data table */
# define addcnt2  (*(unsigned char * ) 0xffec11)   /* Group2 transmit counter */
# define ndat_tab3 ((unsigned char * ) 0xffec12)   /* Outout data table */
# define addcnt3  (*(unsigned char * ) 0xffec17)   /* Group3 transmit counter */

/****************************************/
/*     MAIN PROGRAM : ppg16mn           */
/****************************************/
void ppg16mn(void)
{
        MSTPCR = 0x17ff;              /* Disable module(PPG,TPU) stop mode */

        addcnt0 = 0;                  /* Transmit counter clear */
        addcnt1 = 0;
        addcnt2 = 0;
        addcnt3 = 0;

        TIOR0H = 0x00;                /* Initialize TIOR0H */
        TIOR1H = 0x00;                /* Initialize TIOR1H */
        TIOR2H = 0x00;                /* Initialize TIOR2H */
        TIOR3H = 0x00;                /* Initialize TIOR3H */

        TGR0A = 0x00c8;               /* Set non overlap time */
        TGR0B = 0x1388;               /* Set PPG output cycle */
        TGR1A = 0x00c8;               /* Set non overlap time */
        TGR1B = 0x2710;               /* Set PPG output cycle */
        TGR2A = 0x00c8;               /* Set non overlap time */
        TGR2B = 0x3a98;               /* Set PPG output cycle */
        TGR3A = 0x00c8;               /* Set non overlap time */
        TGR3B = 0x4e20;               /* Set PPG output cycle */

        TPU_TCR0 = 0x40;              /* Initialize TCR0 */
        TPU_TCR1 = 0x40;              /* Initialize TCR1 */
        TCR2 = 0x40;                  /* Initialize TCR2 */
        TCR3 = 0x40;                  /* Initialize TCR3 */

        PODRH = 0;                    /* Set first output data */
        PODRL = 0;                    /* Set first output data */
```

```
        P1DDR = 0xff;                /* Port1: output */
        P2DDR = 0xff;                /* Port2: output */
        NDERH = 0xff;                /* Enable transmit data */
        NDERL = 0xff;                /* Enable transmit data */

        PCR = 0xe4;                  /* Set output toriga */
        PMR = 0xff;                  /* Set non overlap mode */

        NDR0 = ndat_tab0[addcnt0++]; /* Set second output data */
        NDR1 = ndat_tab1[addcnt1++]; /* Set second output data */
        NDR2 = ndat_tab2[addcnt2++]; /* Set second output data */
        NDR3 = ndat_tab3[addcnt3++]; /* Set second output data */

        TIER0 = 0x41;                /* Initialize TIER0 */
        TIER1 = 0x41;                /* Initialize TIER1 */
        TIER2 = 0x41;                /* Initialize TIER2 */
        TIER3 = 0x41;                /* Initialize TIER3 */
        set_imask_ccr(0);                /* Enable interrupt */
        TSTR = 0x0f;                 /* Start TCNT0-3 */
        while(1);                    /* Loop */
}
/*********************************/
/*   INTERRUPT PROGRAM: setdat0   */
/*********************************/
#pragma interrupt (setdat0)
void setdat0(void)
{
        TSR0_BP.TGFA0 = 0;           /* Clear TGFA0 request*/
        NDR0 = ndat_tab0[addcnt0++]; /* Set next output data */
          if(addcnt0 == 6)           /* All output end ? */
          TSTR_BP.CST0 = 0;          /* Stop TCNT0 */
}

/*********************************/
/*   INTERRUPT PROGRAM: setdat1   */
/*********************************/
#pragma interrupt (setdat1)
void setdat1(void)
{
        TSR1_BP.TGFA1 = 0;           /* Clear TGFA1 request*/
        NDR1 = ndat_tab1[addcnt1++]; /* Set next output data */
          if(addcnt1 == 6)           /* All output end ? */
          TSTR_BP.CST1 = 0;          /* Stop TCNT1 */
}

/*********************************/
/*   INTERRUPT PROGRAM: setdat2   */
/*********************************/
#pragma interrupt (setdat2)
void setdat2(void)
{
        TSR2_BP.TGFA2 = 0;           /* Clear TGFA2 request */
        NDR2 = ndat_tab2[addcnt2++]; /* Next output data */
          if(addcnt2 == 6)           /* All output end ? */
          TSTR_BP.CST2 = 0;          /* Stop TCNT2 */
}
```

```
/*********************************/
/*   INTERRUPT PROGRAM: setdat3   */
/*********************************/
#pragma interrupt (setdat3)
void setdat3(void)
{
        TSR3_BP.TGFA3 = 0;            /* Clear TGFA3 request */
        NDR3 = ndat_tab3[addcnt3++];  /* Set next output data */
          if(addcnt3 == 6)            /* All output end ? */
          TSTR_BP.CST3 = 0;           /* Stop TCNT3 */
}
```

Specifications

1. One-byte data is transferred asynchronously between an H8S/2655 chip and H8/3314 chip, as shown in figure 1.
2. The communication format is 9600 bps, 8-bit data, one stop bit, and no parity.
3. Communication control is performed by means of RTS and CTS.

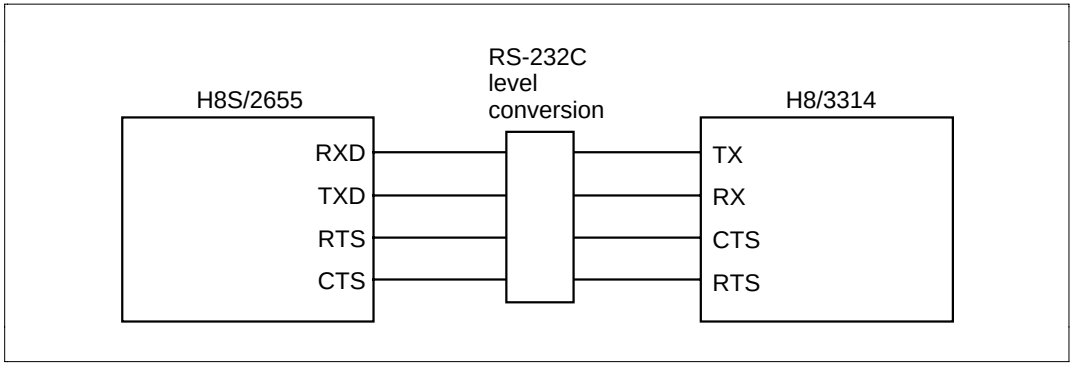


Figure 1 H8S/2655 Asynchronous SCI Block Diagram

Functions Used

1. In this sample task, data transmission/reception is performed using SCI0. Port 3 is used for the communication control pins (RTS, CTS).

   a. Figure 2 shows the SCI transmission block diagram for this sample task. Data is transmitted to the H8/3314 using the following functions:

      - A function that performs data communication asynchronously, establishing synchronization character by character (asynchronous mode)
      - A function that generates an interrupt when transmission is completed (TEI interrupt)

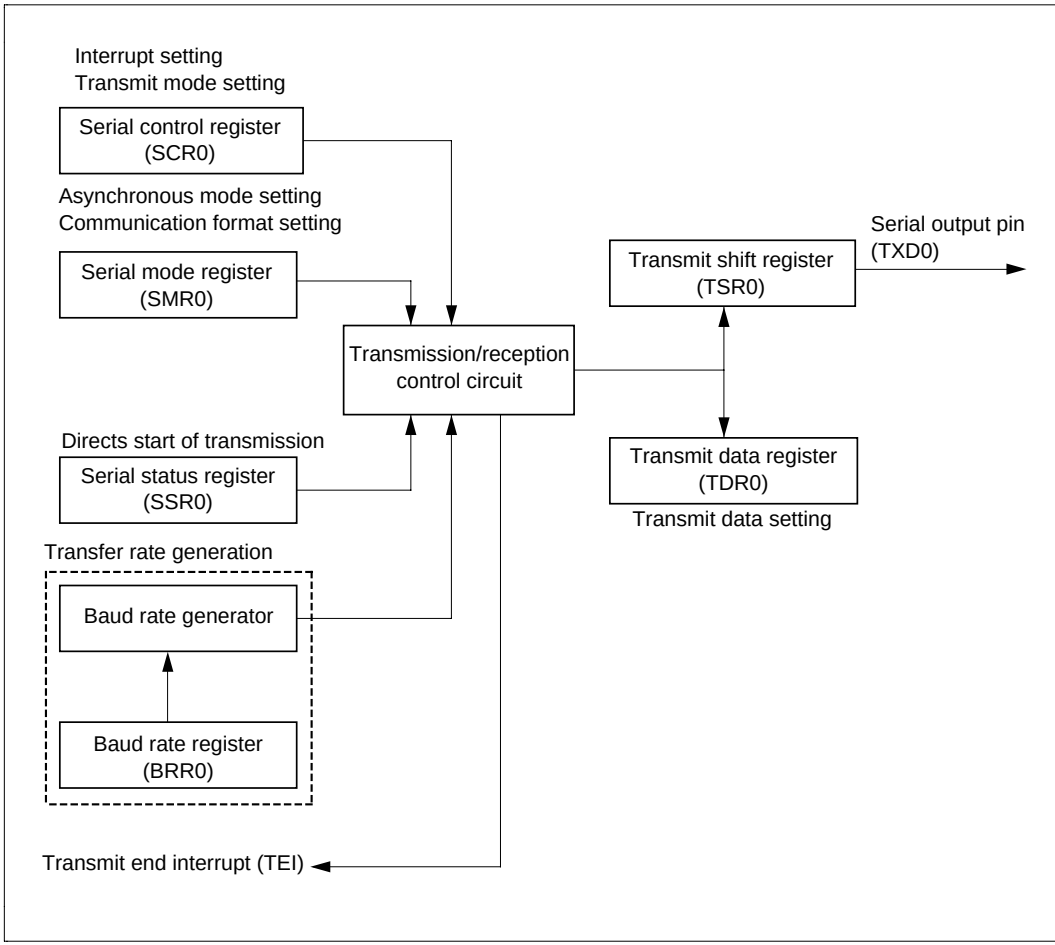


Figure 2 SCI Transmission Block Diagram

b. Figure 3 shows the SCI reception block diagram for this sample task. Data is received from the H8/3314 using the following functions:

- A function that performs data communication asynchronously, establishing synchronization character by character (asynchronous mode)
- A function that generates an interrupt when reception is completed (RXI interrupt)

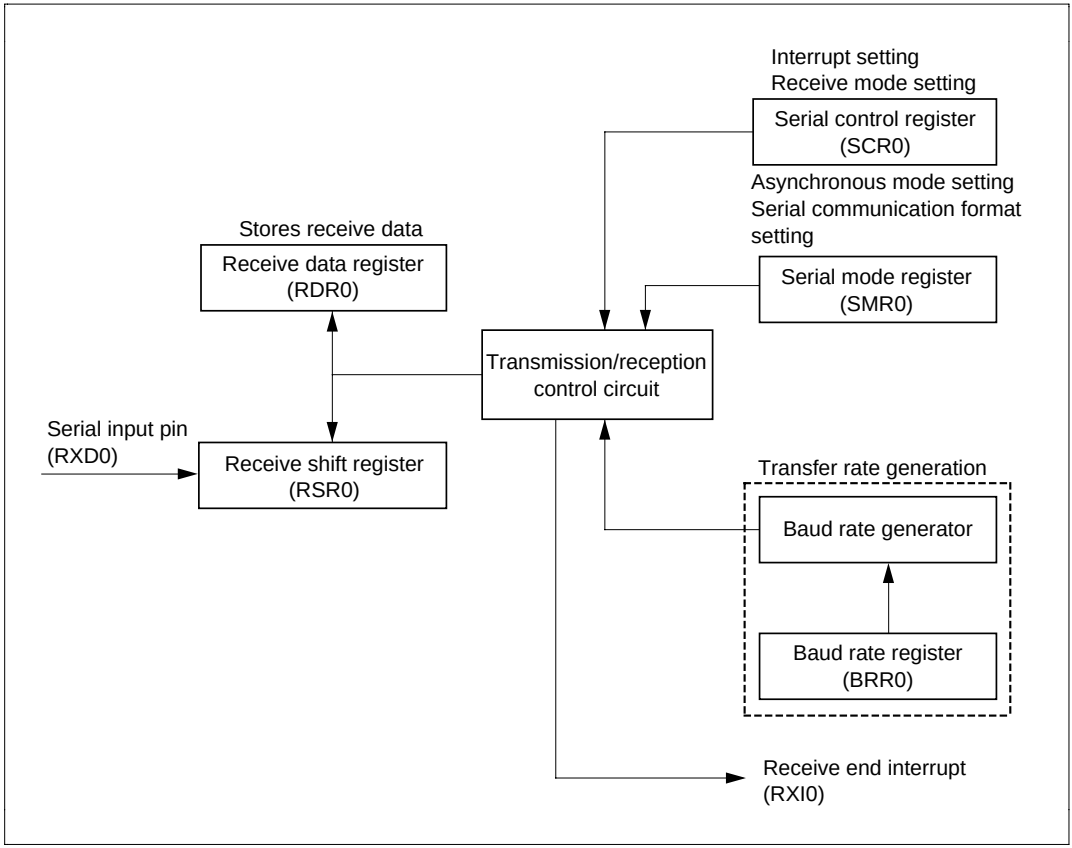


Figure 3 SCI Reception Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform interfacing to an H8/3314 chip.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
RXD0	Receives data from console
TXD0	Transmits data to console
SMR0	Sets SCI to asynchronous mode, and sets communication format
SCR0	Enables transmit/receive interrupts and sets SCI to transmit/receive mode
SSR0	Directs start of transmission with TDRE
RDR0	Holds data received from console
TDR0	Holds data to be transmitted to console
BRR0	Transfer rate setting

Operation

Figure 4 shows the principles of the operation performed by this task. Interfacing to an H8/3314 is performed by means of hardware and software processing, using the timing shown in this figure.

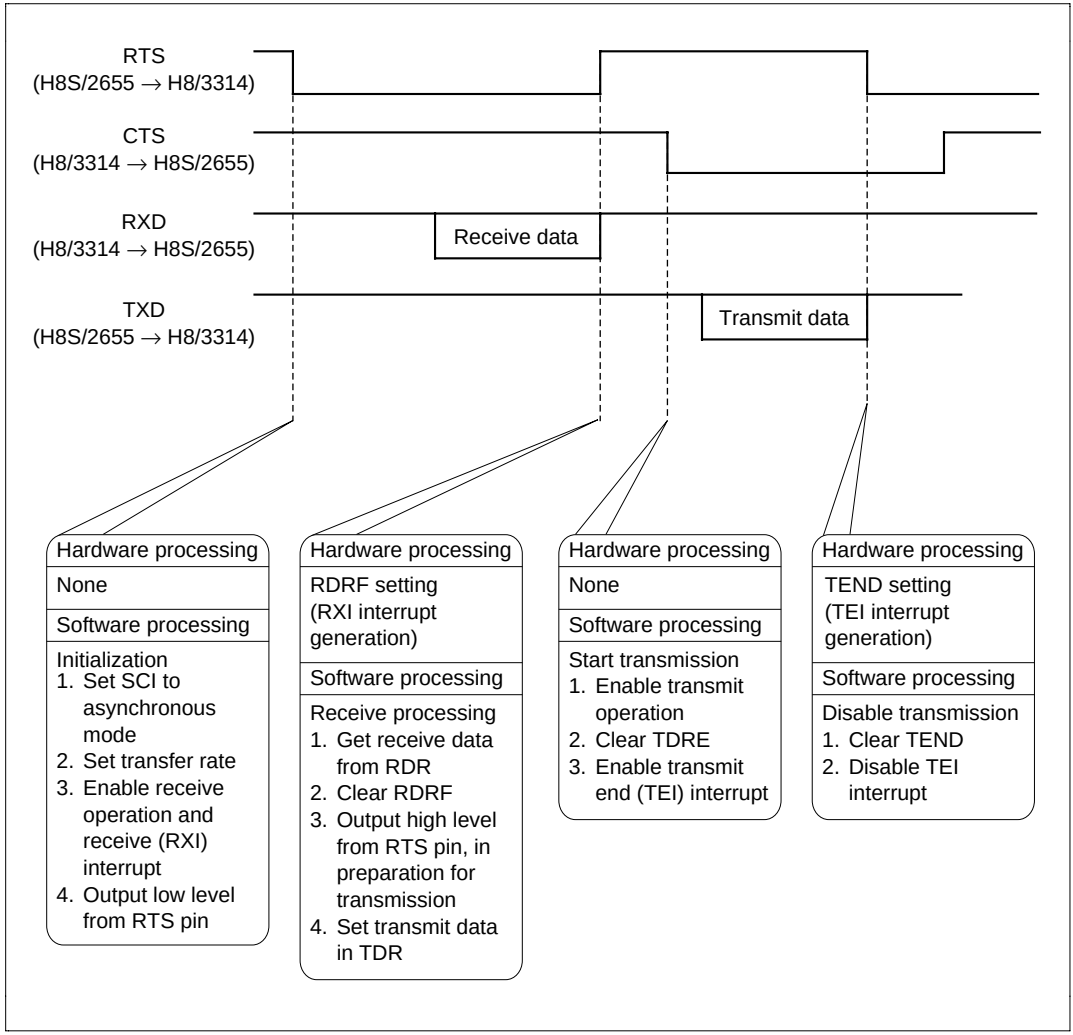


Figure 4 Principles of Asynchronous SCI Operation

Software

1. Modules

Module Name	Label	Function
Main routine	ASCMN	SCI initialization, transmission/reception management
Data receive end	ASCRX	Initiated by RXI interrupt; performs data reception
Data transmit end	ASCTE	Initiated by TEI interrupt; reports completion of transmission

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/Output
rcv_data	Holds data received from console	Unsigned char	Data receive end	Output
			Main routine	Input
rxendf	Flag indicating completion of reception	Unsigned char	Data receive end	Output
	1: Reception completed 0: Reception in progress		Main routine	Input
txendf	Flag indicating completion of transmission	Unsigned char	Data transmit end	Output
	1: Transmission completed 0: Transmission in progress		Main routine	Input

3. Internal Registers Used

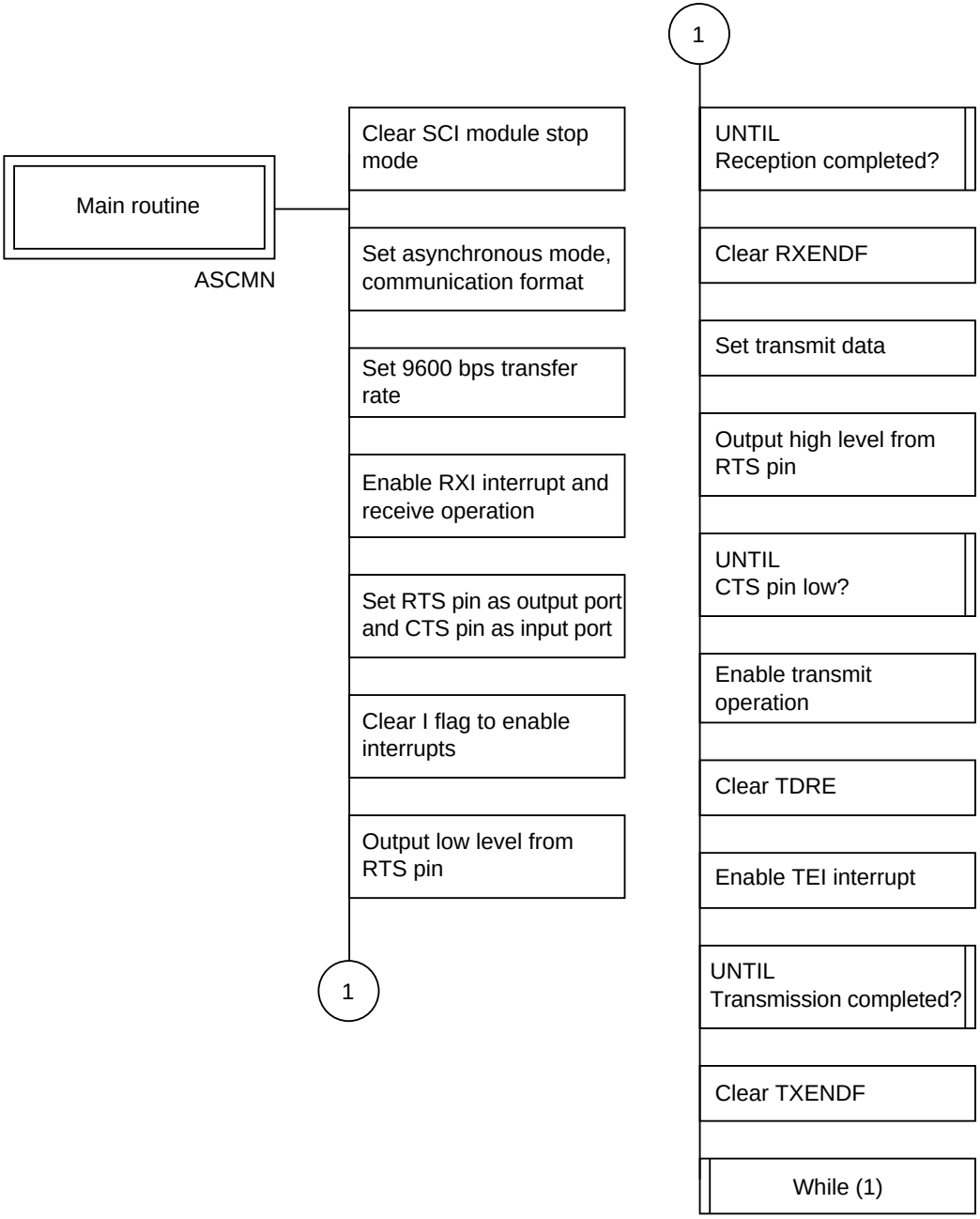
Register Name	Function	Module
SMR0	Sets SCI mode (asynchronous) and communication format, and selects clock for baud rate generator (ø clock input)	Main routine
SCR0	Enables interrupts (RXI, TEI) and sets SCI transmission/reception enabling	Main routine
SSR0	Directs start of transmission by clearing TDRE (b7)	Main routine
RDR0	Holds data received from console	Data receive end
TDR0	Holds data to be transmitted to console	Main routine
BRR0	Transfer rate setting	Main routine
P3DDR	Port 3 input/output setting	Main routine
P3DR	Performs RTS and CTS pin manipulation	Main routine
MSTPCR	Clears SCI module stop mode	Main routine

4. RAM Used

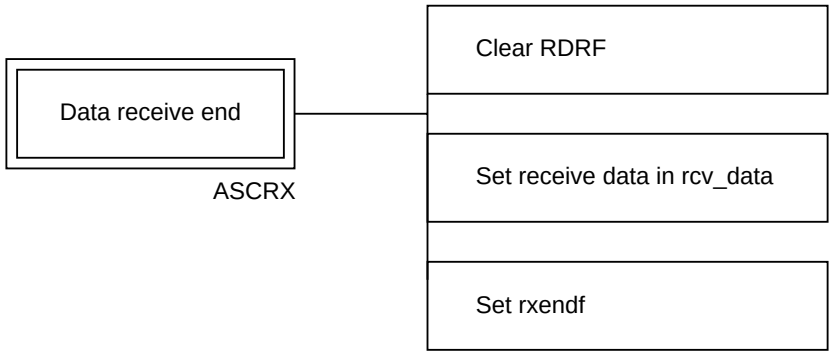
This sample task does not use any RAM apart from the arguments.

PAD

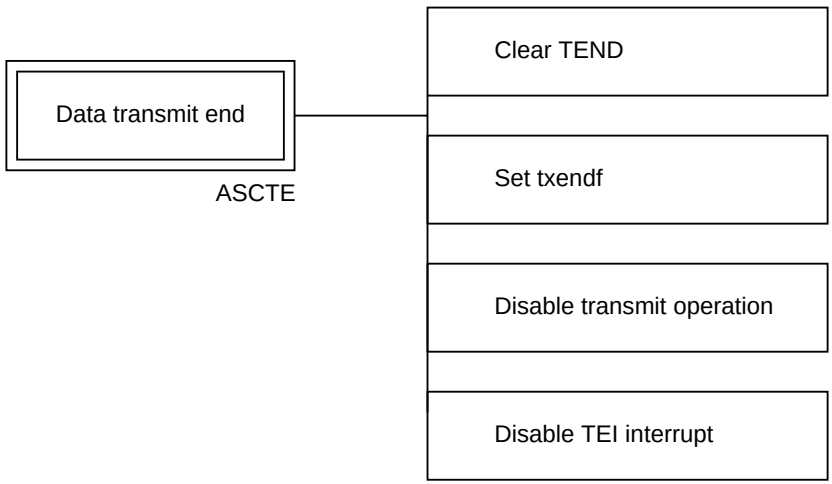
1. Main routine



2. Data receive end



3. Data transmit end



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void ASCMN(void);
#pragma interrupt (ASCRX)
#pragma interrupt (ASCTE)

/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/

# define rcv_data  (*(unsigned char * )0xffec00) /* Receive data from console */
# define rxendf    (*(unsigned char * )0xffec01) /* Receive end flag */
# define txendf    (*(unsigned char * )0xffec02) /* Transmit end flag */

/*********************************/
/*     MAIN PROGRAM: ASCMN       */
/*********************************/
void ASCMN(void)
        {
        MSTPCR = 0xffdf;         /* Disable module(SCI0) stop mode */
        SMR0 = 0;                /* Initialize SMR0 */
        BRR0 = 64;               /* Set 9600bps */
        SCR0 = 0x50;             /* Enable RXI interrupt */
        P3DDR = 0xC9;            /* RTS(P33):output port CTS(P31):input port */
        P3DR = 0xFF;

        set_imask_ccr(0);        /* Enable interrupt */
        P3DR_BP.P33DR=0;         /* Output RTS low */

        while (rxendf == 0);     /* Receive complete ? */
        rxendf = 0;
        TDR0 = rcv_data;         /* Set transmit data (receive data) */
        P3DR_BP. P33DR=1;        /* Output RTS high */

        while (PORT3_BP. P31== 1);

        SCR0_BP. TE0 = 1;        /* Enable transmit */
        SSR0_BP. TDRE0 = 0;      /* Start transmit */
        SCR0_BP. TEIE0 = 1;      /* Enable TEI interrupt */

        while (txendf==0);       /* Transmit complete? */
        txendf = 0;

        while(1);                /* Loop */
        }
```

```
/*********************************/
/*   INTERRUPT PROGRAM: ASCRX     */
/*********************************/
void ASCRX(void)
        {
        SSR0_BP. RDRF0 = 0;      /* Clear RDRF */
        rcv_data = RDR0;         /* Get receive data from RDR0 */
        rxendf = 1;              /* Set rxendf */
        }

/*********************************/
/*   INTERRUPT PROGRAM: ASCTE     */
/*********************************/
void ASCTE(void)
        {
        SSR0_BP. TDRE0 = 0;      /* TEND clear */
        txendf = 1;              /* Set txendf */
        SCR0_BP. TE0 = 0;        /* Disable transmit */
        SCR0_BP. TEIE0 = 0;      /* Disable TEI interrupt */
        }
```

Specifications

1. One-byte data is transferred between an H8S/2655 chip and H8/3314 chip, as shown in figure 1.

2. The clock synchronous communication format is used for data transmission/reception. The master H8S/2655 supplies the clock to the slave H8/3314.

3. The H8S/2655 transmits data to the H8/3314 and receives data from the H8/3314 simultaneously.

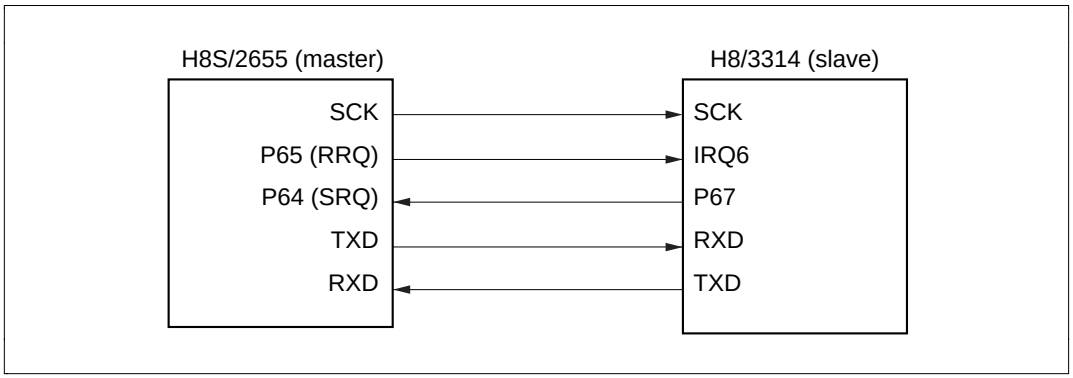


Figure 1 H8S/2655 Clock Synchronous SCI Block Diagram

Functions Used

1. In this sample task, data transmission/reception is performed using SCI1. Port 6 is used for the communication control pins (RRQ, SRQ).

   a. Figure 2 shows the SCI transmission block diagram for this sample task. Simultaneous transmit and receive operations are performed using the following functions:

   - A function that performs serial data communication in synchronization with a clock (clock synchronous mode)
   - A function that performs transmission and reception simultaneously (simultaneous transmit/receive operation)
   - A function that generates an interrupt when reception is completed (RXI interrupt)

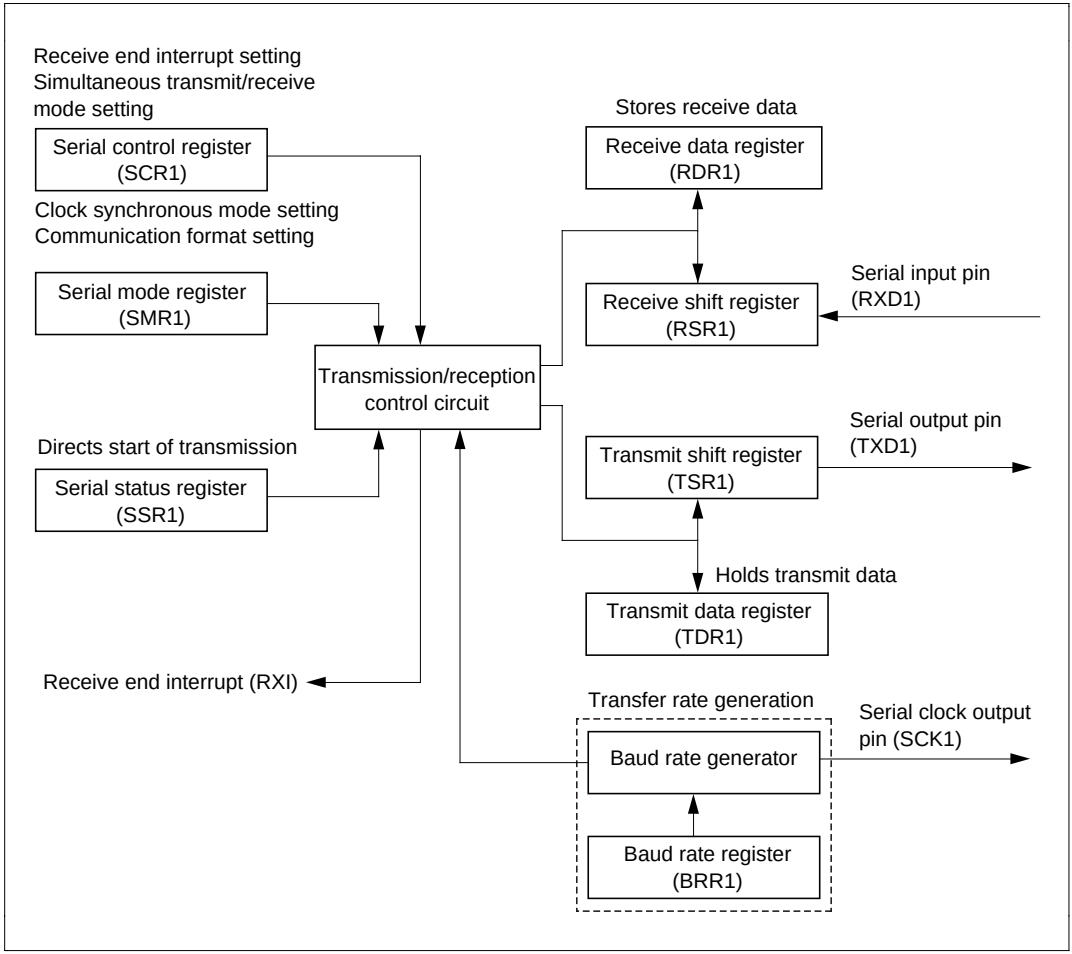


Figure 2 SCI Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform simultaneous data transmission to and reception from an H8/3314 chip.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
SCK1	Transmits serial clock
RXD1	Receives data from H8/3314
TXD1	Transmits data to H8/3314
SMR1	Sets SCI to clock synchronous mode
SCR1	Enables receive interrupt and sets SCI to transmit/receive mode
SSR1	Directs start of transmission with TDRE bit
RDR1	Holds data received from H8/3314
TDR1	Holds data to be transmitted to H8/3314
BRR1	Transfer rate setting
P6DDR	Sets port 6 input/output
P6DR	Performs RRQ transmission and SRQ reception

Operation

Figure 3 shows the principles of the operation performed by this task. Interfacing to an H8/3314 is performed by means of hardware and software processing, using the timing shown in this figure.

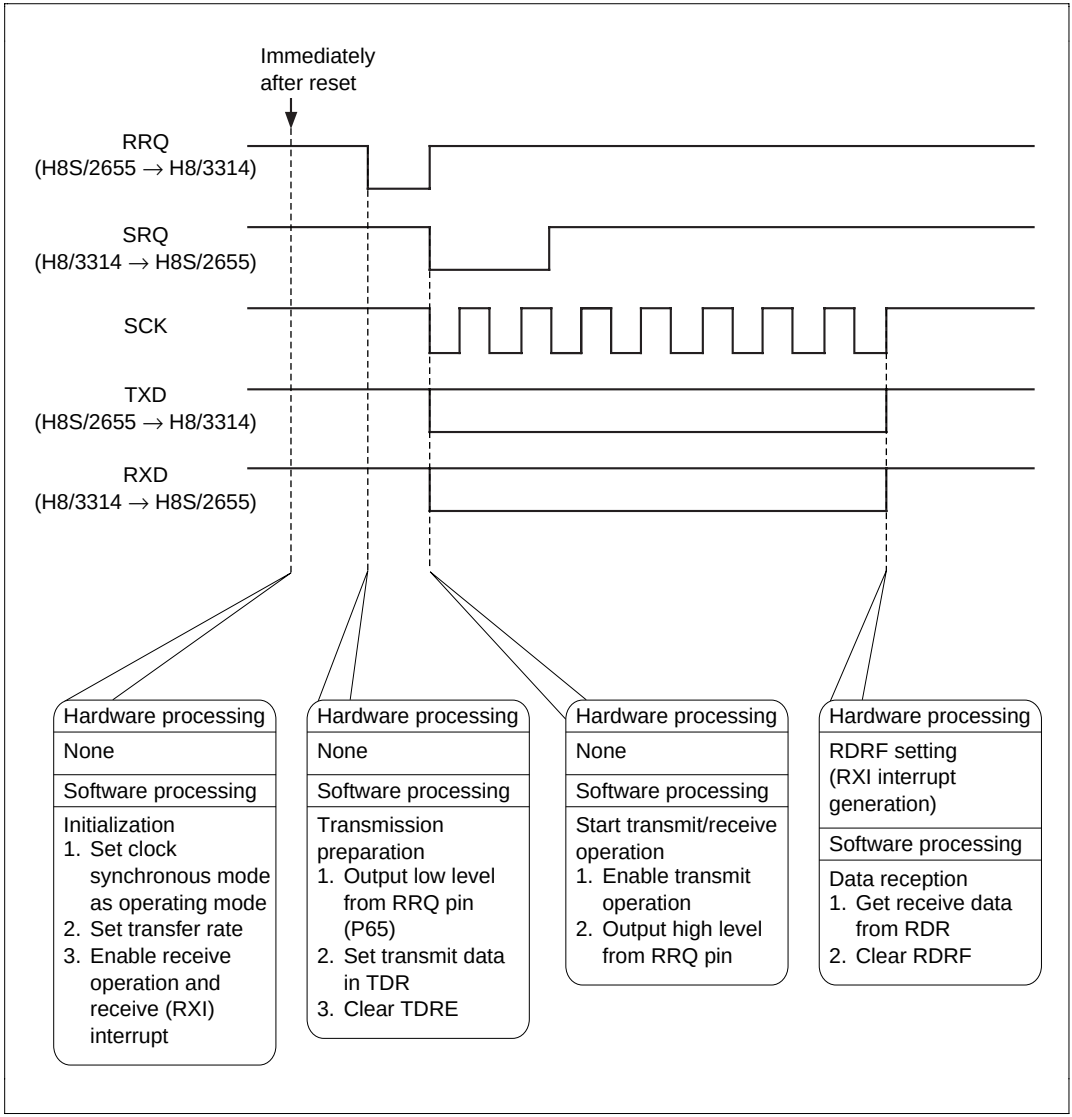


Figure 3 Principles of Simultaneous Transmit/Receive Operation

Software

1. Modules

Module Name	Label	Function
Main routine	simtrmn	SCI initialization, transmission/reception management
Data receive end	rxend	Initiated by RXI interrupt; performs data reception

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
revend	Flag indicating completion of reception 1: Reception completed 0: Reception in progress	Unsigned char	Data receive end	Output
			Main routine	Input

3. Internal Registers Used

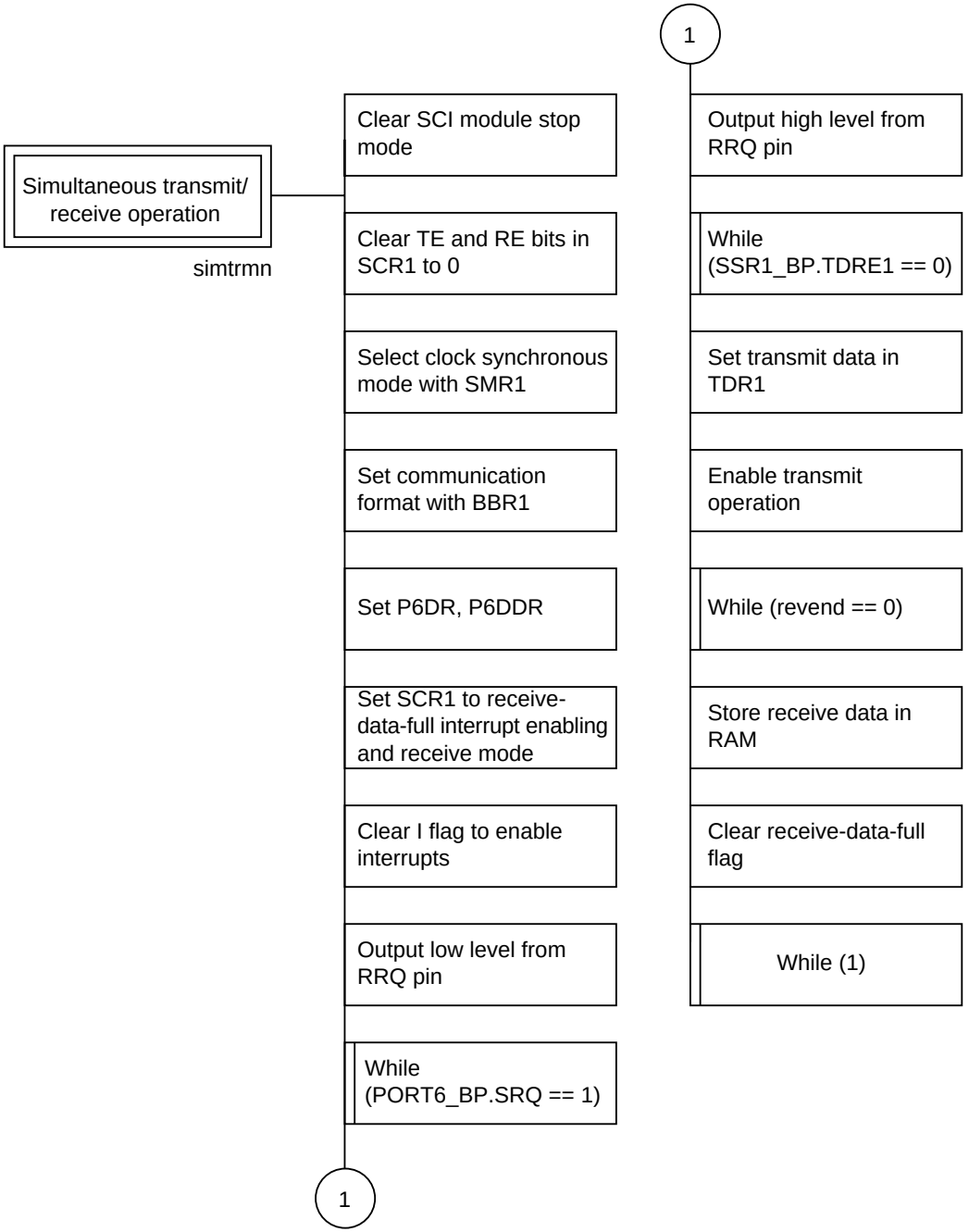
Register Name	Function	Module
SMR1	Sets SCI mode (clock synchronous) and communication format, and selects clock for baud rate generator (ø clock input)	Main routine
SCR1	Enables interrupt (RXI) and sets SCI transmission/ reception enabling	Main routine
SSR1	Directs start of transmission by clearing TDRE	Main routine
RDR1	Holds data received from H8/3314	Data receive end
TDR1	Holds data to be transmitted to H8/3314	Main routine
BRR1	Transfer rate setting	Main routine
P6DDR	Port 6 input/output setting	Main routine
P6DR	Performs RRQ and SRQ pin manipulation	Main routine
MSTPCR	Clears SCI module stop mode	Main routine

4. RAM Used

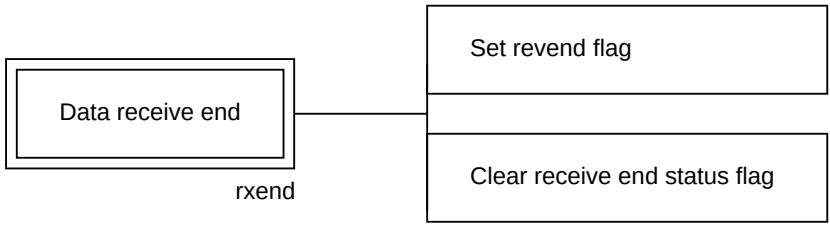
Label/Register Name	Function	Data Length	Module
rvdata	Holds received data	Unsigned char	Main routine
trdata	Holds data to be transmitted	Unsigned char	Main routine

PAD

1. Main routine



2. Data receive end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTCOL                    */
/****************************************/
void simtrmn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
#define trdata (*(unsigned char *)0xffec00)
#define rvdata (*(unsigned char *)0xffec01)
#define revend (*(unsigned char *)0xffec02)

/****************************************/
/*     MAIN PROGRAM : simtrmn           */
/****************************************/
void simtrmn(void)
{
        MSTPCR = 0xffbf;
        SCR1 = 0x00;                    /* select clock mode */
        SMR1 = 0x80;                    /* init SCI1 */
        BRR1 = 0x04;                    /* set 1MBPS */
        P6DDR = 0x20;                   /* init port */
        P6DR = 0x20;
        SCR1 = 0x70;                    /* enable RX,TX,RIE */
        set_imask_ccr(0);

        P6DR_BP.RRQ = 0;                /* "Low"-> RRQ */

        while(PORT6_BP.SRQ == 1);

