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.

SuperH™ RISC engine Peripheral LSI

HD64413A Q2SD

(Quick 2D Graphics Renderer with Synchronous DRAM Interface)

Application Note

ADE-502-070
Rev. 1.0
10/12/1999
Hitachi, Ltd.

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

Preface

HD64413A (Q2SD: Quick 2D Graphics Renderer with Synchronous DRAM Interface) allows various expressions in drawing processing by replacing a part of drawing which was performed by using the software (functions) of SuperH™* with hardware (drawing).

This application note describes the points in providing an interface with SuperH for using HD64413A (Q2SD) and together with some sample programs, as a reference for designing software.

Incidentally, HD64413A (Q2SD) a chip set for SuperH, takes over the drawing commands of HD64412 (Q2i), the second product of the Q Series (Quick Series), as well as its functions of background screen display and suspension/resumption of drawing, and adds video capture functions.

Note: * SuperH is a trademark of Hitachi, Ltd.

Additionally, all the sample programs shown here are prepared under the following conditions.

Operating environment:
SH7709 solution engine (MS7709SE01)
Q2SD daughter board (MS4413DB01) Hitachi ULSI Systems Co., Ltd.

- SuperH conditions
  SH7709 (HD6417709)
  Operating frequency: Internal 80MHz; External 20MHz
  Main memory capacity: 32Mbyte DRAM (MH5164805TT6×4)

- Q Series conditions
  Q2SD (HD64413A)
  CLK0 frequency: 66MHz
  CLK1 frequency: 7.15909MHz (14.31818 for sample 7.c only)
  Display size: 640×240 pixels(640×480 pixels for sample 7.c only)
  Scan mode: Non-interlace mode (interlace sink & video mode for sample 7.c only)
  Expression color: 65536 colors (16 bit/pixel)
  UGM capacity: 32Mbyte SDRAM (μ/PD4564323G5×1)
  NTSC encoder: CXA2075M

- Sample program creating environment
  Program language: C language
  Compiler: Hitachi SH Series C compiler
  SH SERIES C Compiler Ver.5.0
  Linkage editor: Hitachi SH Series linkage editor
  H SERIES LINKAGE EDITOR Ver.5.3
  Object converter: Hitachi H Series object converter
  H SERIES SYSROF STYPE OBJECT CONVERTER Ver.1.5B

Note: The company names and product names mentioned here are the trademarks and registered product names of each company.
Also, the sample programs introduced in the application note hereof were prepared with an aim to evaluate HD64413A. Accordingly, it should be noted that use of these programs in your system, as are, in whole or in part, is not permitted.
The sample circuits shown in this document are intended only as examples of typical applications. As such, Hitachi, Ltd., accepts no responsibility for loss or damage arising from the use of these circuits.

Contents

Section 1 Example of System Configuration

HD64413A, a chip set of SuperH, can configure a system by directly connecting it together with SDRAM.

Also, for dot clock signal CLK1 for display and the operating clock (MCLK) of HD64413A, a non-synchronous clock can be used within a range their frequencies satisfy the relationship MCLK≥2×CLK1.

The display size is determined by the maximum clock frequency that can be inputted to CLK1. For example, the display size when HD64413A operates under non-interlace mode is roughly 320×240 to 480×240 dots, and under interlace sink & video mode, it is about 640×480 dots.

A synthetic display with external video signals can be performed by setting HD64413A to TV-synchronous mode and by supplying HSYNC, VSYNC, ODDF and CKL1 from an external device to HD64413A.

Also, a video display can be performed by video-capturing by digitally encoding video signals.