        P6DR_BP.RRQ = 1;                /* "High"-> RRQ */
        while(SSR1_BP.TDRE1 == 0);
        TDR1 = trdata;                  /* set transmit data to TDR */
        SSR1 &= 0x7f;                   /* start transmit */

        while(revend == 0);             /* receive complete? */

        rvdata = RDR1;                  /* set receive data */
        SSR1 &= 0xbf;

        while(1);
}

/****************************************/
/*    NAME : rxend(data receive end)    */
/****************************************/
#pragma interrupt(rxend)
void rxend(void)
{
        revend = 0x01;
        SSR1 &= 0xbf;
}
```

Specifications

1. Data transmission and reception is performed between an H8S/2655 chip and two H8/3314 chips using a common serial communication line, as shown in figure 1.

2. When the H8S/2655 is the transmitting device, data is transmitted to the H8/3314s, and each H8/3314 receives only data for that device. When the H8S/2655 is the receiving device, it compares ID data with its own ID and receives data if the IDs match.

3. The format for data transmission/reception is 9600 bps, 8-bit data, one stop bit, and no parity.

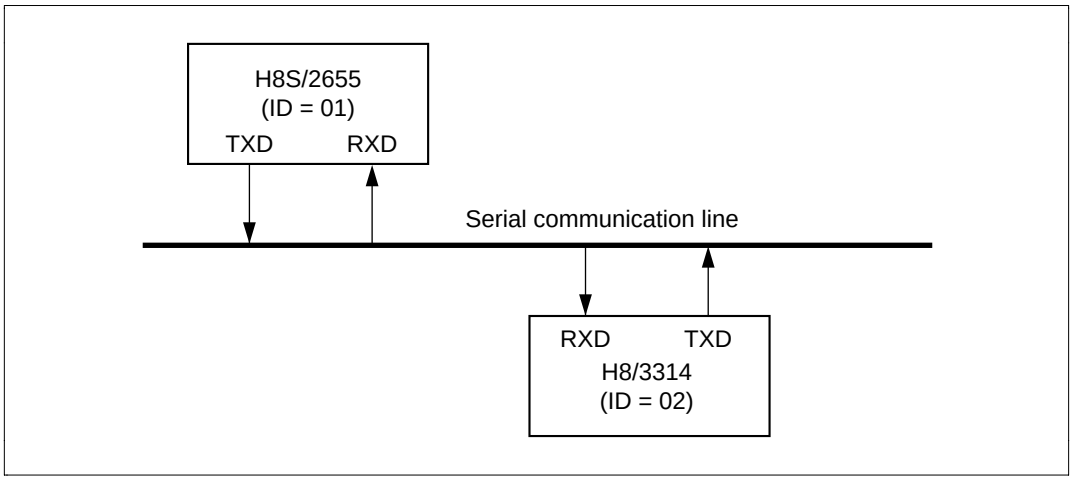


Figure 1 Block Diagram of Asynchronous SCI Using Multiprocessor Function

Functions Used

1. In this sample task, multiprocessor communication is performed using the SCI's multiprocessor communication function.

   a. Figure 2 shows the transmitting device SCI block diagram for this sample task. Transmission is performed using the following SCI functions:

   - A function that performs data communication asynchronously, establishing synchronization character by character (asynchronous mode)
   - A function that performs communication using a format that includes a multiprocessor bit (multiprocessor communication function)
   - A function that generates an interrupt at the start of transmission (TXI interrupt)

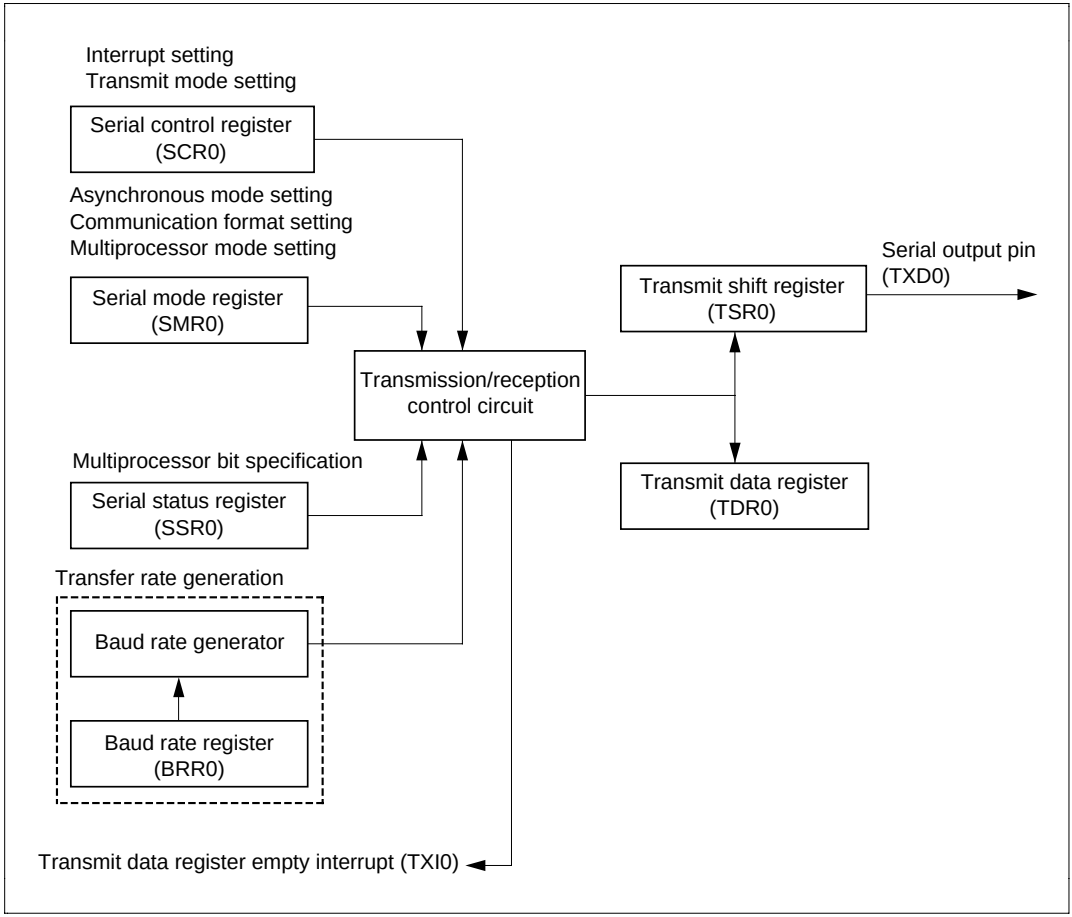


Figure 2 Transmitting Device SCI Block Diagram

b. Figure 3 shows the receiving device SCI block diagram for this sample task. Reception is performed using the following SCI functions:

- A function that performs data communication asynchronously, establishing synchronization character by character (asynchronous mode)
- A function that performs communication using a format that includes a multiprocessor bit (multiprocessor communication function)
- A function that generates an interrupt when a multiprocessor bit is received (multiprocessor interrupt)
- A function that generates an interrupt when reception is completed (RXI interrupt)

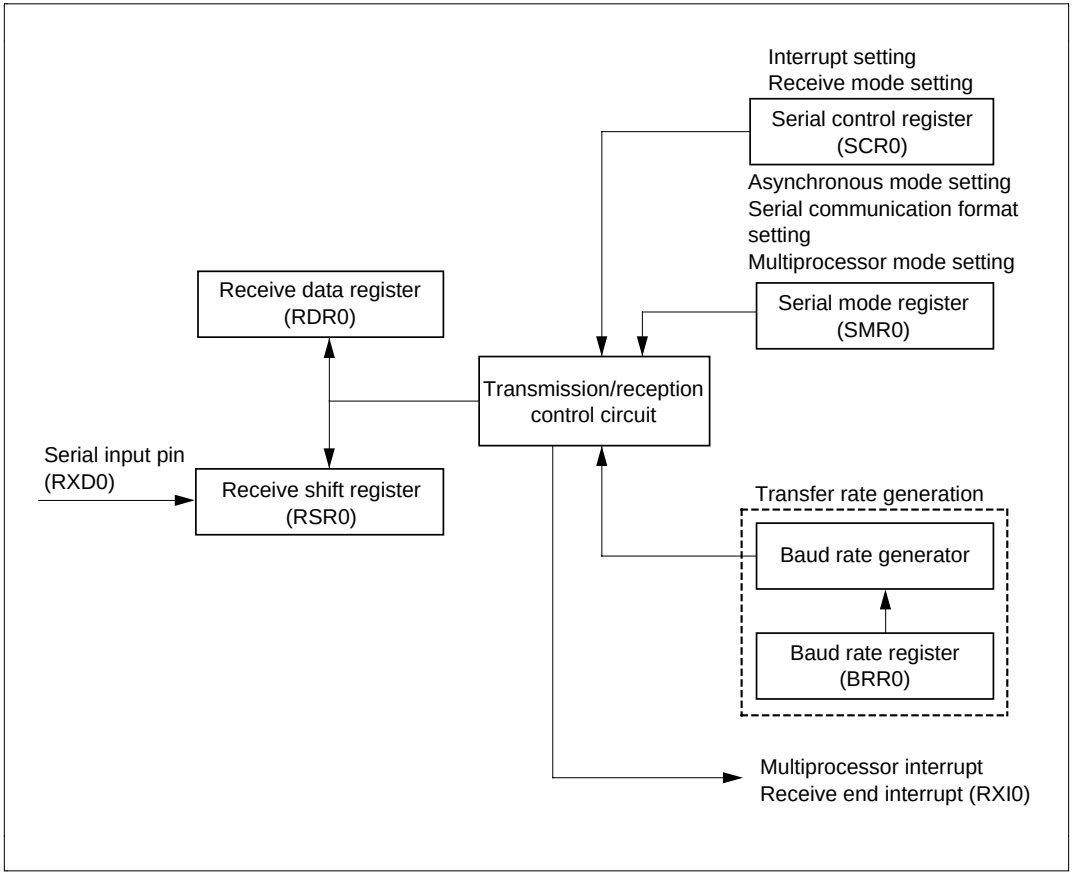


Figure 3 Receiving Device SCI Block Diagram

2. Table 1 shows the function assignments for this sample task. SCI functions are assigned as shown in this table to perform multiprocessor communication.

Table 1 SCI Function Assignments

SCI Functions	Function
RXD0	Receives data from H8/3314
TXD0	Transmits data to H8/3314
SMR0	Sets SCI to asynchronous mode and multiprocessor mode
SCR0	Enables transmit/receive interrupts and sets SCI to transmit/receive mode
SSR0	Starts transmission/sets multiprocessor bit
RDR0	Holds data received from H8/3314
TDR0	Holds data to be transmitted to H8/3314
BRR0	Transfer rate setting

Operation

1. Transmission

Figure 4 shows the principles of the transmit operation performed by this task. Data is transmitted to the receiving H8/3314 by means of hardware and software processing, using the timing shown in this figure.

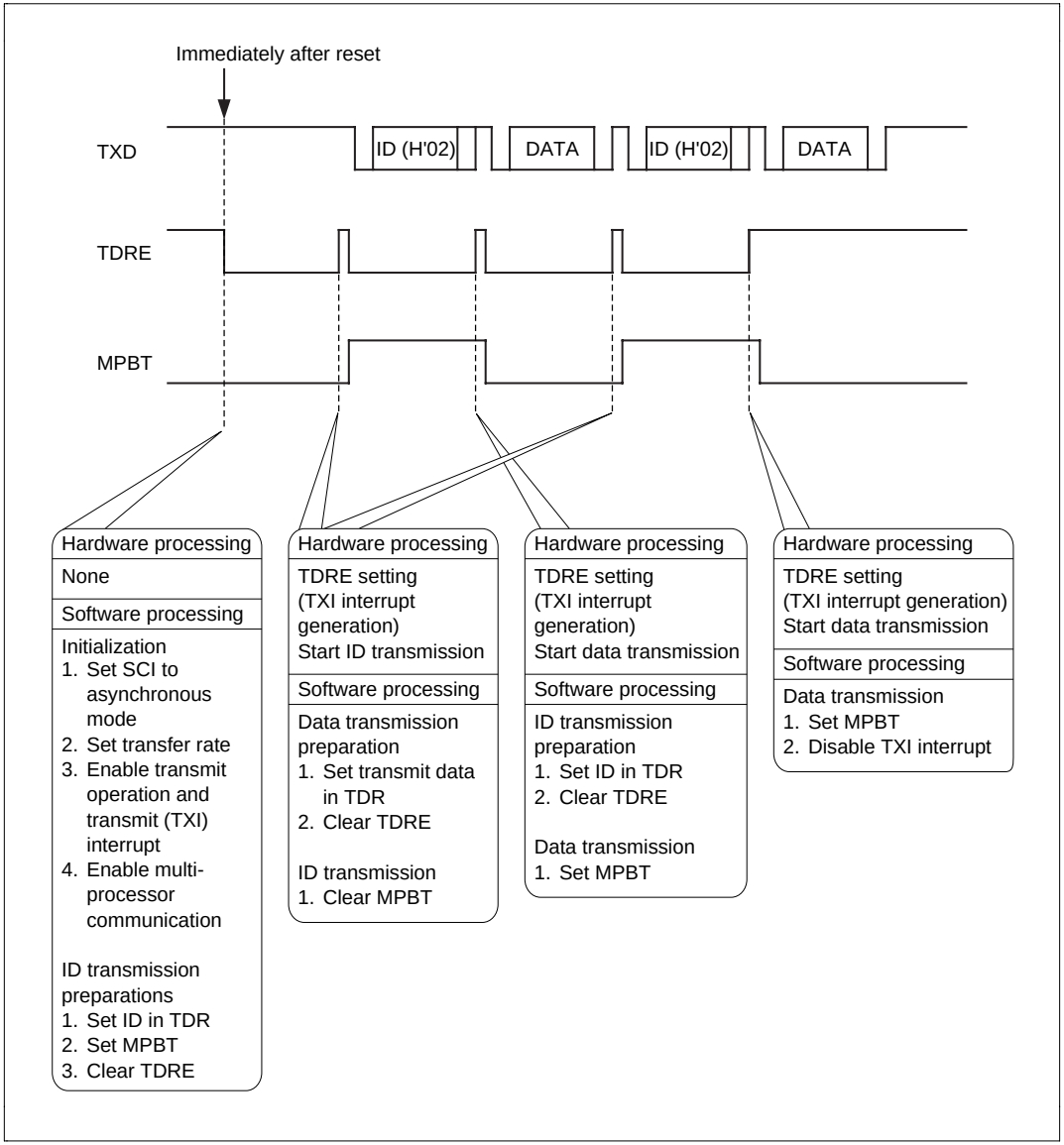


Figure 4 Principles of Multiprocessor Communication (Transmitting Device) Operation

2. Reception

Figure 5 shows the principles of the receive operation performed by this task. Data is received from the transmitting device by means of hardware and software processing, using the timing shown in this figure.

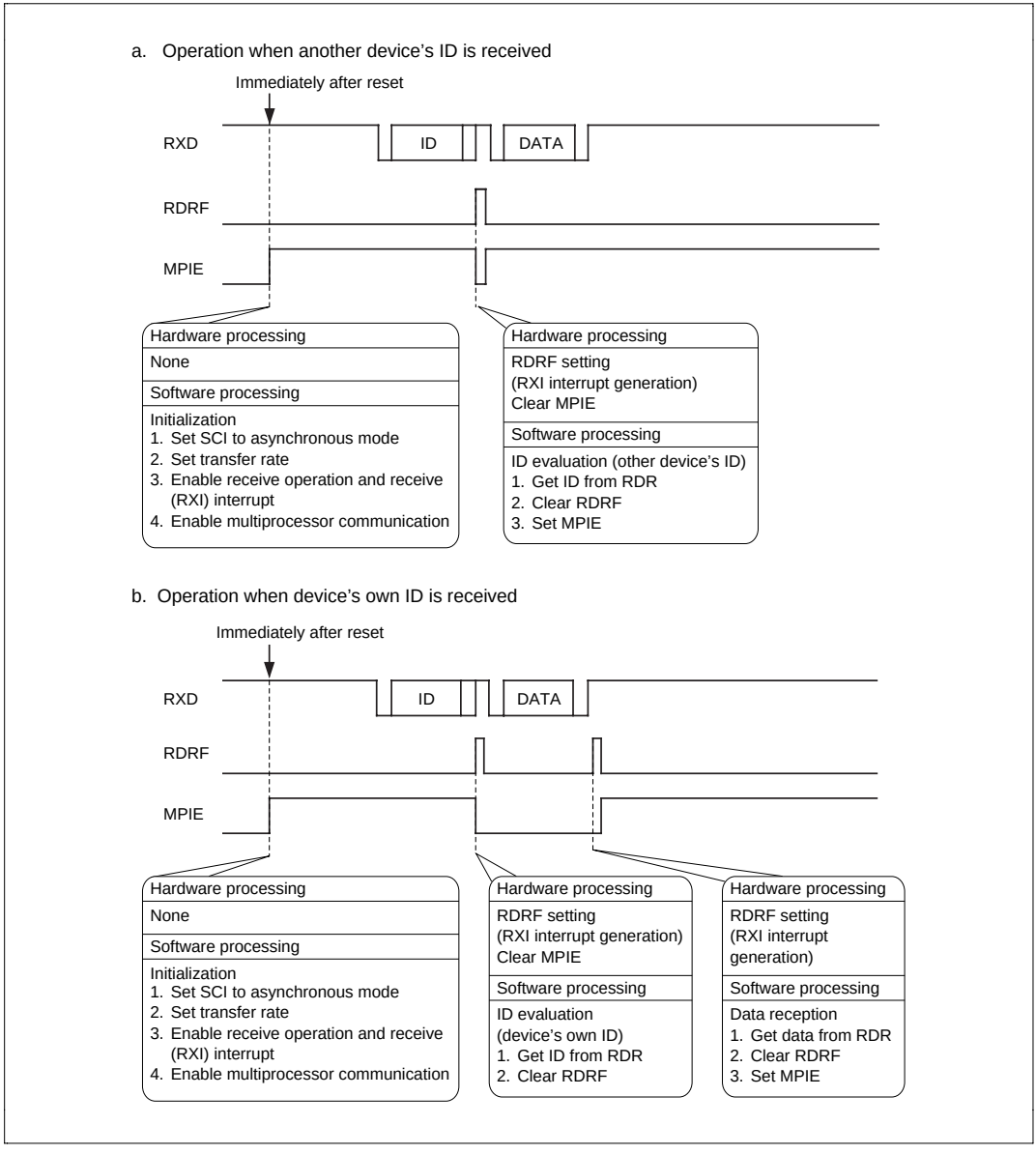


Figure 5 Principles of Multiprocessor Communication (Receiving Device) Operation

Software

1. Transmitting device software

a. Modules

Module Name	Label	Function
Main routine	MPMASMN	SCI initialization
Data transmission	MPSCITX	Initiated by TXI interrupt; performs ID and data transmission

b. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
txdata	Buffer that holds ID or data to be transmitted to receiving H8/3314	Unsigned char	Main routine	Output
			Data transmission	Input
txendf	Indicates end of transmission	Unsigned char	Main routine	Input
	1: End of transmission 0: Transmission in progress		Data transmission	Output

c. Internal Registers Used

Register Name	Function	Module
SMR0	Sets SCI mode (asynchronous) and communication format, and selects clock for baud rate generator (ø clock input)	Main routine
SCR0	Enables interrupt (TXI) and sets SCI transmission enabling/disabling	Main routine
		Transmit end
SSR0	Directs start of transmission by clearing TDRE (b7)	Main routine
		Data transmission
TDR0	Holds ID or data to be transmitted to receiving H8/3314	Main routine
		Data transmission
BRR0	Transfer rate setting	Main routine
MSTPCR	Clears SCI module stop mode	Main routine

d. RAM Used

Label	Module	Data Length	Function
txcnt	Data transmission	Unsigned char	Count of transmitted data

2. Receiving device software

a. Modules

Module Name	Label	Function
Main routine	MPSRVMN	SCI initialization
Data reception	MPSCIRX	Initiated by RXI interrupt; performs ID and data reception

b. Arguments

Label	Function	Data Length	Module	Input/ Output
rcv_data	Holds received ID or data	Unsigned char	Data reception	Output
			Main routine	Input
idrcvf	Flag indicating reception of device's own ID	Unsigned char	Data reception	Output
	1: ID received 0: ID not received		Main routine	Input
dtrcvf	Flag indicating data reception	Unsigned char	Data reception	Output
	1: Data received 0: Data not received		Main routine	Input

c. Internal Registers Used

Register Name	Function	Module
SMR0	Sets SCI mode (asynchronous) and communication format, and selects clock for baud rate generator (ø clock input)	Main routine
SCR0	Enables interrupt (RXI) and sets SCI reception enabling	Main routine
RDR0	Holds ID or data received from transmitting H8/3314	Data reception
BRR0	Transfer rate setting	Main routine
MSTPCR	Clears SCI module stop mode	Main routine

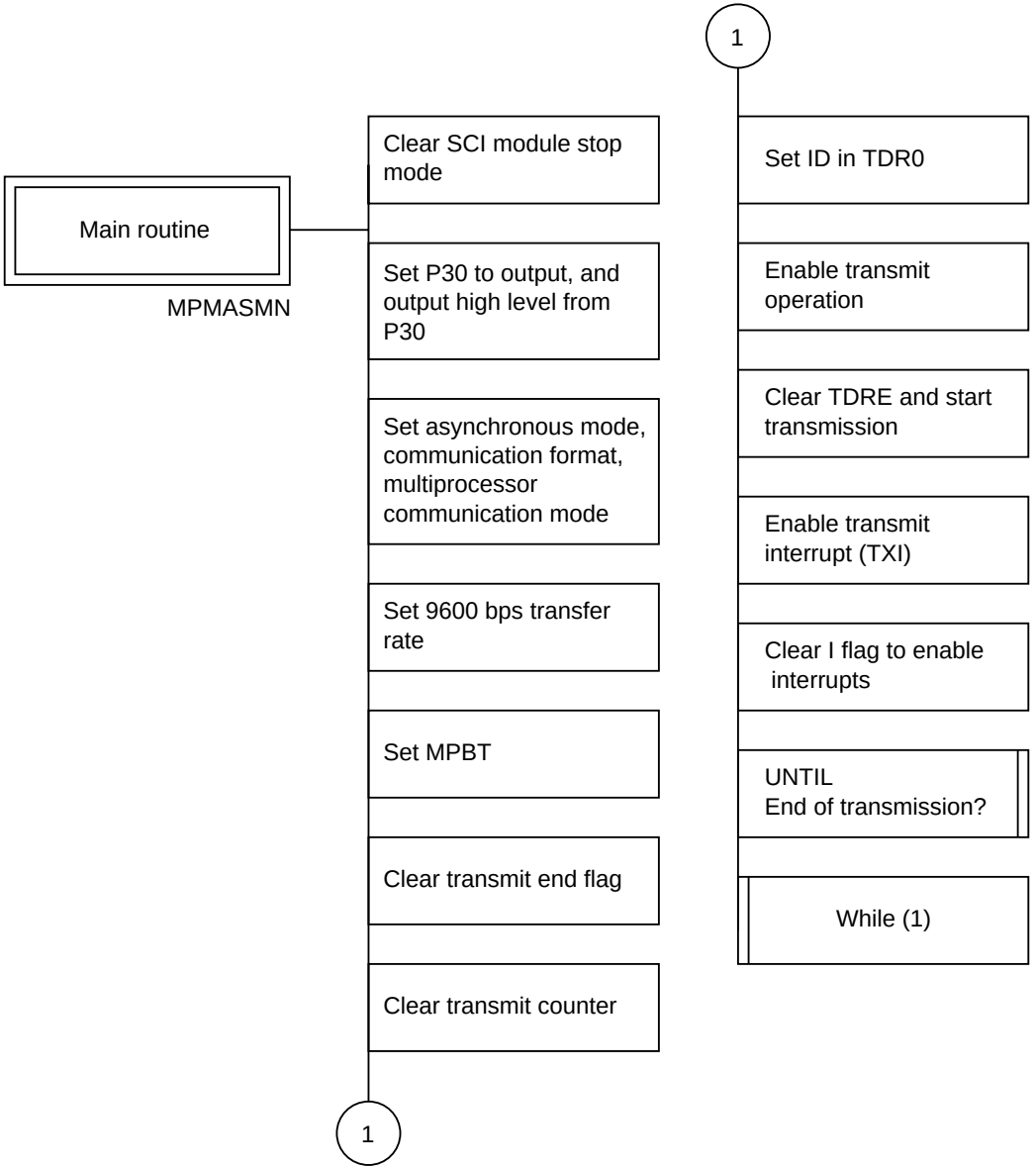
d. RAM Used

Label	Module	Data Length	Function
rxid	Main routine	Unsigned char	Holds received ID
rxdata	Main routine	Unsigned char	Holds received data
myid	Data reception	Unsigned char	Holds device’s own ID

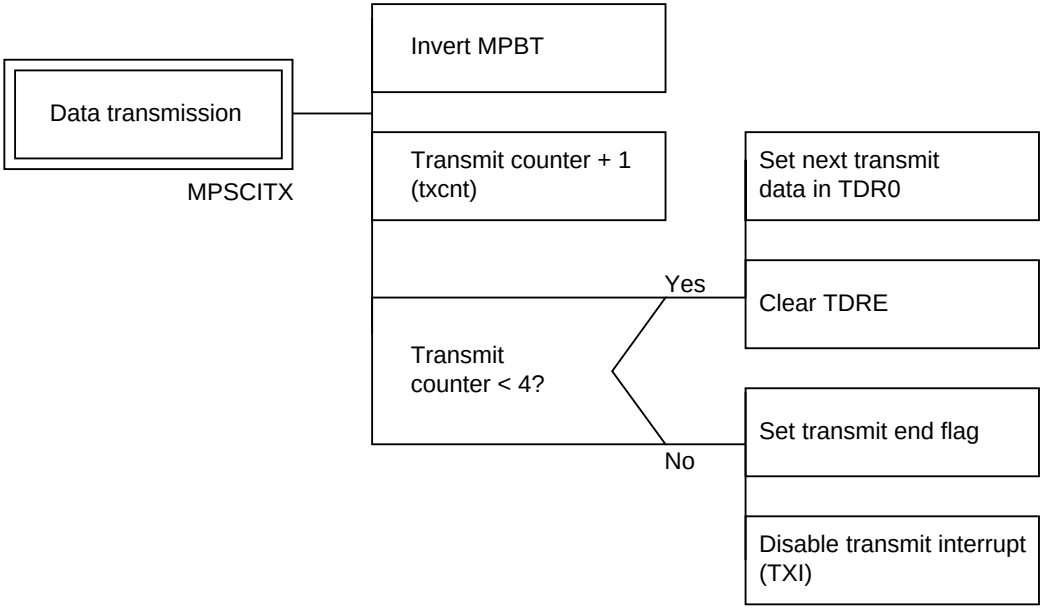
PAD

1. Transmitting device

   a. Main routine

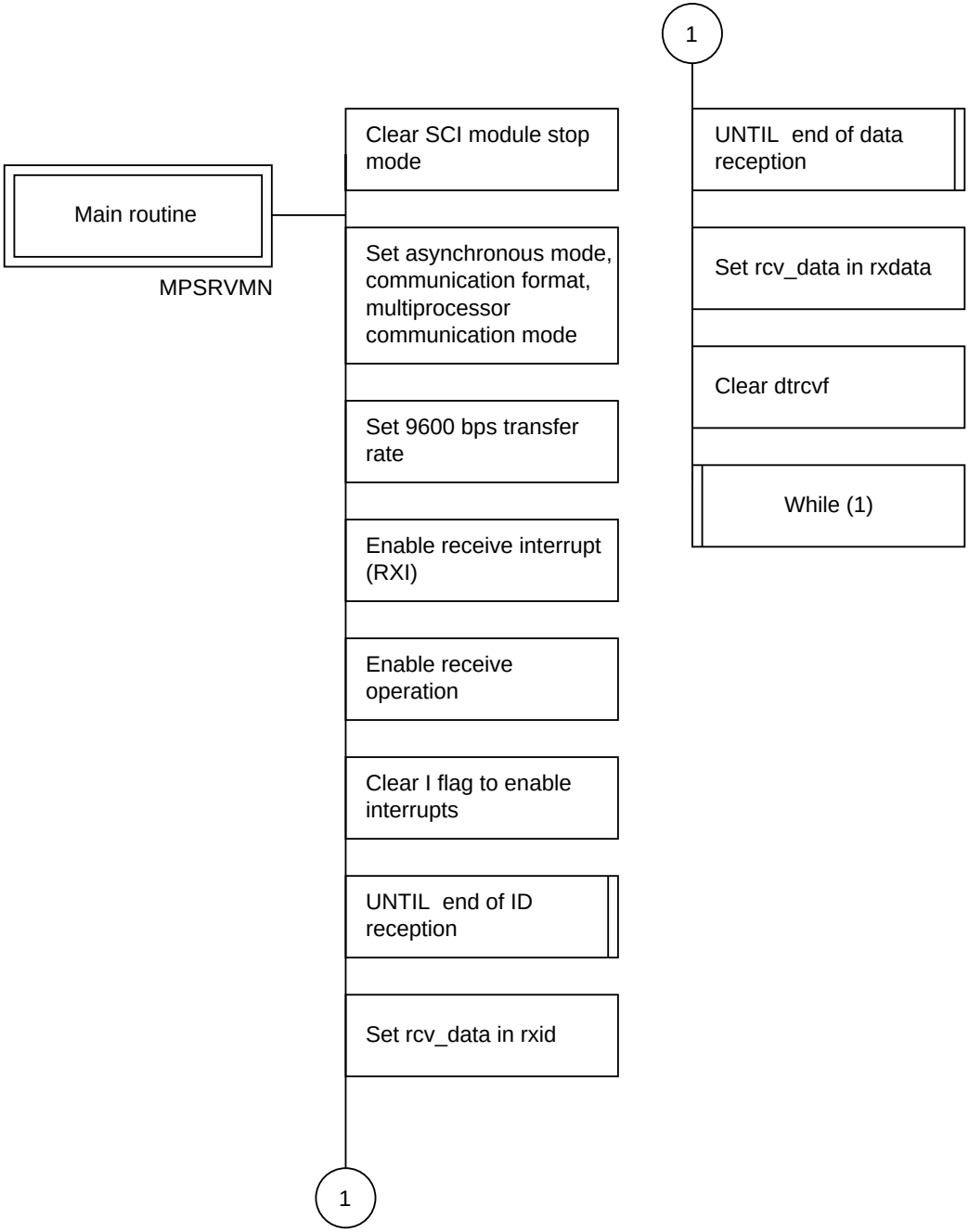


b. Data transmission

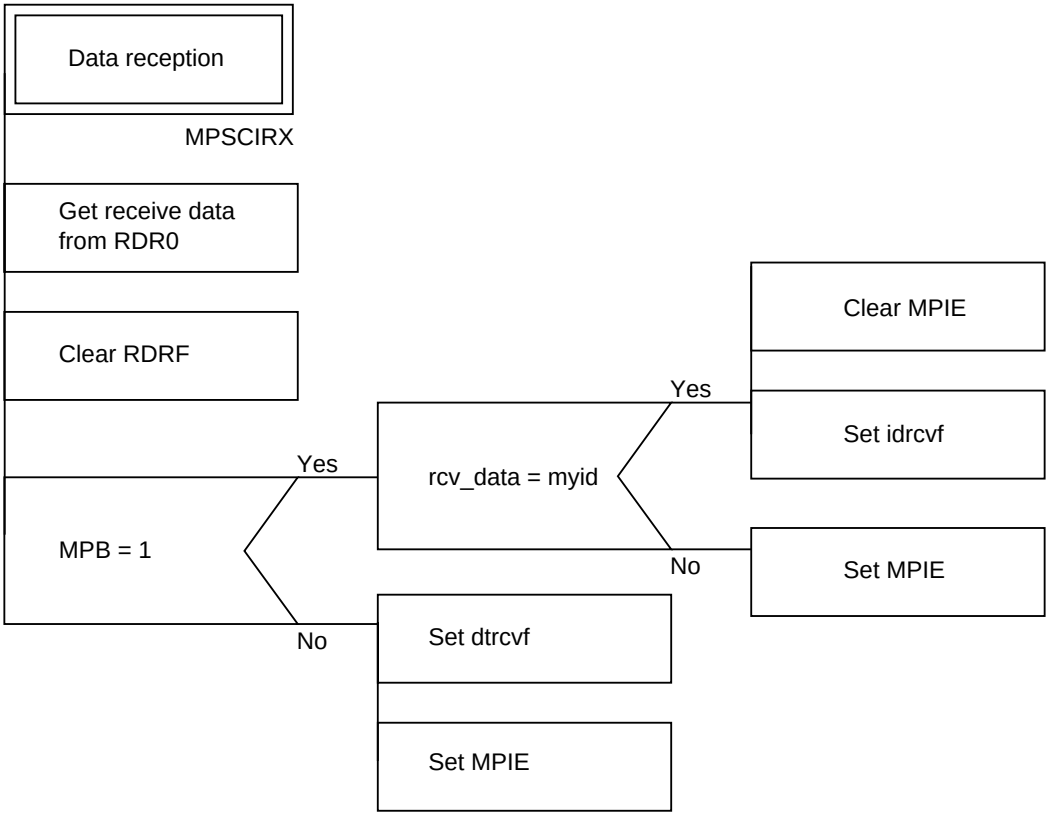


2. Receiving device

a. Main routine



b. Data reception



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void MPMASMN(void);
#pragma interrupt (MPSCITX)

/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/
# define txdata  ((unsigned char * )0xffec00)  /* Transmit data */
# define txcnt   (*(unsigned char * )0xffec04) /* Transmit counter */
# define txendf  (*(unsigned char * )0xffec05) /* Transmit end flag */


/*********************************/
/*     MAIN PROGRAM: MPAMASMN    */
/*********************************/
void MPMASMN(void)
{

        MSTPCR = 0xffdf;         /* Disable module(SCI0) stop mode*/
        P3DDR_BP.P30DDR = 1;     /* P30 high output */
        P3DR_BP.P30DR = 1;
        SMR0 = 0x04;             /* Set multi processor mode */
        BRR0 = 64;               /* Set 9600bps */
        SSR0_BP.MPBT0=1;         /* Set MPBT0 */
        txendf = 0;              /* Set txendf=0 */
        txcnt = 0;               /* Set txcnt=0 */
        TDR0 = txdata[txcnt];    /* Set ID(H'01) to TDR0 */
        SCR0_BP. TE0 = 1;        /* Enable transmit */
        SSR0_BP.TDRE0=0;         /* Start transmit */
        SCR0_BP.TIE0=1;          /* Enable TXI */
        set_imask_ccr(0);        /* Enable interrupt */

        while (txendf == 0);

        while (1);
}


/*********************************/
/*   INTERRUPT PROGRAM: MPSCITX  */
/*********************************/
void MPSCITX(void)
{
       SSR0_BP. MPBT0 = -SSR0_BP. MPBT0;/* Invert MPBT0 */
       txcnt = txcnt + 1;

       if (txcnt<4)
              {
              TDR0 = txdata[txcnt];    /* Load next tans data */
              SSR0_BP.TDRE0=0;         /* Start transmit */
```

```
                }
                else
                {
                txendf = 1;
                SCR0_BP. TIE0=0;          /* Disable TXI */
                }
}
```

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void MPSRVMN(void);
#pragma interrupt (MPSCIRX)
/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/
# define rcv_data   (*(unsigned char * )0xffec00)  /* Receive ID,data */
# define idrcvf     (*(unsigned char * )0xffec01)  /* ID code compare flag */
# define dtrcvf     (*(unsigned char * )0xffec02)  /* Data receive flag */
# define rxid       (*(unsigned char * )0xffec03)  /* Receive ID code */
# define rxdata     (*(unsigned char * )0xffec04)  /* Receive data */
# define myid       (*(unsigned char * )0xffec05)  /* My ID code */
/*********************************/
/*     MAIN PROGRAM: MPSRVMN     */
/*********************************/
void MPSRVMN(void)
{
        MSTPCR = 0xffdf;         /* Disable module(SCI0) stop mode*/
        SMR0 = 0x04;             /* Set multi processor mode */
        BRR0 = 64;               /* Set 9600bps */
        SCR0_BP.MPIE0=1;         /* Enable multiprocessor com. */
        SCR0_BP.RIE0=1;          /* Enable RXI */
        SCR0_BP.RE0=1;           /* Enable receive */
        set_imask_ccr(0);        /* Enable interrupt */

        while (1)
        {
        while (idrcvf == 0);     /* Receive ID ? */
        rxid = rcv_data;         /* Store ID code */
        idrcvf = 0;              /* Clear idrcvf */

        while (dtrcvf == 0);     /* Data receive? */
        rxdata = rcv_data;       /* Store data */
        dtrcvf = 0;              /* Clear dtrcvf */
        }
}
/*********************************/
/*   INTERRUPT PROGRAM: MPSCIRX  */
/*********************************/
void MPSCIRX(void)
{
        rcv_data =RDR0;                  /* Store receive data */
        SSR0_BP. RDRF0 = 0;              /* Clear RDRF */

        if (SSR0_BP.MPB0 == 1)           /* MPB0 = 1 */
        {
            if (rcv_data == myid )       /* Receive ID = my ID */
            {
            SCR0_BP.MPIE0 =0;            /* Clear MPIE0 */
            idrcvf = 1;                  /* Set idrcvf*/
            }
            else
```

```
            SCR0_BP.MPIE0=1;            /* Set MPIE0 */

        }
        else
        {
        dtrcvf = 1;                     /* Set dtrcvf */
        SCR0_BP.MPIE0=1;                /* Set MPEIE0 */
        }
}
```

Specifications

1. Voltages are input to the H8S/2655 on four channels for A/D conversion, as shown in figure 1, and the results are stored in RAM.

2. The A/D converter is activated by an external trigger.

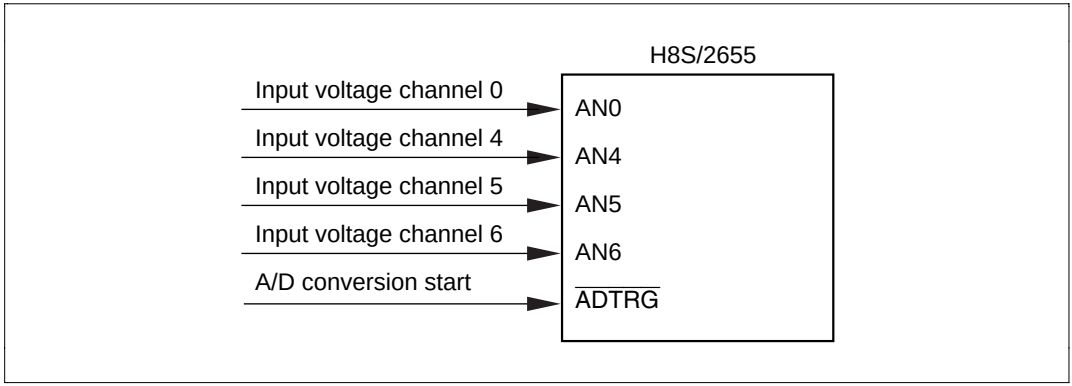


Figure 1 Block Diagram of Voltage Measurement by H8S/2655

Functions Used

1. Figure 2 shows the block diagram for 4-channel A/D conversion. This sample task uses the following A/D converter functions:

   a. A function that performs A/D conversion on four channels (AN0, AN4, AN5, and AN6) automatically without software intervention (group scan mode)

   b. A function that transfers the conversion result to a separate ADDR on completion of conversion for a particular channel (buffered operation)

   c. Initiation of A/D conversion via the external trigger pin (ADTRG)

   d. A function that generates an interrupt at the end of A/D conversion

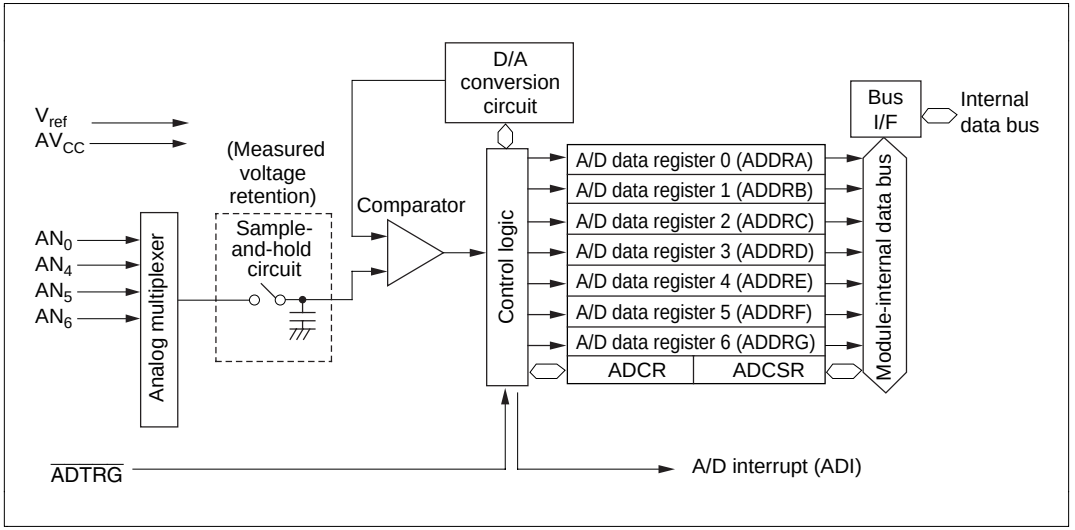


Figure 2 A/D Converter Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform A/D conversion.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
ADCSR	A/D conversion channel selection (group) and status indication
ADCR	Start trigger signal selection and operating mode (scan) setting
ADDRA to ADDRG	Store A/D conversion results
$\overline{\text{ADTRG}}$	A/D external trigger input pin

Operation

Figure 3 shows the principles of the operation. As shown in this figure, the A/D converter is started by external trigger $\overline{\text{ADTRG}}$, and performs A/D conversion repeatedly on four channels, AN0, AN4, AN5, and AN6. The ADST bit remains at 1 until cleared to 0 by software, and while this bit is 1, A/D conversion is performed repeatedly on the selected input channels. Buffered operation (with one 4-stage buffer) is used for this purpose. The A/D conversion results held in ADDRA to ADDRG are stored a 140-byte RAM area comprising SCN0 to SCN6.

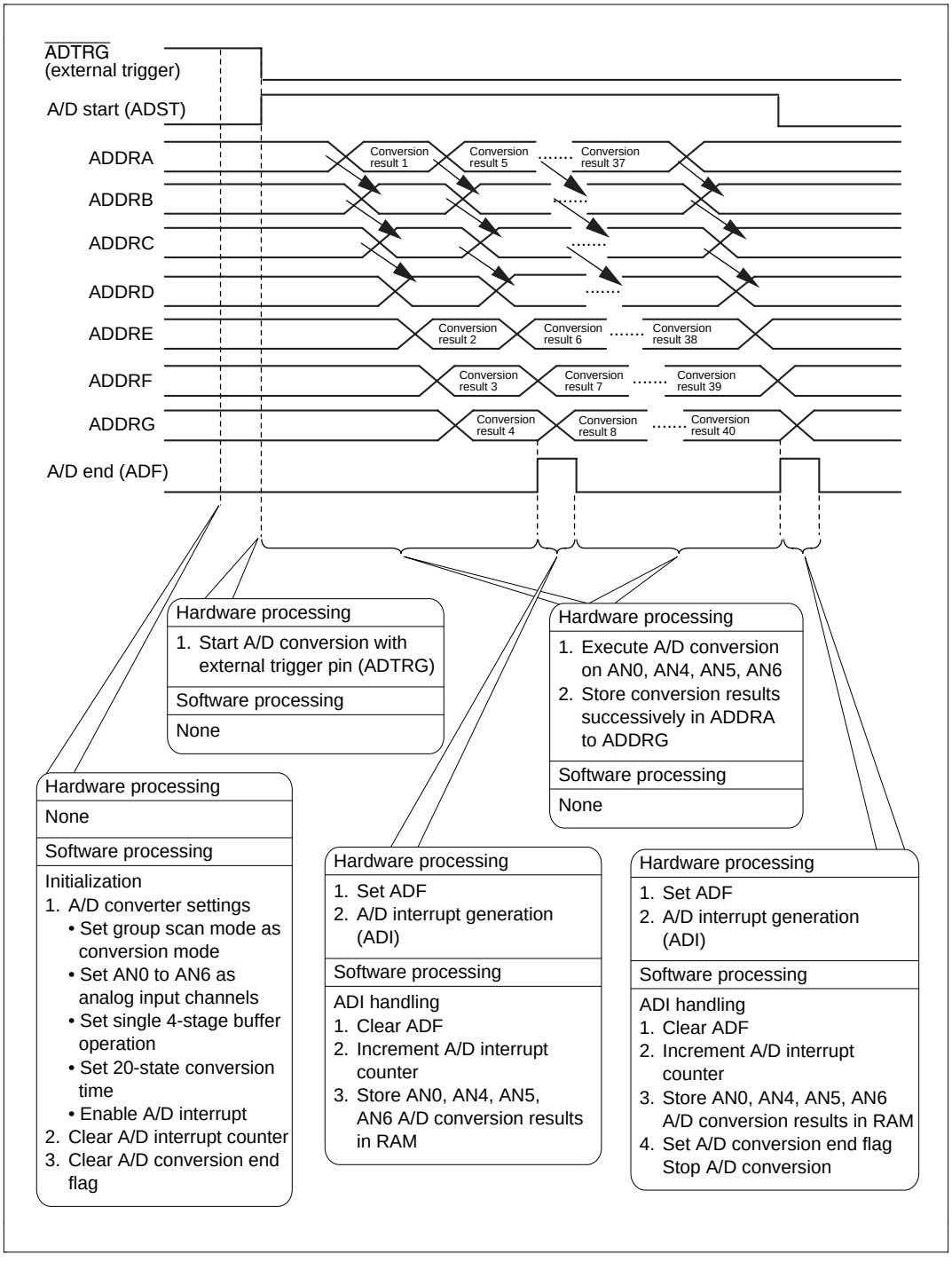


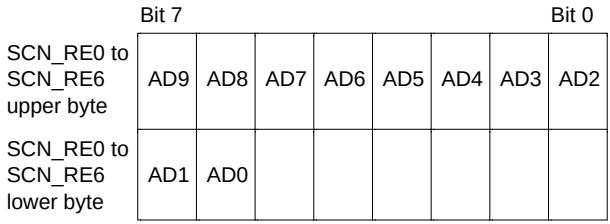
Figure 3 Principles of Scan Mode A/D Conversion Operation

Software

1. Modules

Module Name	Label	Function
Main routine	ADSCNMN	A/D converter and externally triggered A/D converter activation settings
A/D interrupt	SCNEND	Initiated by ADI; stores A/D conversion results in RAM and stops A/D conversion

2. Arguments

Label/ Register Name	Function	Data Length	Module	Input/ Output
scn	Holds 4-channel A/D conversion results 10-bit conversion result is stored as follows:  	Unsigned short	A/D interrupt	Output
scn_endf	Flag indicating that 4-channel A/D conversion has all ended 1: End of A/D conversion 0: A/D conversion in progress	Unsigned short	A/D interrupt Main routine	Output Input

3. Internal Registers Used

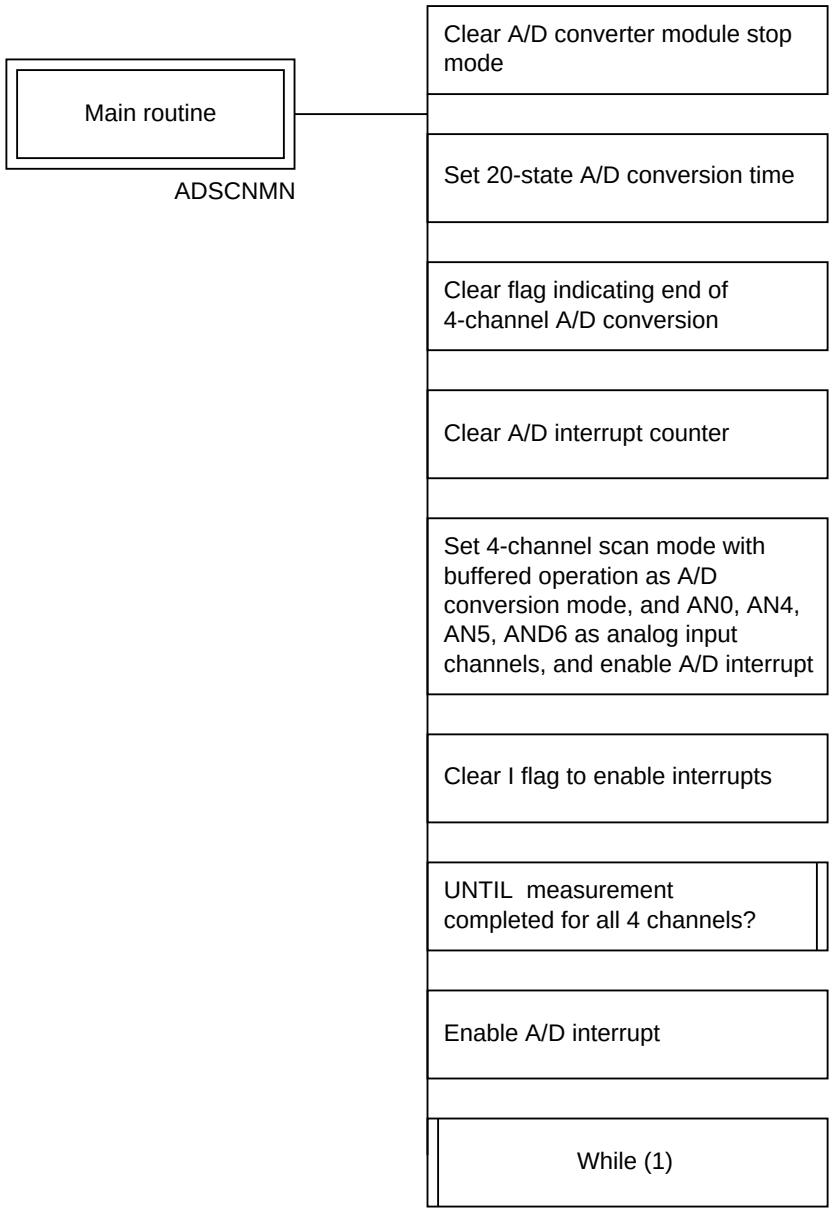
Register Name	Function	Module
ADCSR	Selection of A/D conversion time, analog input channels, and enabling/disabling of A/D interrupt at end of A/D conversion	Main routine A/D interrupt
ADCR	Selection of A/D conversion mode (scan mode) and buffered operation	Main routine
ADDR0 to ADDRG	Store A/D conversion results	A/D interrupt
MSTPCR	Clears A/D converter module stop mode	

4. RAM Used

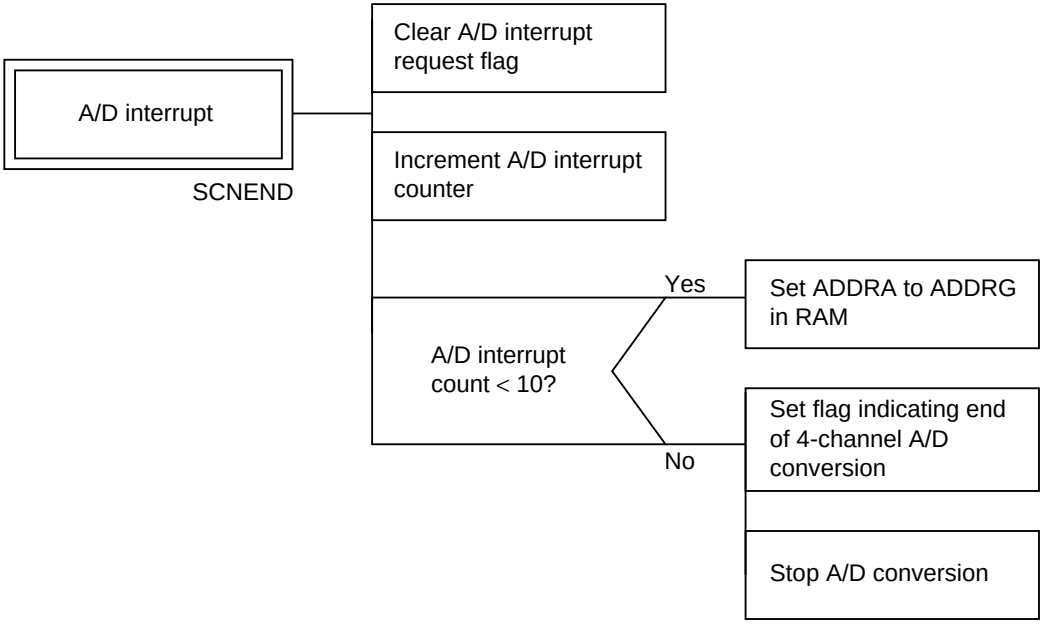
Module	Label	Function
A/D interrupt	adicnt	Counts A/D interrupts
A/D interrupt	scn_cnt	Counter for storing data in RAM from start address

PAD

1. Main routine



2. A/D interrupt



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void ADSCNMN(void);
#pragma interrupt (SCNEND)

/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/

# define scn       ((unsigned short * )0xffec00) /* Result of A/D conversion */
# define scn_cnt   (*(unsigned char * )0xffec8c) /* Work */
# define scn_endf  (*(unsigned char * )0xffec8d) /* A/D conversion end flag */
# define adicnt    (*(unsigned char * )0xffec8e) /* A/D conversion counter */

/*********************************/
/*     MAIN PROGRAM: ADSCNMN     */
/*********************************/
void ADSCNMN(void)
{
        MSTPCR = 0x3dff;         /* Disable module(A/D) stop mode */
        scn_endf = 0;
        scn_cnt = 0;
        adicnt = 0;
        ADCR = 0x3b;             /* Initialize ADCR */
        ADCSR = 0x4E;            /* Initialize ADCSR */
        set_imask_ccr(0);        /* Enable interrupt */

        while (scn_endf == 0)    /* A/D conversion finish ? */

        ADCSR_BP.ADIE=1;         /* Enable A/D interrupt */

        while (1);               /* Loop */
}

/*********************************/
/*   INTERRUPT PROGRAM: SCNEND   */
/*********************************/
void SCNEND(void)
{
        ADCSR_BP.ADF =0;         /* Clear ADF */

        if (adicnt<10)
                {
                scn[scn_cnt++] = ADDRA; /* Set RAM address to store the data */
                scn[scn_cnt++] = ADDRB;
                scn[scn_cnt++] = ADDRC;
                scn[scn_cnt++] = ADDRD;
                scn[scn_cnt++] = ADDRE;
                scn[scn_cnt++] = ADDRF;
                scn[scn_cnt++] = ADDRG;
```

```
                adicnt = adicnt+1;

                }
                else
                {
                scn_endf = 1;   /* Set scn_endf */
                ADCSR_BP.ADST=0;/* Stop A/D conversion */
                }
}
```

Specifications

Pulse output is performed as shown in figure 1 by transferring 30-byte data (comprising five 6-byte blocks) set in ROM to I/O ports each time a falling edge is detected in an external signal.

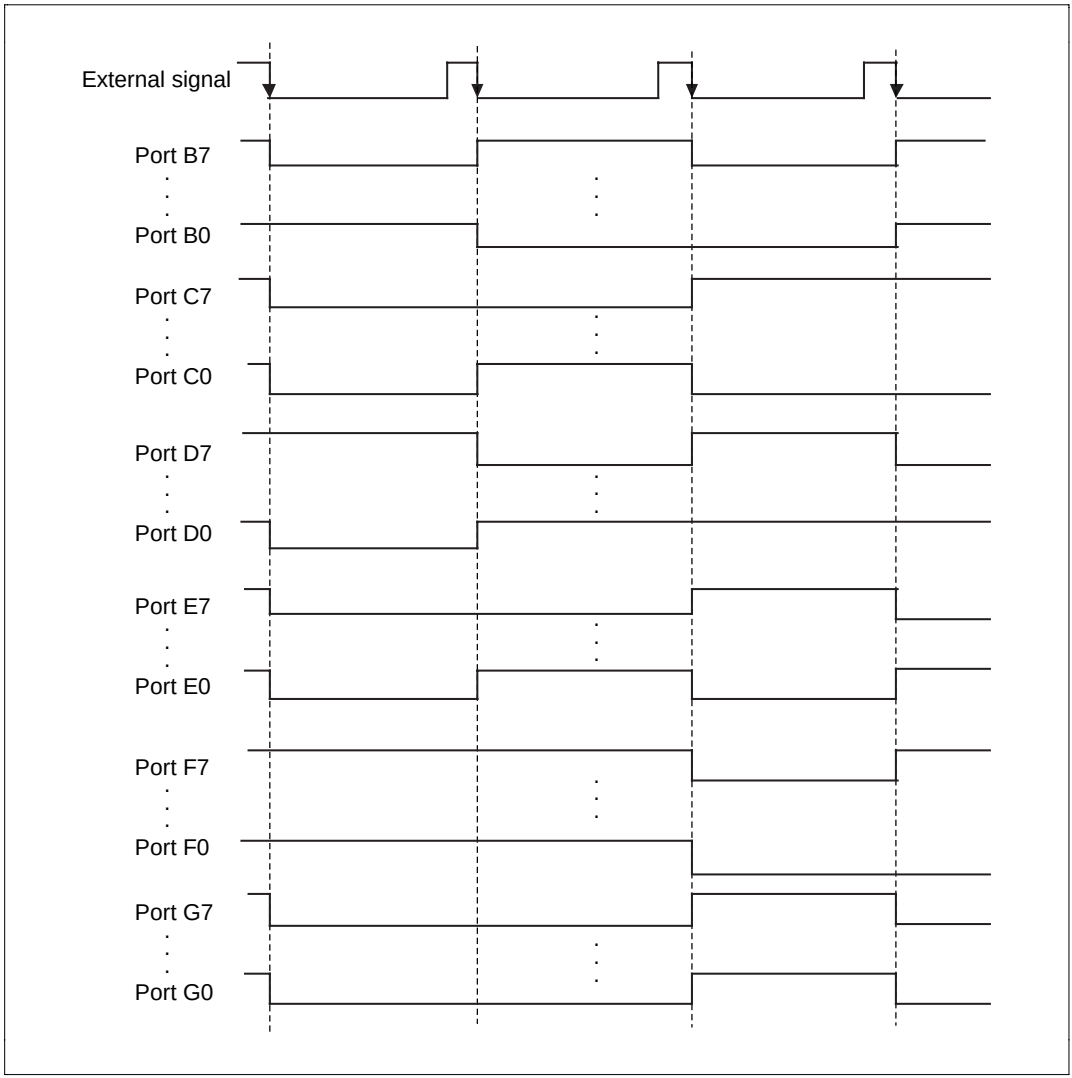


Figure 1 Example of Waveform Output

Functions Used

1. In this sample task, the DTC is activated on each falling edge of IRQ0, and 6-byte data is output to ports B to G.

    a. Figure 2 shows the DTC block diagram for this sample task. The following functions are used to perform block transfer:

    - A function that activates the DTC by means of an external request (DTC activation by IRQ)
    - A function that transfers data in block units when the DTC is activated (block transfer mode)

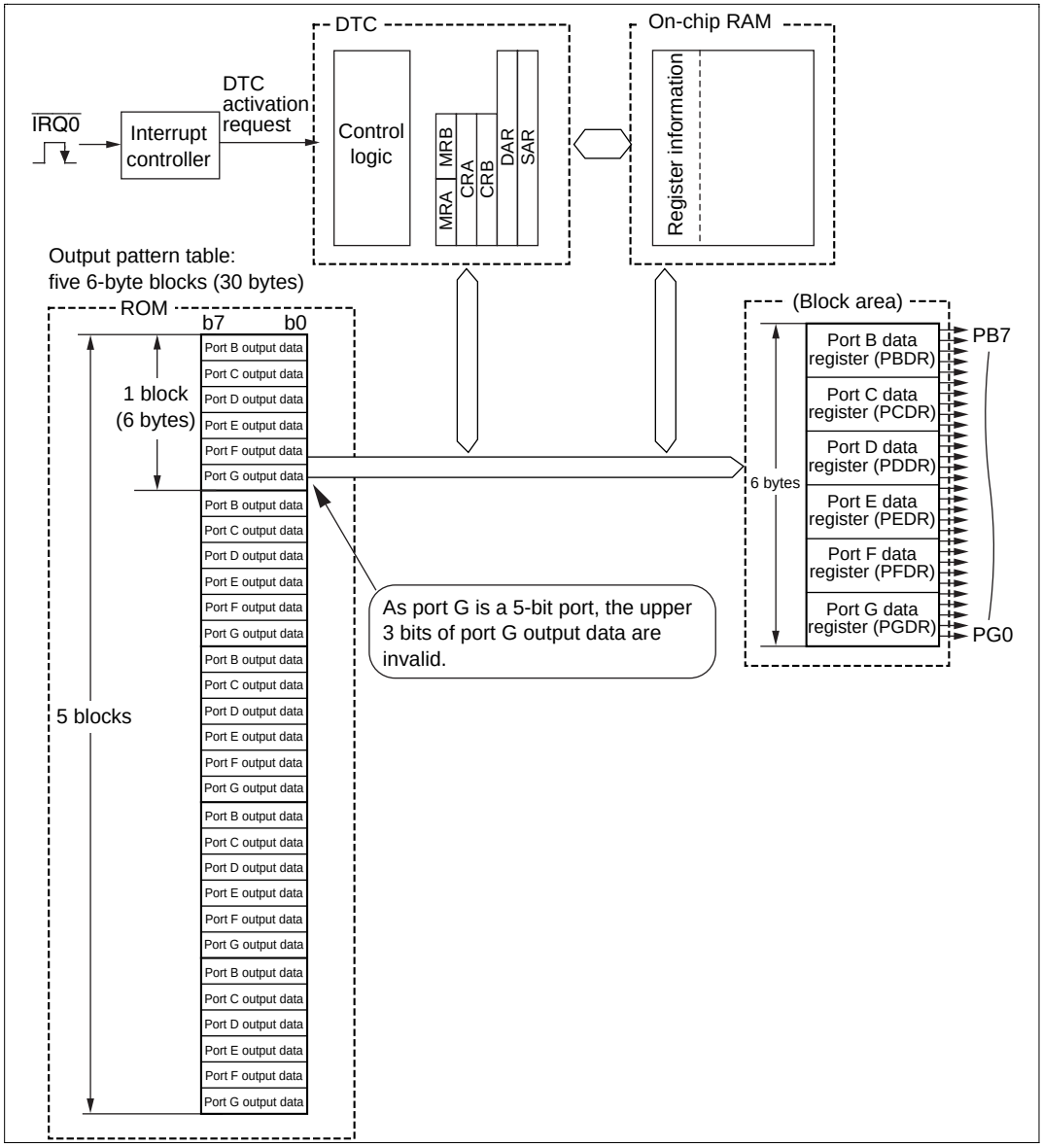


Figure 2 Block Diagram of Block Transfer by DTC

b. Figure 3 shows the DTC vector table and memory allocation. DTC register information is located from address H'FFF800 in the following order: MRA, SAR, MRB, DAR, CRA, CRB.

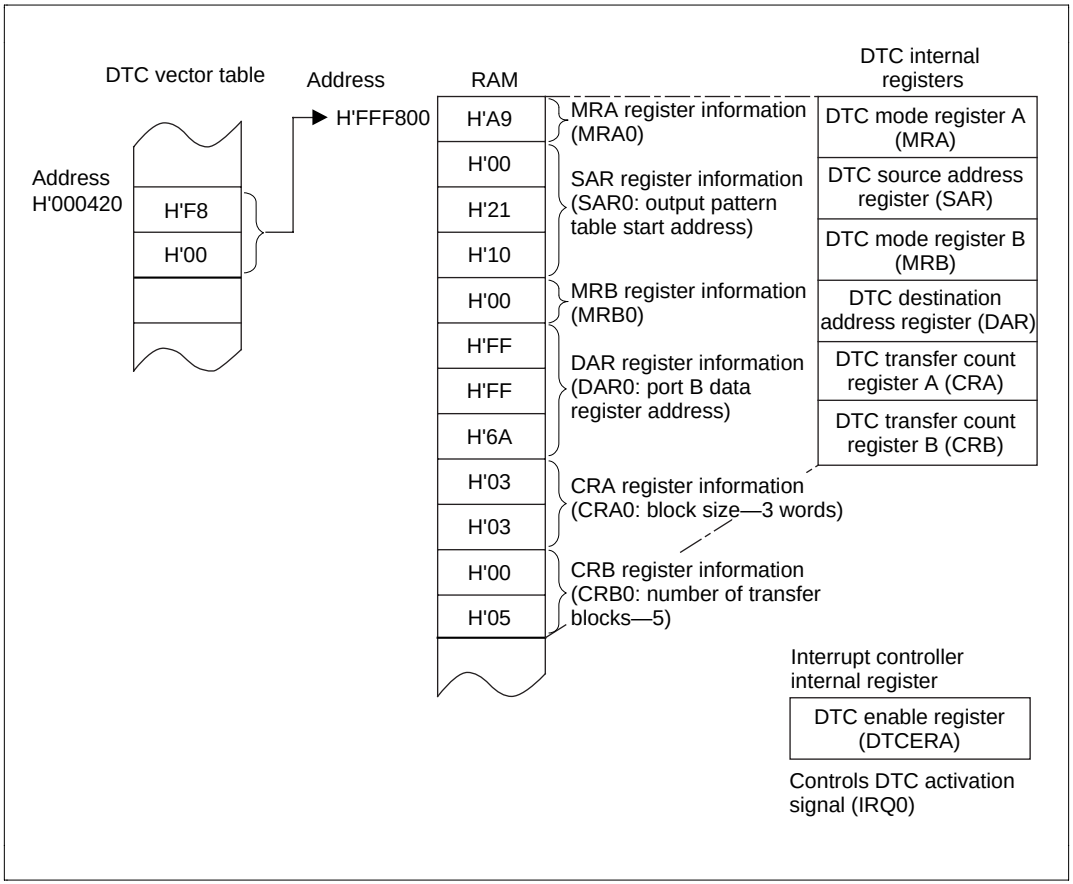


Figure 3 Example of DTC Vector Table and Memory Allocation

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform block transfer.

Table 1 H8S/2655 Function Assignments

DTC Functions	Function
MRA,B	DTC mode control
SAR	Transfer source address setting
DAR	Transfer destination address setting
CRA	Data transfers number setting
CRB	Transfer number setting in block transfer mode
DTCER	Controls enabling/disabling of DTC activation by each interrupt source

Operation

1. Figure 4 shows the principles of block transfer operation using the DTC. Block transfer is performed by means of hardware and software processing, using the timing shown in this figure.

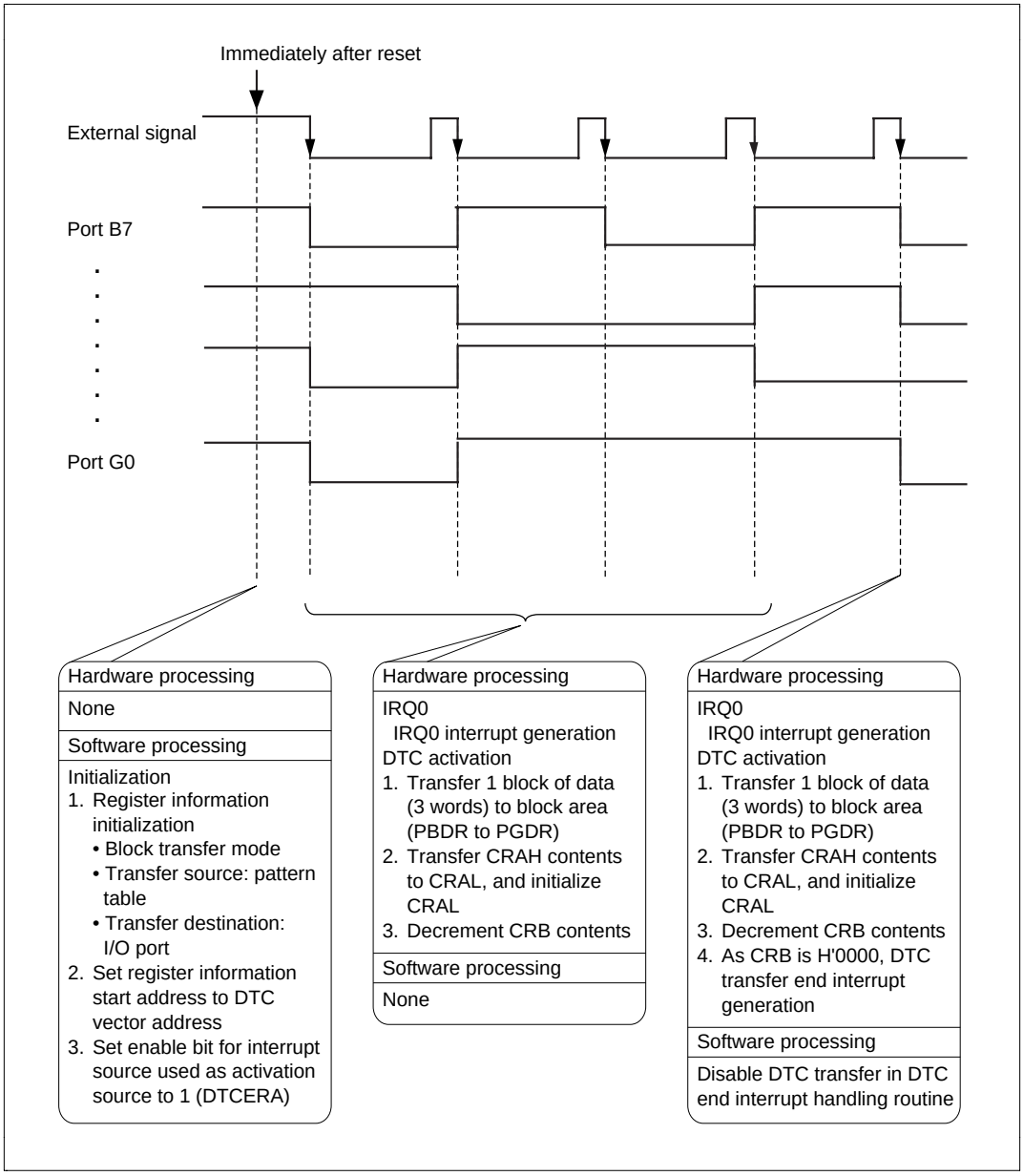


Figure 4 Principles of Block Transfer Operation

Software

1. Modules

Module Name	Label	Function
Main routine	blkmn	DTC initialization
Transfer end	txend	Initiated by DTC transfer end interrupt; disables DTC transfer

2. Arguments

This sample task does not use any arguments between modules.

3. Internal Registers Used

Register Name	Function	Module
DTCER	Enables DTC activation by IRQ0 interrupt	Main routine
MSTPCR	Controls DTC module stop mode	Main routine
ISCRL	Sets interrupt request on falling edge of IRQ0	Main routine
IER	Enables IRQ0 interrupt	Main routine
ISR	Indicates IRQ0 input status	Main routine

4. RAM Used

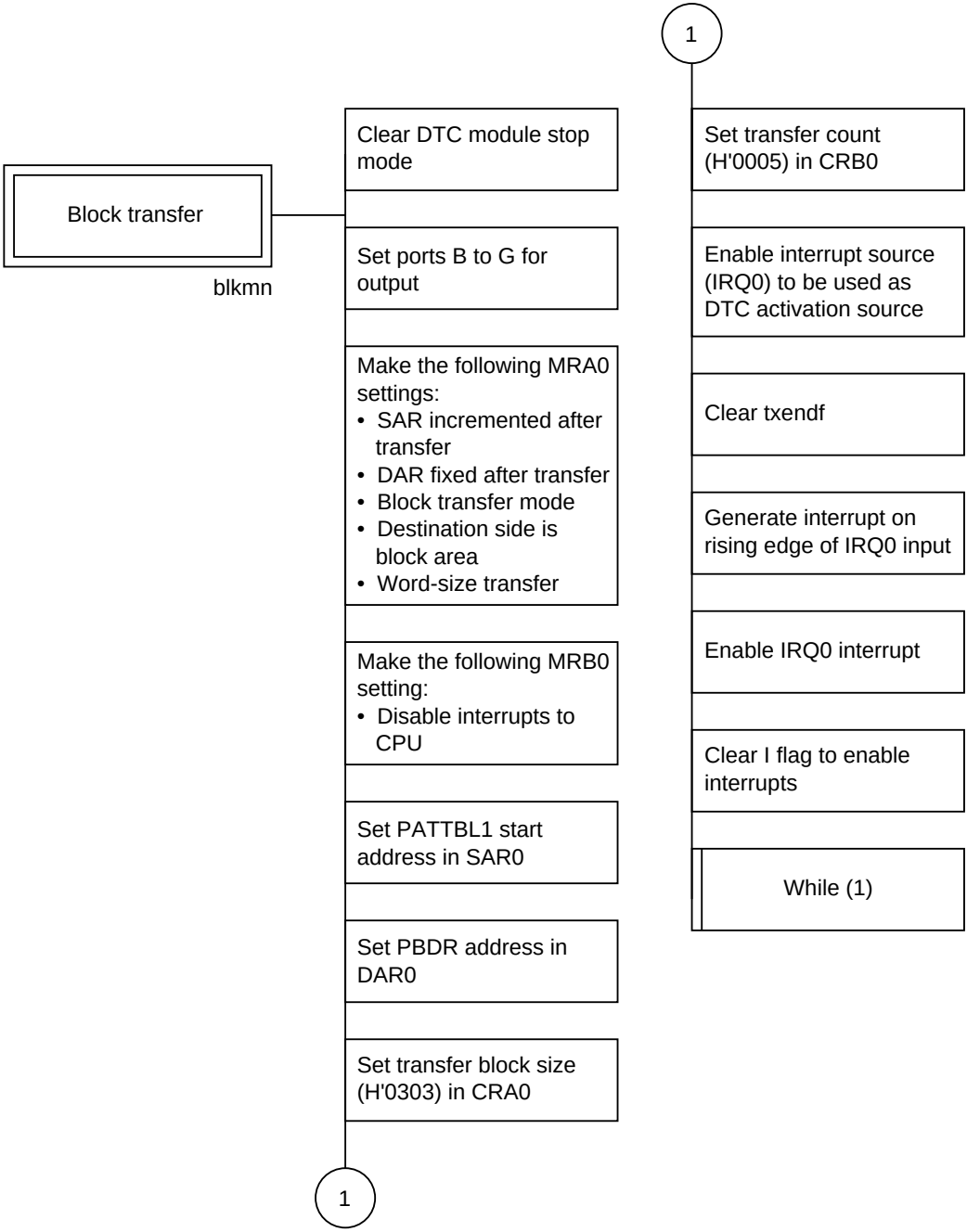
Label	Function	Data Length	Module
MRA0	Sets DTC0 to block transfer mode	Unsigned char	Main routine
MRB0	Disables interrupts to CPU	Unsigned char	Main routine
SAR0	Specifies transfer source address (PATTBL1)	Unsigned long	Main routine
DAR0	Specifies transfer destination address (PBDR)	Unsigned long	Main routine
CRA0	Specifies block size	Unsigned short	Main routine
CRB0	Specifies number of transfer blocks	Unsigned short	Main routine
txendf	Transfer end flag	Unsigned char	Transfer end

5. Data Table

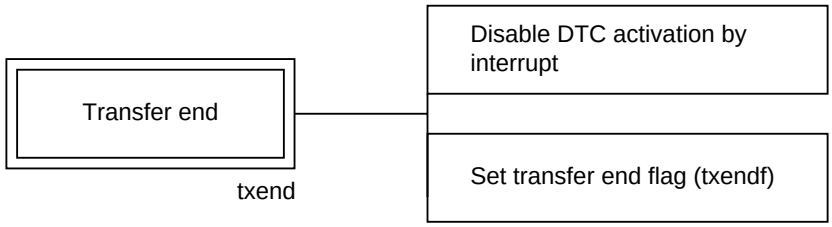
Table Name	Function	Data Length	Data Capacity
PATTBL1	Output pattern setting	Unsigned short	15 words

PAD

1. Main routine



2. Transfer end



Program List

```
#include <machine.h>
#include "..\h8sapn\h8s.h"


/****************************************/
/*           PROTCOL                    */
/****************************************/
void blkmn(void);
#pragma interrupt (txend)

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
#define txendf  (*(volatile unsigned char *)0xffec00)
#define SAR0    (*(volatile unsigned long *)0xfff800)
#define MRA0    (*(volatile unsigned char *)0xfff800)
#define DAR0    (*(volatile unsigned long *)0xfff804)
#define MRB0    (*(volatile unsigned char *)0xfff804)
#define CRA0    (*(volatile unsigned short *)0xfff808)
#define CRB0    (*(volatile unsigned short *)0xfff80a)

/****************************************/
/*         DATA TABLE                   */
/****************************************/
const unsigned short PATTBL1[5][3] =
        {{0x1111,0x2222,0x3333,},{0x4444,0x5555,0x6666},
        {0x7777,0x8888,0x9999},{0x1010,0x2020,0x3030},
        {0x4040,0x5050,0x6060}};
        /* Output data table */

/****************************************/
/*        MAIN PROGRAM : blkmn          */
/****************************************/
void blkmn(void)
{
        MSTPCR = 0x3fff;        /* Disable module(DTC) stop mode*/
        PBDDR = 0xff;           /* PB-PG : output */
        PCDDR = 0xff;
        PDDDR = 0xff;
        PEDDR = 0xff;
        PFDDR = 0xff;
        PGDDR = 0xff;
        SAR0 = (long)(PATTBL1); /* Set base address */
        DAR0 = (long)(&PBDR);   /* Set excute address */
        MRA0 = 0xa9;            /* Block translation mode */
        MRB0 = 0x00;            /* Initialize MRB0 */
        CRA0 = 0x0303;          /* Set excute count */
        CRB0 = 0x0005;          /* Set block excute count */
        DTCERA_BP.IRQ0 = 1;     /* Enable DTC */

        txendf = 0;             /* Clear DTC end flag */

        ISCRL = 0x01;           /* Initialize ISCRL */
        ISR_BP.IRQ0F = 0;       /* Clear IRQ flag */
```

```
        IER = 0x01;             /* Enable IRQ0 interrupt */
        set_imask_ccr(0);       /* Enable interrupt */
        while(txendf == 0);
        while(1);
}

/*************************************/
/*        NAME : txend               */
/*************************************/
void txend(void)
{
        DTCERA_BP.IRQ0 = 0;     /* Disable DTC */
        txendf =1;              /* Set DTC end flag */
}
```

Specifications

1. On detection of a port falling edge, the DTC is activated and transfers a 128-byte block, as shown in figure 1.
2. The transfer area is H'A00000 to H'A000FF.
3. A 20 Hz H8S/2655 internal operating frequency is used.

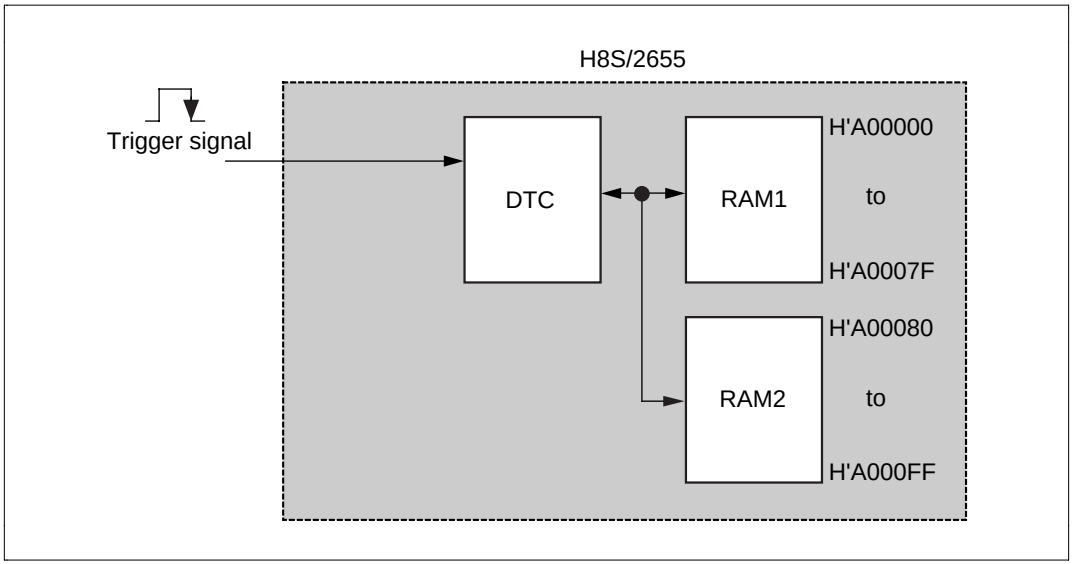


Figure 1 Block Diagram of Software-Activated Data Transfer

Functions Used

1. In this sample task, the DTC is activated by software, and transfers 128-byte data to RAM.

   a. Figure 2 shows the DTC block diagram for this sample task. The following functions are used to perform data transfer:

      - A function that activates the DTC by means of software (DTC activation by software)
      - The ability to send an interrupt request to the CPU at the end of data transfer

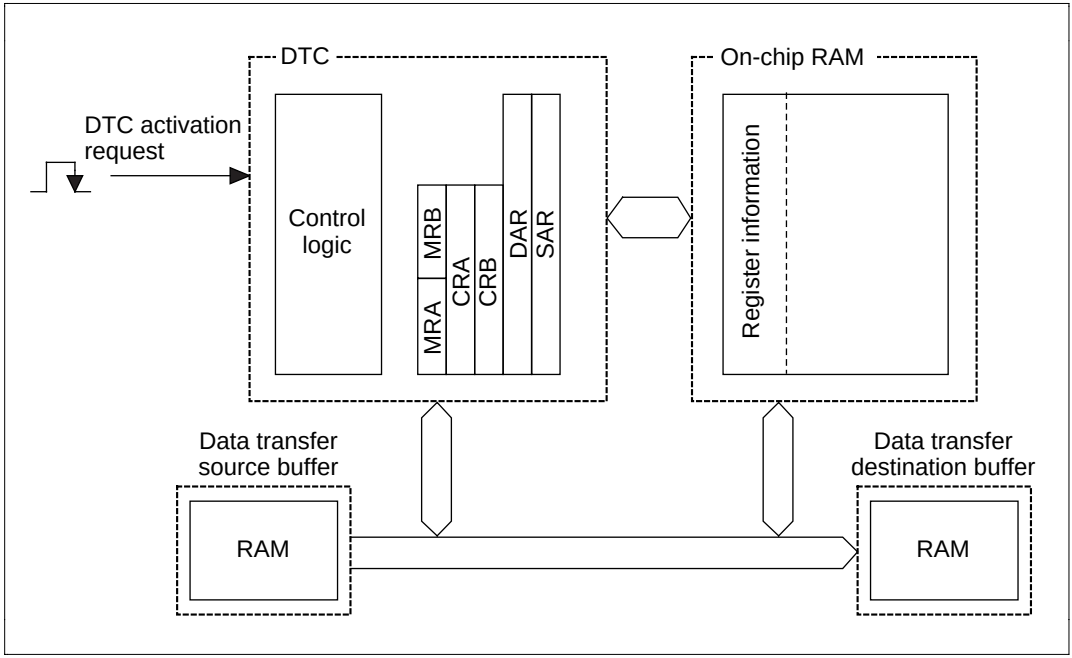


Figure 2 Block Diagram of Software-Activated Data Transfer

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform block transfer.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
MRA,B	DTC mode control
SAR	Transfer source address setting
DAR	Transfer destination address setting
CRA	Data transfer number setting
DTCER	Controls enabling/disabling of DTC activation by each interrupt source
P3DR	Trigger signal input

Operation

Figure 3 shows the principles of data transfer operation using the DTC. Block transfer is performed by means of hardware and software processing, using the timing shown in this figure.

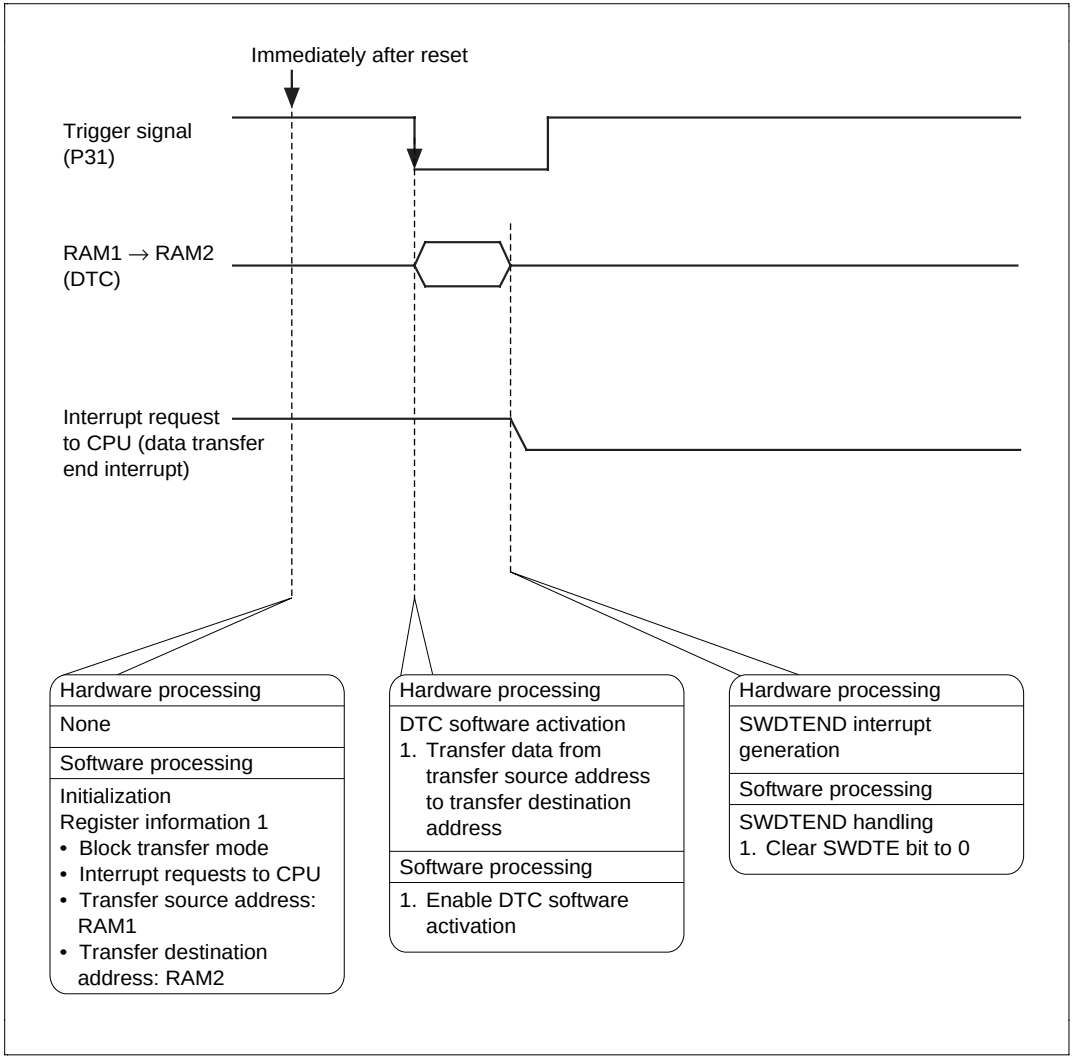


Figure 3 Principles of Software-Activated Data Transfer Operation

Software

1. Modules

Module Name	Label	Function
Main routine	dtcsftmn	DTC initialization
Transfer end	trsend	Initiated by DTC transfer end interrupt; sets transfer end flag

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
trs_end	Flag indicating end of transfer	Unsigned char	Data transfer end	Output
	1: End of transfer 0: Transfer in progress		Main routine	Input
err	Flag indicating DTC activation error	Unsigned char	Main routine	Output
	1: Activation failure 0: Activated			

3. Internal Registers Used

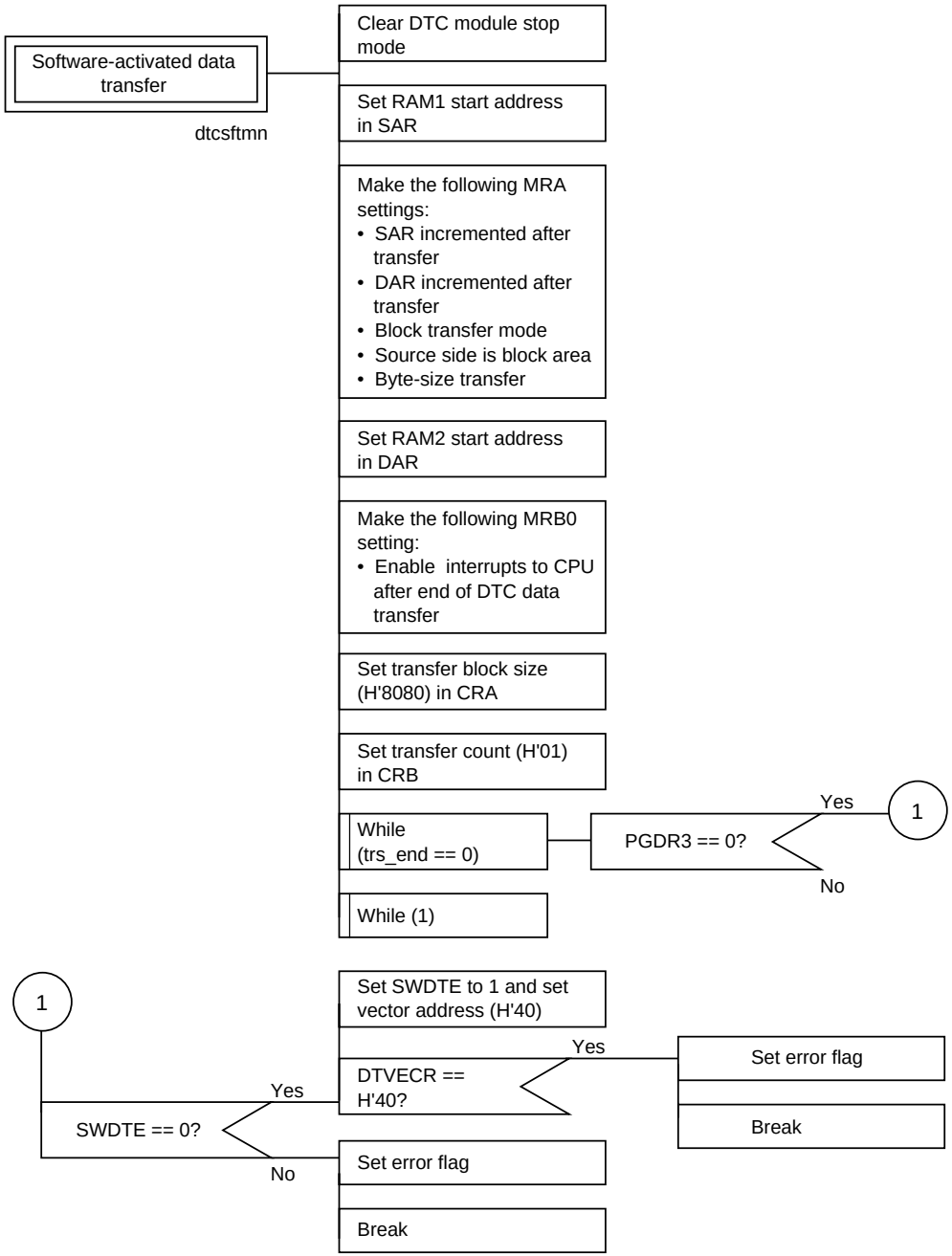
Register Name	Function	Module
DTVECR	Enables DTC activation by software	Main routine
MSTPCR	Controls DTC module stop mode	Main routine

4. RAM Used

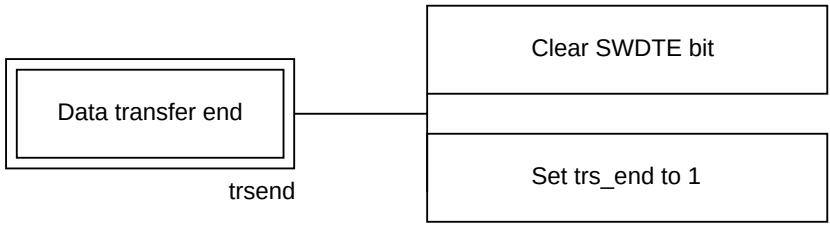
Label	Function	Data Length	Module
MRA	Sets DTC to block transfer mode	Unsigned char	Main routine
MRB	Enables interrupt to CPU after data transfer	Unsigned char	Main routine
SAR	Specifies transfer source address (RAM1)	Unsigned long	Main routine
DAR	Specifies transfer destination address (RAM2)	Unsigned long	Main routine
CRA	Specifies block size (H'8080)	Unsigned short	Main routine
CRB	Specifies number of transfer blocks (H'0001)	Unsigned short	Main routine

PAD

1. Main routine



2. Data transfer end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTCOL                    */
/****************************************/
void dtcsftmn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
#define trs_end (*(volatile unsigned char *)0xffec00)
#define err (*(volatile unsigned char *)0xffec01)

volatile struct databuf
{
               unsigned char  ram1[128];
               unsigned char  ram2[128];
};
#define dat (*(struct databuf *)0xA00000)

#define MRA (*(volatile unsigned char *)0xfff800)
#define SAR (*(volatile unsigned long *)0xfff800)
#define MRB (*(volatile unsigned char *)0xfff804)
#define DAR (*(volatile unsigned long *)0xfff804)
#define CRA (*(volatile unsigned short *)0xfff808)
#define CRB (*(volatile unsigned short *)0xfff80a)
```

```
/****************************************/
/*        MAIN PROGRAM : dtcsftmn       */
/****************************************/
void dtcsftmn(void)
{
       MSTPCR = 0x3fff;
       SAR = (long)(&dat.ram1);
       MRA = 0xa8;
       DAR = (long)(&dat.ram2);
       MRB = 0x40;
       CRA = 0x8080;
       CRB = 0x0001;

       while (trs_end == 0)
       {
              if (P3DR_BP.P31DR == 0)
              {
                     if (DTVECR_BP.SWDTE == 0){
                            DTVECR = 0xc0;

                            if (DTVECR_BP.VECR != 0x40)
                                   err = 1;
                                   break;
                     }
                     else{
                            err = 1;
                            break;
                     }
              }
       }

       while(1);
}

/****************************************/
/*        NAME : trsend                 */
/****************************************/
#pragma interrupt(trsend)
void trsend(void)
{
        DTVECR_BP.SWDTE = 0;
        trs_end = 1;
}
```

Specifications

1. Using the DMAC single address mode, transfer is performed between external space specified by either the transfer source or transfer destination address, and an external device selected by the DACK strobe, without regard to the address, as shown in figure 1.

2. The DMAC is activated by detection of a falling edge in an external signal.

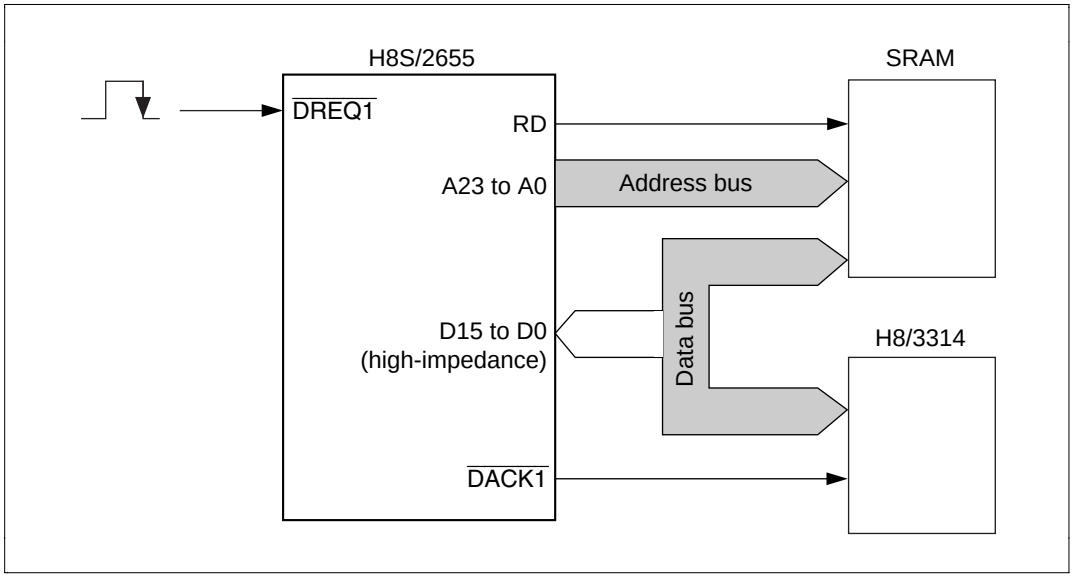


Figure 1 Single Address Mode Data Bus

Functions Used

1. In this sample task, the DMAC single address mode (idle mode specification) is used to transfer data from external memory (SRAM) to an external device (H8/3314).

   a. Figure 2 shows the DMAC block diagram for this sample task. The following DMAC functions are used to perform block transfer:

      - A function that activates the DMAC in response to an external request (DMAC activation by DREQ)
      - Execution of the specified number of one-byte or one-word transfers between external memory and an external device in response to a single transfer request (single address mode)

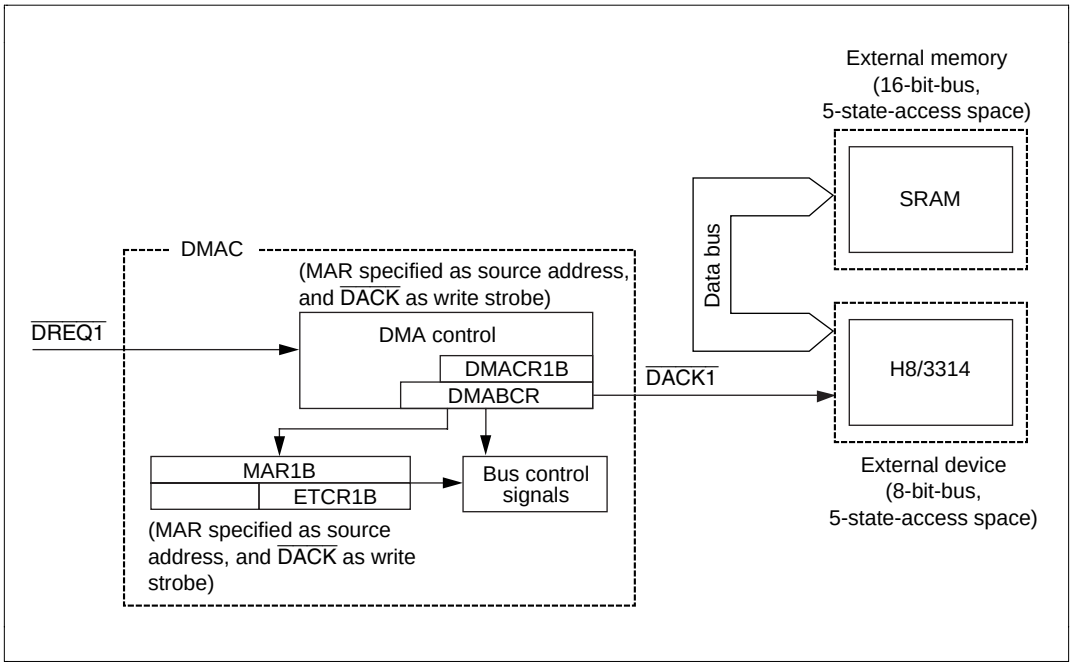


Figure 2 DMA Controller Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform block transfer.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
DREQ1	Inputs external pulse used as DMAC activation trigger
DACK1	Data transfer acknowledge
DMABCR	Controls operation of each channel
DMACR1B	Sets DMAC to idle mode
MAR1B	Transfer source address setting
ETCR1B	Transfer number setting

Operation

Figure 3 shows the principles of the operation. One-byte transfer is performed from external 16-bit, 5-state-access space to external device 8-bit, 5-state-access space by means of H8S/2655 hardware and software processing, as shown in this figure.

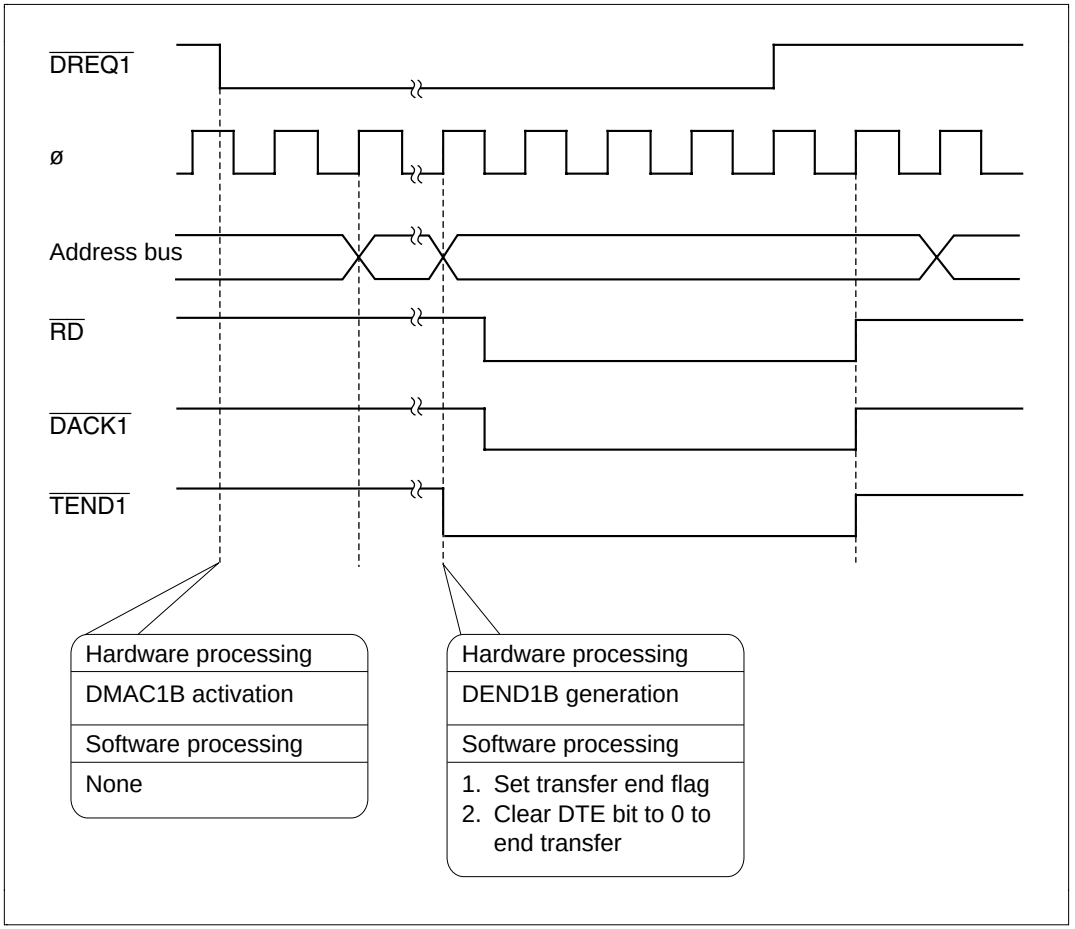


Figure 3 Principles of Single Address Mode (Byte Read) Transfer Operation

Software

1. Modules

Module Name	Label	Function
Main routine	singlemn	DMAC initialization
Data transfer end	transend	Sets transfer end flag

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
status	Flag indicating end of data transfer	Unsigned char	Main routine	Input
	1: End of transfer 0: Operation in progress		Transfer end	Output

3. Internal Registers Used

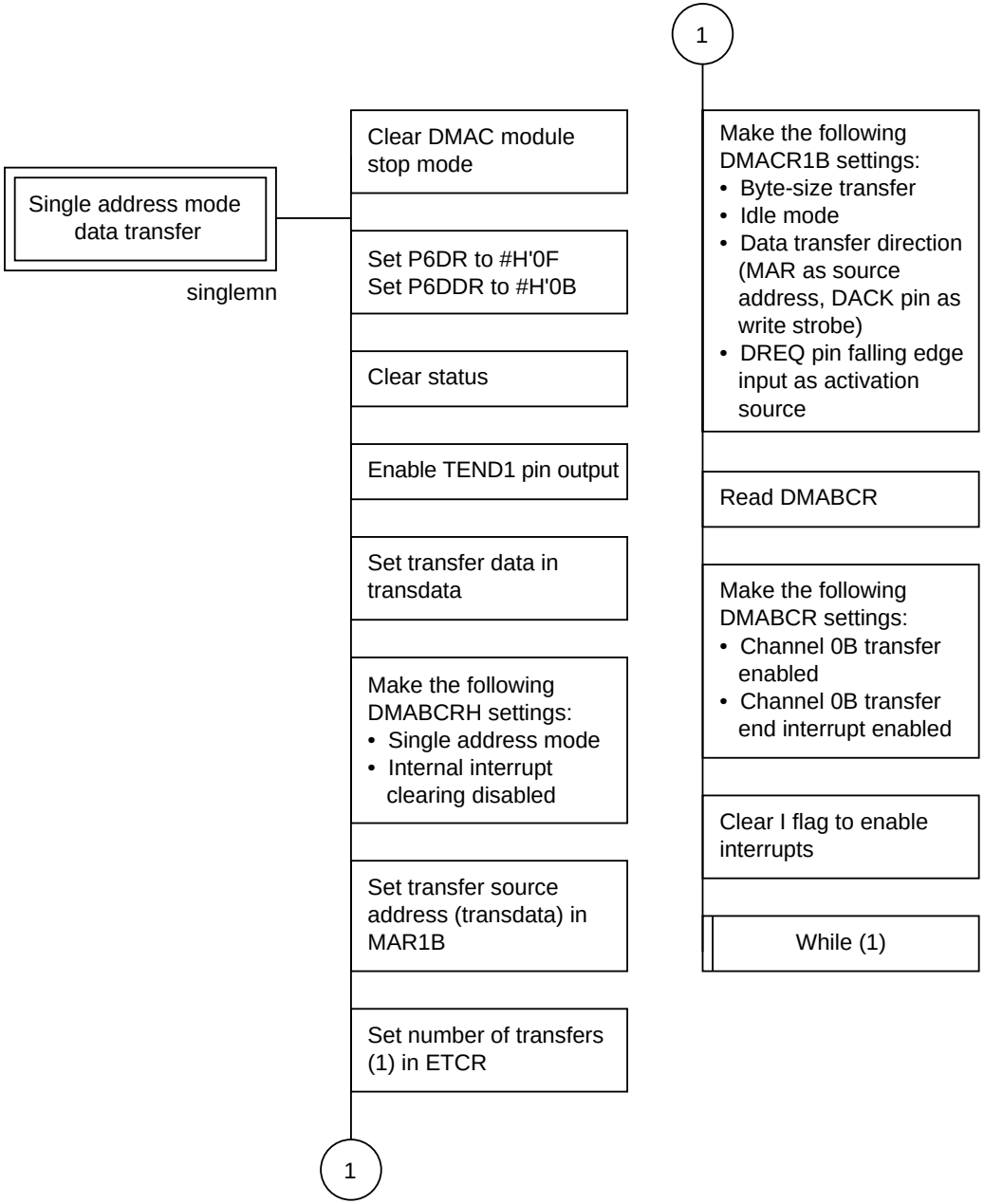
Register Name	Function	Module
DMABCRH	Sets DMAC1B to single address mode in short address mode	Main routine
DMABCRL	Enables data transfer	Main routine
DMACR1B	Makes the following DMACR settings: • Byte-size transfer • Idle mode • MAR incremented after data transfer • Data transfer direction (MAR as source address, DACK pin as write strobe) • DREQ pin falling edge input as activation source	Main routine
MAR1B	Transfer source address setting	Main routine
ETCR	Transfer number setting	Main routine
MSTPCR	Clears DMAC module stop mode	Main routine

4. RAM Used

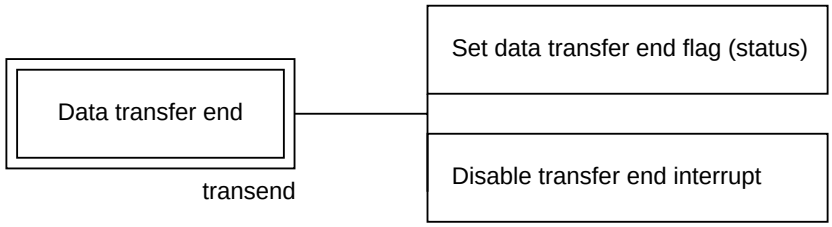
Label/Register Name	Function	Data Length	Module
transdata	Holds data to be transferred	Unsigned char	Main routine

PAD

1. Main routine



2. Data transfer end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*          PROTCOL                     */
/****************************************/
void singlemn(void);

/****************************************/
/*          RAM ALLOCATION              */
/****************************************/
#define status (*(volatile unsigned char *)0xffec00)
#define transdata (*(volatile unsigned char *)0x800000)

/****************************************/
/*      MAIN PROGRAM : singlemn         */
/****************************************/
void singlemn(void)
{
        MSTPCR = 0x7fff;

        status = 0x00;                  /* Clear user flag */

        P6DR = 0x0F;
        P6DDR = 0x0B;                   /* Set corresponding port to output */

        DMATCR =  0x20;                 /* Low output from TEND1 after end of
                                        transfer */
        transdata = 0xaa;               /* Set transfer data */

        DMABCRH = 0x20;                 /* Set ch1B to short address mode
                                           Set ch1B to single address mode
                                           Set ch1B to internal interrupt clear
disabled */
        MAR1B = (long)(&transdata);     /* Enter transfer source address */
        ETCR1B = 0x0001;                /* Enter number of transfers */
        DMACR1B = 0x22;                 /* Byte-size transfer, idle mode,
                                        MAR incremented, DREQ1 falling edge */
        DMABCRL |= 0x88;                /* Enable ch1B data transfer, enable
                                        ch1B transfer end interrupt */
        set_imask_ccr(0);

        while(1);
}


/****************************************/
/*    NAME : transend(set end flag)     */
/****************************************/
#pragma interrupt(transend)
void transend(void)
{
        status = 0x01;                  /* Clear user flag */
        DMABCRL &= 0x77;                /* Clear flag */

}
```