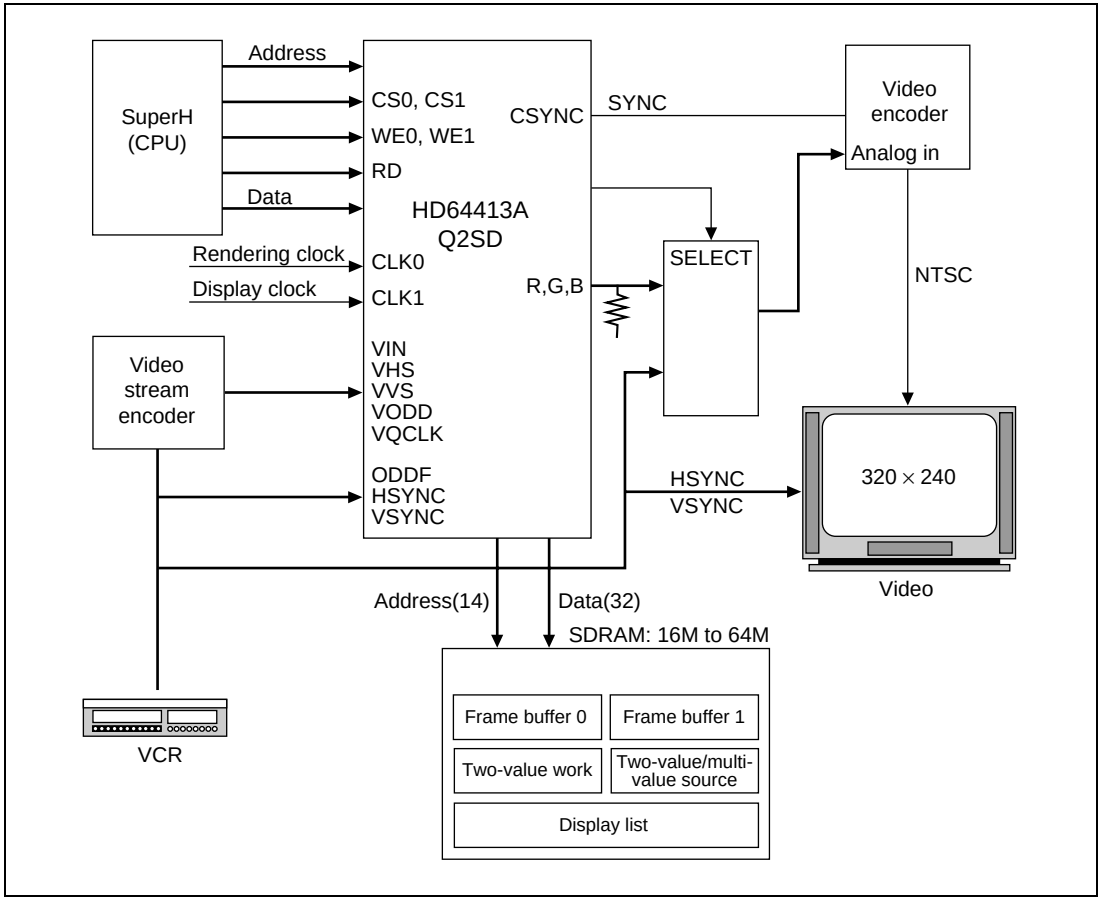


Figure 1.1 Example of System Configuration Overview

Section 2 Interface with SuperH

2.1 Determination of Clock

The clocks supplied to HD64413A are clocks inputted to the CLK1 pin and those inputted to the CLK0 pin. The clocks inputted to the CLK1 pin are used as clocks for display control, and the clocks inputted to the CLK0 pin are used as operating clocks.

1. For the CLK0 pin, the following clock type a or b can be used.

   a. Method for using clocks outputted from CKI0 pin of SuperH

      When a SuperH (SH-3, SH-4) operating with 3.3V is used as the CPU, clocks outputted from the CKIO pin can be used as input clocks for the CLK0 pin.

      Also, to increase the fan-out of CKIO pin, input the output clock of the CKIO pin to the CLK0 pin of HD64413A via a buffer circuit.

   b. Method for using clocks other than those outputted from CKIO pin of CPU

      Clocks of 3.3V level can be used as input clocks for the CLK0 pin.

2. Input clocks for the CLK1 pin must satisfy the following conditions:

   $$MCLK[Hz] \geq 2 \times CLK1\ [Hz]\ (CLK1 \leq 33.3MHz)$$

   $$MCLK = N \times CLK0\ (N: \text{either of multiple 1, 2 or 4})$$

2.2 Setting of Software Weight

The software weight cycle of SuperH is determined by the relationship of the external bus operating frequency (CKI0) of SuperH and the internal operating frequency (MCLK) of HD64413A.

Set the software cycle so that SuperH can detect the $\overline{\text{WAIT}}$ signal outputted by HD64413A, taking into consideration the AC timing of both SuperH and HD64413A.