Specifications

1. An arbitrary number of 50% duty pulses are output, as shown in figure 1.
2. At 20 MHz operation, a pulse cycle of 1.2 µs to 102 µs can be set in 0.4 µs units, and the number of pulses output can be set from 1 to 256.

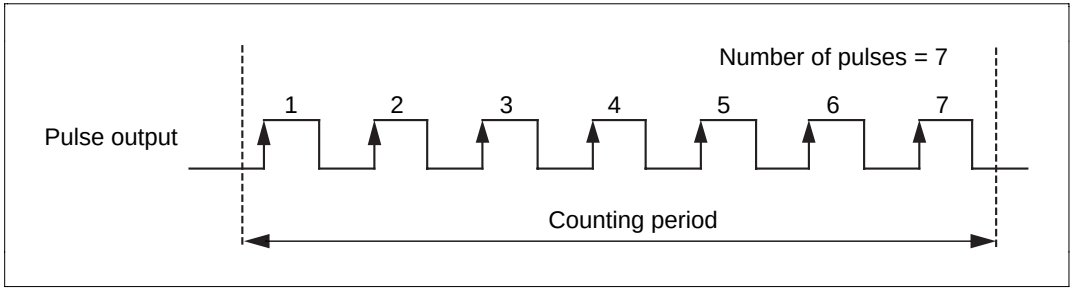


Figure 1 Pulse Output Timing

Functions Used

1. Figure 2 shows the 8-bit timer block diagram for this sample task. This task uses the following functions:

   a. A function that cascades two 8-bit timer channels, and counts channel 0 compare-matches with the channel 1 timer (compare-match count mode)

   b. A function that generates an interrupt at the specified count

   This sample task uses these functions as shown in figure 2 to count pulse rising edges.

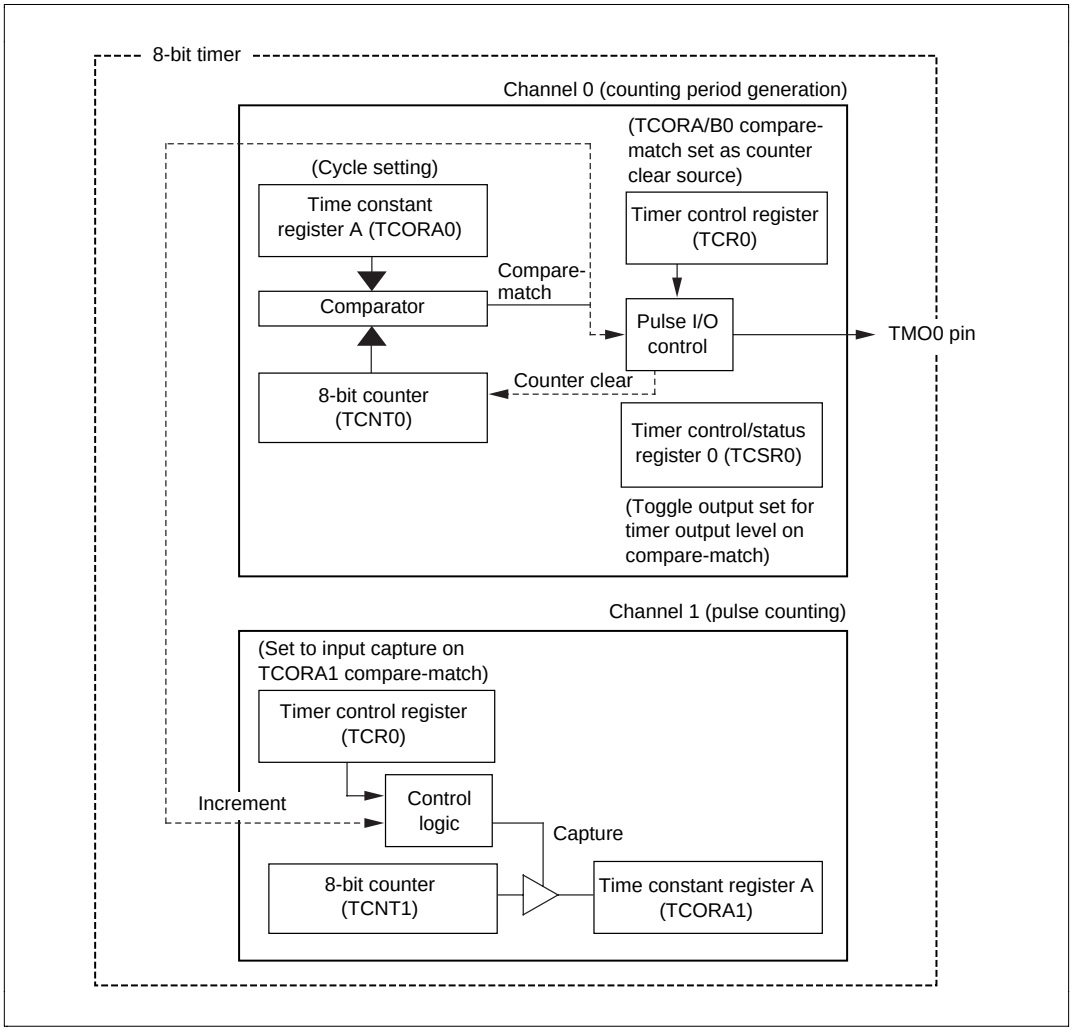


Figure 2 Output Pulse Counting Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform pulse counting.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function	Function
TCNT0	For compare-match A/B generation
TCORA0	For compare-match A generation
TCORB0	For compare-match B generation
TCSR0	1 output on each compare-match A; 0 output on each compare-match B
TMO0	Timer output pin (compare-match output)
TCR0	Counter clearing by compare-match A: input clock selection (ø/8)
TCNT1	Counts channel 0 compare-match A occurrences
TCORA1	For compare-match A generation
TCR1	Counter clearing by compare-match A: sets compare-match (A) interrupt enabled

Operation

Figure 3 shows the principles of the operation. Pulses are counted by means of H8S/2655 hardware and software processing, as shown in this figure.

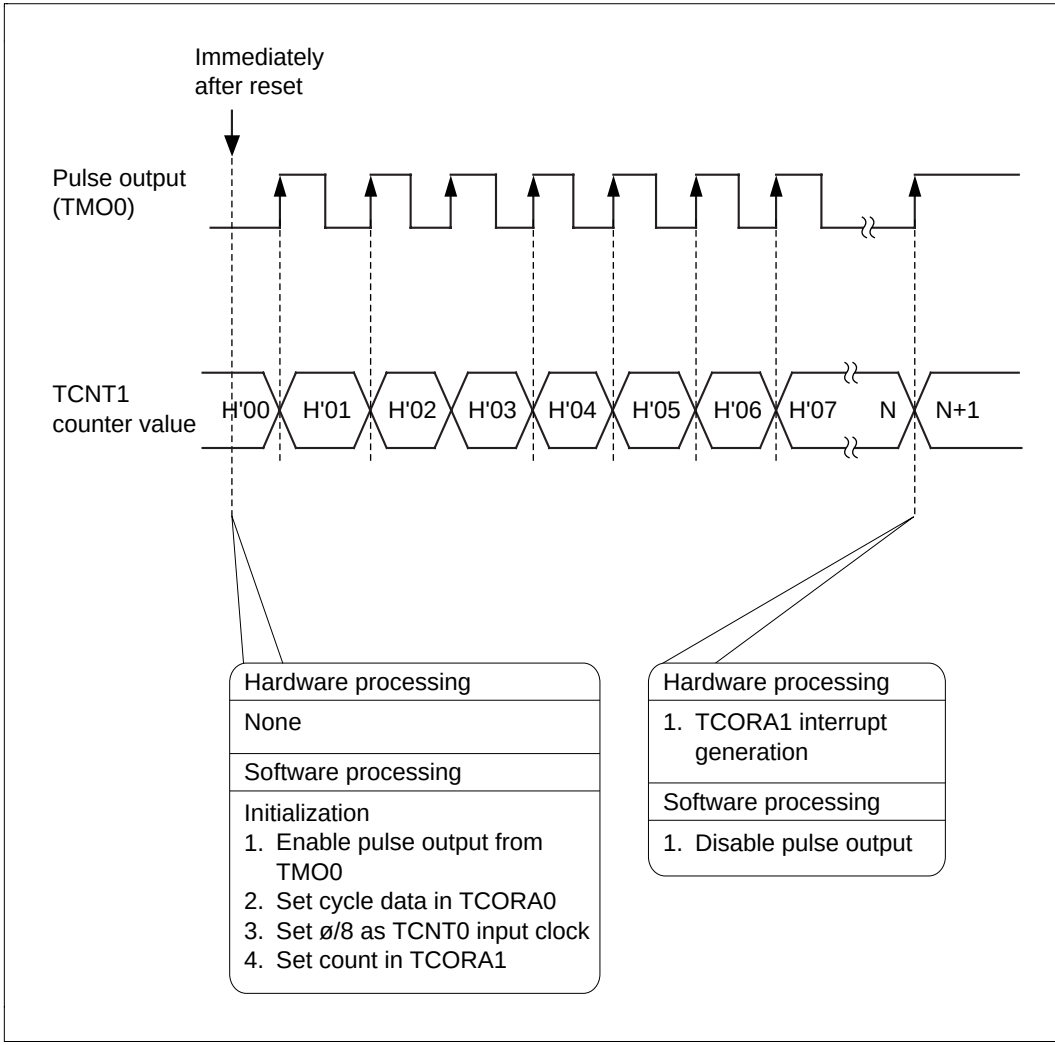


Figure 3 Principles of Pulse Counting Operation

Software

1. Modules

Module Name	Label	Function
Main routine	pulsemn	8-bit timer initialization
Pulse output end	pulend	Initiated by TCORA1 interrupt; sets number of pulses set in TCNT1 as output argument

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
pulse_cycle	Pulse cycle setting	1 byte	Main routine	Input
pulse_count	Pulse count setting	1 byte	Main routine	Input

3. Internal Registers Used

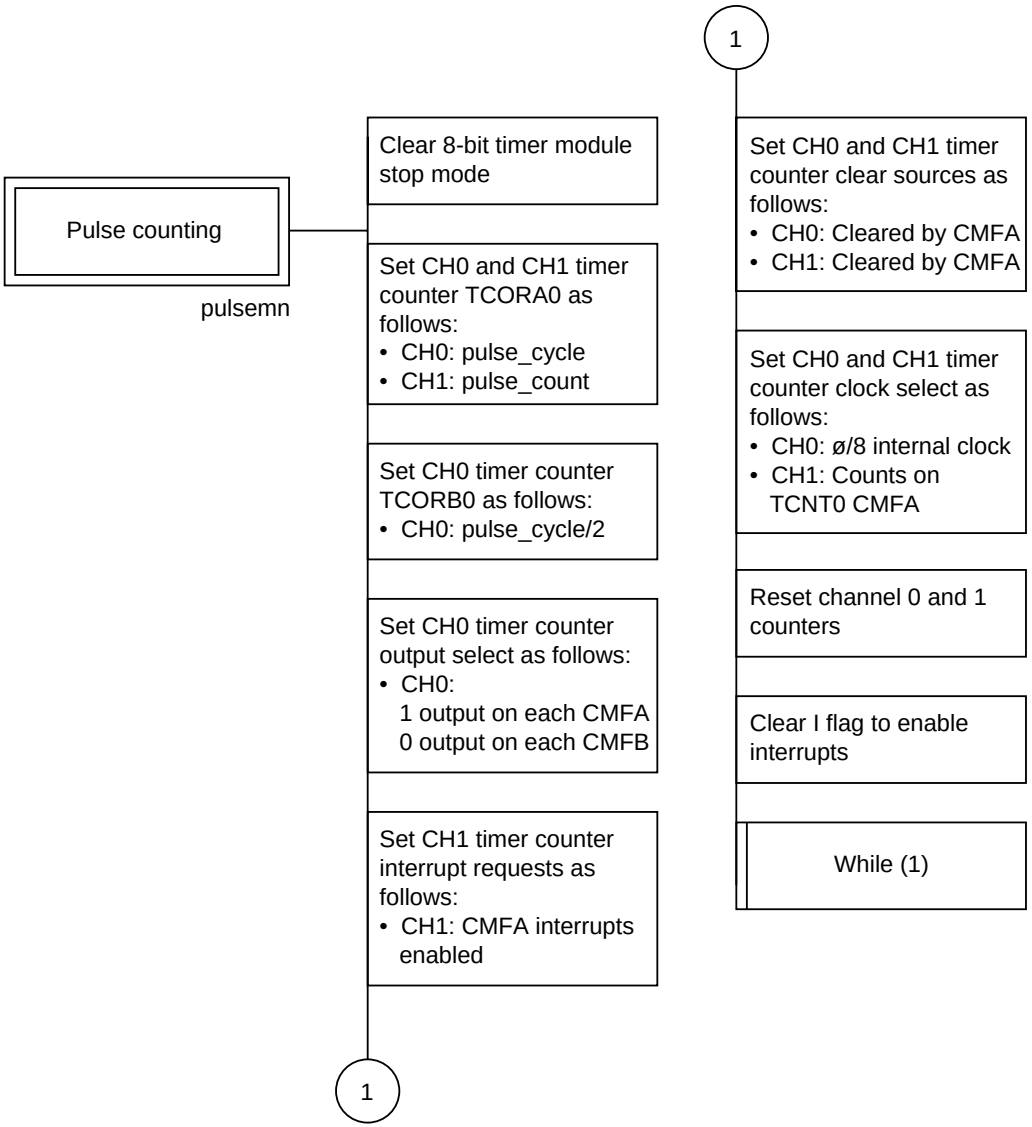
Register Name	Function	Module
TCORA0	For compare-match A generation	Main routine
TCORB0	For compare-match B generation	Main routine
TCSR0	1 output on each compare-match A; 0 output on each compare-match B	Main routine
TCR0	Counter clearing by compare-match A	Main routine
	Input clock selection (ø/8)	Main routine
TCR1	Counts channel 0 compare-match A occurrences	Main routine
	Counter clearing by compare-match A	Main routine
	Sets compare-match (A) interrupt enabled	Main routine
TCORA1	For compare-match A generation	Main routine, count end
MSTPCR	Clears 8-bit timer module stop mode	Main routine

4. RAM Used

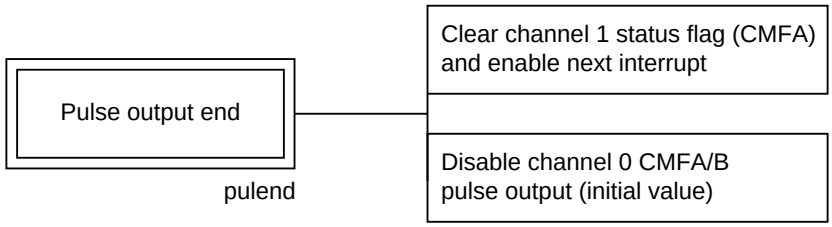
This task does not use any RAM apart from the arguments.

PAD

1. Main routine



2. Pulse output end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTOCOL                   */
/****************************************/
void pulsemn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
# define pulse_cycle (*(unsigned char * ) 0xffec00)
# define pulse_count (*(unsigned char * ) 0xffec01)

/****************************************/
/*      MAIN PROGRAM : pulsemn          */
/****************************************/
void pulsemn(void)
{
        MSTPCR = 0xefff;                /* disable module stop mode*/

        TCORA0 = pulse_cycle;           /* set pulse cycle time */
        TCORB0 = pulse_cycle/2;         /* set "low"pulse time */
        TCORA1 = pulse_count;           /* set pulse counter */

        TCSR0 = 0x06;                   /* initialize TCSR0 */
        TCSR1 = 0x10;                   /* initialize TCSR1 */

        TCR0 = 0x09;                    /* Initialize TCR0 */
        TCR1 = 0x4c;                    /* initialize TCR1 */

        TCNT0 = 0;                      /* reset counter */
        TCNT1 = 0;

        set_imask_ccr(0);

        while(1);                       /* loop */
}

/****************************************/
/*      NAME : pulend(output disble)    */
/****************************************/
#pragma interrupt (pulend)
void pulend(void)
{
        TCSR0 = 0;              /* output disable */
}
```

4.1 High-Speed Data Output TPU, PPG, DMAC

Specifications

1. 12-bit data is output each time the rising edge of an external signal is detected, as shown in figure 1.

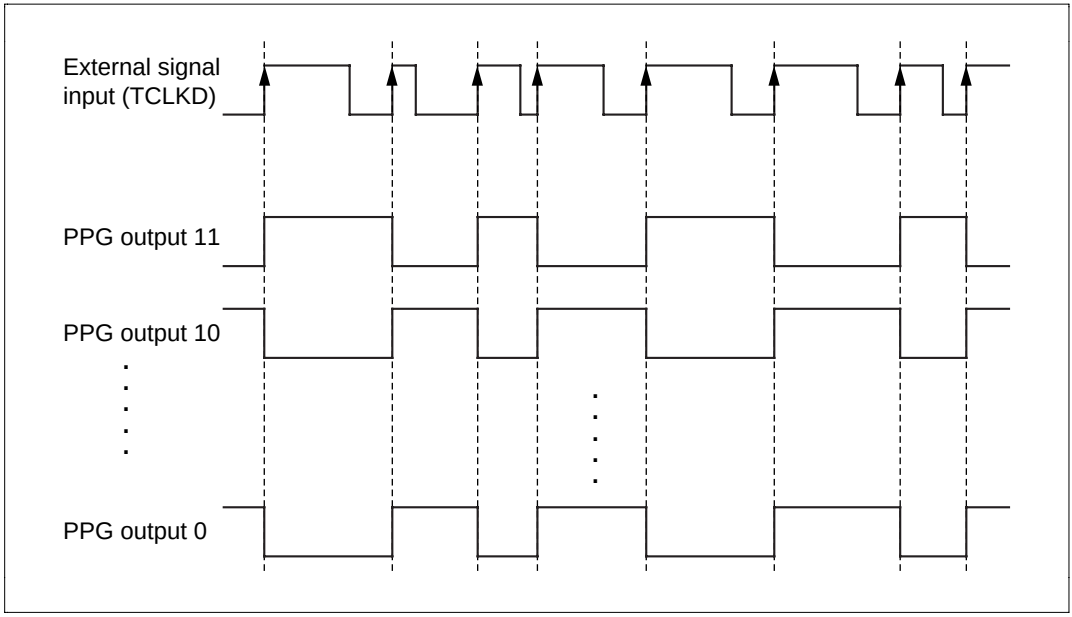


Figure 1 Example of 12-Bit Data Output

Functions Used

1. Figure 2 shows the on-chip function block diagram for this sample task. The following H8S/2655 functions are used to perform high-speed data output:

   Output pattern data table

   The data patterns to be output from the PPG are set in RAM.

   TPU

   DMAC0A and the PPG are activated on each occurrence o compare-match A.

   (H'0000 is set in TGRA, and incrementing on external clock rising edges is specified)

   DMAC0A

   Activated by TPU compare-match A; transfers output data from the output pattern data table to NDR.

   PPG

   Activated by TPU compare-match A; outputs 12-bit data.

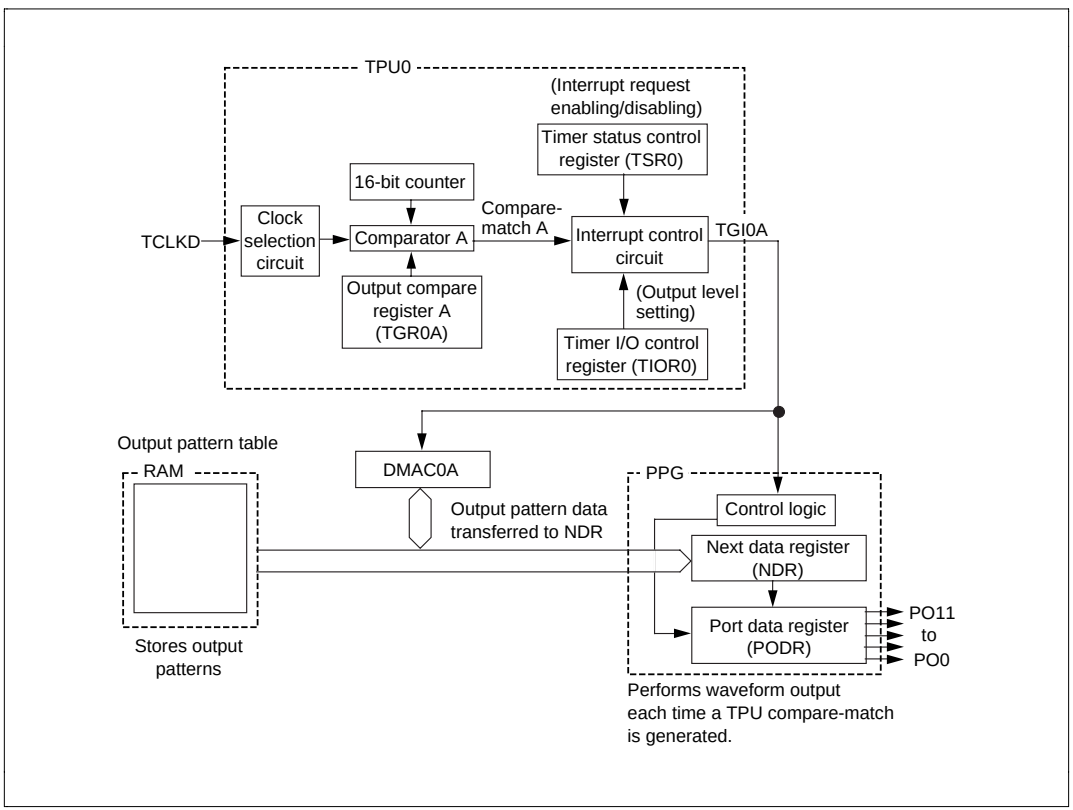


Figure 2 High-Speed Data Output Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform high-speed data output.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
TPU0	TCLKD	External signal input pin
	TCNT0	16-bit counter
	TGR0A	Output compare register
	TCR0	Count clock selection, counter clear source selection
	TIOR0	Sets TGR0A as output compare register
	TSR0	Indicates compare-match or overflow generation
DMAC0A	DMABCR H/L	Controls operation of each channel
	DMACR0A	Controls DMAC0A operation
	MAR0A	Output pattern data table start address setting
	IORA0A	NDR address setting
	ETCR0A	Transfer number setting
PPG	PODRH	Stores PPG output group 2 and 3 output data
	PODRL	Stores PPG output group 0 and 1 output data
	PCR	PPG output trigger signal selection
	NDERH	Enables PPG outputs PO15 to PO8
	NDERL	Enables PPG outputs PO7 to PO0
	NDRH	Stores next PPG output data
	NDRL	Stores next PPG output data
	PO11 to PO8	Group 2 pulse output pins
	PO7 to PO4	Group 1 pulse output pins
	PO3 to PO0	Group 0 pulse output pins

Operation

Figure 3 shows the principles of the operation. High-speed data output is performed by means of H8S/2655 hardware and software processing as shown in the figure.

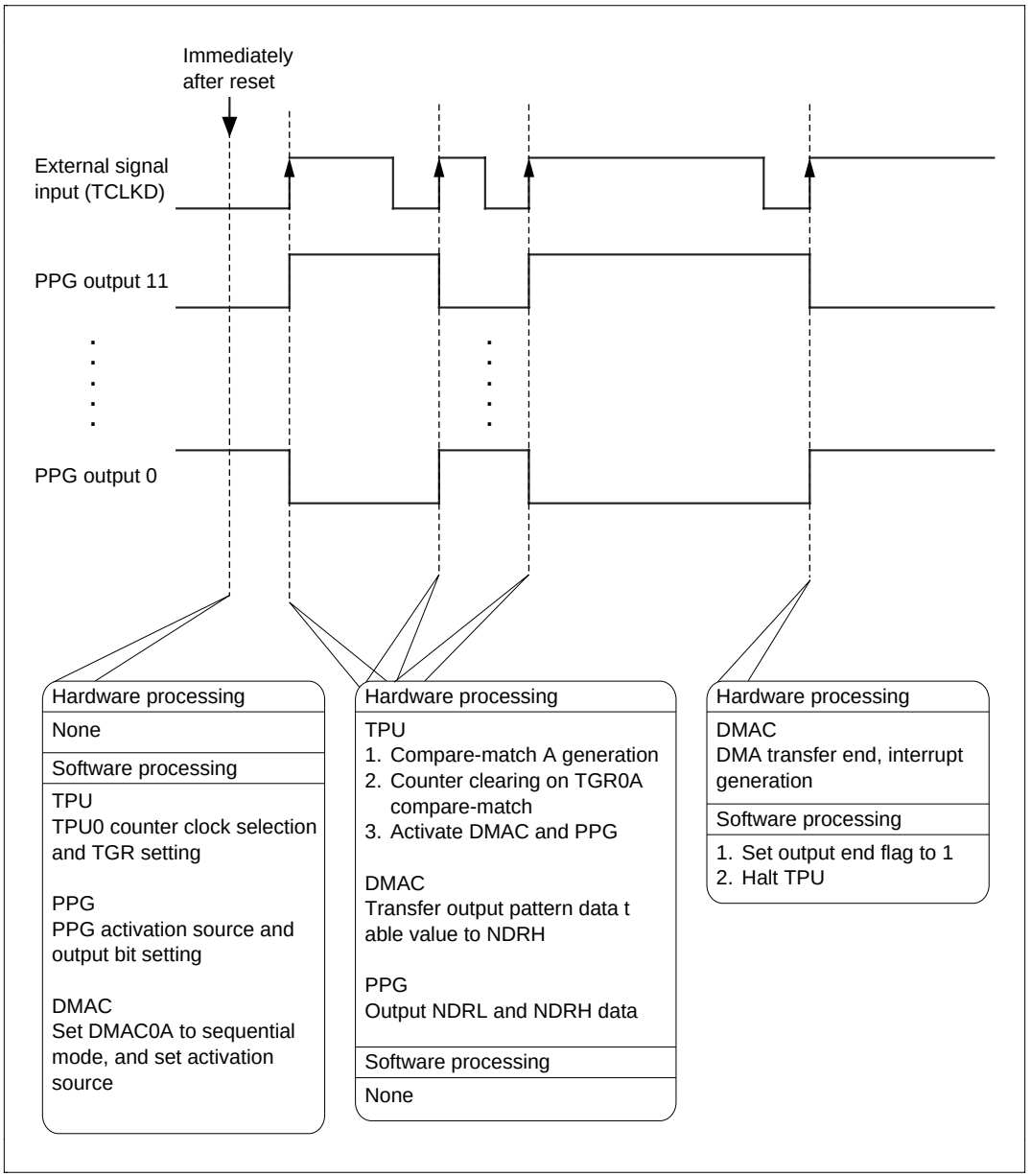


Figure 3 Principles of High-Speed Data Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	STPCMN	TPU, PPG, DMAC0A initialization
DMA interrupt	DMAEND	Sets output end flag

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
dma_fla	Flag indicating end of all output	Unsigned char	Main routine	Input
	0: Output of all bits not completed 1: Output of all bits completed		DMA interrupt	Output

3. Internal Registers Used

On-Chip Function	Register Name	Function
TPU0	TGR0A	Holds output compare value (H'0000)
	TCR0	Makes the following TPU settings: • Counter clearing by TGR0A compare-match • Counting on external signal rising edges • Counting using TCLKD pin
	TIOR0	Sets TGR0A as output compare register, disables pin output
	TIER0	Enables TGI0A interrupt
	TSTR	Starts TCNT0 count operation
PPG	PODRH	Stores PO11 to PO8 output data
	PODRL	Stores PO7 to PO0 output data
	TPMR	Sets normal operation for PO11 to PO0
	TPCR	Sets TPU0 compare-match as PO11 to PO0 output trigger
	NDERH	Enables PPG outputs PO11 to PO8
	NDERL	Enables PPG outputs PO7 to PO0
	NDRL	Stores next output pattern data
	NDRH	Stores next output pattern data
DMAC0A	DMACR0A	Sets word-size as data size Sets incrementing of MAR Sets sequential mode for data transfer Sets TPU0 compare-match A as activation source
	DMABCRH/L	Data transfer and transfer end interrupt enabling/disabling setting
	MAR0A	Output pattern data table transfer source address setting
	IOAR0A	NDRH address (transfer destination) setting
	ETCR0A	Transfer number setting
	MSTPCR	Clears DMAC, TPU, PPG module stop mode

4. RAM Used

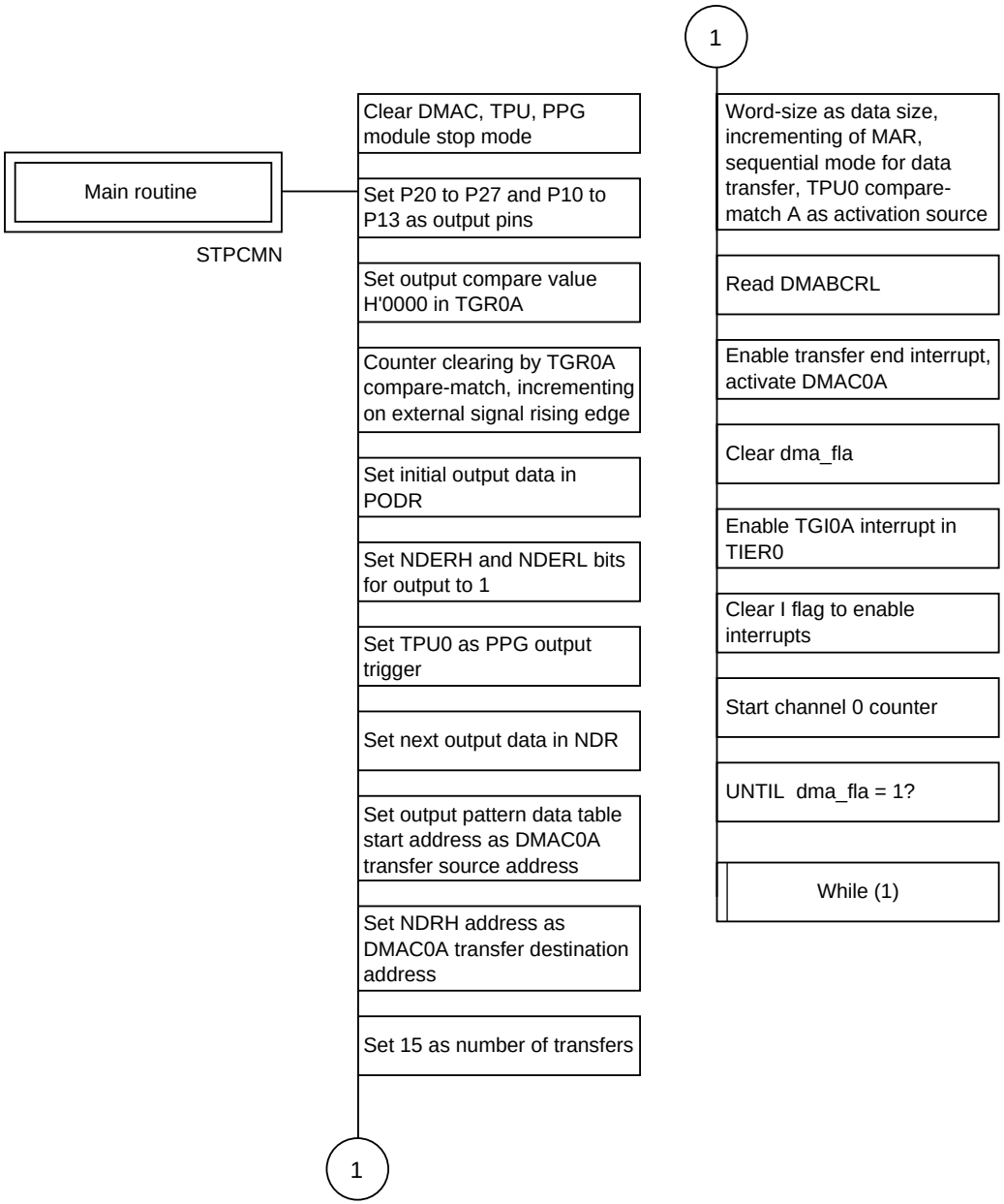
This sample task does not use RAM.

5. Data Table

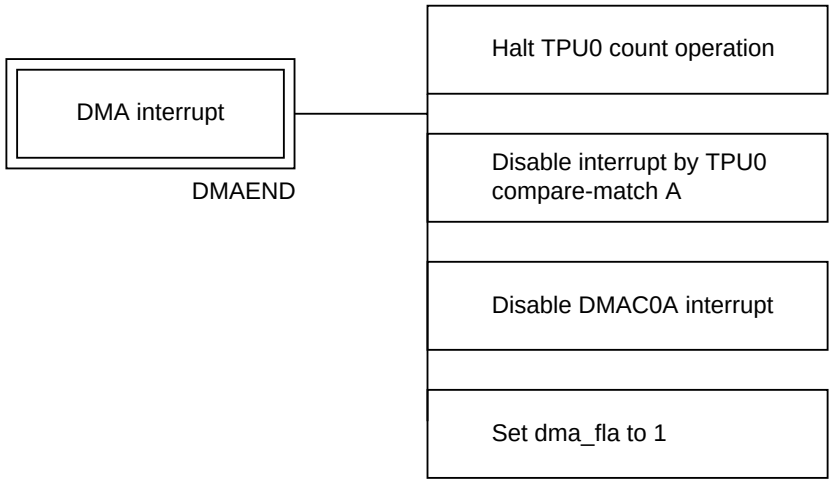
Table Name	Function	Data Length	Data Capacity
opat_tab	Stores data to be output by PPG	Unsigned short	30 bytes

PAD

1. Main routine



2. DMA interrupt



Program List

```
#include <machine.h>
#include "H8S.H"
/*********************************/
/*           PROTOCOL            */
/*********************************/
void STPCMN(void);
#pragma interrupt (DMAEND)
/*********************************/
/*     SYMBOL DEFINITIONS        */
/*********************************/

# define opt_tab   ((unsigned short * )0xffec00) /* Output data table */
# define dma_fla   (*(unsigned char * )0xffec1e) /* DMAC end flag */

/*********************************/
/*     MAIN PROGRAM: STPCM       */
/*********************************/
void STPCMN(void)
{
        MSTPCR = 0x17ff;         /* Disable module(DMA,TPU,PPG) stop mode*/

        P1DDR = 0x4f;            /* P1:output port */
        P2DDR = 0xff;            /* P2:output port */

        TIOR0H = 0x00;           /* Initialize TIOR0H */
        TGR0A = 0x0000;          /* Set non overlap time */
        TPU_TCR0 = 0x27;         /* Initialize TCR0 */

        PODRH = 0x00;            /* Output first data */
        PODRL = 0x00;            /* Output first data */
        NDERH = 0x0f;            /* Enable next data output */
        NDERL = 0xff;            /* Enable next data output */
        PCR = 0x00;              /* Output toriga TPU0'S compare match */
        NDRH = 0xff;             /* Set second output data */
        NDRL = 0xff;             /* Set second output data */

        MAR0A_W = opt_tab;       /* Set base address */
        IOAR0A = 0xff4c;         /* Set excute address */
        ETCR0A = 0x000f;         /* Set excute count */
        DMACR0A = 0x88;          /* Initialize DMACR0A */
        DMABCRH = 0x01;          /* Initialize DMABCRH */
        DMABCRL |= 0x11;         /* Initialize DMABCRL */

        dma_fla = 0;             /* Clear dma_fla */
        TIER0_BP.TGIEA0=1;       /* Enable TGI0EA interrupt */
        set_imask_ccr(0);        /* Enable interrupt */
        TSTR = 0x01;             /* Start TCNT0 */
        while(dma_fla==0);       /* DMAC end? */
        while(1);                /* Loop */
}

/*********************************/
/*   INTERRUPT PROGRAM: DMAEND   */
/*********************************/
```

```
void DMAEND(void)
{
        TSTR_BP.CST0 = 0;         /* Stop TCNT0 */
        TIER0_BP.TGIEA0=0;        /* Disable timer compare match interrupt */
        DMABCR_BP.DTIE0A=0;       /* Disable DMAC end interrupt */
        DMABCR_BP.DTE0A=0;        /* Disable data translation */
        dma_fla =1;               /* Set dma_fla */
}
```

Specifications

1. The H8S/2655's SCI is set to clock synchronous mode, and performs continuous transmission and continuous reception of 48-byte data to/from an H8/3314 chip.

2. The DMAC is used to transfer data from memory to TDR and from RDR to memory without CPU intervention.

3. The transmitting device is the clock master.

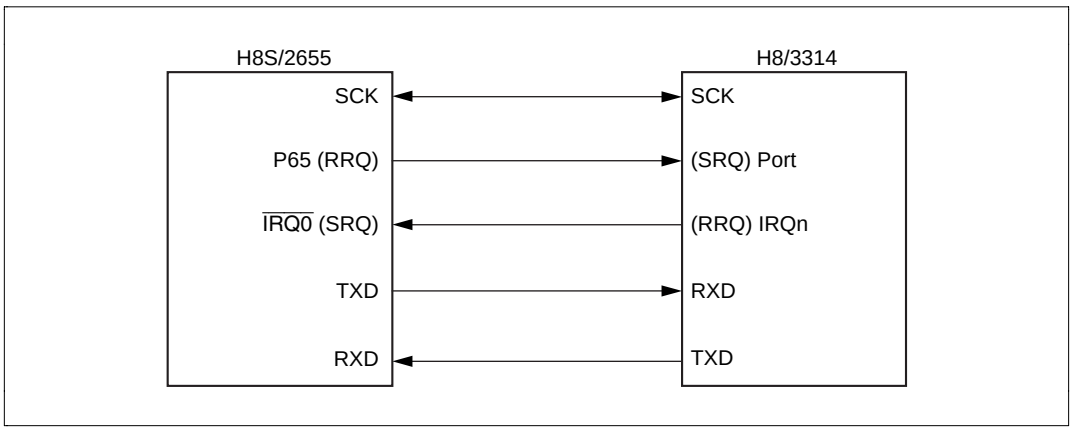


Figure 1 H8S/2655 Clock Synchronous SCI Block Diagram

Functions Used

1. Figure 2 shows the H8S/2655 on-chip functions used by this sample task. DMAC0A, DMAC0B, and SCI1 are used as shown in this figure to perform high-speed serial communication.

   Data Buffer

   Buffer RAM that stores the transmit/receive data

   DMAC0A

   Operates in sequential mode. DMAC0A is activated by an SCI transmit end interrupt, and transfers the contents of the transmit data buffer to the SCI.

   DMAC0B

   Operates in sequential mode. DMAC0B is activated by an SCI receive end interrupt, and transfers receive data to the receive data buffer.

SCI1

Performs serial data transmission and reception.

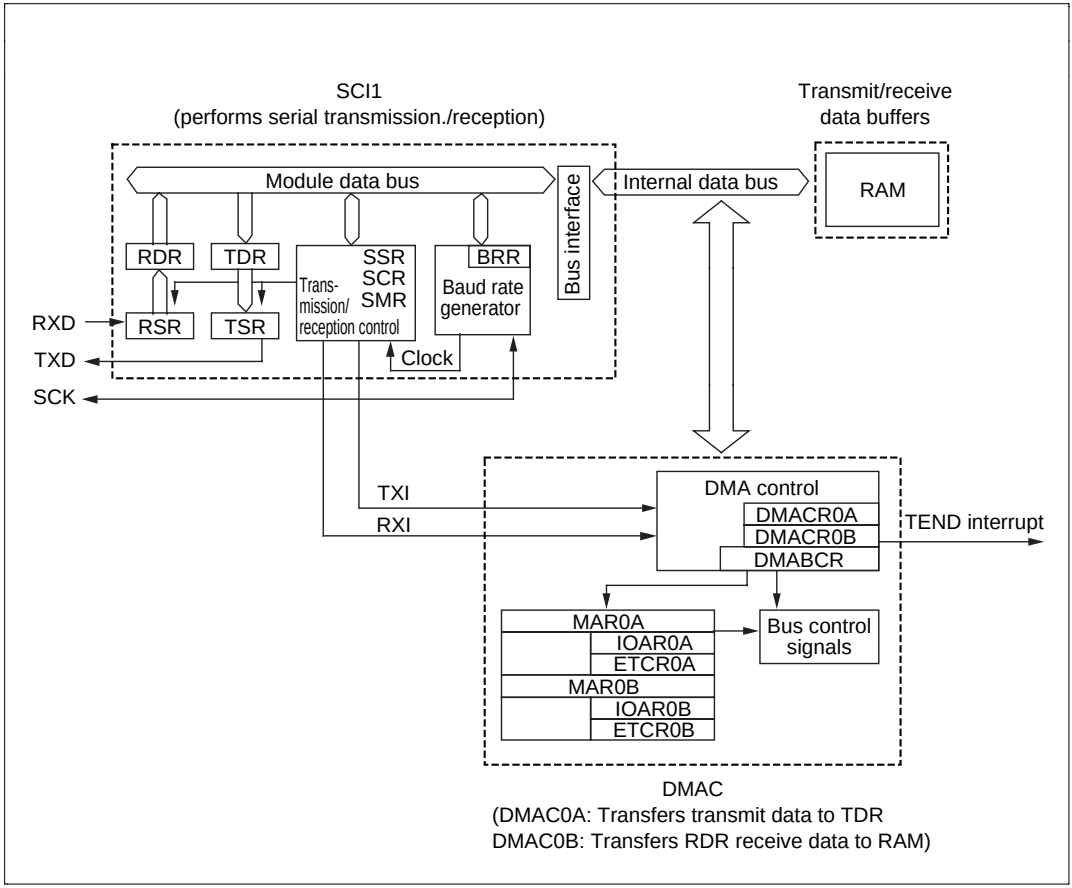


Figure 2 SCI Continuous Transmission/Reception Block Diagram

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to transfer transmit/receive data without CPU intervention.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
Interrupt controller	ISCRL	Selects interrupt generation on falling edge of IRQ0 input
	IER	Enables IRQ0 interrupt
	ISR	Indicates IRQ0 interrupt request status
SCI1	SCK1	Transmits serial clock; also receives serial clock during reception
	RXD1	Receive data input pin
	TXD1	Transmit data output pin
	SMR1	Sets SCI to clock synchronous mode
	SCR1	Transmission and reception settings
	SSR1	Indicates transmission/reception status
	RDR1	Stores received data
	TDR1	Holds data to be transmitted
	BRR1	Transfer rate setting
Port 6	P6DDR	Port 6 input/output setting
	P6DR	RRQ transmission
DMAC	DMABCR	Controls operation of each channel
	DMACR0A	Controls DMAC0A operation
	MAR0A	Transfer source address (data buffer) setting
	IOAR0A	Transfer destination address (TDR) setting
	ETCR0A	Transfer number setting
	DMACR0B	Controls DMAC0B operation
	MAR0B	Transfer destination address (data buffer) setting
	IOAR0B	Transfer source address (RDR) setting
	ETCR0B	Transfer number setting

Operation

1. Data transmission

Figure 3 shows the principles of operation in data transmission. Interfacing is performed by controlling the I/O ports and clock synchronous SCI, using the timing shown in this figure.

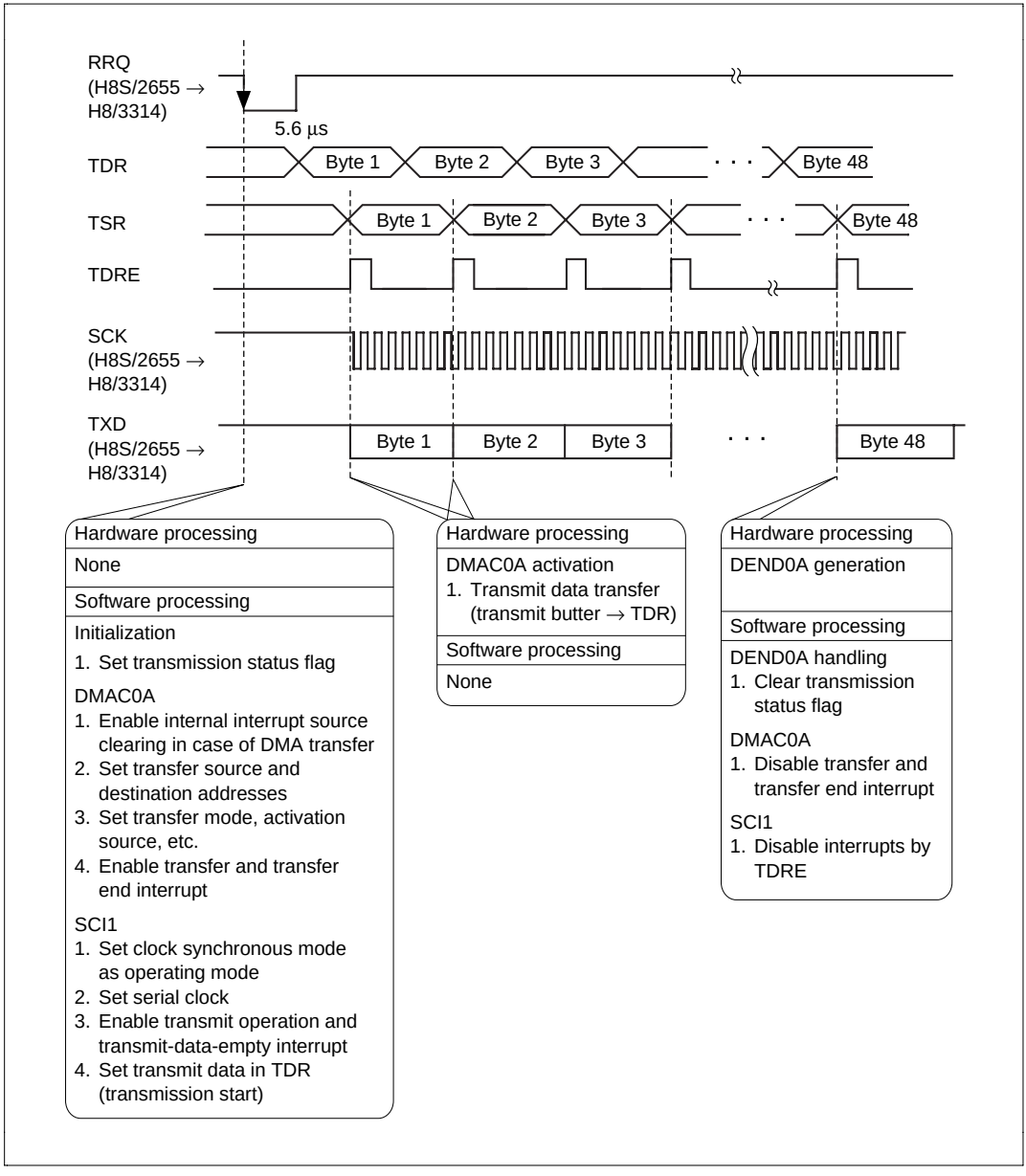


Figure 3 Principles of Data Transmission Operation

2. Data reception

Figure 4 shows the principles of operation in data reception. Interfacing is performed by controlling the I/O ports and clock synchronous SCI, using the timing shown in this figure.

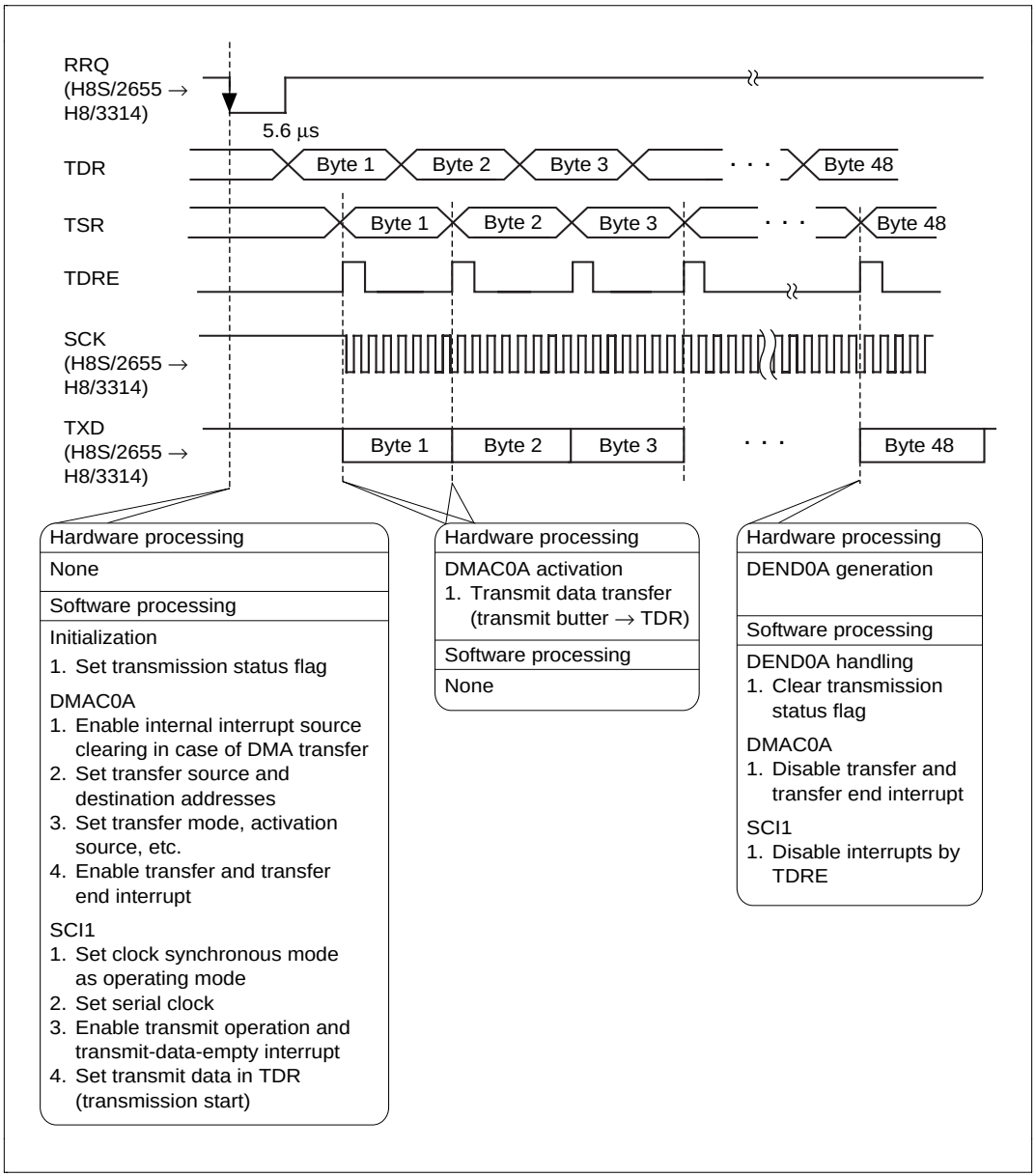


Figure 3 Principles of Data Reception Operation

Software

1. Modules

Module Name	Label	Function
Main routine	hiscimn	I/O port, SCI, DMAC initialization
Data transmission	txstart	Enables DMAC transfer, and starts SCI transmit operation
Data reception	rxstart	Initiated by IRQ0 interrupt: enables DMAC transfer and starts SCI receive operation
Transmit end	txend	Initiated by DMAC0A transfer end interrupt; disables stat_tx clearing and transmission processing
Receive end	rxend	Initiated by DMAC0B transfer end interrupt; disables stat_rx clearing and reception processing

2. Arguments

Label/Register Name	Function	Data Length	Module	Input/ Output
stat_tx	Flag indicating transmission in progress	Unsigned char	Data transmission Data reception	Output Input
stat_rx	Flag indicating reception in progress	Unsigned char	Data transmission Data reception	Input Output

3. Internal Registers Used

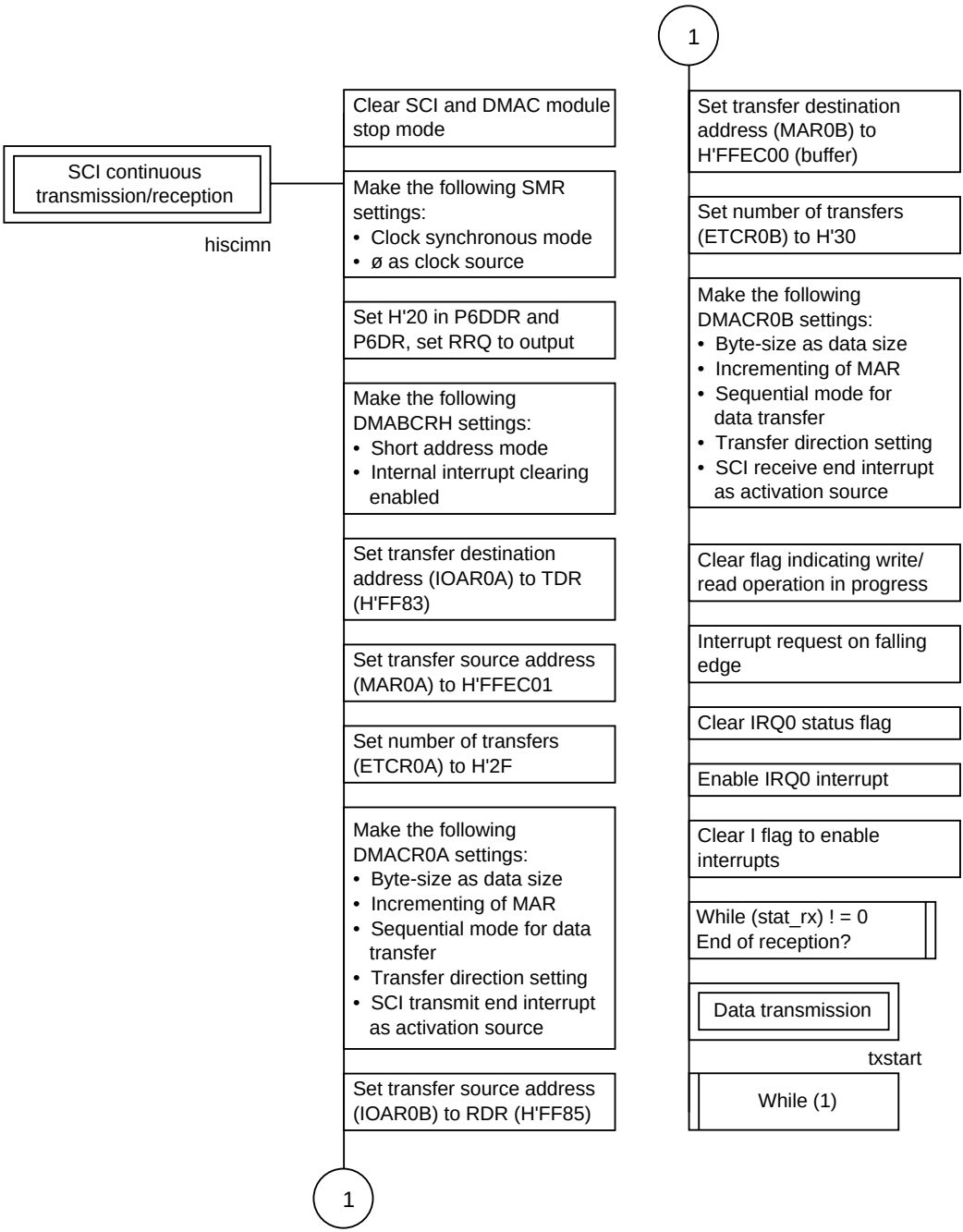
On-Chip Function	Register Name	Function
SCI1	SMR1	Makes the following SCI settings: • Sets clock synchronous mode as SCI operating mode • Sets ø as baud rate generator clock source
	SCR1	Makes the following SCI settings for transmission and reception: • Transmit operation Enables transmit-data-empty interrupt Enables transmit operation SCK pin as serial clock output • Receive operation Enables receive-data-full interrupt Enables receive operation SCK pin as serial clock input
	SSR1	• Transmit operation Clears TDRE, starting transmit operation • Receive operation

		Clears RDRF, enabling receive operation
On-Chip Function	**Register Name**	**Function**
SCI1	RDR1	Stores received data
	TDR1	Holds data to be transmitted
	BRR1	Transfer rate setting
DMAC	DMABCR	Makes the following DMAC0A and DMAC0B settings: • Sets short address mode as operating mode • Enables internal interrupt source clearing in case of DMA transfer • Enables data transfer and transfer end interrupt
	DMACR0A	Makes the following DMAC0A settings: • Sets byte-size as data size • Sets incrementing of MAR • Sets sequential mode for data transfer • Sets data transfer direction (channel 0A: MAR → IOAR) • Sets SCI transmit end interrupt as activation source
	MAR0A	Transmit buffer address setting
	IOAR0A	TDR address setting
	ETCR0A	Transfer number setting
	DMACR0B	Makes the following DMAC0B settings: • Sets byte-size as data size • Sets incrementing of MAR • Sets sequential mode for data transfer • Sets data transfer direction (channel 0B: IOAR → MAR) • Sets SCI receive end interrupt as activation source
	MAR0B	Receive buffer address setting
	IOAR0B	RDR address setting
	ETCR0B	Transfer number setting
I/O	P6DDR	Port 6 input/output setting
	P6DR	RRQ transmission
Interrupt controller	IER	Enables IRQ0 interrupt
	ISCR	Sets interrupt request generation on falling edge of IRQ0 input
	ISR	Indicates IRQ0 input status
	MSTPCR	Clears SCI and DMAC module stop mode

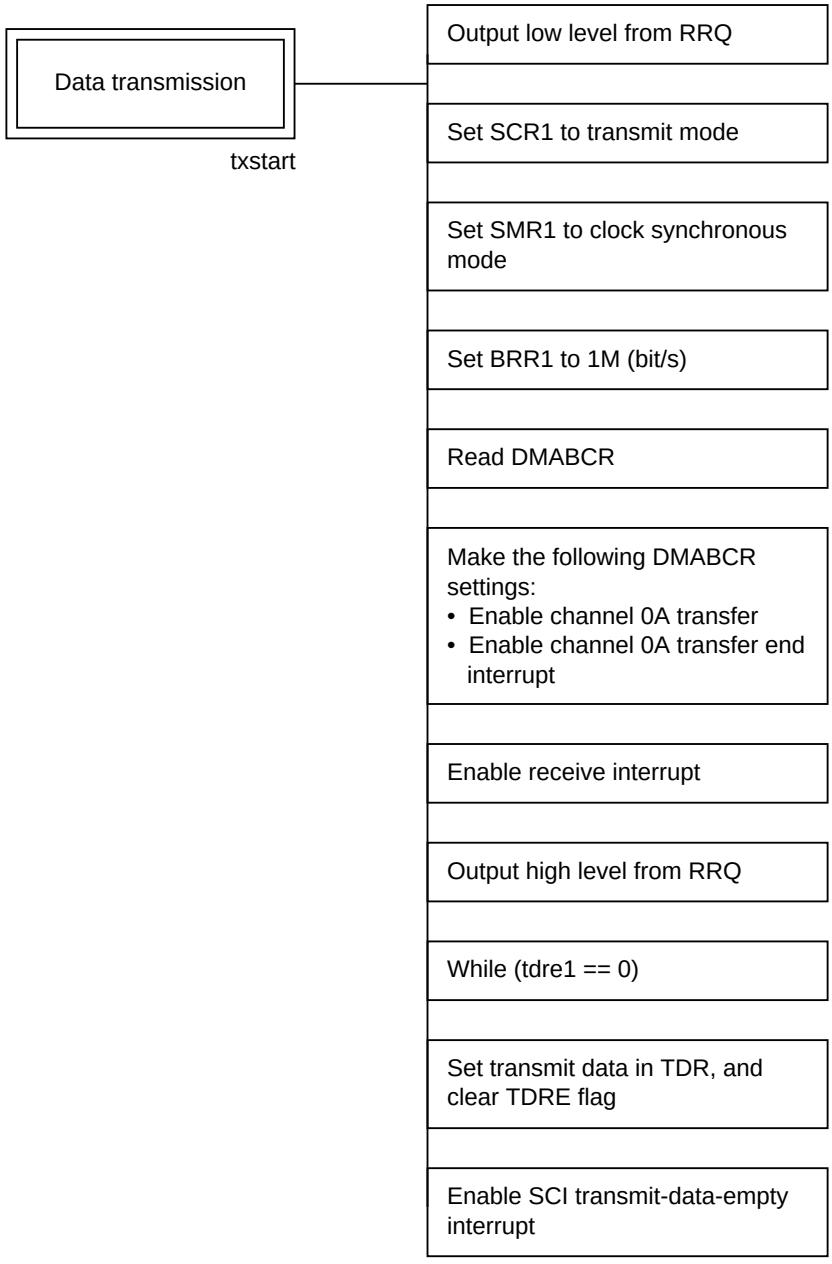
4. RAM Used

Label/Register Name	Function	Data Length	Module
buffer	Stores transmit/receive data	48 bytes	Data transmission

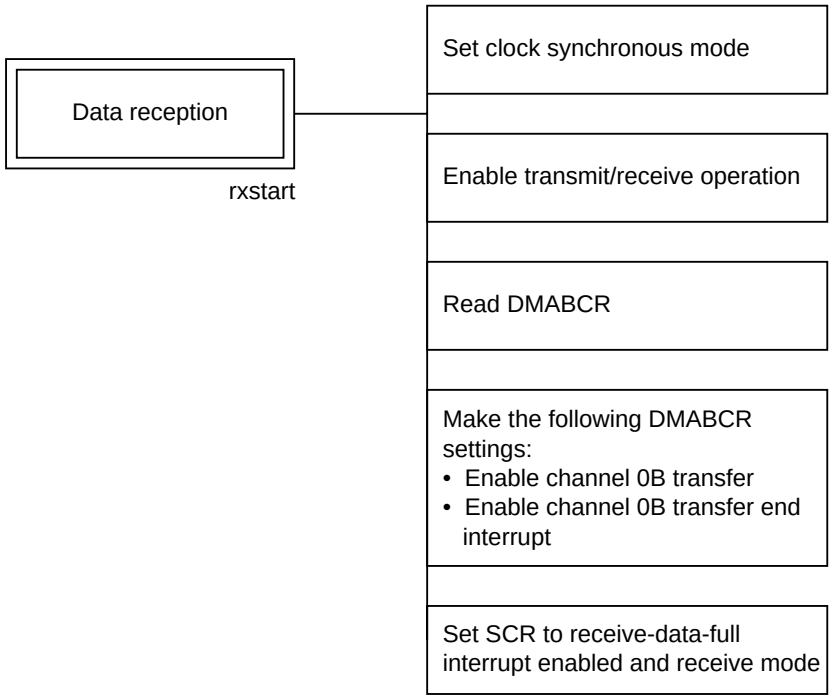
1. Main routine



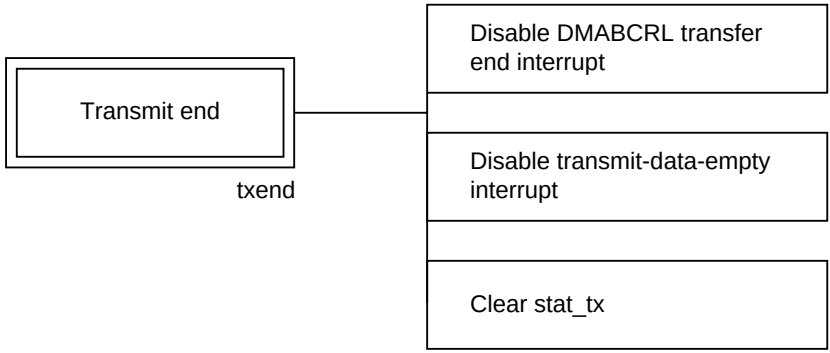
2. Data transmission



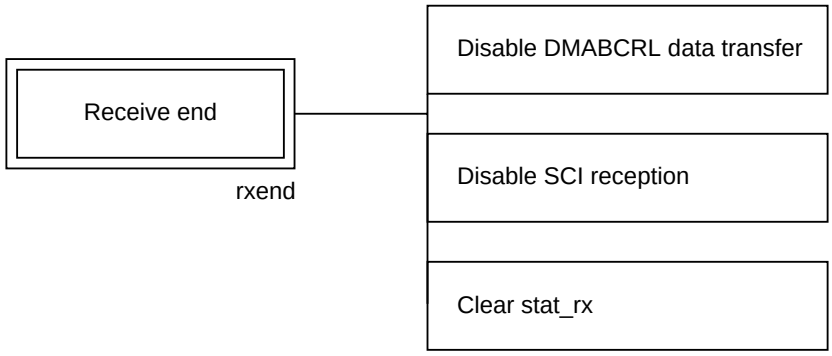
3. Data reception



4. Transmit end



5. Receive end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTCOL                    */
/****************************************/
void hiscimn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
#define buffer (*(volatile unsigned char *)0xffec00)
#define stat_rx (*(volatile unsigned char *)0xffec30)
#define stat_tx (*(volatile unsigned char *)0xffec31)

/****************************************/
/*     MAIN PROGRAM : hiscimn           */
/****************************************/
void hiscimn(void)
{
        MSTPCR = 0x7fbf;
        SCR1 = 0x00;
        P6DDR = 0x20;                   /* init port */
        P6DR = 0x20;

        DMABCRH = 0x03;
        IOAR0A = 0xff83;
        MAR0A = 0xffec01;
        ETCR0A = 0x2f;
        DMACR0A = 0x06;

        IOAR0B = 0xff85;
        MAR0B = (long)(&buffer);
        ETCR0B = 0x30;
        DMACR0B = 0x17;

        stat_rx = 0xff;
        stat_tx = 0xff;

        ISCRL = 0x01;
        ISR_BP.IRQ0F = 0;
        IER = 0x01;

        set_imask_ccr(0);

        while(stat_rx != 0);            /* receive complete? */
        txstart();

        while(1);
}
```

```
/****************************************/
/* NAME : txstart(set transmit data )     */
/****************************************/
txstart()
{
        P6DR_BP.RRQ = 0;
        SCR1 = 0x00;                    /* select clock mode */
        SMR1 = 0x80;                    /* init SCI1 */
        BRR1 = 0x04;                    /* set 1MBPS */
        DMABCRL |= 0x11;
        SCR1 = 0x20;
        P6DR_BP.RRQ = 1;

        while(SSR1_BP.TDRE1 == 0);
        TDR1 = buffer;                  /* set transmit data to TDR */
        SSR1_BP.TDRE1 = 0;              /* start transmit */
        SCR1_BP.TIE1 = 1;
}

/****************************************/
/* NAME : rxstart(set SCI to receive operation) */
/****************************************/
#pragma interrupt(rxstart)
void rxstart(void)
{
        SCR1 &= 0xcf;
        SMR1 = 0x80;                    /* init SCI1 */
        DMABCRL |= 0x22;
        SCR1 = 0x52;
}

/****************************************/
/* NAME : txend(set transmit end flag)    */
/****************************************/
#pragma interrupt(txend)
void txend(void)
{
        DMABCRL &= 0xee;
        SCR1_BP.TIE1 = 0;
        stat_tx = 0;
}

/****************************************/
/* NAME : rxend(set receive end flag)     */
/****************************************/
#pragma interrupt(rxend)
void rxend(void)
{
        SCR1 = 0x00;
        DMABCRL &= 0xdd;
        stat_rx = 0;
}
```

4.3 Four-Phase Stepping Motor Application Example

TPU, PPG, DTC

Specifications

1. The H8S/2655's TPU, PPG, and DTC on-chip functions are used to control four 4-phase stepping motors, as shown in figure 1.
2. The stepping motors are controlled by means of two-phase excitation.
3. This task repeats the following cycle of stepping motor operations: stop → forward → stop → reverse → stop.
4. The task performs slew-up and slew-down processing without software intervention.
5. A through-current prevention period is provided for driver protection.

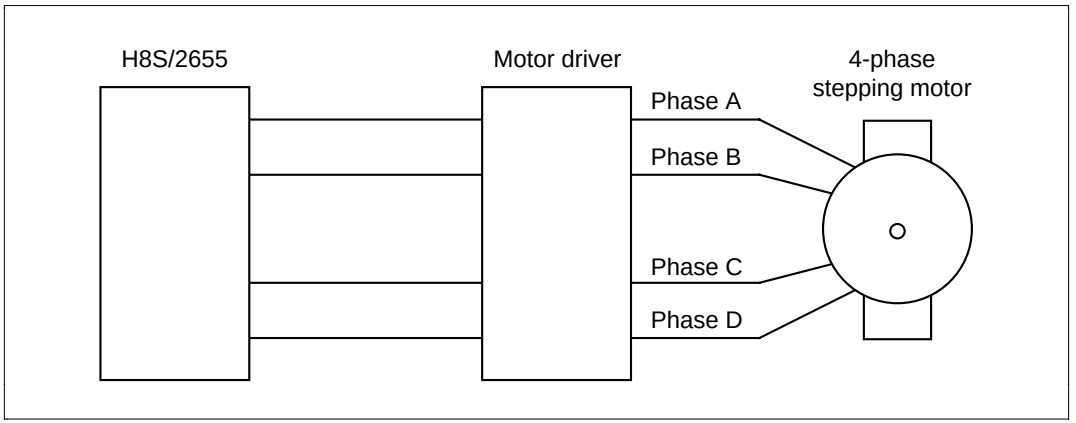


Figure 1 Four-Phase Stepping Motor Control Circuit Diagram

Design Concept

1. Example of stepping motor operation

   Figure 2 shows an example of 4-phase stepping motor operation using two-phase excitation. The operating sequence is as follows:

   a. When a pulse is high, the corresponding phase is excited, as shown in figure 2.

   b. In (1) in the figure, phases D and A are excited simultaneously, and the rotor is positioned midway between these two phases.

c. In (2) in the figure, phases A and B are excited simultaneously, and the rotor is positioned midway between these two phases. Two-phase excitation continues in this way, with adjacent phases excited in succession (phases D and A, A and B, B and C, and C and D), so rotating the rotor.

d. For reverse operation, the rotor is rotated in the opposite direction by exciting the phases in the order D and C → C and B → B and A → A and D.

e. The stepping motor is stopped by continuing excitation of the last phase of forward or reverse operation for a given period.

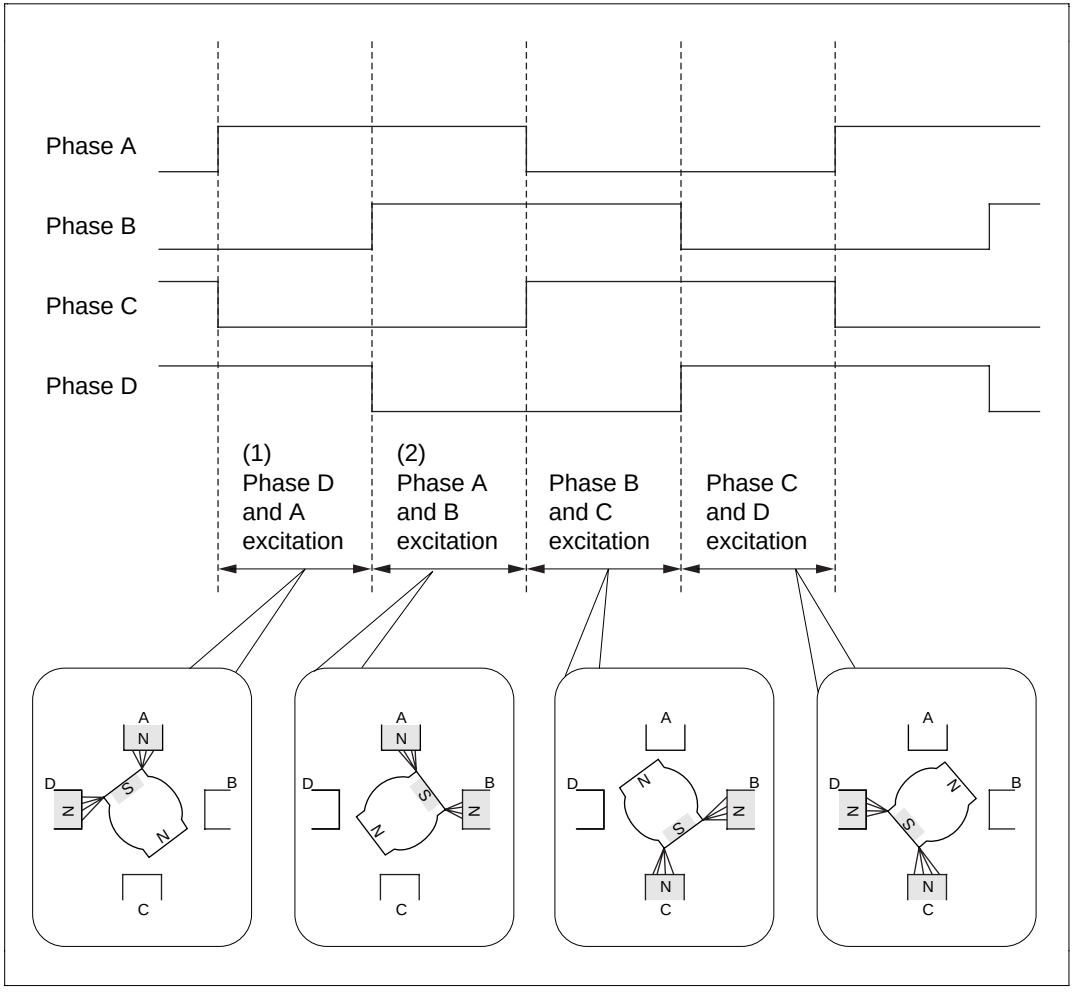


Figure 2 Example of Stepping Motor Operation

2. Non-overlap time

As shown in figure 3, a through-current prevention period (non-overlap time) *n* is inserted when the output pattern is switched. Providing a time lag in this way eliminates the risk of damage to the driver due to turn-off delay when switching the excitation phase.

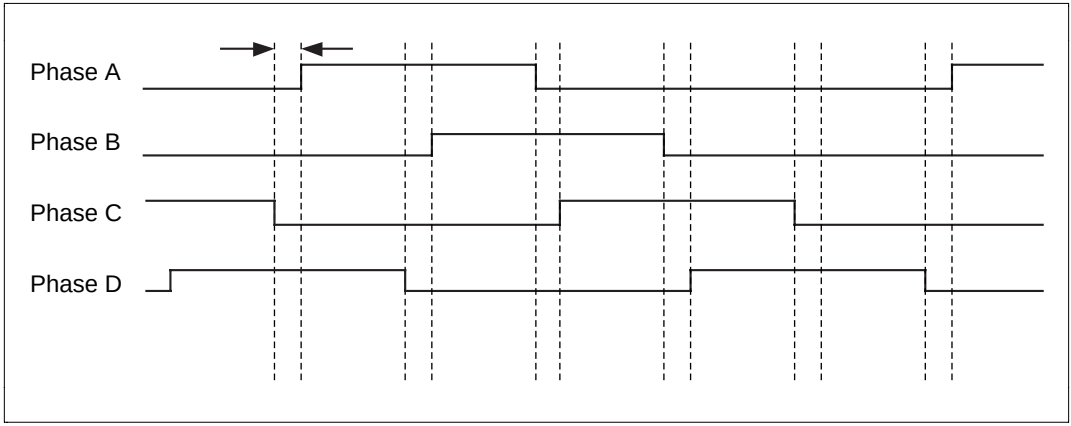


Figure 3 Example of Non-Overlap Time Output

3. Slew-up and slew-down

Pulses are output with controlled acceleration and deceleration as shown in figure 4. Slewing up and down in this way prevents motor step-out.

If short-cycle pulses are output suddenly when the motor is run, it may not be able to keep up with the load, and so fail to rotate. Slew-up and slew-down operations are performed to avoid this problem.

The principles of the operation are as follows:

a. The set number of pulses are output while gradually shortening the pulse cycle (slew-up).

b. The set number of pulses are output with a fixed pulse cycle (constant-speed operation).

c. The set number of pulses are output while gradually lengthening the pulse cycle (slew-down).

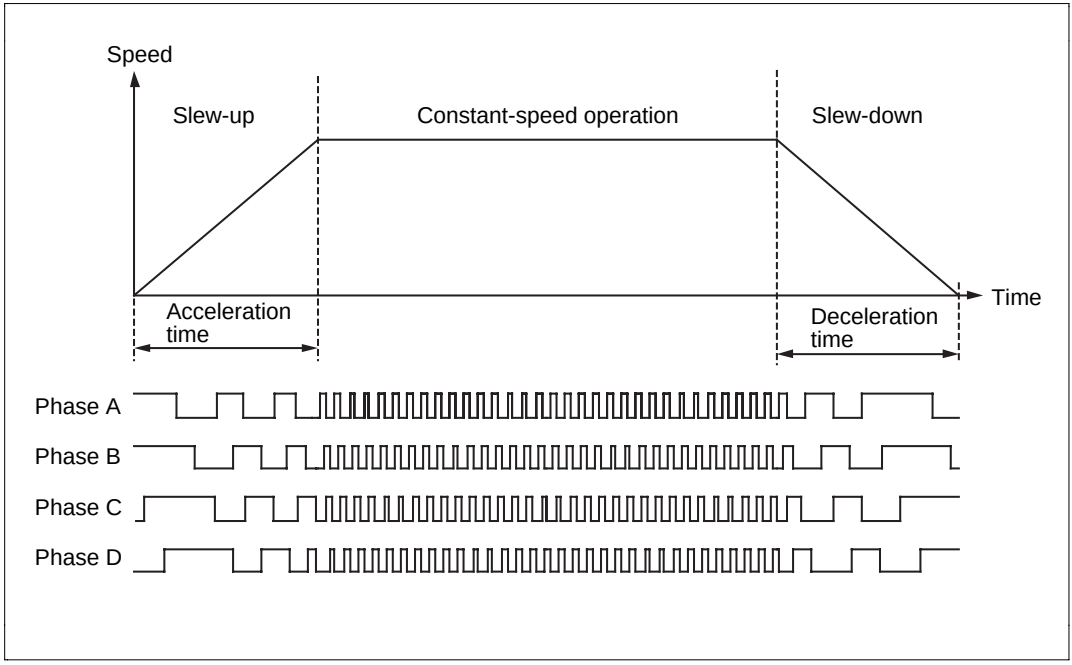


Figure 4 Slew-Up and Slew-Down

Functions Used

1. This sample task uses the H8S/2655's DTC to transfer a 4-phase output pattern to the PPG and the pulse cycle to TGRB in the TPU on generation of TPU compare-match A, and generate 4-phase pulse output.

   a. Figure 5 shows a block diagram of the H8S/2655 on-chip functions used by this sample task. The following functions are used to generate motor output waveforms.

      - DTC

        Activated by TPU compare-match A.

        Transfers output data from the output pattern data table to NDR in the PPG. After this transfer, the DTC transfers pulse cycle data from the cycle data table to TGRB in the TPU using chain transfer.

      - TPU

        Compare-match A: Activates the DTC and PPG.

        Compare-match B: Performs timer counter clearing, and also activates the PPG.

      - PPG

        In this sample task, the PPG outputs 16-bit pulses with a non-overlap interval.

        Compare-match B: Performs pulse output with a waveform changing from high to low. Output of pulses with a waveform changing from low to high is held pending.

        Compare-match A: Performs low-to-high output held pending on compare-match B (with the delay specified by TGRA).

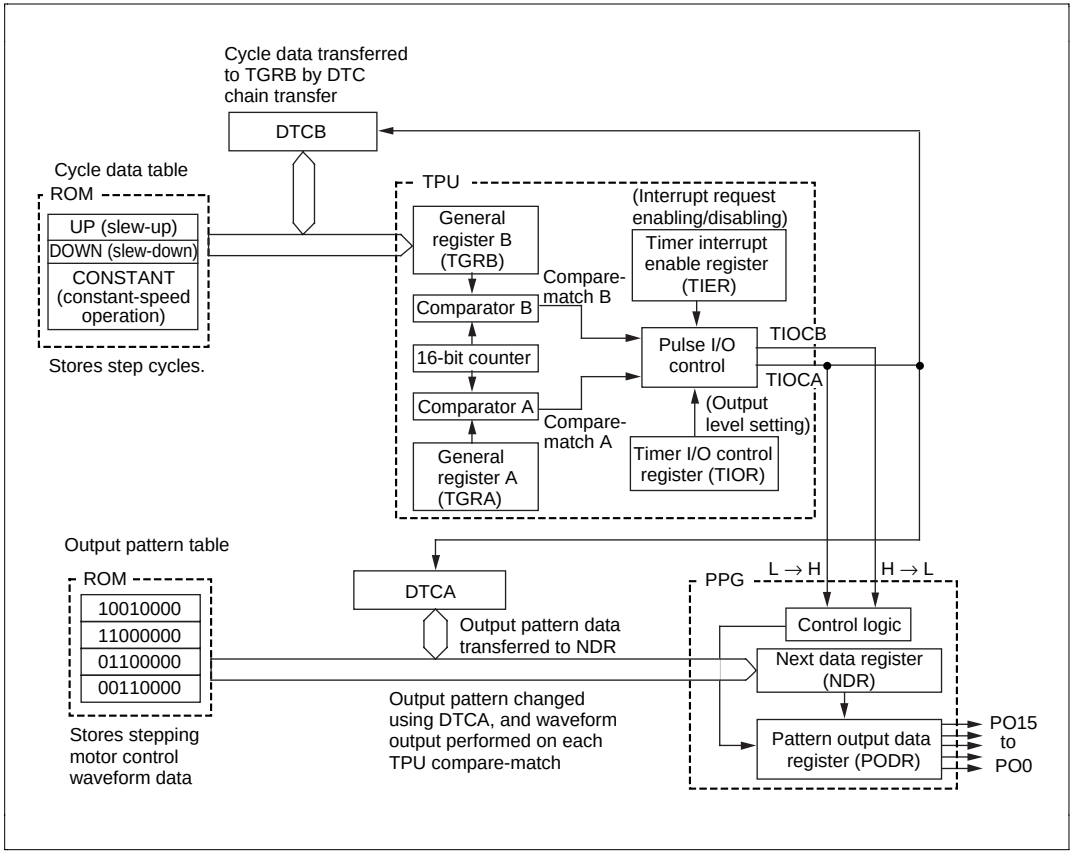


Figure 5 Four-Phase Stepping Motor Control Block Diagram

b. Figure 6 shows the DTC vector table and memory allocation. DTC register information is located from address H'FFF800 in the following order: MRA, SAR, MRB, DAR, CRA, CRB.

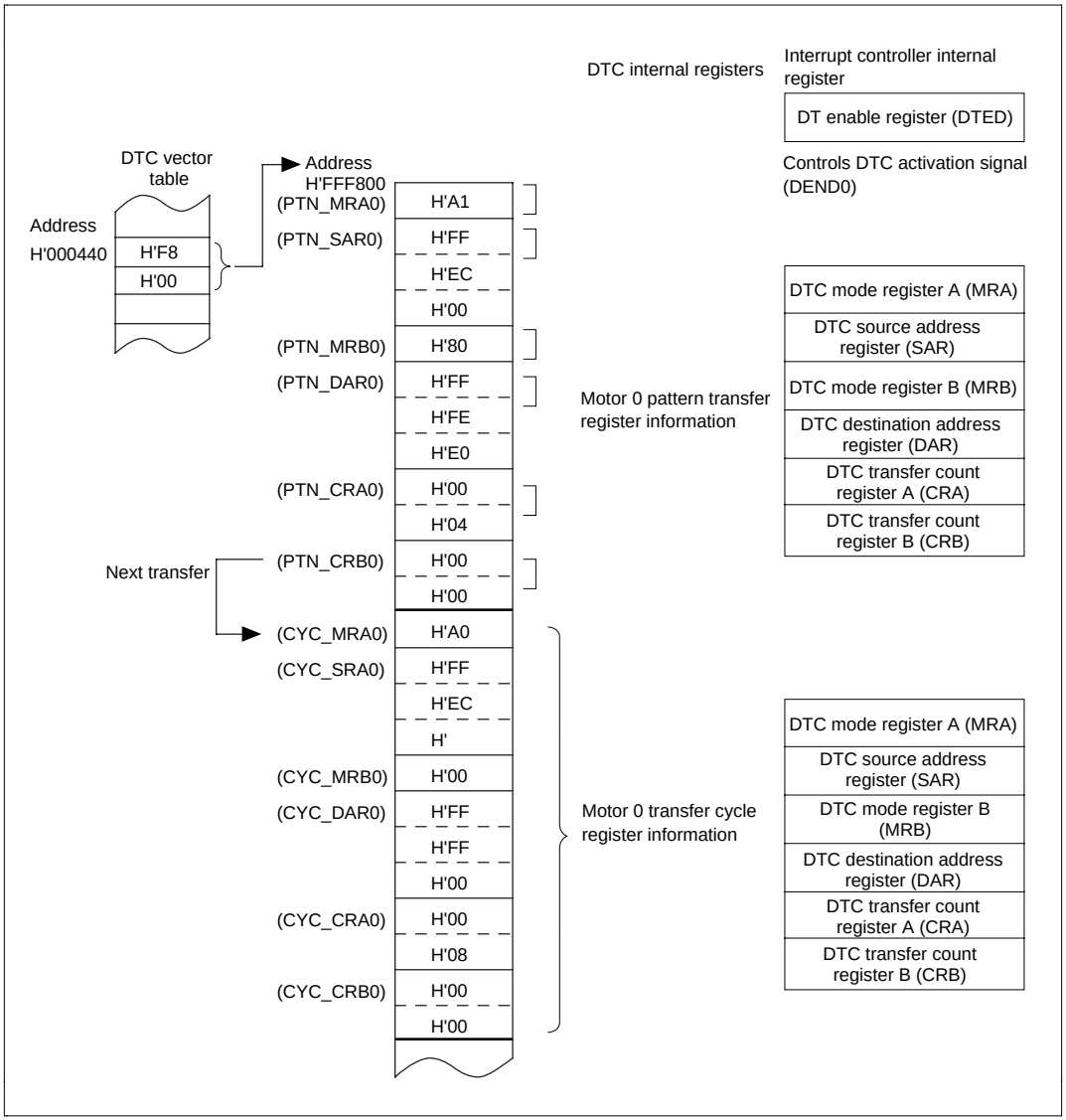


Figure 6 Example of DTC Vector Table and Memory Allocation

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to generate two-phase excitation stepping motor output waveforms.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
TPU	TCNT	16-bit counter
	TGRA	Output compare register A
	TGRB	Output compare register B
	TCR	Count clock selection, counter clear source selection
	TIOR	Sets TGRA and TGRB as output compare registers
	TSR	Compare-match and overflow status register
DTC	MRA	Controls DTC operating mode
	MRB	Specifies DTC chain transfer
	DTCER	Enabling/disabling of DTC activation by each interrupt source
	DTVECR	Sets vector number for software activation interrupt
	SAR	Transfer source address setting
	DAR	Transfer destination address setting
	CRA	Transfer number setting
PPG	PODR	Stores PPG output data
	NDR	Stores next PPG output data
	PO15 to PO0	4-phase stepping motor output waveform generation

Operation

1. Example of 4-phase pulse output

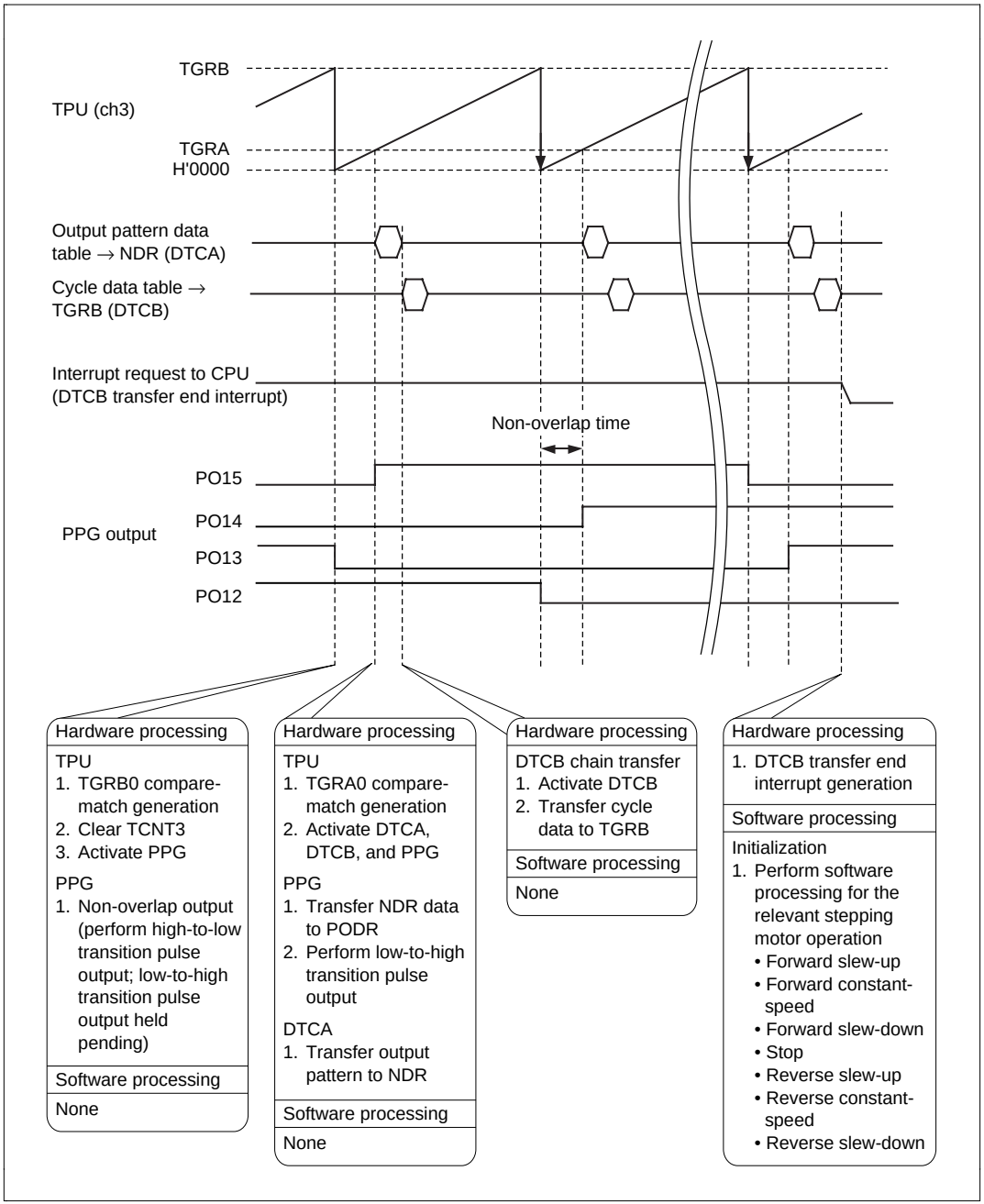


Figure 7 Stepping Motor Timing

Software

1. Modules

Module Name	Label	Function
Main routine	stp4mn	TPU, PPG, DTC initialization, forward slew-up operation setting
Motor control	mtrctl0 mtrctl1 mtrctl2 mtrctl3	Control motor operations
Forward slew-up	fslueup0 to fslueup3	Initiated after end of reverse stop operation; performs forward slew-up operation DTC transfer mode switching
Forward constant-speed	fconst0 to fconst3	Initiated after end of forward slew-up operation; performs forward constant-speed operation DTC transfer mode switching
Forward slew-down	fsldwn0 to fsldwn3	Initiated after end of forward constant-speed operation; performs forward slew-down operation DTC transfer mode switching
Forward stop	fstop0 to fstop3	Initiated after end of forward slew-down operation; performs forward stop operation DTC transfer mode switching
Reverse slew-up	rslueup0 to rslueup3	Initiated after end of forward stop operation; performs reverse slew-up operation DTC transfer mode switching
Reverse constant-speed	rconst0 to rconst3	Initiated after end of reverse slew-up operation; performs reverse constant-speed operation DTC transfer mode switching
Reverse slew-down	rsludwn0 to rsludwn3	Initiated after end of reverse constant-speed operation; performs reverse slew-down operation DTC transfer mode switching
Reverse stop	rstop0 to rstop3	Initiated after end of reverse slew-down operation; performs reverse stop operation DTC transfer mode switching

2. Arguments

This task does not use any arguments.