Here a case in which CKI0=20MHz and MCLK=66MHz are used using SH-3 is described. As shown in the figure below, by setting the software cycle (Tw) of SuperH to 2, the rule of tWTS and tWTH, which governs the $\overline{\text{WAIT}}$ pin of SuperH can be observed and the hardware cycle (Twx) between SuperH and HD64413A can be defined. (tWAS1=3tcyc0+15ns (MAX))

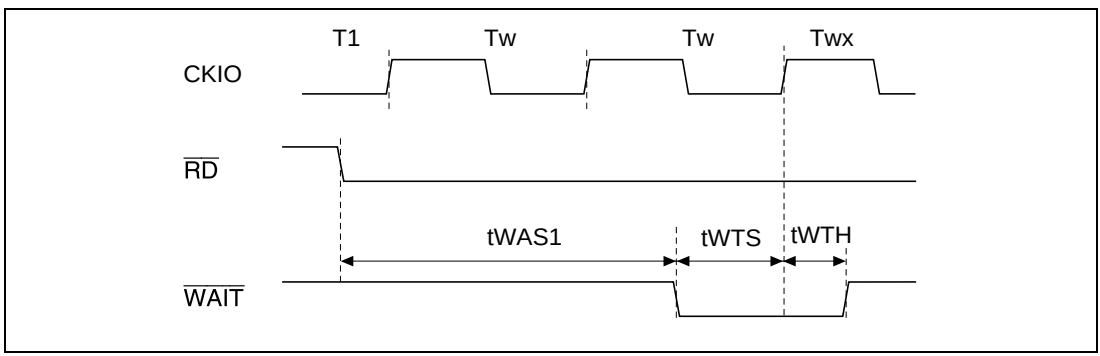


Figure 2.1 Example of Interface Timing

2.3 Special Notes on Connection

When connecting SuperH to HD64413A, note the following:

1. When the initial value of $\overline{\text{CS}}$ pin of SuperH is an input port and signals connected to the $\overline{\text{CS0}}$ and $\overline{\text{CS1}}$ pins of HD64413A are generated from this pin, pull up the $\overline{\text{CS}}$ pin of SuperH so that the voltage level not to become unstable after canceling hardware reset.
2. When the DMAC built in SuperH is used and the DACK pin is set to active high by initial value setting, connect the inverted signal of DACK pin to the DACK pin of HD64413A by the external circuit. Incidentally, use the DACK pin set to active high, as is.
3. When using SH-4 for SuperH, invert signals outputted from the $\overline{\text{WAIT}}$ pin of HD64413A by the external circuit and input them to the $\overline{\text{RDY}}$ pin of SH-4.

2.4 Initialization Procedures of Address-Mapped Register

Standard procedures of setting initial value to the address-mapped register of HD64413A is described below. Follow the steps 1 to 4.

1. Set SRES=0, DRES=1 and DEN=0 to the system control register and stop the display synchronous operation.

   Additionally, do not allow access to UGM by SuperH and DMAC after setting these values and before the display synchronous operations start.
2. Set initial values to registers between register addresses H'002 to H'025 and 02B. In particular, depending on the initial values of each bit of 02B, initial values are required to be set to the registers related to these bits. For details, see the HD64413A Q2SD User's Manual.
3. When displaying 8-bit/pixel displays or performing cursor display by combining GBM2 to 0, set initial values to the color palette register.
4. Set SRES=0 and DRES=0 to the system control register and start display synchronous operations.

   By setting this way, SuperH can make access to UGM. Additionally, to enable the graphics drawn by HD64413A to be checked, normally the DBM of system control register is specified with auto rendering mode or manual display change mode.

An example of initializing the address-mapped register is shown in the ginit function in the sample programs.

2.5 Memory Assignment

2.5.1 Memory Mapping of HD64413A

The address-mapped registers of HD64413A and UGM are mapped in the cache through space of memory space of SuperH. An example of memory map using a 64-Mbit synchronous RAM as UGM is shown below. Also, the A22 to A1 pins of HD64413A require the UGM address of HD64413A to be inputted directly. In this example, A1 to A22 are used as address signals for directly showing the UGM address. For example, when access is made by SuperH to address H'000000 of UGM, clear all the A22 to A1 pins of HD64413A to “0.”

In the figure 2.2, UGM is placed from H'A8000000 so that access is made to the cache through space when SuperH makes access to UGM.