On-Chip Function	Register Name	Function
TPU	TGRA	Non-overlap time setting
	TGRB	Timer cycle setting
	TIER	Enables TGFA interrupt
	TCR	Makes the following TPU settings: • Counter clearing by TGRB compare-match • Counting on ø internal clock
	TIOR	Sets TGRA and TRGB as output compare registers, disables pin output
	TSTR	Enables TCNT count operation
DTC	DTCER	Enables DTC activation by TGIA interrupt
PPG	PODR	Stores output pattern data
	PMR	Sets PO15 to PO0 as non-overlap outputs
	PCR	Selects pulse output trigger signal for each group Group 3: TPU3 compare-match Group 2: TPU2 compare-match Group 1: TPU1 compare-match Group 0: TPU0 compare-match
	NDERL	Enables PPG outputs PO7 to PO0
	NDERH	Enables PPG outputs PO15 to PO8
	NDRL	Stores next output pattern data for PO7 to PO0
	NDRH	Stores next output pattern data for PO15 to PO8
	MSTPCR	Clears TPU, DTC, and PPG module stop mode

4. RAM Used

Label	Function	Data Length	Module
nextmode0 nextmode1 nextmode2 nextmode3	Indicate the stepping motor operation status H'01: Forward slew-up control H'02: Forward constant-speed control H'03: Forward slew-down control H'04: Forward stop control H'05: Reverse slew-up control H'06: Reverse constant-speed control H'07: Reverse slew-down control H'08: Reverse stop control	Unsigned char	Motor control
PTN_MRA0	Sets repeat mode as transfer mode	Unsigned char	Main routine
PTN_MRB0	Enables chain transfer	Unsigned char	Main routine
PTN_SAR0	Transfer source address (PATTBL) setting	Unsigned long	Main routine
PTN_DAR0	Transfer destination address (NDR0) setting	Unsigned long	Main routine
PTN_CRA0	Block size setting	Unsigned short	Main routine
CYC_MRA0	Sets normal mode as transfer mode	Unsigned char	Main routine
CYC_MRB0	Enables interrupt request to CPU at end of transfer	Unsigned char	Main routine
CYC_SAR0	Transfer source address (UPTBL) setting	Unsigned long	Main routine
CYC_DAR0	Transfer destination address (TGRB0) setting	Unsigned long	Main routine
CYC_CRA0	Transfer number setting	Unsigned short	Main routine
PTN_MRA1	Sets repeat mode as transfer mode	Unsigned char	Main routine
PTN_MRB1	Enables chain transfer	Unsigned char	Main routine
PTN_SAR1	Transfer source address (PATTBL) setting	Unsigned long	Main routine
PTN_DAR1	Transfer destination address (NDR1) setting	Unsigned long	Main routine
PTN_CRA1	Block size setting	Unsigned short	Main routine
CYC_MRA1	Sets normal mode as transfer mode	Unsigned char	Main routine
CYC_MRB1	Enables interrupt request to CPU at end of transfer	Unsigned char	Main routine
CYC_SAR1	Transfer source address (UPTBL) setting	Unsigned long	Main routine
CYC_DAR1	Transfer destination address (TGRB1) setting	Unsigned long	Main routine
CYC_CRA1	Transfer number setting	Unsigned short	Main routine
PTN_MRA2	Sets repeat mode as transfer mode	Unsigned char	Main routine
PTN_MRB2	Enables chain transfer	Unsigned char	Main routine
PTN_SAR2	Transfer source address (PATTBL) setting	Unsigned long	Main routine
PTN_DAR2	Transfer destination address (NDR2) setting	Unsigned long	Main routine

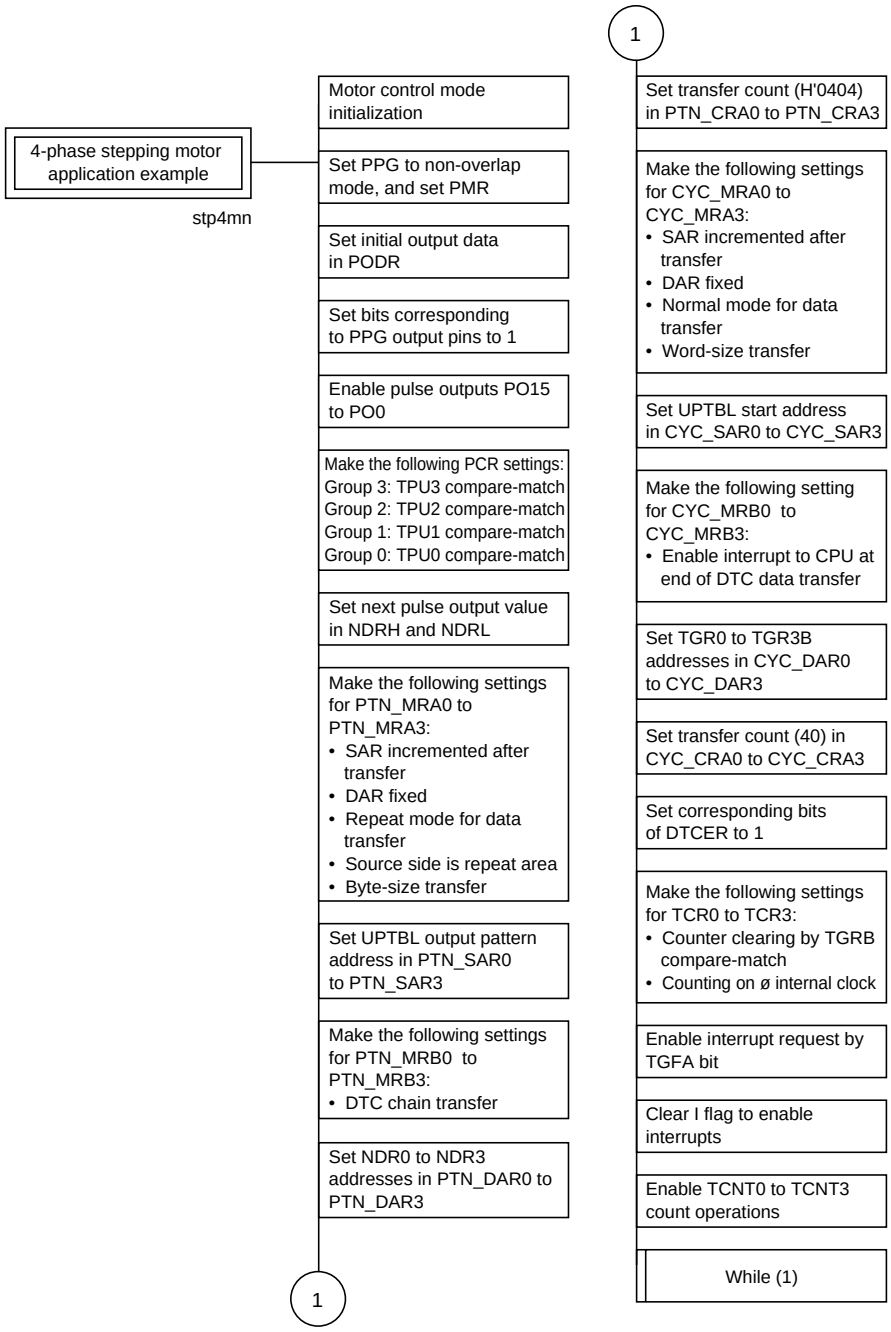
Label	Function	Data Length	Module
PTN_CRA2	Block size setting	Unsigned short	Main routine
CYC_MRA2	Sets normal mode as transfer mode	Unsigned char	Main routine
CYC_MRB2	Enables interrupt request to CPU at end of transfer	Unsigned char	Main routine
CYC_SAR2	Transfer source address (UPTBL) setting	Unsigned long	Main routine
CYC_DAR2	Transfer destination address (TGRB2) setting	Unsigned long	Main routine
CYC_CRA2	Transfer number setting	Unsigned short	Main routine
PTN_MRA3	Sets repeat mode as transfer mode	Unsigned char	Main routine
PTN_MRB3	Enables chain transfer	Unsigned char	Main routine
PTN_SAR3	Transfer source address (PATTBL) setting	Unsigned long	Main routine
PTN_DAR3	Transfer destination address (NDR3) setting	Unsigned long	Main routine
PTN_CRA3	Block size setting	Unsigned short	Main routine
CYC_MRA3	Sets normal mode as transfer mode	Unsigned char	Main routine
CYC_MRB3	Enables interrupt request to CPU at end of transfer	Unsigned char	Main routine
CYC_SAR3	Transfer source address (UPTBL) setting	Unsigned long	Main routine
CYC_DAR3	Transfer destination address (TGRB3) setting	Unsigned long	Main routine
CYC_CRA3	Transfer number setting	Unsigned short	Main routine

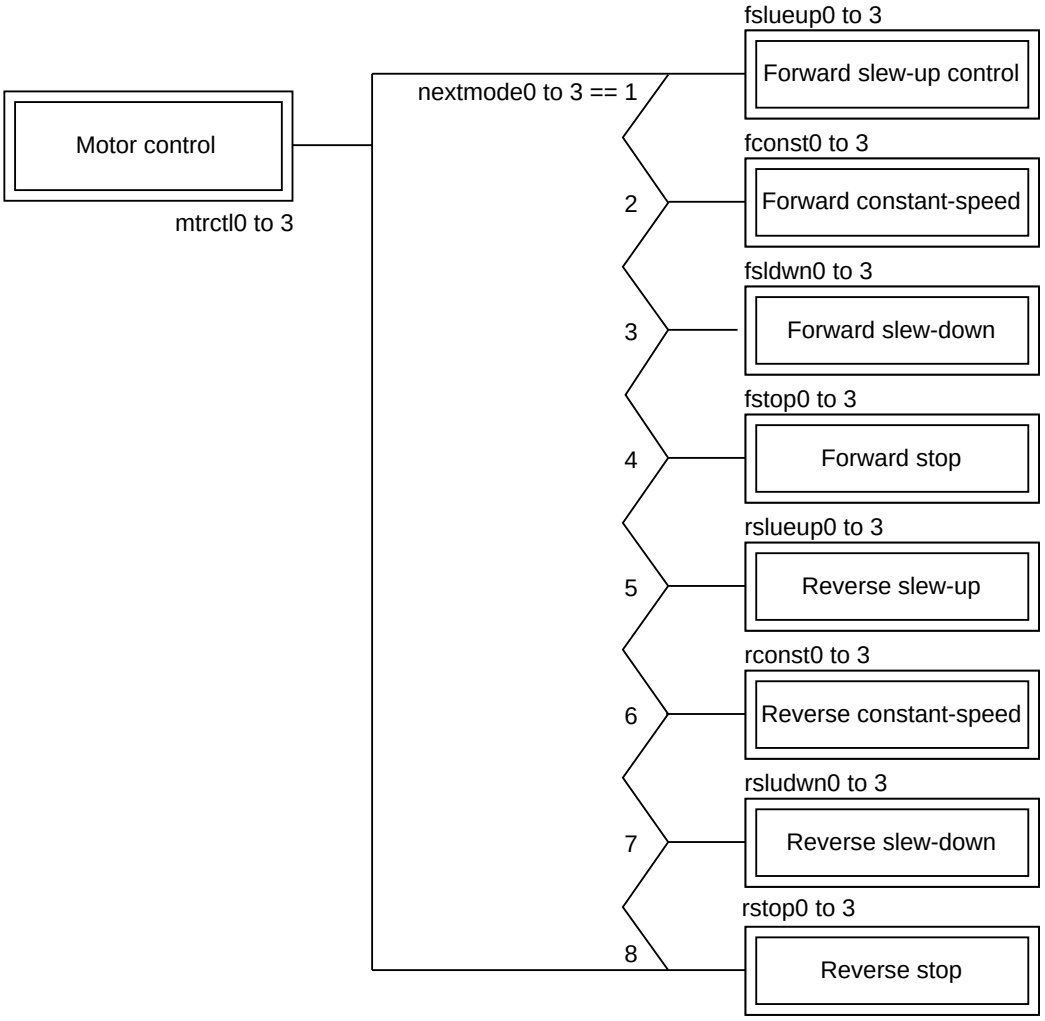
5. Data Tables

Table Name	Function	Data Length	Data Capacity
PATTBL0, PATTBL1	Hold patterns for 4-phase pulse output	Unsigned char	4 bytes
UPTBL DOWNTBL CNSTBL	Hold data for changing step cycle	Unsigned short	121 words

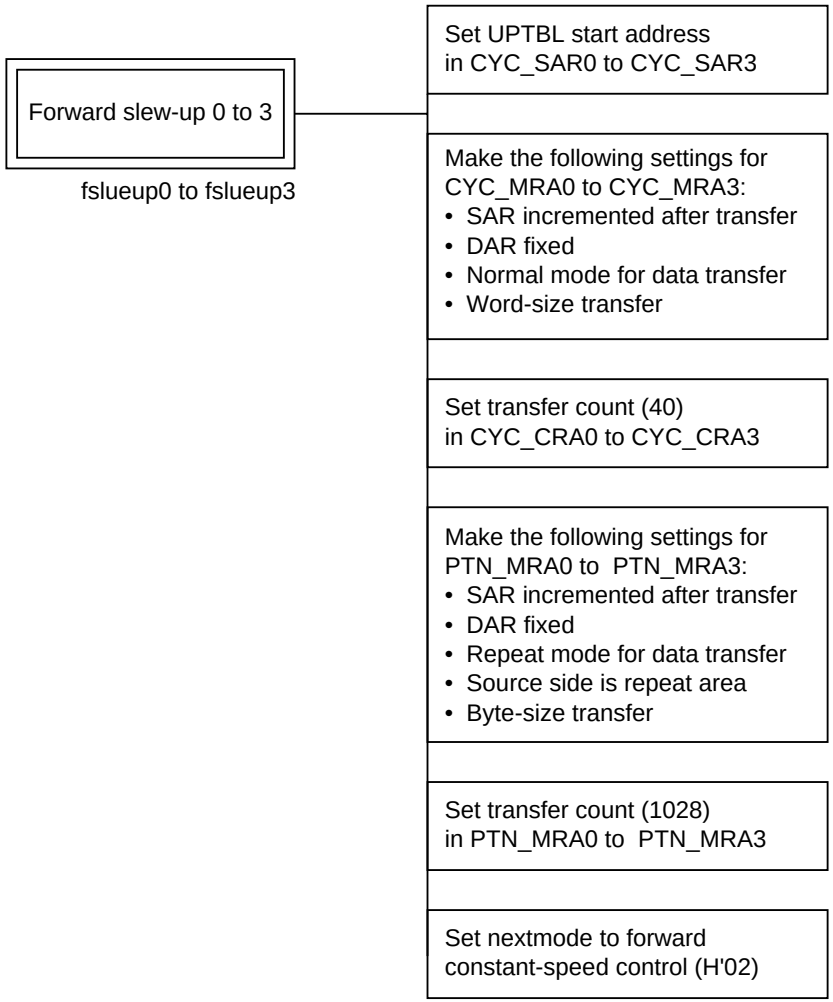
PAD

1. Main routine

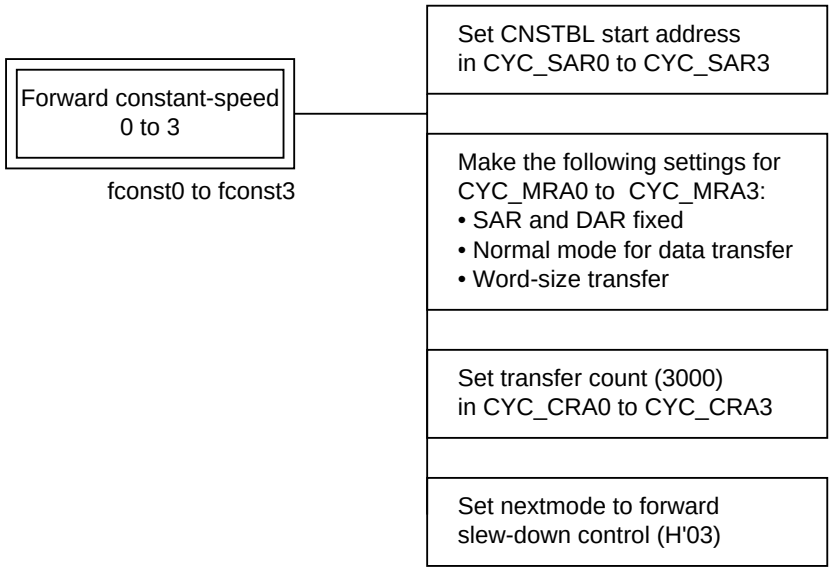



Motor control
mtrctl0 to 3
nextmode0 to 3 == 1
2
3
4
5
6
7
8
fslueup0 to 3
Forward slew-up control
fconst0 to 3
Forward constant-speed
fsldwn0 to 3
Forward slew-down
fstop0 to 3
Forward stop
rslueup0 to 3
Reverse slew-up
rconst0 to 3
Reverse constant-speed
rsludwn0 to 3
Reverse slew-down
rstop0 to 3
Reverse stop

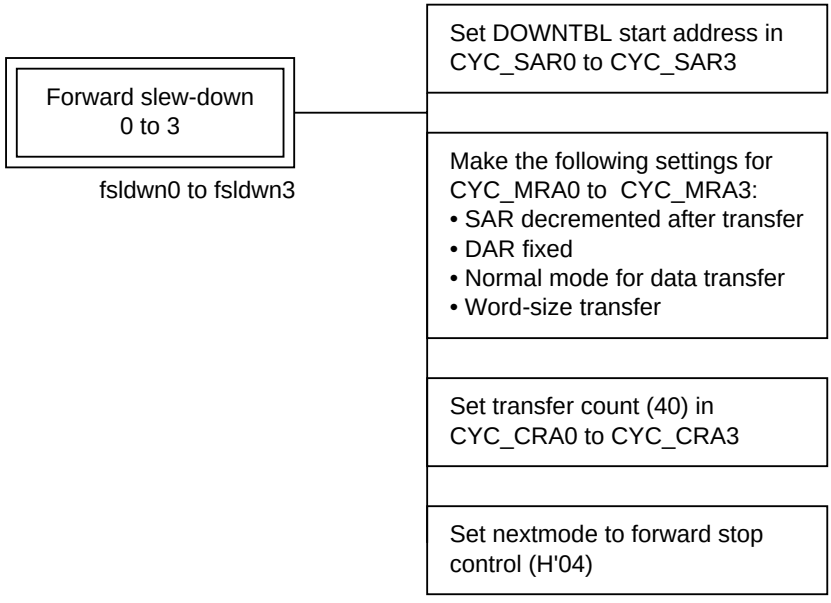
3. Forward slew-up 0 to 3



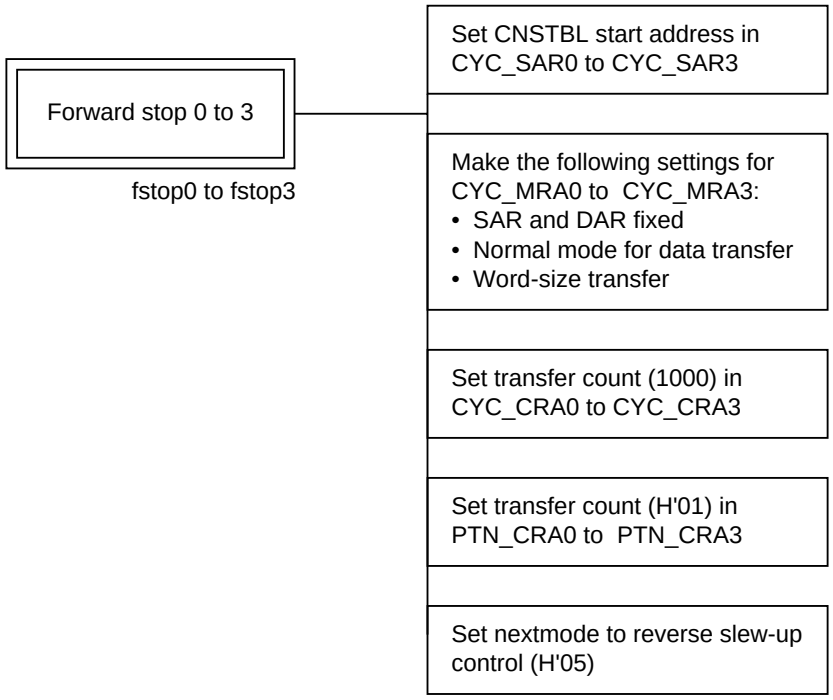
4. Forward constant-speed 0 to 3



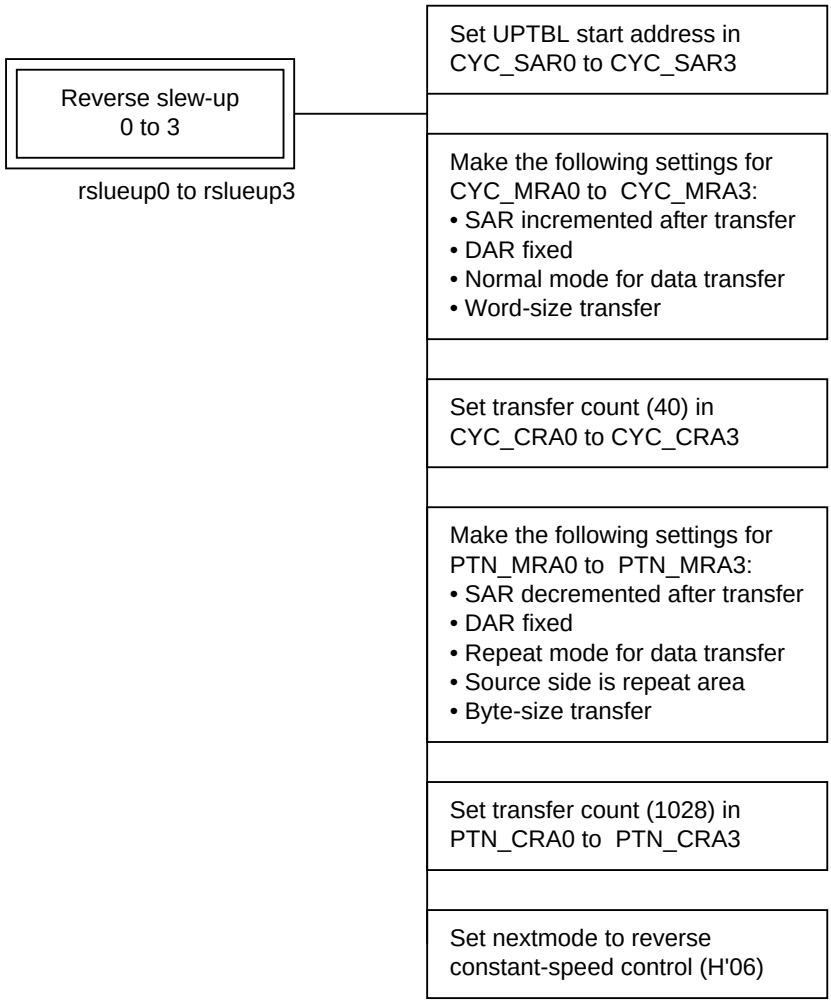
5. Forward slew-down 0 to 3



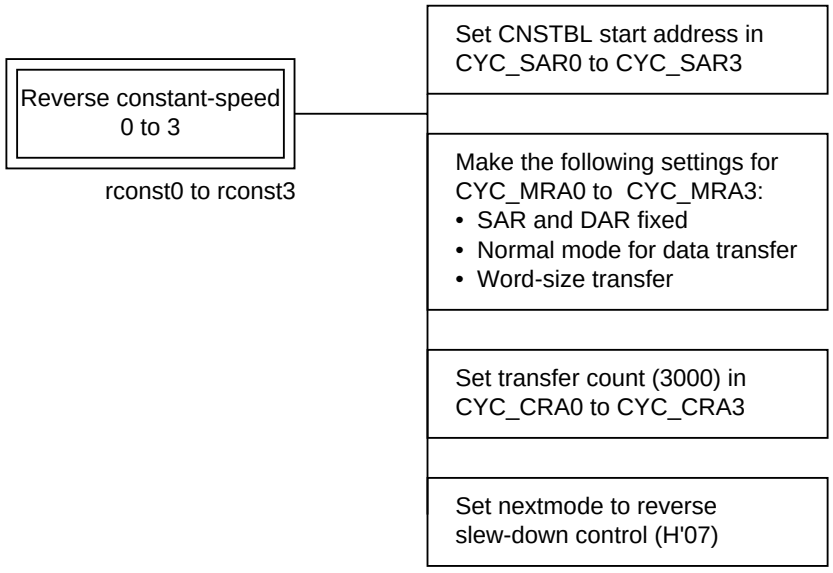
6. Forward stop 0 to 3



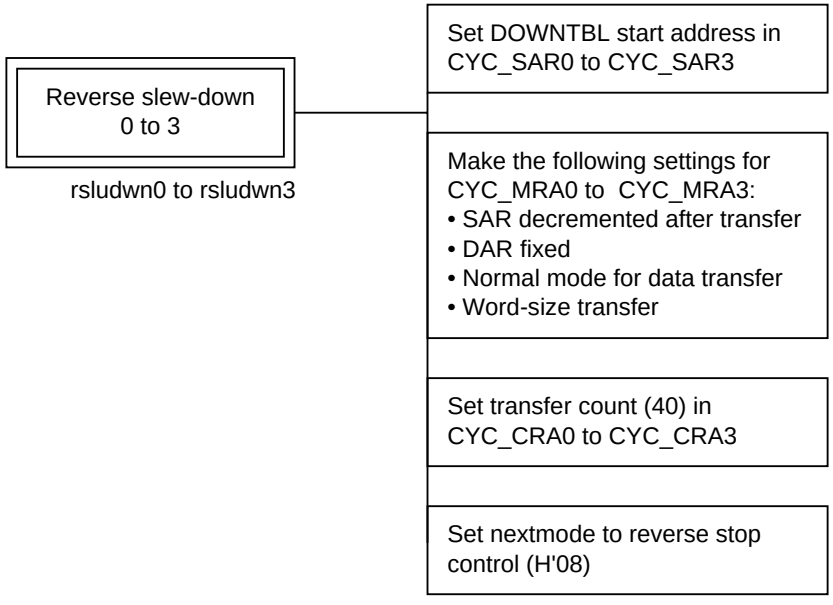
7. Reverse slew-up 0 to 3



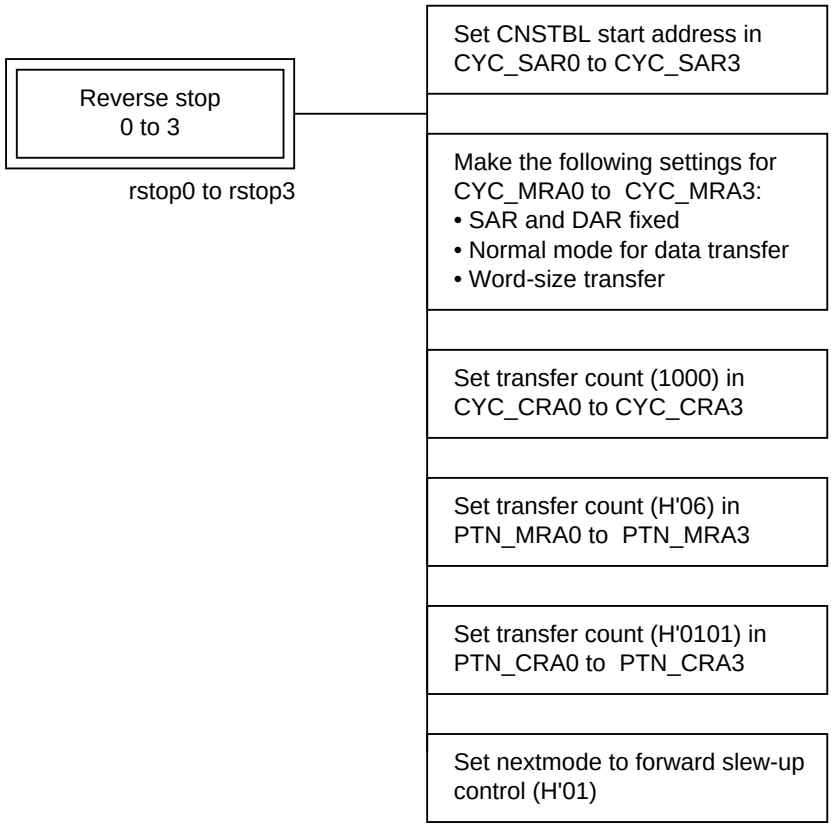
8. Reverse constant-speed 0 to 3



9. Reverse slew-down 0 to 3



10. Reverse stop 0 to 3



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*           PROTOCOL                   */
/****************************************/
void stp4mn(void);

/****************************************/
/*           RAM ALLOCATION             */
/****************************************/

#define PTN0_MRA (*(volatile unsigned char *)0xfff800)
#define PTN0_SAR (*(volatile unsigned long *)0xfff800)
#define PTN0_MRB (*(volatile unsigned char *)0xfff804)
#define PTN0_DAR (*(volatile unsigned long *)0xfff804)
#define PTN0_CRA (*(volatile unsigned short *)0xfff808)
#define PTN0_CRB (*(volatile unsigned short *)0xfff80a)

#define CYC0_MRA (*(volatile unsigned char *)0xfff80c)
#define CYC0_SAR (*(volatile unsigned long *)0xfff80c)
#define CYC0_MRB (*(volatile unsigned char *)0xfff810)
#define CYC0_DAR (*(volatile unsigned long *)0xfff810)
#define CYC0_CRA (*(volatile unsigned short *)0xfff814)
#define CYC0_CRB (*(volatile unsigned short *)0xfff816)

#define PTN1_MRA (*(volatile unsigned char *)0xfff818)
#define PTN1_SAR (*(volatile unsigned long *)0xfff818)
#define PTN1_MRB (*(volatile unsigned char *)0xfff81c)
#define PTN1_DAR (*(volatile unsigned long *)0xfff81c)
#define PTN1_CRA (*(volatile unsigned short *)0xfff820)
#define PTN1_CRB (*(volatile unsigned short *)0xfff822)

#define CYC1_MRA (*(volatile unsigned char *)0xfff824)
#define CYC1_SAR (*(volatile unsigned long *)0xfff824)
#define CYC1_MRB (*(volatile unsigned char *)0xfff828)
#define CYC1_DAR (*(volatile unsigned long *)0xfff828)
```

```
#define CYC1_CRA (*(volatile unsigned short *)0xfff82c)
#define CYC1_CRB (*(volatile unsigned short *)0xfff82e)

#define PTN2_MRA (*(volatile unsigned char *)0xfff830)
#define PTN2_SAR (*(volatile unsigned long *)0xfff830)
#define PTN2_MRB (*(volatile unsigned char *)0xfff834)
#define PTN2_DAR (*(volatile unsigned long *)0xfff834)
#define PTN2_CRA (*(volatile unsigned short *)0xfff838)
#define PTN2_CRB (*(volatile unsigned short *)0xfff83a)

#define CYC2_MRA (*(volatile unsigned char *)0xfff83c)
#define CYC2_SAR (*(volatile unsigned long *)0xfff83c)
#define CYC2_MRB (*(volatile unsigned char *)0xfff840)
#define CYC2_DAR (*(volatile unsigned long *)0xfff840)
#define CYC2_CRA (*(volatile unsigned short *)0xfff844)
#define CYC2_CRB (*(volatile unsigned short *)0xfff846)

#define PTN3_MRA (*(volatile unsigned char *)0xfff848)
#define PTN3_SAR (*(volatile unsigned long *)0xfff848)
#define PTN3_MRB (*(volatile unsigned char *)0xfff84c)
#define PTN3_DAR (*(volatile unsigned long *)0xfff84c)
#define PTN3_CRA (*(volatile unsigned short *)0xfff850)
#define PTN3_CRB (*(volatile unsigned short *)0xfff852)

#define CYC3_MRA (*(volatile unsigned char *)0xfff854)
#define CYC3_SAR (*(volatile unsigned long *)0xfff854)
#define CYC3_MRB (*(volatile unsigned char *)0xfff858)
#define CYC3_DAR (*(volatile unsigned long *)0xfff858)
#define CYC3_CRA (*(volatile unsigned short *)0xfff85c)
#define CYC3_CRB (*(volatile unsigned short *)0xfff85e)

#define nextmode0 (*(volatile unsigned char *)0xfff860)
#define nextmode1 (*(volatile unsigned char *)0xfff861)
#define nextmode2 (*(volatile unsigned char *)0xfff862)
#define nextmode3 (*(volatile unsigned char *)0xfff863)
```

```
/****************************************/
/*         DATA TABLE                   */
/****************************************/
const unsigned char PATTBL0[4] = {0x6f,0x3f,0x9f,0xcf};
const unsigned char PATTBL1[4] = {0xf6,0xf3,0xf9,0xfc};

/*
const unsigned short UPTBL[40] = {  47723, 43885, 40848, 38365, 36286,
                            34513, 32977, 31629, 30434, 29365,
                            28402, 27527, 26729, 25996, 25321,
                            24695, 24114, 23572, 23065, 22589,
                            22141, 21720, 21321, 20944, 20585,
                            20245, 19921, 19612, 19317, 19035,
                            18765, 18506, 18258, 18019, 17790,
                            17569, 17356, 17150, 16952, 16760     };
const unsigned short DOWNTBL[] = { 16760 };
const unsigned short CNSTBL[] = { 16760 };

*/

const unsigned short UPTBL[12] = {  35350, 14637, 11225, 9462,
                            8337, 7537, 6937, 6450, 6062,
                            5725, 5450, 5212    };

const unsigned short DOWNTBL[] = { 5000 };
const unsigned short CNSTBL[] = { 5000 };


/****************************************/
/*        MAIN PROGRAM : stp4mn         */
/****************************************/
void stp4mn(void)
{
      MSTPCR = 0x97ff;
      nextmode0 = 0x02;
      nextmode1 = 0x02;
      nextmode2 = 0x02;
      nextmode3 = 0x02;
```

```
/* PPG INITIALIZE */
      PMR = 0xff;
      PODRH = 0xcc;
      PODRL = 0xcc;
      P1DDR = 0xff;
      P2DDR = 0xff;
      NDERH = 0xff;
      NDERL = 0xff;
      PCR = 0xe4;
      NDR3 = 0xcf;
      NDR2 = 0xfc;
      NDR1 = 0xcf;
      NDR0 = 0xfc;

/* DTC INITIALIZE */
      PTN0_SAR = (long)PATTBL1;
      PTN1_SAR = (long)PATTBL0;
      PTN2_SAR = (long)PATTBL1;
      PTN3_SAR = (long)PATTBL0;
      PTN0_MRA = 0x86;
      PTN1_MRA = 0x86;
      PTN2_MRA = 0x86;
      PTN3_MRA = 0x86;
      PTN0_DAR = (long)(&NDR0);
      PTN1_DAR = (long)(&NDR1);
      PTN2_DAR = (long)(&NDR2);
      PTN3_DAR = (long)(&NDR3);
      PTN0_MRB = 0x80;
      PTN1_MRB = 0x80;
      PTN2_MRB = 0x80;
      PTN3_MRB = 0x80;
      PTN0_CRA = 0x0404;
      PTN1_CRA = 0x0404;
      PTN2_CRA = 0x0404;
      PTN3_CRA = 0x0404;
```

```
        CYC0_SAR = (long)UPTBL;
        CYC1_SAR = (long)UPTBL;
        CYC2_SAR = (long)UPTBL;
        CYC3_SAR = (long)UPTBL;
        CYC0_MRA = 0x81;
        CYC1_MRA = 0x81;
        CYC2_MRA = 0x81;
        CYC3_MRA = 0x81;
        CYC0_DAR = (long)(&TGR0B);
        CYC1_DAR = (long)(&TGR1B);
        CYC2_DAR = (long)(&TGR2B);
        CYC3_DAR = (long)(&TGR3B);
        CYC0_MRB = 0x00;
        CYC1_MRB = 0x00;
        CYC2_MRB = 0x00;
        CYC3_MRB = 0x00;
        CYC0_CRA = 13;
        CYC1_CRA = 13;
        CYC2_CRA = 13;
        CYC3_CRA = 13;
        DTCERB_BP.TGI0A = 1;
        DTCERB_BP.TGI1A = 1;
        DTCERC_BP.TGI2A = 1;
        DTCERC_BP.TGI3A = 1;

    /* TPU INITIALIZE */
        TPU_TCR0 = 0x42;
        TPU_TCR1 = 0x42;
        TCR2 = 0x42;
        TCR3 = 0x42;
        TIER0 = 0x41;
        TIER1 = 0x41;
        TIER2 = 0x41;
        TIER3 = 0x41;
        TGR0A = 100;
        TGR1A = 100;
        TGR2A = 100;
        TGR3A = 100;
        TGR0B = 62500;
        TGR1B = 62500;
        TGR2B = 62500;
        TGR3B = 62500;

        set_imask_ccr(0);
        TSTR = 0x0f;
        while(1);
    }
```

```
/****************************************/
/*        NAME : mtrcntl0                */
/****************************************/
#pragma interrupt(mtrcntl0)
void mtrcntl0(void)
{
        switch(nextmode0){
            case 1:
                  fslueup0();
                  break;
            case 2:
                  fconst0();
                  break;
            case 3:
                  fsldwn0();
                  break;
            case 4:
                  fstop0();
                  break;
            case 5:
                  rslueup0();
                  break;
            case 6:
                  rconst0();
                  break;
            case 7:
                  rsldwn0();
                  break;
            case 8:
                  rstop0();
                  break;
            default:
                  break;
            }

        DTCERB_BP.TGI0A = 1;
        TSR0 &= 0xfe;
}
```

```
/*********** forward slue-up0 *************/
fslueup0()
{
      CYC0_SAR = (long)UPTBL;
      CYC0_MRA = 0x81;
      CYC0_CRA = 0x000d;
      PTN0_SAR = (long)PATTBL1;
      PTN0_MRA = 0x86;
      PTN0_CRA = 0x0404;
      nextmode0++;
}

/******** forward constant speed0 *********/
fconst0()
{
      CYC0_SAR = (long)CNSTBL;
      CYC0_MRA = 0x01;
      CYC0_CRA = 0x0bb9;
      PTN0_SAR = (long)PATTBL1;
      PTN0_MRA = 0x86;
      PTN0_CRA = 0x0404;
      nextmode0++;
}

/********** forward slue-down0 ************/
fsldwn0()
{
      CYC0_SAR = (long)DOWNTBL;
      CYC0_MRA = 0xc1;
      CYC0_CRA = 0x000d;
      PTN0_SAR = (long)PATTBL1;
      PTN0_MRA = 0x86;
      PTN0_CRA = 0x0404;
      nextmode0++;
}
/************* forward stop0 **************/
fstop0()
{
      CYC0_SAR = (long)UPTBL;
      CYC0_MRA = 0x01;
      CYC0_CRA = 0x03e9;
      PTN0_SAR = (long)PATTBL1;
      PTN0_MRA = 0x06;
      PTN0_CRA = 0x0404;
      nextmode0++;
}
```

```
/*********** reverse slue-up0 *************/
rslueup0()
{
      CYC0_SAR = (long)UPTBL;
      CYC0_MRA = 0x81;
      CYC0_CRA = 0x000d;
      PTN0_SAR = (long)(PATTBL1+3);
      PTN0_MRA = 0xc6;
      PTN0_CRA = 0x0404;
      nextmode0++;
}

/******** reverse constant speed0 *********/
rconst0()
{
      CYC0_SAR = (long)CNSTBL;
      CYC0_MRA = 0x01;
      CYC0_CRA = 0x0bb9;
      PTN0_SAR = (long)(PATTBL1+3);
      PTN0_MRA = 0xc6;
      PTN0_CRA = 0x0404;
      nextmode0++;
}

/********** reverse slue-down0 ************/
rsldwn0()
{
      CYC0_SAR = (long)DOWNTBL;
      CYC0_MRA = 0xc1;
      CYC0_CRA = 0x000d;
      PTN0_SAR = (long)(PATTBL1+3);
      PTN0_MRA = 0xc6;
      PTN0_CRA = 0x0404;
      nextmode0++;
}

/************* reverse stop0 **************/
rstop0()
{
      CYC0_SAR = (long)UPTBL;
      CYC0_MRA = 0x01;
      CYC0_CRA = 0x03e9;
      PTN0_SAR = (long)(PATTBL1+3);
      PTN0_MRA = 0x06;
      PTN0_CRA = 0x0404;
      nextmode0 = 0x01;
}
```

```
/****************************************/
/*        NAME : mtrcntl1                */
/****************************************/
#pragma interrupt(mtrcntl1)
void mtrcntl1(void)
{
        switch(nextmode1){
            case 1:
                  fslueup1();
                  break;
            case 2:
                  fconst1();
                  break;
            case 3:
                  fsldwn1();
                  break;
            case 4:
                  fstop1();
                  break;
            case 5:
                  rslueup1();
                  break;
            case 6:
                  rconst1();
                  break;
            case 7:
                  rsldwn1();
                  break;
            case 8:
                  rstop1();
                  break;
        default:
                  break;
        }

        DTCERB_BP.TGI1A = 1;
        TSR1 &= 0xfe;
}
```

```
/*********** forward slue-up1 *************/
fslueup1()
{
      CYC1_SAR = (long)UPTBL;
      CYC1_MRA = 0x81;
      CYC1_CRA = 0x000d;
      PTN1_SAR = (long)PATTBL0;
      PTN1_MRA = 0x86;
      PTN1_CRA = 0x0404;
      nextmode1++;
}

/******** forward constant speed1 *********/
fconst1()
{
      CYC1_SAR = (long)CNSTBL;
      CYC1_MRA = 0x01;
      CYC1_CRA = 0x0bb9;
      PTN1_SAR = (long)PATTBL0;
      PTN1_MRA = 0x86;
      PTN1_CRA = 0x0404;
      nextmode1++;
}

/********** forward slue-down1 ************/
fsldwn1()
{
      CYC1_SAR = (long)DOWNTBL;
      CYC1_MRA = 0xc1;
      CYC1_CRA = 0x000d;
      PTN1_SAR = (long)PATTBL0;
      PTN1_MRA = 0x86;
      PTN1_CRA = 0x0404;
      nextmode1++;
}

/************* forward stop1 **************/
fstop1()
{
      CYC1_SAR = (long)UPTBL;
      CYC1_MRA = 0x01;
      CYC1_CRA = 0x03e9;
      PTN1_SAR = (long)PATTBL0;
      PTN1_MRA = 0x06;
      PTN1_CRA = 0x0404;
      nextmode1++;
}
```

```
/*********** reverse slue-up1 *************/
rslueup1()
{
      CYC1_SAR = (long)UPTBL;
      CYC1_MRA = 0x81;
      CYC1_CRA = 0x000d;
      PTN1_SAR = (long)(PATTBL0+3);
      PTN1_MRA = 0xc6;
      PTN1_CRA = 0x0404;
      nextmode1++;
}

/******** reverse constant speed1 *********/
rconst1()
{
      CYC1_SAR = (long)CNSTBL;
      CYC1_MRA = 0x01;
      CYC1_CRA = 0x0bb9;
      PTN1_SAR = (long)(PATTBL0+3);
      PTN1_MRA = 0xc6;
      PTN1_CRA = 0x0404;
      nextmode1++;
}
/********** reverse slue-down1 ************/
rsldwn1()
{
      CYC1_SAR = (long)DOWNTBL;
      CYC1_MRA = 0xc1;
      CYC1_CRA = 0x000d;
      PTN1_SAR = (long)(PATTBL0+3);
      PTN1_MRA = 0xc6;
      PTN1_CRA = 0x0404;
      nextmode1++;
}

/************* reverse stop1 **************/
rstop1()
{
      CYC1_SAR = (long)UPTBL;
      CYC1_MRA = 0x01;
      CYC1_CRA = 0x03e9;
      PTN1_SAR = (long)(PATTBL0+3);
      PTN1_MRA = 0x06;
      PTN1_CRA = 0x0404;
      nextmode1 = 0x01;
}
```

```
/*****************************************/
/*        NAME : mtrcntl2                */
/*****************************************/
#pragma interrupt(mtrcntl2)
void mtrcntl2(void)
{
        switch(nextmode2){
            case 1:
                  fslueup2();
                  break;
            case 2:
                  fconst2();
                  break;
            case 3:
                  fsldwn2();
                  break;
            case 4:
                  fstop2();
                  break;
            case 5:
                  rslueup2();
                  break;
            case 6:
                  rconst2();
                  break;
            case 7:
                  rsldwn2();
                  break;
            case 8:
                  rstop2();
                  break;
            default:
                  break;
            }

        DTCERC_BP.TGI2A = 1;
        TSR2 &= 0xfe;
}
```

```
/*********** forward slue-up2 *************/
fslueup2()
{
            CYC2_SAR = (long)UPTBL;
            CYC2_MRA = 0x81;
            CYC2_CRA = 0x000d;
            PTN2_SAR = (long)PATTBL1;
            PTN2_MRA = 0x86;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/******** forward constant speed2 *********/
fconst2()
{
            CYC2_SAR = (long)CNSTBL;
            CYC2_MRA = 0x01;
            CYC2_CRA = 0x0bb9;
            PTN2_SAR = (long)PATTBL1;
            PTN2_MRA = 0x86;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/********** forward slue-down2 ************/
fsldwn2()
{
            CYC2_SAR = (long)DOWNTBL;
            CYC2_MRA = 0xc1;
            CYC2_CRA = 0x000d;
            PTN2_SAR = (long)PATTBL1;
            PTN2_MRA = 0x86;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/************* forward stop2 **************/
fstop2()
{
            CYC2_SAR = (long)UPTBL;
            CYC2_MRA = 0x01;
            CYC2_CRA = 0x03e9;
            PTN2_SAR = (long)PATTBL1;
            PTN2_MRA = 0x06;
            PTN2_CRA = 0x0404;
            nextmode2++;
}
```

```
/*********** reverse slue-up2 *************/
rslueup2()
{
            CYC2_SAR = (long)UPTBL;
            CYC2_MRA = 0x81;
            CYC2_CRA = 0x000d;
            PTN2_SAR = (long)(PATTBL1+3);
            PTN2_MRA = 0xc6;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/******** reverse constant speed2 *********/
rconst2()
{
            CYC2_SAR = (long)CNSTBL;
            CYC2_MRA = 0x01;
            CYC2_CRA = 0x0bb9;
            PTN2_SAR = (long)(PATTBL1+3);
            PTN2_MRA = 0xc6;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/********** reverse slue-down2 ************/
rsldwn2()
{
            CYC2_SAR = (long)DOWNTBL;
            CYC2_MRA = 0xc1;
            CYC2_CRA = 0x000d;
            PTN2_SAR = (long)(PATTBL1+3);
            PTN2_MRA = 0xc6;
            PTN2_CRA = 0x0404;
            nextmode2++;
}

/************* reverse stop2 **************/
rstop2()
{
            CYC2_SAR = (long)UPTBL;
            CYC2_MRA = 0x01;
            CYC2_CRA = 0x03e9;
            PTN2_SAR = (long)(PATTBL1+3);
            PTN2_MRA = 0x06;
            PTN2_CRA = 0x0404;
            nextmode2 = 0x01;
}
```

```
/****************************************/
/*        NAME : mtrcntl3                */
/****************************************/
#pragma interrupt(mtrcntl3)
void mtrcntl3(void)
{
        switch(nextmode3){
                        case 1:
                              fslueup3();
                              break;
                        case 2:
                              fconst3();
                              break;
                        case 3:
                              fsldwn3();
                              break;
                        case 4:
                              fstop3();
                              break;
                        case 5:
                              rslueup3();
                              break;
                        case 6:
                              rconst3();
                              break;
                        case 7:
                              rsldwn3();
                              break;
                        case 8:
                              rstop3();
                              break;
                        default:
                              break;
            }

            DTCERC_BP.TGI3A = 1;
            TSR3 &= 0xfe;
}
```

```
/*********** forward slue-up3 *************/
fslueup3()
{
            CYC3_SAR = (long)UPTBL;
            CYC3_MRA = 0x81;
            CYC3_CRA = 0x000d;
            PTN3_SAR = (long)PATTBL0;
            PTN3_MRA = 0x86;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/******** forward constant speed3 *********/
fconst3()
{
            CYC3_SAR = (long)CNSTBL;
            CYC3_MRA = 0x01;
            CYC3_CRA = 0x0bb9;
            PTN3_SAR = (long)PATTBL0;
            PTN3_MRA = 0x86;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/********** forward slue-down3 ************/
fsldwn3()
{
            CYC3_SAR = (long)DOWNTBL;
            CYC3_MRA = 0xc1;
            CYC3_CRA = 0x000d;
            PTN3_SAR = (long)PATTBL0;
            PTN3_MRA = 0x86;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/************* forward stop3 **************/
fstop3()
{
            CYC3_SAR = (long)UPTBL;
            CYC3_MRA = 0x01;
            CYC3_CRA = 0x03e9;
            PTN3_SAR = (long)PATTBL0;
            PTN3_MRA = 0x06;
            PTN3_CRA = 0x0404;
            nextmode3++;
}
```

```
/*********** reverse slue-up3 *************/
rslueup3()
{
            CYC3_SAR = (long)UPTBL;
            CYC3_MRA = 0x81;
            CYC3_CRA = 0x000d;
            PTN3_SAR = (long)(PATTBL0+3);
            PTN3_MRA = 0xc6;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/******** reverse constant speed3 *********/
rconst3()
{
            CYC3_SAR = (long)CNSTBL;
            CYC3_MRA = 0x01;
            CYC3_CRA = 0x0bb9;
            PTN3_SAR = (long)(PATTBL0+3);
            PTN3_MRA = 0xc6;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/********** reverse slue-down3 ************/
rsldwn3()
{
            CYC3_SAR = (long)DOWNTBL;
            CYC3_MRA = 0xc1;
            CYC3_CRA = 0x000d;
            PTN3_SAR = (long)(PATTBL0+3);
            PTN3_MRA = 0xc6;
            PTN3_CRA = 0x0404;
            nextmode3++;
}

/************* reverse stop3 **************/
rstop3()
{
            CYC3_SAR = (long)UPTBL;
            CYC3_MRA = 0x01;
            CYC3_CRA = 0x03e9;
            PTN3_SAR = (long)(PATTBL0+3);
            PTN3_MRA = 0x06;
            PTN3_CRA = 0x0404;
            nextmode3 = 0x01;
}
```

Specifications

1. The A/D converter and DMAC are activated by a TPU conversion start trigger, A/D conversion of voice signals is performed, and the results are transferred to RAM by the DMAC, as shown in figure 1.
2. The transfer areas are H'A00000 to H'A0FFFF and H'A10000 to H'A1FFFF.
3. The address is activated by a TPU TGRA compare-match.
4. A 20 Hz H8S/2655 internal operating frequency is used.

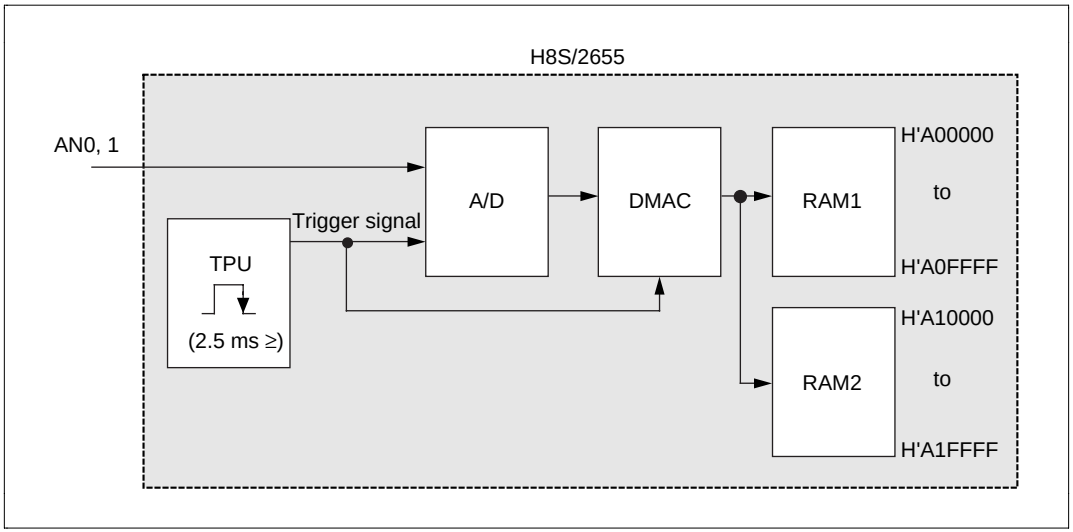


Figure 1 Block Diagram of Timer-Triggered A/D Conversion

Functions Used

1. Figure 2 shows the DMAC, A/D, and TPU block diagram for this sample task.

   The following DMAC function is used to transfer A/D conversion results to RAM:

   a. Activation of DMAC operation by a TPU compare-match A interrupt

   The following A/D converter functions are used to perform signal sampling:

   a. The ability to start conversion when triggered by the TPU
   b. Simultaneous sampling of input voltages on two channels (simultaneous sampling operation)

The following TPU function is used to perform signal sampling:

a. The ability to generate an A/D converter conversion start trigger

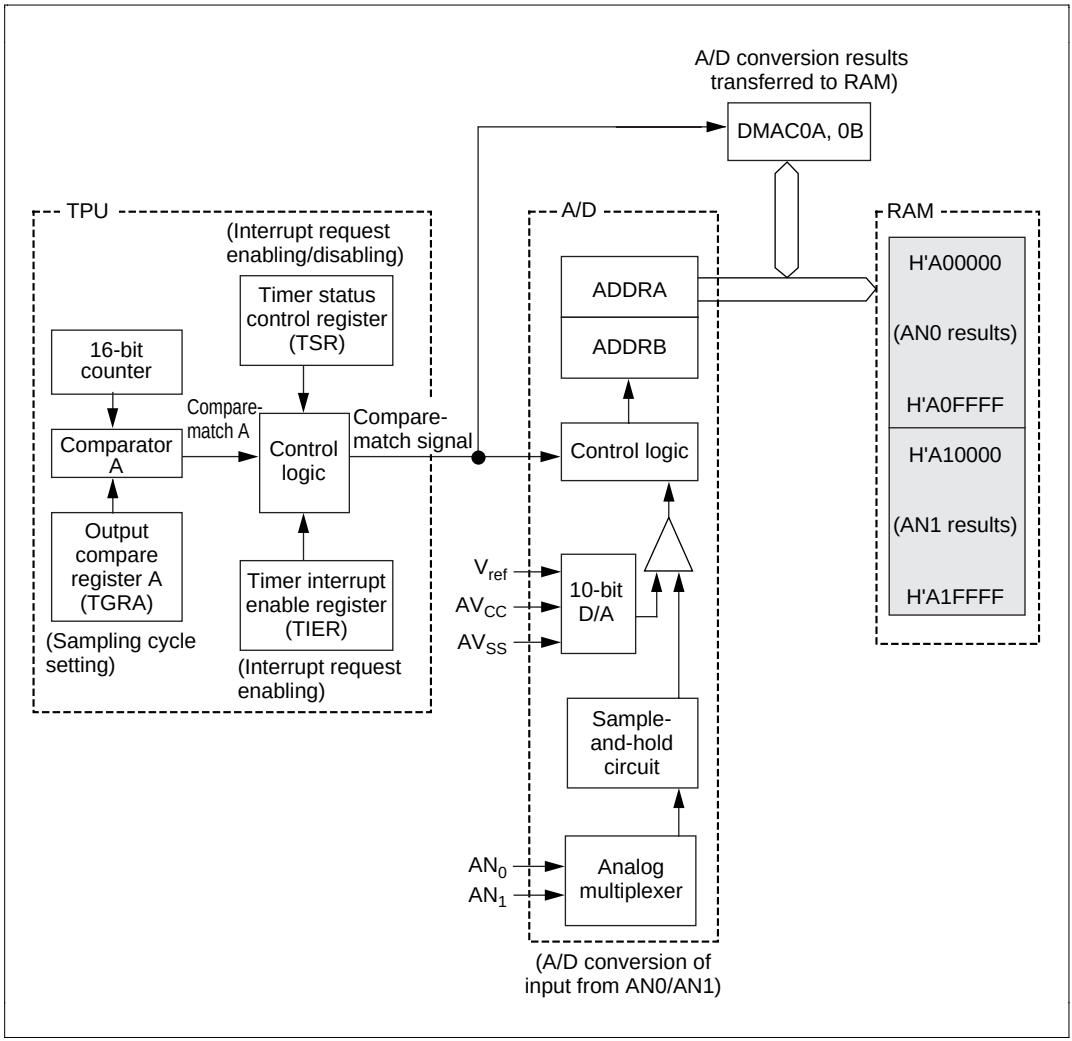


Figure 2 Block Diagram of Timer-Triggered A/D Conversion

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to transfer A/D conversion results to RAM.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
A/D	AN0, 1	Analog signal input pins
	ADDRA, ADDRB	Store A/D conversion results
DMAC	DMABCR	Controls operation of each channel
	DMACR	Sets sequential mode as transfer mode
	MAR	Transfer source address setting
	IOAR	Transfer destination address setting
	ETCR	Transfer number setting
TPU	TGR	Cycle setting
	TCR	Selects clock, counter clear source, etc.
	TIOR	Sets TGR as output compare register

Operation

Figure 3 shows the principles of the operation. A/D conversion values are stored in RAM by means of H8S/2655 hardware processing as shown in this figure.

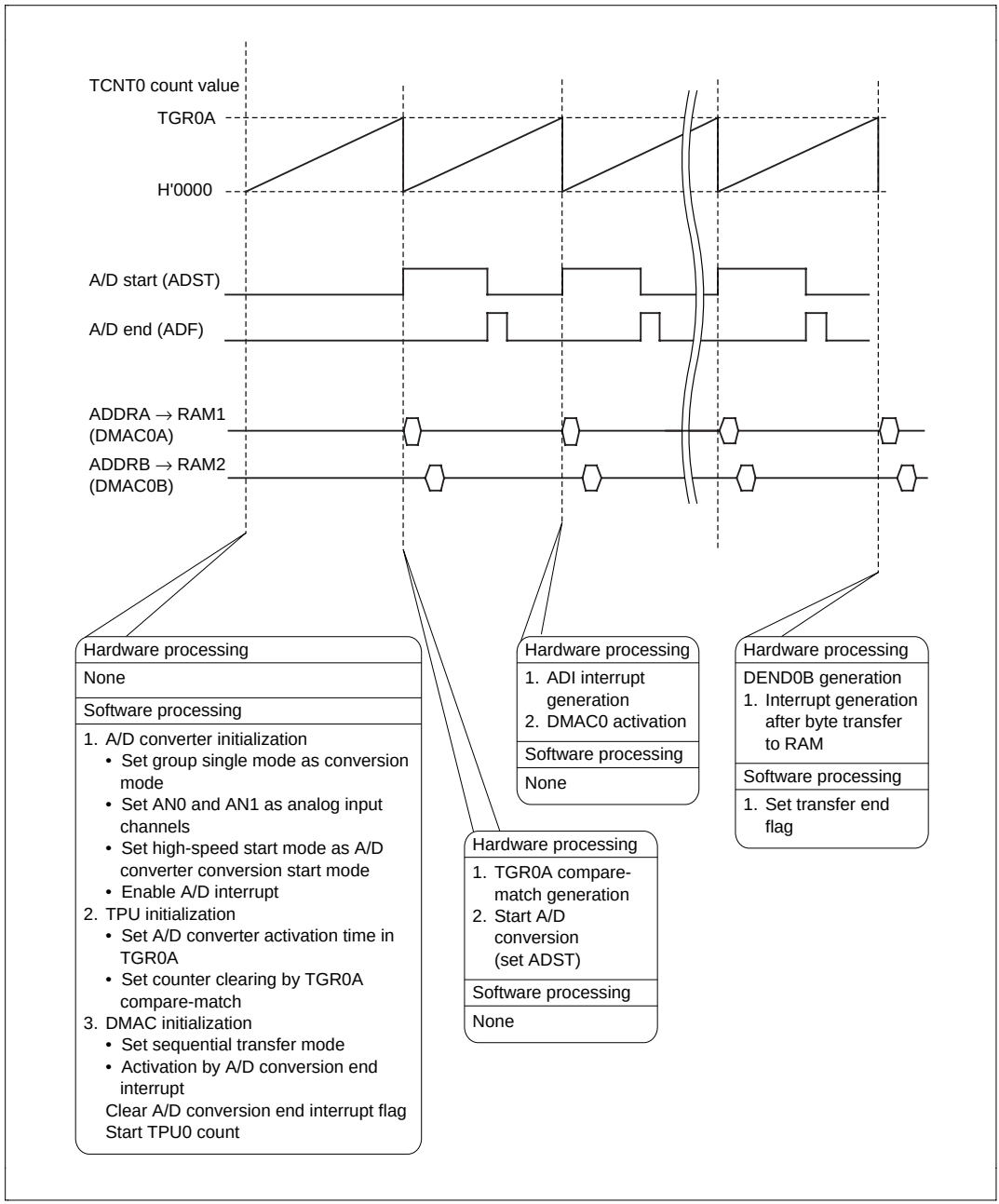


Figure 3 Principles of Timer-Triggered A/D Conversion Operation

Software

1. Modules

Module Name	Label	Function
Main routine	tpuadmn	TPU and DMAC initialization, setting of RAM used
A/D conversion end	adend	A/D conversion end flag setting

2. Arguments

Label	Function	Data Length	Module	Input/ Output
ad_end	Indicates end of data transfer from H'A00000 to H'A1FFFF 1: End of data transfer 0: Data transfer in progress	Unsigned char	Main routine A/D conversion end	Input Output
ad_data	AN0 and AN1 A/D conversion results are stored in byte units starting in addata0 and addata1, respectively, by DMA transfer The conversion result is transferred to RAM as follows: Upper bits: AD9 AD8 AD7 AD6 AD5 AD4 AD3 AD2	Unsigned char	Main routine	Input
sum_cyc	Setting of timer value corresponding to A/D conversion sampling cycle Cycle (ns) = Timer counter value × ø cycle (50 ns at 20 MHz operation)	Unsigned short	Main routine	Input

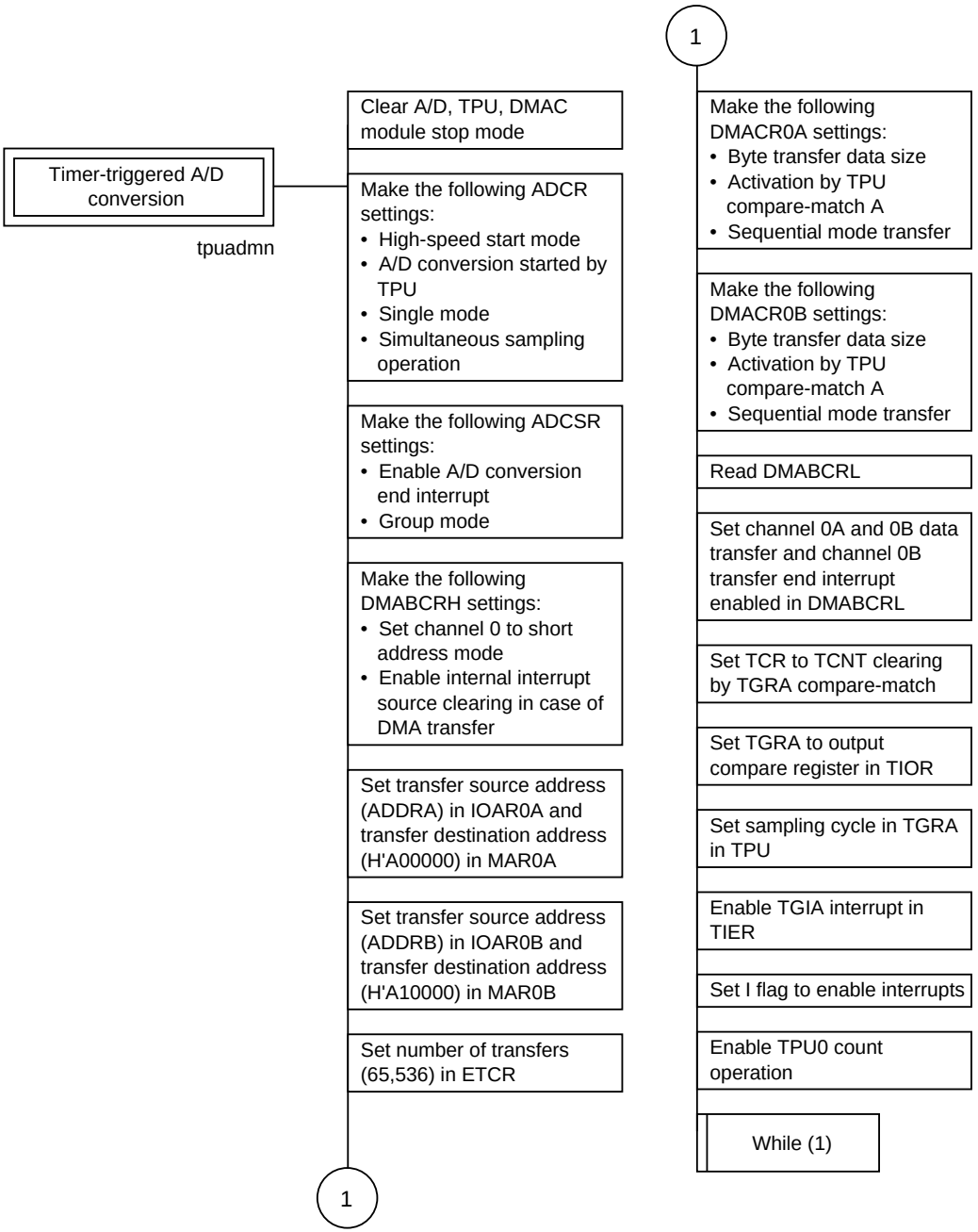
On-Chip Function	Register Name	Function
TPU	TGRA	A/D conversion sampling cycle setting
	TIER	Enables TGIEA interrupt’
	TCR	Makes the following TPU0 settings: • Counter clearing by TGRA compare-match • Counting on ø internal clock
	TIOR	Sets TGRA as output compare register, disables pin output
	TSTR	Enables TCNT0 count operation
DMAC	DMABCR	Controls operation of each channel
	DMACR0A	Makes the following DMAC0A settings: • Byte-size transfer • Sequential mode • Internal interrupt source clearing in case of DMA transfer enabled • Data transfer and transfer end interrupt enabled
	DMACR0B	Makes the following DMAC0B settings: • Byte-size transfer • Sequential mode • Internal interrupt source clearing in case of DMA transfer enabled • Data transfer and transfer end interrupt enabled
	IOAR0	Transfer source address setting
	MAR0	Transfer destination address setting
	ETCR0	Transfer number (H'0000) setting
A/D	ADCR	ADCR is set as follows: • High-speed start mode • A/D conversion started by TPU0 • Single mode • Simultaneous sampling operation
	ADCSR	ADCSR is set as follows: • A/D conversion end interrupt enabled • Group mode • AN0 and AN1 set as input channels
	MSTPCR	Clears module stop mode

4. RAM Used

This task does not use any RAM apart from the arguments.

PAD

1. Main routine



2. A/D conversion end

Program List

```
/****************************************/
/*          FILE NAME : ap21.c          */
/****************************************/
#include <machine.h>
#include <h8s.h>

/****************************************/
/*          PROTCOL                     */
/****************************************/
void tpuadmn(void);

/****************************************/
/*          RAM ALLOCATION              */
/****************************************/
#define ad_end (*(volatile unsigned char *)0xffec00)
#define sum_cyc (*(volatile unsigned short *)0xffec01)
volatile struct ad_data
{
                unsigned char    data1[65536];
                unsigned char    data2[65536];
};
#define ad (*(struct ad_data *)0xA00000)

/****************************************/
/*       MAIN PROGRAM : tpuadmn         */
/****************************************/
void tpuadmn(void)
{
        MSTPCR = 0x5dff;
                ADCR = 0x54;
        ADCSR = 0x49;

                DMABCRH = 0x03;
                IOAR0A = (long)(&ADDRA);
                IOAR0B = (long)(&ADDRB);
                MAR0A = (long)(&ad.data1);
                MAR0B = (long)(&ad.data2);
                ETCR0A = 0x0000;
                ETCR0B = 0x0000;
                DMACR0A = 0x11;
                DMACR0B = 0x11;
                DMABCRL |= 0x32;

                TPU_TCR0 = 0x20;
                TIOR0H = 0x00;
                TGR0A = sum_cyc;
                TIER0 = 0xc0;

                set_imask_ccr(0);

                TSTR = 0x01;
                while(1);
}
```

```
/*****************************************/
/*    NAME : adend(set end flag)          */
/*****************************************/
#pragma interrupt(adend)
void adend(void)
{
                TSTR = 0x00;
                ad_end = 0x01;
                DMABCRL &= 0xcd;
                ADCSR_BP.ADIE = 0;
}
```

Specifications

1. The DMAC is activated by TPU channels 0 and 1, and D/A conversion is performed on data stored in RAM, as shown in figure 1.
2. The RAM area is H'A00000 to H'A1FFFF.
3. A 20 Hz H8S/2655 internal operating frequency is used.

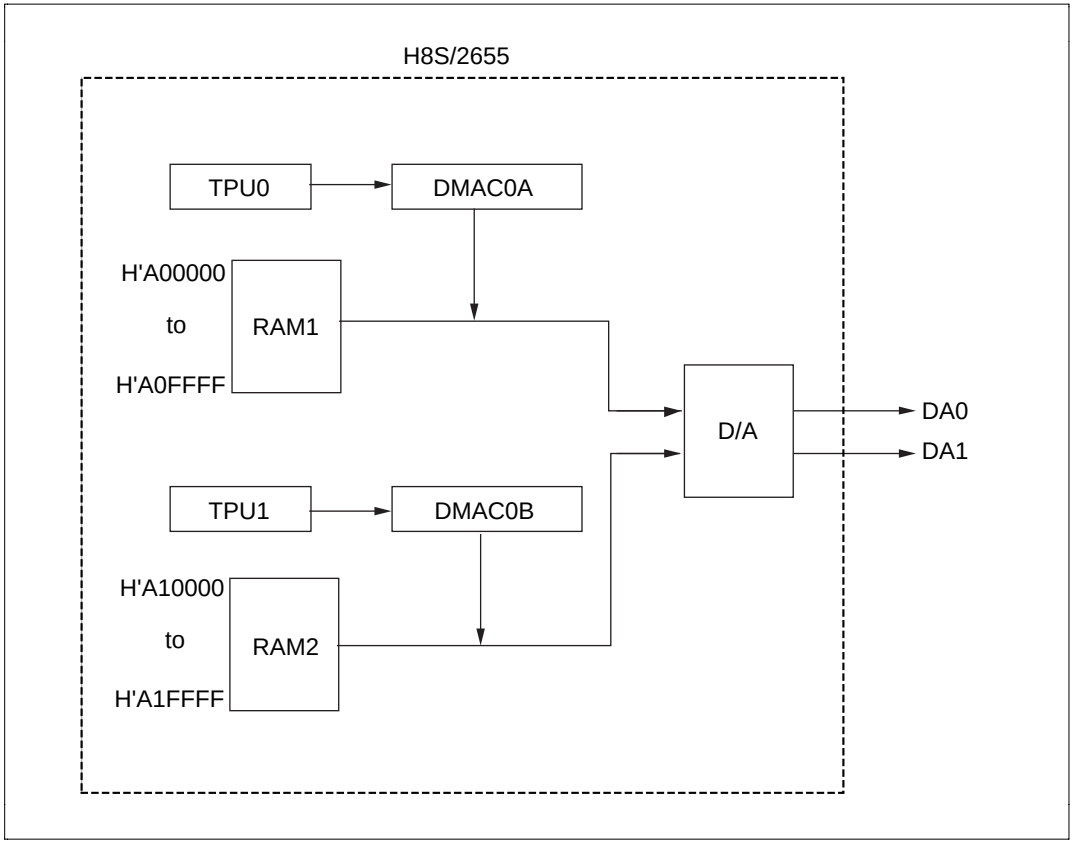


Figure 1 D/A Conversion Block Diagram

Functions Used

1. Figure 2 shows the DMAC, D/A, and TPU block diagram for this sample task.

   The following H8S/2655 functions are used to perform D/A conversion

   DMAC

   Activated by TPU compare-match A; transfers data from D/A DADR to data buffer.

   TPU

   Channels 0 and 1 operate simultaneously, and activate the DMAC.

   The timer counter is cleared by each channel 1 compare-match A.

   D/A

   Immediately conversion data is written to DADR, D/A conversion is started and the conversion result is output after the elapse of the conversion time. The analog conversion voltage range can be set, with AV_{CC} as the reference voltage.

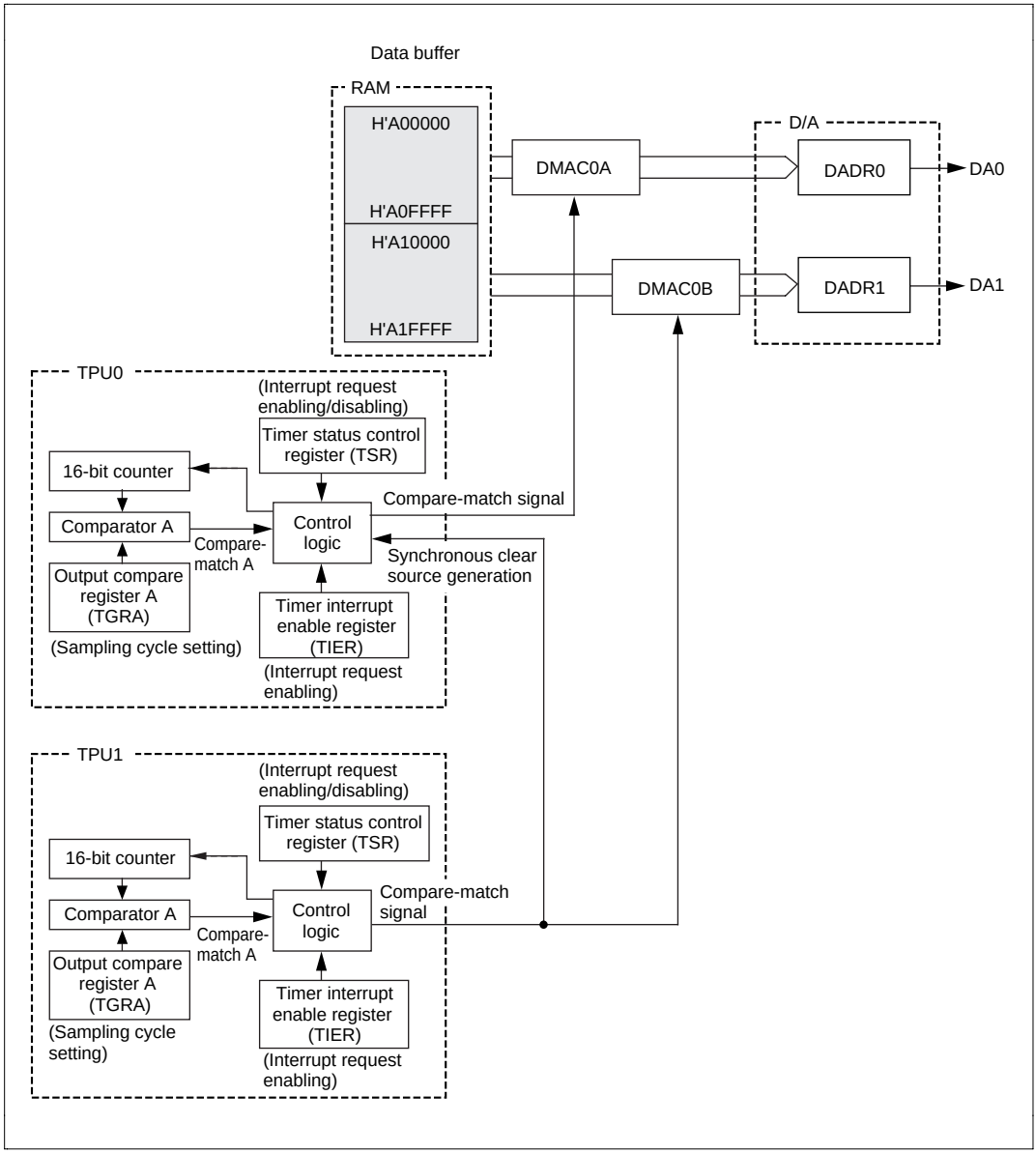


Figure 2 Block Diagram of Analog Output Circuit

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to transfer D/A conversion results to RAM.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
TPU	TCNT0	16-bit counter
	TGR0A	Output compare register
	TCR0	Selects counter clock and counter clear source
	TSR0	Indicates compare-match and overflow status
	TIER0	Selects interrupt enabling/disabling
	TCNT1	16-bit counter
	TGR1A	Output compare register
	TCR1	Selects counter clock and counter clear source
	TSR1	Indicates compare-match and overflow status
	TIER1	Selects interrupt enabling/disabling
	TSYR	Sets simultaneous operation of channels 0 and 1
DMAC	DMABCR	Controls operation of each channel
	DMACR0	Sets sequential mode as transfer mode
	MAR0A	Data start address setting
	MAR0B	Data start address setting
	IOAR0A	DADR0 address setting
	IOAR0B	DADR1 address setting
	ETCR0A	Transfer number setting
	ETCR0B	Transfer number setting
D/A	DADR0	Stores data for conversion (AN0 side)
	DADR1	Stores data for conversion (AN1 side)
	DACR	Controls D/A converter operation
	AV_{CC}	Analog block power supply and reference voltage
	AV_{SS}	Analog block ground and reference voltage
	DA0	Analog output
	DA1	Analog output

Operation

Figure 3 shows the principles of the operation. D/A conversion is performed by means of H8S/2655 hardware and software processing as shown in this figure.

1. Analog output

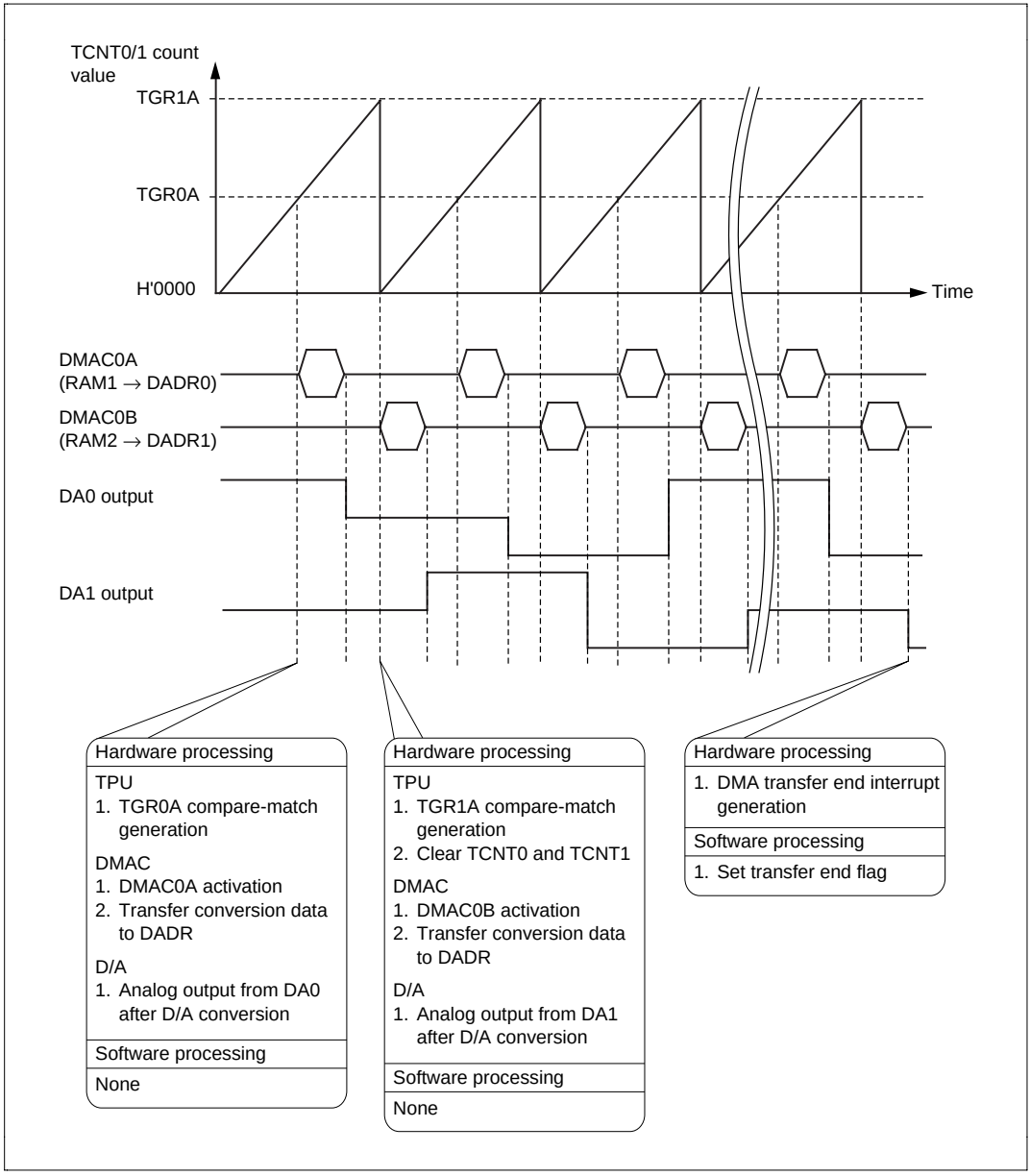


Figure 3 Principles of Analog Output Operation

Software

1. Modules

Module Name	Label	Function
Main routine	dacvtmn	TPU, DMAC, D/A initialization, setting of RAM used
D/A conversion end	datrend	D/A conversion end flag setting

2. Arguments

Label	Function	Data Length	Module	Input/ Output
da_end	Indicates end of data transfer from H'A00000 to H'A1FFFF 1: End of data transfer 0: Data transfer in progress	Unsigned char	Main routine D/A conversion end	Input Output

3. Internal Registers Used

On-Chip Function	Register Name	Function
TPU	TGR0A	D/A conversion sampling cycle setting
	TIER0	Enables TGIA interrupt'
	TCR0	Makes the following TPU0 settings: • Synchronous clearing • Counting on ø internal clock
	TIOR0	Sets TGR0A as output compare register, disables pin output
	TGR1A	D/A conversion sampling cycle setting
	TIER1	Enables TGIA interrupt'
	TCR1	Makes the following TPU0 settings: • Counter clearing by TGR1A compare-match • Counting on ø internal clock
	TIOR1	Sets TGR1A as output compare register, disables pin output
	TSTR	Enables TCNT0 and TCNT1 count operation
	TSYR	Sets synchronous operation of channels 0 and 1

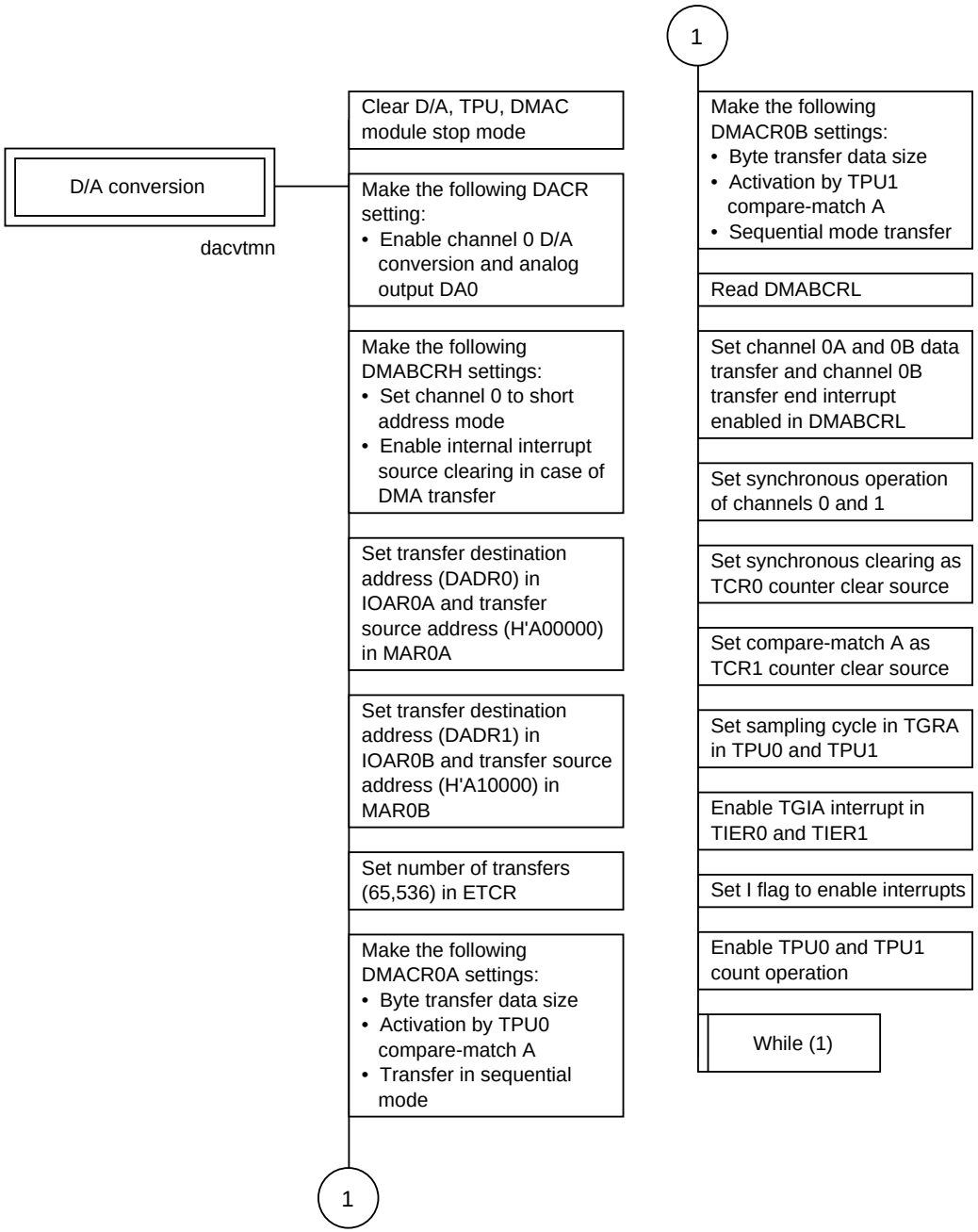
On-Chip Function	Register Name	Function
DMAC	DMABCR	Controls operation of each channel
	DMACR0A	Makes the following DMAC0A settings: • Byte-size transfer • Sequential mode • Internal interrupt source clearing in case of DMA transfer enabled • Data transfer enabled
	DMACR0B	Makes the following DMAC0B settings: • Byte-size transfer • Sequential mode • Internal interrupt source clearing in case of DMA transfer enabled • Data transfer and transfer end interrupt enabled
	MAR0A	Transfer source address (RAM1 start address) setting
	MAR0B	Transfer source address (RAM2 start address) setting
	IOAR0A	Transfer destination address (DADR0) setting
	IOAR0B	Transfer destination address (DADR1) setting
	ETCR0A	Transfer number (H'0000) setting
	ETCR0B	Transfer number (H'0000) setting
D/A	DACR0	DACR is set as follows: • Channel 0 D/A conversion and analog output DA0 enabled
	DADR0	Stores data for conversion
	DADR1	Stores data for conversion
	MSTPCR	Clears module stop mode

4. RAM Used

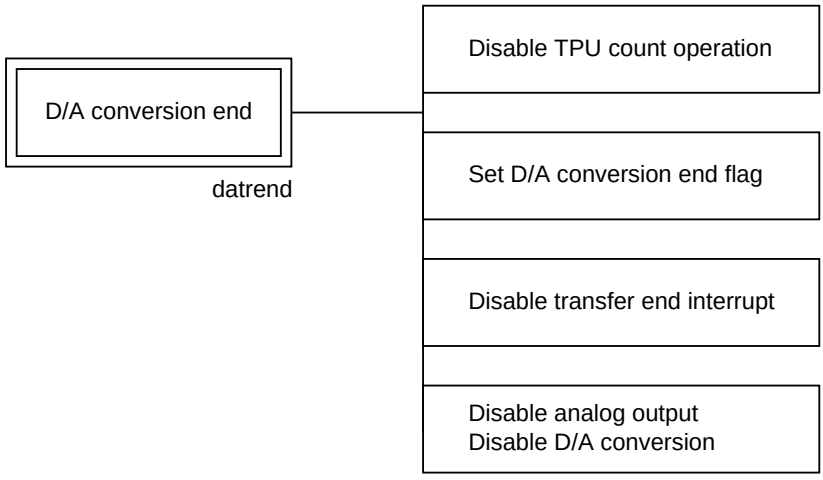
Label	Function	Data Length	Data Capacity
da_data1, da_data2	Stores D/A conversion data	Unsigned char	128 kbytes

PAD

1. Main routine



2. D/A conversion end



Program List

```
#include <machine.h>
#include <h8s.h>

/****************************************/
/*          PROTCOL                     */
/****************************************/
void dacvtmn(void);

/****************************************/
/*          RAM ALLOCATION              */
/****************************************/
#define trs_end (*(volatile unsigned char *)0xffec00)
#define da (*(struct da_data *)0xA00000)
volatile struct da_data
{
                unsigned char    data1[65536];
                unsigned char    data2[65536];
};

/****************************************/
/*       MAIN PROGRAM : dacvtmn         */
/****************************************/
void dacvtmn(void)
{
        MSTPCR = 0x5bff;
        DACR = 0x5f;

        DMABCRH = 0x03;
        IOAR0A = (long)(&DADR0);
        IOAR0B = (long)(&DADR1);
        MAR0A = (long)(&da.data1);
        MAR0B = (long)(&da.data2);
        ETCR0A = 0x0000;
        ETCR0B = 0x0000;
        DMACR0A = 0x08;
        DMACR0B = 0x09;
        DMABCRL |= 0x32;

        TSYR = 0x03;
        TPU_TCR0 = 0xe0;
        TPU_TCR1 = 0x20;
        TGR0A = 0x00c8;
        TGR1A = 0x0190;
        TIER0 = 0x41;
        TIER1 = 0x41;

        set_imask_ccr(0);

        TSTR = 0x03;
        while(1);
}
```

```
/*****************************************/
/*    NAME : datrend(set end flag)       */
/*****************************************/
#pragma interrupt(datrend)
void datrend(void)
{
        TSTR = 0x00;
        trs_end = 0x01;
        DMABCRL &= 0xcd;
        DACR = 0x1f;

}
```

4.6 Simultaneous DTC, DMAC, and CPU Activation

DTC, DMAC, TPU

Specifications

1. The DTC, DMAC, and CPU are activated each time a timer compare-match occurs, as shown in figure 1.

   The DTC transfers data from a data table (ROM) to NDR in the PPG, where pulse output is performed.

   The DMAC transfers 512-byte data stored ion RAM1 to RAM2.

   The CPU monitors the port status and stops DTC and DMAC transfers when the port goes low. However, interrupts continue to be sent to the CPU.

2. The data to be transferred by the DMAC is stored in addresses H'A00000 to H'A002FF.

3. A 20 Hz H8S/2655 internal operating frequency is used.

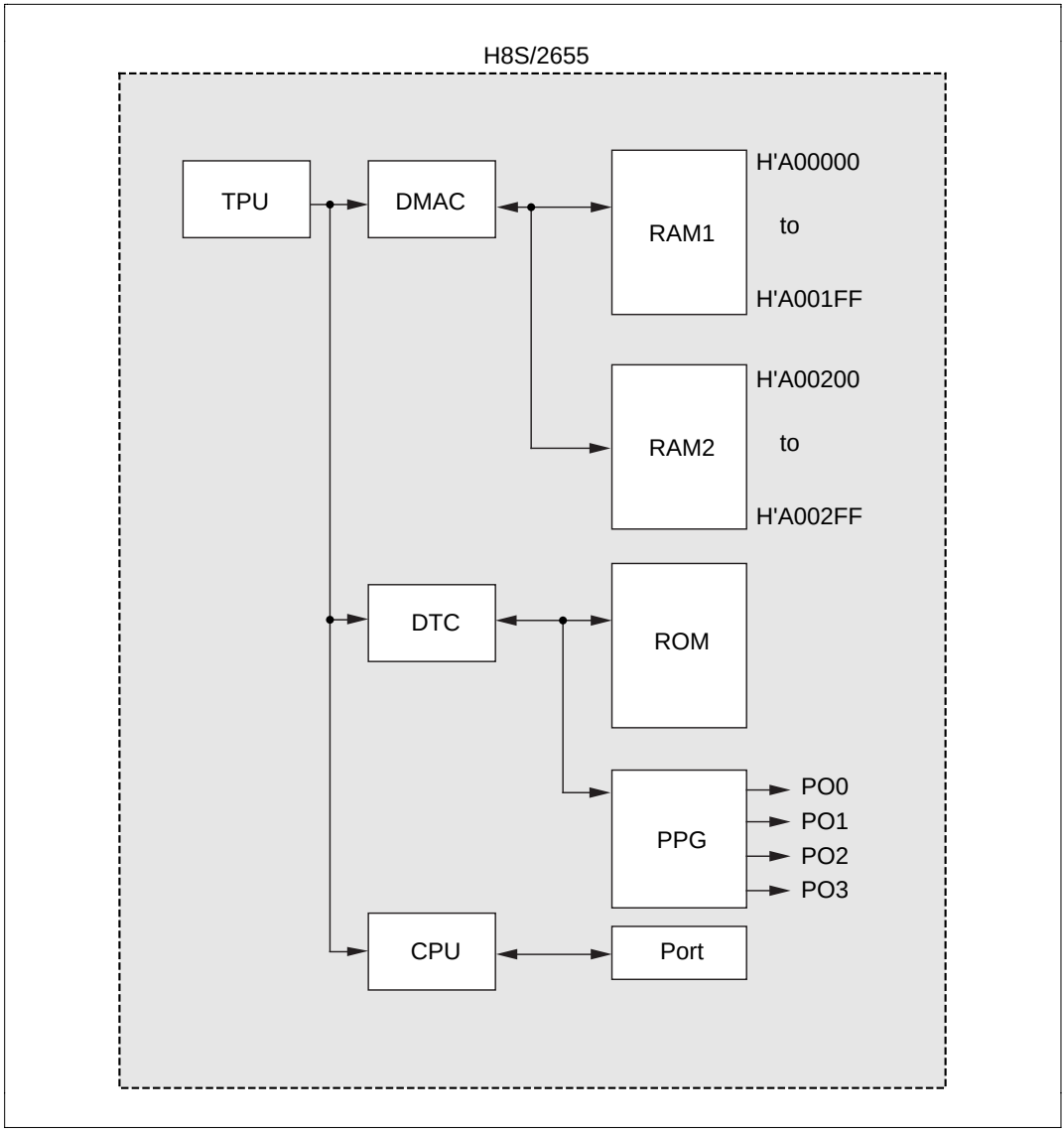


Figure 1 Block Diagram of Simultaneous DTC, DMAC, and CPU Activation

Functions Used

1. In this sample task, the DMAC, DTC, and CPU are activated each time a TPU compare-match occurs.

   a. Figure 2 shows a block diagram of the H8S/2655 on-chip functions used by this sample task.

      The following functions are used to perform to activate the DTC, DMAC, and CPU simultaneously, perform data transfer, and monitor the port status.

      TPU

      Generates compare-matches, DTC and DMAC transfer requests, and CPU interrupt requests.

      DMAC:

      Activated by a TPU compare-match; transfers 512-byte data from RAM1 to RAM2.

      DTC

      Activated by a TPU compare-match; transfers 4-byte data from a data table to NDR in the PPG.

      CPU

      Executes interrupt handling on a TPU compare-match. During interrupt handling, the CPU monitors the port status and controls DMAC and DTC transfers.

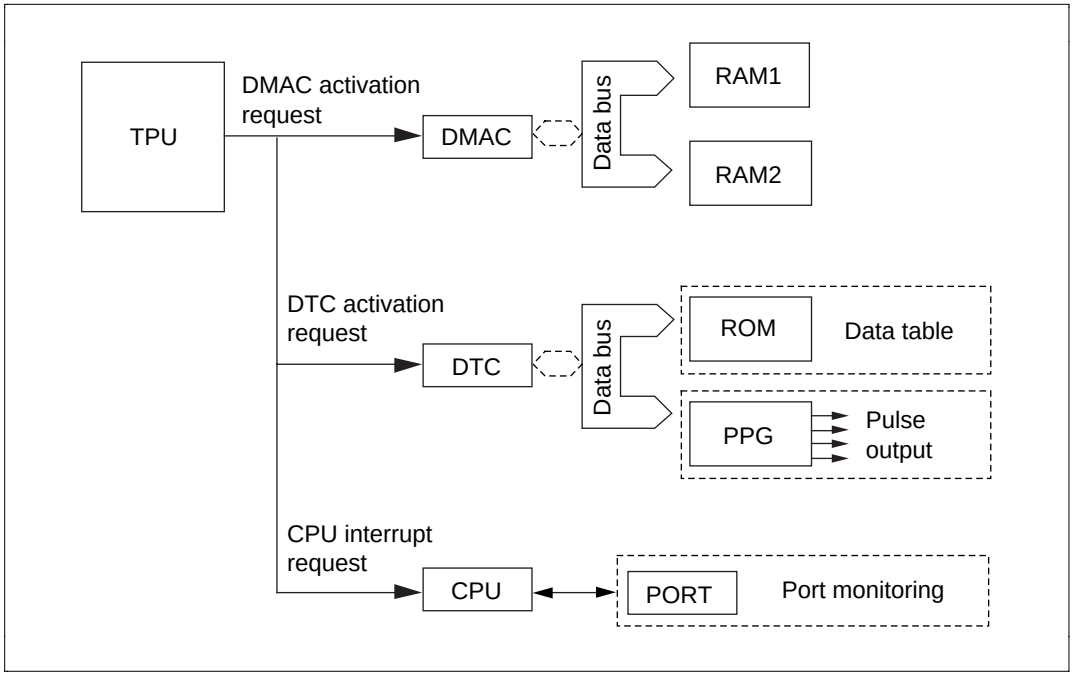


Figure 2 Block Diagram of Simultaneous DTC, DMAC, and CPU Activation

2. Table 1 shows the function assignments for this sample task. H8S/2655 functions are assigned as shown in this table to perform data transfer.

Table 1 H8S/2655 Function Assignments

H8S/2655 Function		Function
DMAC	DMABCR	Makes the following DMAC0A settings: • Full address mode as transfer mode • Internal interrupt source clearing in case of DMA transfer disabled • Data transfer and transfer end interrupt enabled
	DMACR0A	Makes the following DMAC0A settings: • Byte-size data • MAR incremented • Block transfer mode for data transfer • Data transfer direction setting (DMAC0A: MAR → IOAR) • TPU0A as activation source
	MAR0A	RAM1 start address (transfer source) setting
	MAR0B	RAM2 start address (transfer destination) setting
	ETCR0A	Transfer number setting
TPU	TCR0	TCNT clearing by compare-match
	TIOR0	Sets compare-match output disabling
	TIER0	Enables compare-match interrupt
	TSR0	TGRA compare-match interrupt request flag setting
DTC	DTCER	Enabling of DTC activation by TGIA interrupt
PPG	NDER	Enables pulse outputs PO0 to PO15
	NDR	Stores next pulse output data
	PCR	Sets TPU channel 0 compare-match as PPG output request
	PMR	Sets direct output for PPG output

Operation

Figure 3 shows the principles of the operation. Simultaneous DTC, DMAC, and CPU interrupt activation is requested by means of H8S/2655 hardware and software processing, using the timing shown in this figure.

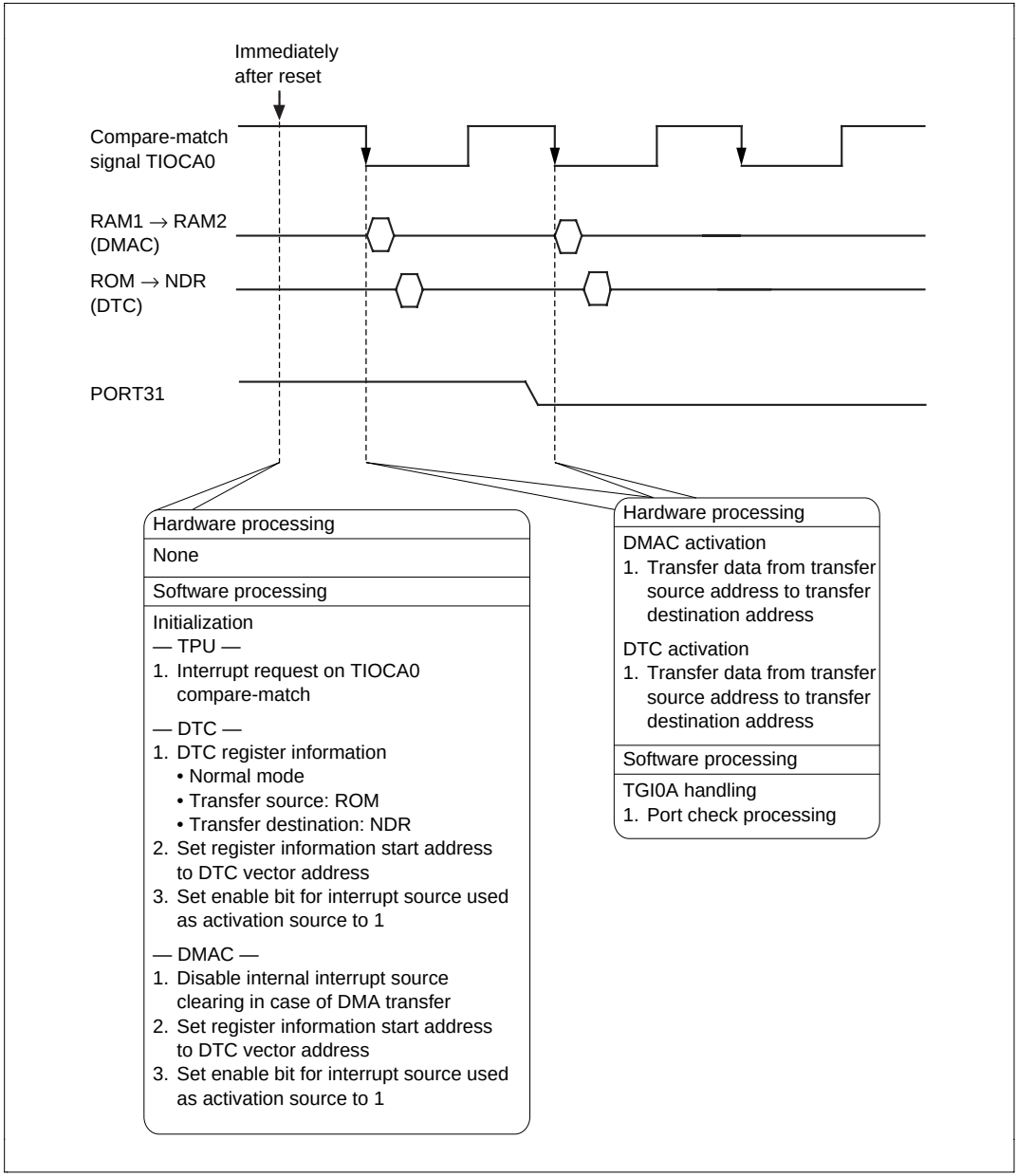


Figure 3 Principles of Simultaneous DTC, DMAC, and CPU Activation Operation

Software

1. Modules

Module Name	Label	Function
Main routine	simbsrmn	TPU, DTC, PPG, DMAC, interrupt handling initialization
Port check	portchk	Port checking and transfer disable processing
Data transfer end	trsend	Data transfer end flag setting

2. Arguments

Register Name	Function	Data Length	Module	Input/ Output
status	Indicates port 31 status 0: Data transfer enabled 1: Data transfer disabled	Unsigned char	Port check	Output
trs_end	Flag indicating end of 512-byte transfer 1: End of data transfer 0: Data transfer in progress	Unsigned char	Data transfer end	Output

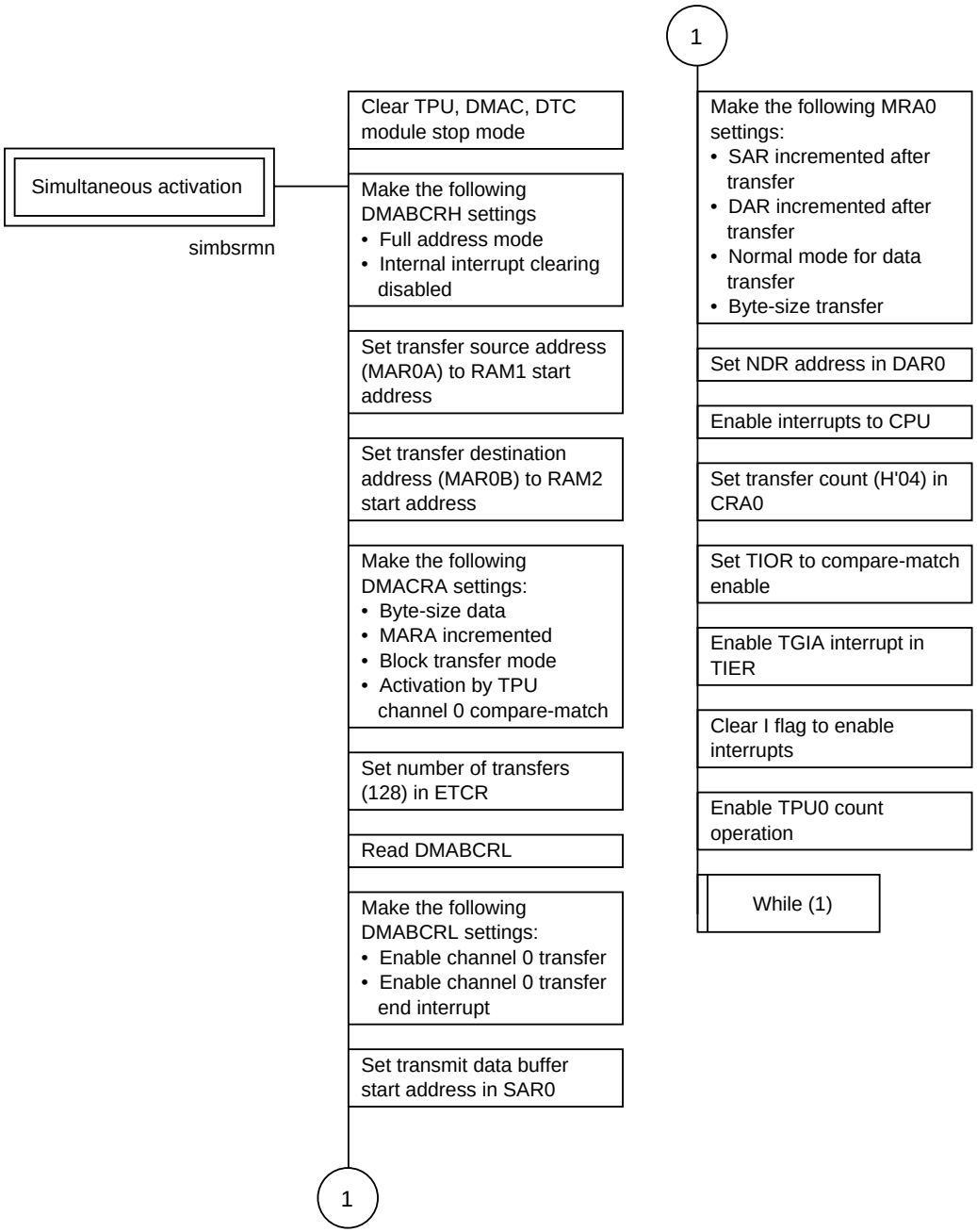
On-Chip Function	Register Name	Function
DMAC	DMABCR	Makes the following DMAC0A settings: • Full address mode as transfer mode • Internal interrupt source clearing in case of DMA transfer disabled • Data transfer and transfer end interrupt enabled
	DMACR0A	Makes the following DMAC0A settings: • Byte-size data • MAR incremented • Block transfer mode for data transfer • Data transfer direction setting (channel 0A: MAR → IOAR) • TPU0A as activation source
	MAR0A	RAM1 start address (trs) setting
	MAR0B	RAM2 start address (rev) setting
	ETCR0A	Transfer number setting
TPU	TCR0	TCNT clearing by compare-match
	TIOR0	Sets compare-match output disabling
	TIER0	Enables compare-match interrupt
	TSR0	TGRA capture interrupt request flag setting
DTC	DTCER	Enabling of DTC activation by TGIA interrupt
PPG	NDER	Enables pulse outputs PO0 to PO15
	NDR	Stores next pulse output data
	PCR	Sets TPU channel 0 compare-match for all pulse output groups
	PMR	Sets direct output for all pulse output groups
	MSTPCR	Controls DTC, TPU, DMAC, PPG module stop mode

4. RAM Used

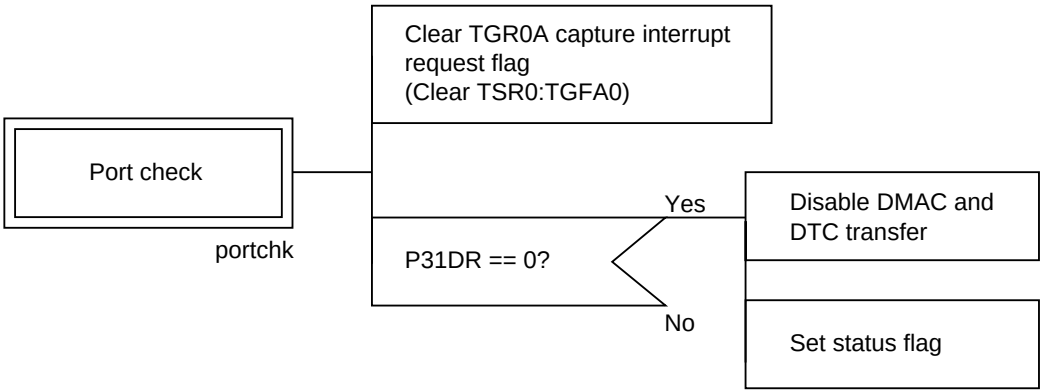
Label	Function	Data Length	Module
MAR0	DTC0 normal mode setting	Unsigned char	Main routine
MRB0	CPU interrupt enabling	Unsigned char	Main routine
SAR0	Transfer source address (PATTBL1) setting	Unsigned long	Main routine
DAR0	Transfer destination address (P1DR) setting	Unsigned long	Main routine
CRA0	Transfer number setting	Unsigned short	Main routine
trs	Stores transmit data	512 bytes	Main routine
rev	Stores receive data	512 bytes	Main routine
PATTBL1	Stores PPG output data	4 bytes	Main routine

PAD

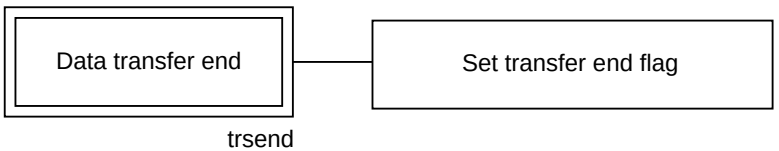
1. Main routine



2. Port check



3. Data transfer end



Program List

```
/****************************************/
/*        FILE NAME : ap17.c            */
/****************************************/
#include <machine.h>
#include "..\h8sapn\h8s.h"
/****************************************/
/*           PROTCOL                    */
/****************************************/
void simbsrmn (void);
#pragma interrupt(trsend)
#pragma interrupt(portchk)
/****************************************/
/*           RAM ALLOCATION             */
/****************************************/
#define status (*(volatile unsigned char *)0xffec00)
#define trs_end (*(volatile unsigned char *)0xffec01)
volatile struct databuf
{
                unsigned char    trs[512];
                unsigned char    rev[512];
};
#define dat (*(struct databuf *)0xa00000)
#define SAR0 (*(volatile unsigned long *)0xfff800)
#define MRA0 (*(volatile unsigned char *)0xfff800)
#define DAR0 (*(volatile unsigned long *)0xfff804)
#define MRB0 (*(volatile unsigned char *)0xfff804)
#define CRA0 (*(volatile unsigned char *)0xfff808)
#define CRB0 (*(volatile unsigned char *)0xfff80a)
const unsigned char PATTBL1[4] = {0xf6,0xf3,0xf9,0xfc};
/****************************************/
/*        MAIN PROGRAM : simbsrmn       */
/****************************************/
void simbsrmn(void)
{
        status = 0;                  /* flag clr */
        trs_end = 0;                 /* flag clr */
        MSTPCR = 0x17ff;             /* Disable module(PPG,DMAC,DTC) stop mode*/
        P1DDR = 0xff;                /* P1,2 : output */
        P2DDR = 0xff;
        NDERH = 0xff;                /* Set next data enable */
        NDERL = 0xff;
        PCR = 0x00;                  /* Set trigger TPU0 compare match */
        PMR = 0xf0;                  /* Set normal mode */
        DMABCRH = 0x40;              /* Initialize DMABCRH */
        MAR0A = (long)(&dat.trs);    /* Set base address */
        MAR0B = (long)(&dat.rev);    /* Set excute address */
        ETCR0A = 0x01ff;             /* Set excute count */
        DMACR0A = 0x20;              /* Initialize DMACR0 */
        DMACR0B = 0x27;
        DMABCRL |= 0x30;             /* Initialize DMABCRL */
        SAR0 = (long)(PATTBL1);      /* Set base address */
        MRA0 = 0x86;                 /* Set repeat mode */
        DAR0 = (long)(&NDRH);        /* Set excute address */
        MRB0 = 0x40;                 /* Initialize MRB */
```

```
        CRA0 = 0x0404;              /* Set excute count */
        DTCERB_BP.TGI0A = 1;        /* Enable DTC */
        TCR0 = 0x20;
        TGR0A = 0x1000;             /* Initialize TGR0A */
        TIOR0H = 0x00;              /* Initialize TIOR0H */
        TIER0 = 0x01;               /* Enable TGI0A interrupt */
        set_imask_ccr(0);           /* Enable interrupt */
        TSTR = 0x01;
        while(1);
}
```

```
/*****************************************/
/*        NAME : portchk                 */
/*****************************************/
void portchk(void)
{
        TSR0_BP.TGFA0 = 0;                      /* Clear TGFA0 flag */

                if (P3DR_BP.P31DR == 1)         /* Check port31 */
                        {
                        status = 1;
                        DMABCRL |= 0x30;        /* Initialize DMABCRL */
                        DTCERB_BP.TGI0A = 1;    /* Disble DTC */
                        }
                else
                        {
                        DMABCRL |= 0x00;        /* Disable DMAC */
                        DTCERB_BP.TGI0A = 0;    /* Disble DTC */
                        }
}
/*****************************************/
/*        NAME : trsend                  */
/*****************************************/
void trsend(void)
{
                ETCR0A = 0x01ff;                /* Set excute count */
                trs_end = 1;                    /* Set DMAC end flag */
}
```

5.1 Internal Register Definitions

H8S/2655 Header File (1) <H8S.H>

```
/*************************************/
/*         SYMBOL DEFNITIONS         */
/*************************************/
struct ISCR_S{                          /* IRQ sense control register */
        unsigned char   IRQ7SCB:1;
        unsigned char   IRQ7SCA:1;
        unsigned char   IRQ6SCB:1;
        unsigned char   IRQ6SCA:1;
        unsigned char   IRQ5SCB:1;
        unsigned char   IRQ5SCA:1;
        unsigned char   IRQ4SCB:1;
        unsigned char   IRQ4SCA:1;
        unsigned char   IRQ3SCB:1;
        unsigned char   IRQ3SCA:1;
        unsigned char   IRQ2SCB:1;
        unsigned char   IRQ2SCA:1;
        unsigned char   IRQ1SCB:1;
        unsigned char   IRQ1SCA:1;
        unsigned char   IRQ0SCB:1;
        unsigned char   IRQ0SCA:1;
};
#define ISCR_BP (*(struct ISCR_S *)0xffff2c)

struct ISR_S{                           /* IRQ0 status register */
        unsigned char   IRQ7F:1;
        unsigned char   IRQ6F:1;
        unsigned char   IRQ5F:1;
        unsigned char   IRQ4F:1;
        unsigned char   IRQ3F:1;
        unsigned char   IRQ2F:1;
        unsigned char   IRQ1F:1;
        unsigned char   IRQ0F:1;
};
#define ISR_BP (*(struct ISR_S *)0xffff2f)

struct IPRA_S{                          /* interrupt priority registerA */
        unsigned char   dummy1:1;
        unsigned char   IPRA6:1;
        unsigned char   IPRA5:1;
        unsigned char   IPRA4:1;
        unsigned char   dummy2:1;
        unsigned char   IPRA2:1;
        unsigned char   IPRA1:1;
        unsigned char   IPRA0:1;
};
#define IPRA_BP (*(struct IPRA_S *)0xfffec4)
```

```
struct DMABCR_S{                        /* DMA band control register */
        unsigned char   FAE1:1;
        unsigned char   FAE0:1;
        unsigned char   SAE1:1;
        unsigned char   SAE0:1;
        unsigned char   DTA1B:1;
        unsigned char   DTA1A:1;
        unsigned char   DTA0B:1;
        unsigned char   DTA0A:1;
        unsigned char   DTE1B:1;
        unsigned char   DTE1A:1;
        unsigned char   DTE0B:1;
        unsigned char   DTE0A:1;
        unsigned char   DTIE1B:1;
        unsigned char   DTIE1A:1;
        unsigned char   DTIE0B:1;
        unsigned char   DTIE0A:1;
};
#define DMABCR_BP (*(struct DMABCR_S *)0xffff06)

struct DTCERA_S{                        /* DTC vector register */
        unsigned char   IRQ0:1;
        unsigned char   IRQ1:1;
        unsigned char   IRQ2:1;
        unsigned char   IRQ3:1;
        unsigned char   IRQ4:1;
        unsigned char   IRQ5:1;
        unsigned char   IRQ6:1;
        unsigned char   IRQ7:1;
};
#define DTCERA_BP (*(struct DTCERA_S *)0xffff30)

struct DTCERB_S{                        /* DTC vector register */
        unsigned char   dummy3:1;
        unsigned char   ADI:1;
        unsigned char   TGI0A:1;
        unsigned char   TGI0B:1;
        unsigned char   TGI0C:1;
        unsigned char   TGI0D:1;
        unsigned char   TGI1A:1;
        unsigned char   TGI1B:1;
};
#define DTCERB_BP (*(struct DTCERB_S *)0xffff31)

struct DTCERC_S{                        /* DTC vector register */
        unsigned char   TGI2A:1;
        unsigned char   TGI2B:1;
        unsigned char   TGI3A:1;
        unsigned char   TGI3B:1;
        unsigned char   TGI3C:1;
        unsigned char   TGI3D:1;
        unsigned char   TGI4A:1;
        unsigned char   TGI4B:1;
};
```

```
#define DTCERC_BP (*(struct DTCERC_S *)0xffff32)

struct DTVECR_S{                        /* DTC vector register */
        unsigned char   SWDTE:1;
        unsigned char   VECR:7;
};
#define DTVECR_BP (*(struct DTVECR_S *)0xffff37)

struct P2DR_S{                          /* port2 data register */
        unsigned char   P27DR:1;
        unsigned char   P26DR:1;
        unsigned char   P25DR:1;
        unsigned char   P24DR:1;
        unsigned char   P23DR:1;
        unsigned char   P22DR:1;
        unsigned char   P21DR:1;
        unsigned char   P20DR:1;
};
#define P2DR_BP (*(struct P2DR_S *)0xffff61)

struct P3DDR_S{
        unsigned char P37:1;
        unsigned char P36:1;
        unsigned char P35DDR:1;
        unsigned char P34DDR:1;
        unsigned char P33DDR:1;
        unsigned char P32DDR:1;
        unsigned char P31DDR:1;
        unsigned char P30DDR:1;
};
#define P3DDR_BP (*(struct P3DDR_S *)0xfffeb2 )

struct P3DR_S{                          /* port3 data register */
        unsigned char   dummy60:1;
        unsigned char   dummy61:1;
        unsigned char   P35DR:1;
        unsigned char   P34DR:1;
        unsigned char   P33DR:1;
        unsigned char   P32DR:1;
        unsigned char   P31DR:1;
        unsigned char   TRG:1;
};
#define P3DR_BP (*(struct P3DR_S *)0xffff62)

struct PORT3_S{
unsigned char dummy43:1;
unsigned char dummy44:1;
unsigned char P35:1;
unsigned char P34:1;
unsigned char P33:1;              /*RTS*/
unsigned char P32:1;
unsigned char P31:1;              /*CTS*/
unsigned char P30:1;
};
```

```
#define PORT3_BP (*(struct PORT3_S *)0xffff52 )

struct P6DR_S{                          /* port6 data register */
        unsigned char   IRQ3:1;
        unsigned char   IRQ2:1;
        unsigned char   RRQ:1;
        unsigned char   IRQ0:1;
        unsigned char   dummy4:1;
        unsigned char   dummy5:1;
        unsigned char   dummy6:1;
        unsigned char   dummy7:1;
};
#define P6DR_BP (*(struct P6DR_S *)0xffff65)

struct PORT6_S{                         /* port6 data register */
        unsigned char   IRQ3:1;
        unsigned char   IRQ2:1;
        unsigned char   RRQ:1;
        unsigned char   SRQ:1;
        unsigned char   dummy4:1;
        unsigned char   dummy5:1;
        unsigned char   dummy6:1;
        unsigned char   dummy7:1;
};
#define PORT6_BP (*(struct PORT6_S *)0xffff55)

struct TIER0_S{                         /* timer interrupt enable
register0 */
        unsigned char   TTGE0:1;
        unsigned char   dummy8:1;
        unsigned char   dummy9:1;
        unsigned char   TCIEV0:1;
        unsigned char   TGIED0:1;
        unsigned char   TGIEC0:1;
        unsigned char   TGIEB0:1;
        unsigned char   TGIEA0:1;
};
#define TIER0_BP (*(struct TIER0_S *)0xffffd4)

struct TSR0_S{                          /* timer status
register0 */
        unsigned char   dummy10:1;
        unsigned char   dummy11:1;
        unsigned char   dummy12:1;
        unsigned char   TCFV0:1;
        unsigned char   TGFD0:1;
        unsigned char   TGFC0:1;
        unsigned char   TGFB0:1;
        unsigned char   TGFA0:1;
};
#define TSR0_BP (*(struct TSR0_S *)0xffffd5)

struct TIER1_S{                         /* timer interrupt enable
register1 */
```

```
        unsigned char   TTGE1:1;
        unsigned char   dummy13:1;
        unsigned char   TCIEU1:1;
        unsigned char   TCIEV1:1;
        unsigned char   dummy14:1;
        unsigned char   dummy15:1;
        unsigned char   TGIEB1:1;
        unsigned char   TGIEA1:1;
};
#define TIER1_BP (*(struct TIER1_S *)0xffffe4)

struct TSR1_S{                          /* timer status
register1 */
        unsigned char   TCFD1:1;
        unsigned char   dummy16:1;
        unsigned char   TCFU1:1;
        unsigned char   TCFV1:1;
        unsigned char   dummy17:1;
        unsigned char   dummy18:1;
        unsigned char   TGFB1:1;
        unsigned char   TGFA1:1;
};
#define TSR1_BP (*(struct TSR1_S *)0xffffe5)

struct TIER2_S{                         /* timer interrupt enable
register2 */
        unsigned char   TTGE2:1;
        unsigned char   dummy19:1;
        unsigned char   TCIEU2:1;
        unsigned char   TCIEV2:1;
        unsigned char   dummy20:1;
        unsigned char   dummy21:1;
        unsigned char   TGIEB2:1;
        unsigned char   TGIEA2:1;
};
#define TIER2_BP (*(struct TIER2_S *)0xfffff4)

struct TSR2_S{                          /* timer status register2 */
        unsigned char   TCFD2:1;
        unsigned char   dummy22:1;
        unsigned char   TCFU2:1;
        unsigned char   TCFV2:1;
        unsigned char   dummy23:1;
        unsigned char   dummy24:1;
        unsigned char   TGFB2:1;
        unsigned char   TGFA2:1;
};
#define TSR2_BP (*(struct TSR2_S *)0xfffff5)

struct TIER3_S{                         /* timer interrupt enable
register3 */
        unsigned char   TTGE3:1;
        unsigned char   dummy25:1;
        unsigned char   dummy26:1;
```

```
        unsigned char   TCIEV3:1;
        unsigned char   TGIED3:1;
        unsigned char   TGIEC3:1;
        unsigned char   TGIEB3:1;
        unsigned char   TGIEA3:1;
};
#define TIER3_BP (*(struct TIER3_S *)0xfffe84)

struct TSR3_S{            /* timer status register3 */
        unsigned char   dummy27:1;
        unsigned char   dummy28:1;
        unsigned char   dummy29:1;
        unsigned char   TCFV3:1;
        unsigned char   TGFD3:1;
        unsigned char   TGFC3:1;
        unsigned char   TGFB3:1;
        unsigned char   TGFA3:1;
};
#define TSR3_BP (*(struct TSR3_S *)0xfffe85)

struct TIER4_S{            /* timer interrupt enable
register4 */
        unsigned char   TTGE4:1;
        unsigned char   dummy30:1;
        unsigned char   TCIEU4:1;
        unsigned char   TCIEV4:1;
        unsigned char   dummy31:1;
        unsigned char   dummy32:1;
        unsigned char   TGIEB4:1;
        unsigned char   TGIEA4:1;
};
#define TIER4_BP (*(struct TIER4_S *)0xfffe94)

struct TSR4_S{                          /* timer status register4 */
        unsigned char   TCFD4:1;
        unsigned char   dummy33:1;
        unsigned char   TCFU4:1;
        unsigned char   TCFV4:1;
        unsigned char   dummy34:1;
        unsigned char   dummy35:1;
        unsigned char   TGFB4:1;
        unsigned char   TGFA4:1;
};
#define TSR4_BP (*(struct TSR4_S *)0xfffe95)

struct TIER5_S{            /* timer interrupt enable
register5 */
        unsigned char   TTGE5:1;
        unsigned char   dummy36:1;
        unsigned char   TCIEU5:1;
        unsigned char   TCIEV5:1;
        unsigned char   dummy37:1;
        unsigned char   dummy38:1;
        unsigned char   TGIEB5:1;
```

```
        unsigned char   TGIEA5:1;
};
#define TIER5_BP (*(struct TIER5_S *)0xfffea4)
struct TSR5_S{                          /* timer status register5 */
        unsigned char   TCFD5:1;
        unsigned char   dummy39:1;
        unsigned char   TCFU5:1;
        unsigned char   TCFV5:1;
        unsigned char   dummy40:1;
        unsigned char   dummy41:1;
        unsigned char   TGFB5:1;
        unsigned char   TGFA5:1;
};
#define TSR5_BP (*(struct TSR5_S *)0xfffea5)

struct TSTR_S{                          /* timer start register */
        unsigned char   dummy42:1;
        unsigned char   dummy43:1;
        unsigned char   CST5:1;
        unsigned char   CST4:1;
        unsigned char   CST3:1;
        unsigned char   CST2:1;
        unsigned char   CST1:1;
        unsigned char   CST0:1;
};
#define TSTR_BP (*(struct TSTR_S *)0xffffc0)

struct SCR0_S{                          /* serial control register0 */
        unsigned char   TIE0:1;
        unsigned char   RIE0:1;
        unsigned char   TE0:1;
        unsigned char   RE0:1;
        unsigned char   MPIE0:1;
        unsigned char   TEIE0:1;
        unsigned char   CKE10:1;
        unsigned char   CKE00:1;
};
#define SCR0_BP (*(struct SCR0_S *)0xffff7a)

struct SSR0_S{                          /* serial status register0 */
        unsigned char   TDRE0:1;
        unsigned char   RDRF0:1;
        unsigned char   OPER0:1;
        unsigned char   FER0:1;
        unsigned char   PER0:1;
        unsigned char   TEND0:1;
        unsigned char   MPB0:1;
        unsigned char   MPBT0:1;
};
#define SSR0_BP (*(struct SSR0_S *)0xffff7c)

struct SCR1_S{                          /* serial control register1 */
        unsigned char   TIE1:1;
        unsigned char   RIE1:1;
```

```
        unsigned char   TE1:1;
        unsigned char   RE1:1;
        unsigned char   MPIE1:1;
        unsigned char   TEIE1:1;
        unsigned char   CKE11:1;
        unsigned char   CKE01:1;
};
#define SCR1_BP (*(struct SCR1_S *)0xffff82)
struct SSR1_S{                          /* serial status register1 */
        unsigned char   TDRE1:1;
        unsigned char   RDRF1:1;
        unsigned char   OPER1:1;
        unsigned char   FER1:1;
        unsigned char   PER1:1;
        unsigned char   TEND1:1;
        unsigned char   MPB1:1;
        unsigned char   MPBT1:1;
};
#define SSR1_BP (*(struct SSR1_S *)0xffff84)

struct SCR2_S{                          /* serial control register2 */
        unsigned char   TIE2:1;
        unsigned char   RIE2:1;
        unsigned char   TE2:1;
        unsigned char   RE2:1;
        unsigned char   MPIE2:1;
        unsigned char   TEIE2:1;
        unsigned char   CKE12:1;
        unsigned char   CKE02:1;
};
#define SCR2_BP (*(struct SCR2_S *)0xffff8a)

struct SSR2_S{                          /* serial status register2 */
        unsigned char   TDRE2:1;
        unsigned char   RDRF2:1;
        unsigned char   OPER2:1;
        unsigned char   FER2:1;
        unsigned char   PER2:1;
        unsigned char   TEND2:1;
        unsigned char   MPB2:1;
        unsigned char   MPBT2:1;
};
#define SSR2_BP (*(struct SSR2_S *)0xffff8c)

struct ADCSR_S{                         /* serial status register2 */
        unsigned char   ADF:1;
        unsigned char   ADIE:1;
        unsigned char   ADST:1;
        unsigned char   CKS:1;
        unsigned char   GRP:1;
        unsigned char   CH2:1;
        unsigned char   CH1:1;
        unsigned char   CH0:1;
};
```

```
#define ADCSR_BP (*(struct ADCSR_S *)0xffffa0)

struct MSTPCR_S{                        /* module stop mode */
        unsigned char   MSTP15:1;       /* DMA controller */
        unsigned char   MSTP14:1;       /* DTC */
        unsigned char   MSTP13:1;       /* TPU */
        unsigned char   MSTP12:1;       /* 8bit timer */
        unsigned char   MSTP11:1;       /* PPG */
        unsigned char   MSTP10:1;       /* D/A */
        unsigned char   MSTP9:1;        /* A/D */
        unsigned char   MSTP8:1;
        unsigned char   MSTP7:1;        /* SCI2 */
        unsigned char   MSTP6:1;        /* SCI1 */
        unsigned char   MSTP5:1;        /* SCI0 */
        unsigned char   MSTP4:1;
        unsigned char   MSTP3:1;
        unsigned char   MSTP2:1;
        unsigned char   MSTP1:1;
        unsigned char   MSTP0:1;
};
#define MSTPCR_BP (*(struct MSTPCR_S *)0xffff3c)
#define ISCRH   (*(volatile unsigned char *)0xffff2c)
#define ISCRL   (*(volatile unsigned char *)0xffff2d)
#define IER     (*(volatile unsigned char *)0xffff2e)
#define ISR     (*(volatile unsigned char *)0xffff2f)
#define IPRA    (*(volatile unsigned char *)0xfffec4)

#define ABWCR   (*(volatile unsigned char *)0xfffed0)
#define ASTCR   (*(volatile unsigned char *)0xfffed1)
#define WCRH    (*(volatile unsigned char *)0xfffed2)
#define WCRL    (*(volatile unsigned char *)0xfffed3)
#define BCRH    (*(volatile unsigned char *)0xfffed4)
#define BCRL    (*(volatile unsigned char *)0xfffed5)
#define MCR     (*(volatile unsigned char *)0xfffed6)
#define DRAMCR  (*(volatile unsigned char *)0xfffed7)

#define MAR0A_B (*(volatile unsigned char **)0xfffee0)
#define MAR0A_W (*(volatile unsigned short **)0xfffee0)
#define MAR0A_L (*(volatile unsigned long **)0xfffee0)
#define MAR0B_B (*(volatile unsigned char **)0xfffee8)
#define MAR0B_W (*(volatile unsigned short **)0xfffee8)
#define MAR0B_L (*(volatile unsigned long **)0xfffee8)

#define MAR0A   (*(volatile unsigned long *)0xfffee0)
#define IOAR0A  (*(volatile unsigned short *)0xfffee4)
#define ETCR0A  (*(volatile unsigned short *)0xfffee6)
#define MAR0B   (*(volatile unsigned long *)0xfffee8)
#define IOAR0B  (*(volatile unsigned short *)0xfffeec)
#define ETCR0B  (*(volatile unsigned short *)0xfffeee)

#define MAR1A   (*(volatile unsigned long *)0xfffef0)
#define IOAR1A  (*(volatile unsigned short *)0xfffef4)
#define ETCR1A  (*(volatile unsigned short *)0xfffef6)
#define MAR1B   (*(volatile unsigned long *)0xfffef8)
```

```
#define IOAR1B  (*(volatile unsigned short *)0xfffefc)
#define ETCR1B  (*(volatile unsigned short *)0xfffefe)

#define DMAWER  (*(volatile unsigned char *)0xffff00)
#define DMATCR  (*(volatile unsigned char *)0xffff01)
#define DMACR0A (*(volatile unsigned char *)0xffff02)
#define DMACR0B (*(volatile unsigned char *)0xffff03)
#define DMACR1A (*(volatile unsigned char *)0xffff04)
#define DMACR1B (*(volatile unsigned char *)0xffff05)

#define DMABCRH (*(volatile unsigned char *)0xffff06)
#define DMABCRL (*(volatile unsigned char *)0xffff07)

#define DTCERA  (*(volatile unsigned char *)0xffff30)
#define DTCERB  (*(volatile unsigned char *)0xffff31)
#define DTCERC  (*(volatile unsigned char *)0xffff32)
#define DTCERD  (*(volatile unsigned char *)0xffff33)
#define DTCERE  (*(volatile unsigned char *)0xffff34)
#define DTCERF  (*(volatile unsigned char *)0xffff35)
#define DTVECR  (*(volatile unsigned char *)0xffff37)

#define P1DDR    (*(volatile unsigned char *)0xfffeb0)
#define P1DR     (*(volatile unsigned char *)0xffff60)
#define P2DDR    (*(volatile unsigned char *)0xfffeb1)
#define P2DR     (*(volatile unsigned char *)0xffff61)
#define P3DDR    (*(volatile unsigned char *)0xfffeb2)
#define P3DR     (*(volatile unsigned char *)0xffff62)
#define P5DDR    (*(volatile unsigned char *)0xfffeb4)
#define P5DR     (*(volatile unsigned char *)0xffff64)
#define P6DDR    (*(volatile unsigned char *)0xfffeb5)
#define P6DR     (*(volatile unsigned char *)0xffff65)
#define PADDR    (*(volatile unsigned char *)0xfffeb9)
#define PADR     (*(volatile unsigned char *)0xffff69)
#define PBDDR    (*(volatile unsigned char *)0xfffeba)
#define PBDR     (*(volatile unsigned char *)0xffff6a)
#define PCDDR    (*(volatile unsigned char *)0xfffebb)
#define PCDR     (*(volatile unsigned char *)0xffff6b)
#define PDDDR    (*(volatile unsigned char *)0xfffebc)
#define PDDR     (*(volatile unsigned char *)0xffff6c)
#define PEDDR    (*(volatile unsigned char *)0xfffebd)
#define PEDR     (*(volatile unsigned char *)0xffff6d)
#define PFDDR    (*(volatile unsigned char *)0xfffebe)
#define PFDR     (*(volatile unsigned char *)0xffff6e)
#define PGDDR    (*(volatile unsigned char *)0xfffebf)
#define PGDR     (*(volatile unsigned char *)0xffff6f)

#define TPU_TCR0 (*(volatile unsigned char *)0xffffd0)
#define TMDR0    (*(volatile unsigned char *)0xffffd1)
#define TIOR0H   (*(volatile unsigned char *)0xffffd2)
#define TIOR0L   (*(volatile unsigned char *)0xffffd3)
#define TIER0    (*(volatile unsigned char *)0xffffd4)
#define TSR0     (*(volatile unsigned char *)0xffffd5)
#define TPU_TCNT0 (*(volatile unsigned short *)0xffffd6)
#define TGR0A    (*(volatile unsigned short *)0xffffd8)
```

```
#define TGR0B    (*(volatile unsigned short *)0xffffda)
#define TGR0C    (*(volatile unsigned short *)0xffffdc)
#define TGR0D    (*(volatile unsigned short *)0xffffde)

#define TPU_TCR1 (*(volatile unsigned char *)0xfffffe0)
#define TMDR1    (*(volatile unsigned char *)0xfffffe1)
#define TIOR1H   (*(volatile unsigned char *)0xfffffe2)
#define TIER1    (*(volatile unsigned char *)0xfffffe4)
#define TSR1     (*(volatile unsigned char *)0xfffffe5)
#define TPU_TCNT1 (*(volatile unsigned short *)0xfffffe6)
#define TGR1A    (*(volatile unsigned short *)0xfffffe8)
#define TGR1B    (*(volatile unsigned short *)0xfffffea)

#define TCR2     (*(volatile unsigned char *)0xffffff0)
#define TMDR2    (*(volatile unsigned char *)0xffffff1)
#define TIOR2H   (*(volatile unsigned char *)0xffffff2)
#define TIOR2L   (*(volatile unsigned char *)0xffffff3)
#define TIER2    (*(volatile unsigned char *)0xffffff4)
#define TSR2     (*(volatile unsigned char *)0xffffff5)
#define TCNT2    (*(volatile unsigned short *)0xffffff6)
#define TGR2A    (*(volatile unsigned short *)0xffffff8)
#define TGR2B    (*(volatile unsigned short *)0xffffffa)

#define TCR3    (*(volatile unsigned char *)0xfffe80)
#define TMDR3   (*(volatile unsigned char *)0xfffe81)
#define TIOR3H  (*(volatile unsigned char *)0xfffe82)
#define TIOR3L  (*(volatile unsigned char *)0xfffe83)
#define TIER3   (*(volatile unsigned char *)0xfffe84)
#define TSR3    (*(volatile unsigned char *)0xfffe85)
#define TCNT3   (*(volatile unsigned short *)0xfffe86)
#define TGR3A   (*(volatile unsigned short *)0xfffe88)
#define TGR3B   (*(volatile unsigned short *)0xfffe8a)
#define TGR3C   (*(volatile unsigned short *)0xfffe8c)
#define TGR3D   (*(volatile unsigned short *)0xfffe8e)

#define TCR4    (*(volatile unsigned char *)0xfffe90)
#define TMDR4   (*(volatile unsigned char *)0xfffe91)
#define TIOR4H  (*(volatile unsigned char *)0xfffe92)
#define TIOR4L  (*(volatile unsigned char *)0xfffe93)
#define TIER4   (*(volatile unsigned char *)0xfffe94)
#define TSR4    (*(volatile unsigned char *)0xfffe95)
#define TCNT4   (*(volatile unsigned short *)0xfffe96)
#define TGR4A   (*(volatile unsigned short *)0xfffe98)
#define TGR4B   (*(volatile unsigned short *)0xfffe9a)

#define TCR5    (*(volatile unsigned char *)0xfffea0)
#define TMDR5   (*(volatile unsigned char *)0xfffea1)
#define TIOR5H  (*(volatile unsigned char *)0xfffea2)
#define TIOR5L  (*(volatile unsigned char *)0xfffea3)
#define TIER5   (*(volatile unsigned char *)0xfffea4)
#define TSR5    (*(volatile unsigned char *)0xfffea5)
#define TCNT5   (*(volatile unsigned short *)0xfffea6)
#define TGR5A   (*(volatile unsigned short *)0xfffea8)
#define TGR5B   (*(volatile unsigned short *)0xfffeaa)
```

```
#define TSTR    (*(volatile unsigned char *)0xffffc0)
#define TSYR    (*(volatile unsigned char *)0xffffc1)


#define PCR     (*(volatile unsigned char *)0xffff46)
#define PMR     (*(volatile unsigned char *)0xffff47)
#define NDERH   (*(volatile unsigned char *)0xffff48)
#define NDERL   (*(volatile unsigned char *)0xffff49)
#define PODRH   (*(volatile unsigned char *)0xffff4A)
#define PODRL   (*(volatile unsigned char *)0xffff4B)
#define NDRH    (*(volatile unsigned char *)0xffff4C)
#define NDRL    (*(volatile unsigned char *)0xffff4D)
#define NDR3    (*(volatile unsigned char *)0xffff4C)
#define NDR2    (*(volatile unsigned char *)0xffff4E)
#define NDR1    (*(volatile unsigned char *)0xffff4D)
#define NDR0    (*(volatile unsigned char *)0xffff4F)


#define TCR0    (*(volatile unsigned char *)0xffffb0)
#define TCSR0   (*(volatile unsigned char *)0xffffb2)
#define TCORA0  (*(volatile unsigned char *)0xffffb4)
#define TCORB0  (*(volatile unsigned char *)0xffffb6)
#define TCNT0   (*(volatile unsigned char *)0xffffb8)

#define TCR1    (*(volatile unsigned char *)0xffffb1)
#define TCSR1   (*(volatile unsigned char *)0xffffb3)
#define TCORA1  (*(volatile unsigned char *)0xffffb5)
#define TCORB1  (*(volatile unsigned char *)0xffffb7)
#define TCNT1   (*(volatile unsigned char *)0xffffb9)

#define SMR0    (*(volatile unsigned char *)0xffff78)
#define BRR0    (*(volatile unsigned char *)0xffff79)
#define SCR0    (*(volatile unsigned char *)0xffff7a)
#define TDR0    (*(volatile unsigned char *)0xffff7b)
#define SSR0    (*(volatile unsigned char *)0xffff7c)
#define RDR0    (*(volatile unsigned char *)0xffff7d)
#define SCMR0   (*(volatile unsigned char *)0xffff7e)

#define SMR1    (*(volatile unsigned char *)0xffff80)
#define BRR1    (*(volatile unsigned char *)0xffff81)
#define SCR1    (*(volatile unsigned char *)0xffff82)
#define TDR1    (*(volatile unsigned char *)0xffff83)
#define SSR1    (*(volatile unsigned char *)0xffff84)
#define RDR1    (*(volatile unsigned char *)0xffff85)
#define SCMR1   (*(volatile unsigned char *)0xffff86)

#define SMR2    (*(volatile unsigned char *)0xffff88)
#define BRR2    (*(volatile unsigned char *)0xffff89)
#define SCR2    (*(volatile unsigned char *)0xffff8a)
#define TDR2    (*(volatile unsigned char *)0xffff8b)
#define SSR2    (*(volatile unsigned char *)0xffff8c)
#define RDR2    (*(volatile unsigned char *)0xffff8d)
#define SCMR2   (*(volatile unsigned char *)0xffff8e)
```

```
#define ADDRA   (*(volatile unsigned short *)0xffff90)
#define ADDRB   (*(volatile unsigned short *)0xffff92)
#define ADDRC   (*(volatile unsigned short *)0xffff94)
#define ADDRD   (*(volatile unsigned short *)0xffff96)
#define ADDRE   (*(volatile unsigned short *)0xffff98)
#define ADDRF   (*(volatile unsigned short *)0xffff9a)
#define ADDRG   (*(volatile unsigned short *)0xffff9c)
#define ADDRH   (*(volatile unsigned short *)0xffff9e)
#define ADCSR   (*(volatile unsigned char *)0xffffa0)
#define ADCR    (*(volatile unsigned char *)0xffffa1)


#define DADR0   (*(volatile unsigned char *)0xffffa4)
#define DADR1   (*(volatile unsigned char *)0xffffa5)
#define DACR    (*(volatile unsigned char *)0xffffa6)


#define MSTPCR  (*(volatile unsigned short *)0xffff3c)
```

H8S/2655 Series Application Note

Publication Date: 1st Edition, September 1997
Published by: Semiconductor and IC Div.
Hitachi, Ltd.
Edited by: Technical Documentation Center
Hitachi Microcomputer System Ltd.