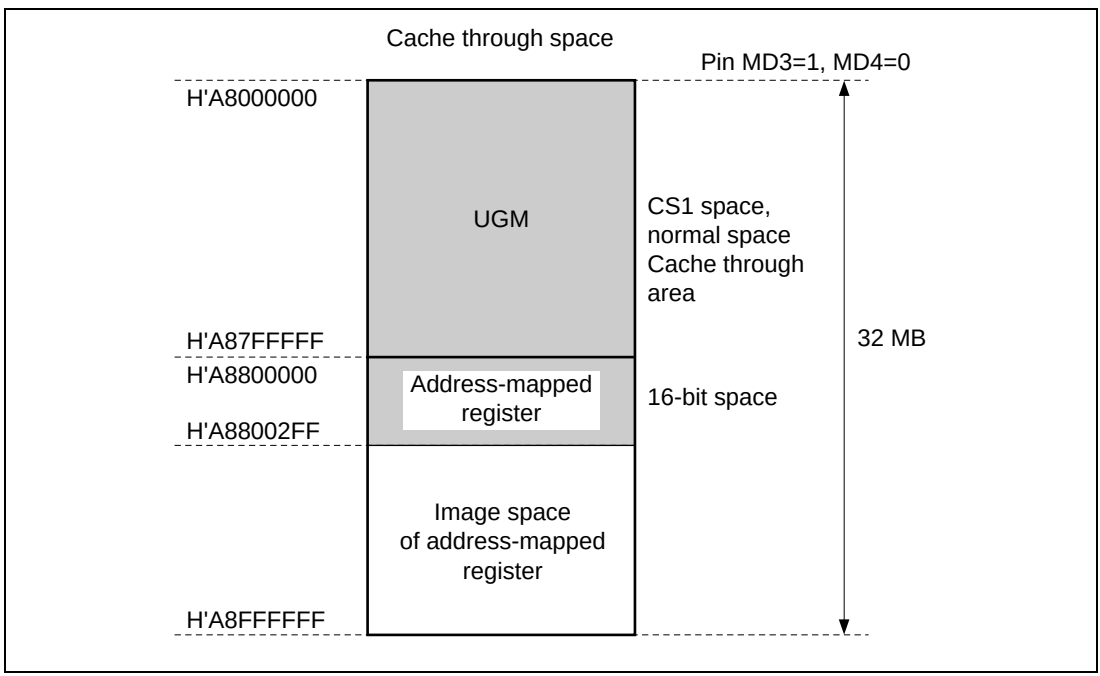


Figure 2.2 Example of Memory Mapping (Using SH7709)

2.5.2 Example of Area Placement in UGM

An example of area placement in UGM is shown below. The display size is 640×240 (640×480 max.) dots.

1. Frame buffer area (FB0, FB1)

   Under double-buffer control, these areas are used as a display area and drawing area (rendering coordinates).

   For the display addresses (DSA0, DSA1) of these areas, set the UGM addresses that correspond to every 256-dot position touching Y-axis.

2. Video store area (V0, V1, V2)

   When using the video capture function, the captured data streams are stored in these areas. The areas are used in order of V0, V1 and V2 each time a synchronous signal is inputted to the $\overline{\text{VVS}}$ pin. Here the read size is 320×240 pixels.

   For the display addresses of these areas (VSAR0 to 2), set the UGM addresses corresponding to every 16-dot position along Y-axis and every 32-dot position along X-axis, considering that UGM is 16 bit/pixel.

   Additionally, when the video capture function and video window are not displayed, these areas are not used and are not necessary.

3. Work area (BWAREA)

   This area is used as a work area. The maximum pixel of X-axis of work coordinates is the pixel quantity specified by the MWX bit of rendering mode register. Accordingly, regardless of the GBM bit of rendering mode register, the memory capacity required as work coordinates is (pixel quantity specified by MWX bit) × (display pixel quantity along Y-axis)/8[Bytes].

   For work area addresses (WASH, WASL), set UGM addresses corresponding to every 16-dot position touching Y-axis.

   Additionally, when the drawing of optional patterns, such as polygons, is not performed, this area is not used and is not necessary.

4. Display list area (DL0, DL1)

   These areas are used for storing display lists. Either one of DLO and DL1 is used as the read area for HD64413A to fetch display lists, and the other as the write area for SuperH to place display lists. DL0 and DL1 are used alternately by software control. Display list start addresses (DLSAH, DLSAL) can be specified with optional word (16-bit) addresses.

5. Cursor 1, 2 area (CU1, CU2)

   These areas are used for storing the shape patterns of cursors. For HD64413A, two cursors can be displayed, so each shape is stored respectively in CU1 and CU2.

   Also, as cursors themselves are displayed as 8-bit/pixel, surely set the display colors of cursors in the color palette.

Additionally, for both CU1 and CU2, the memory capacity used is 2kB.

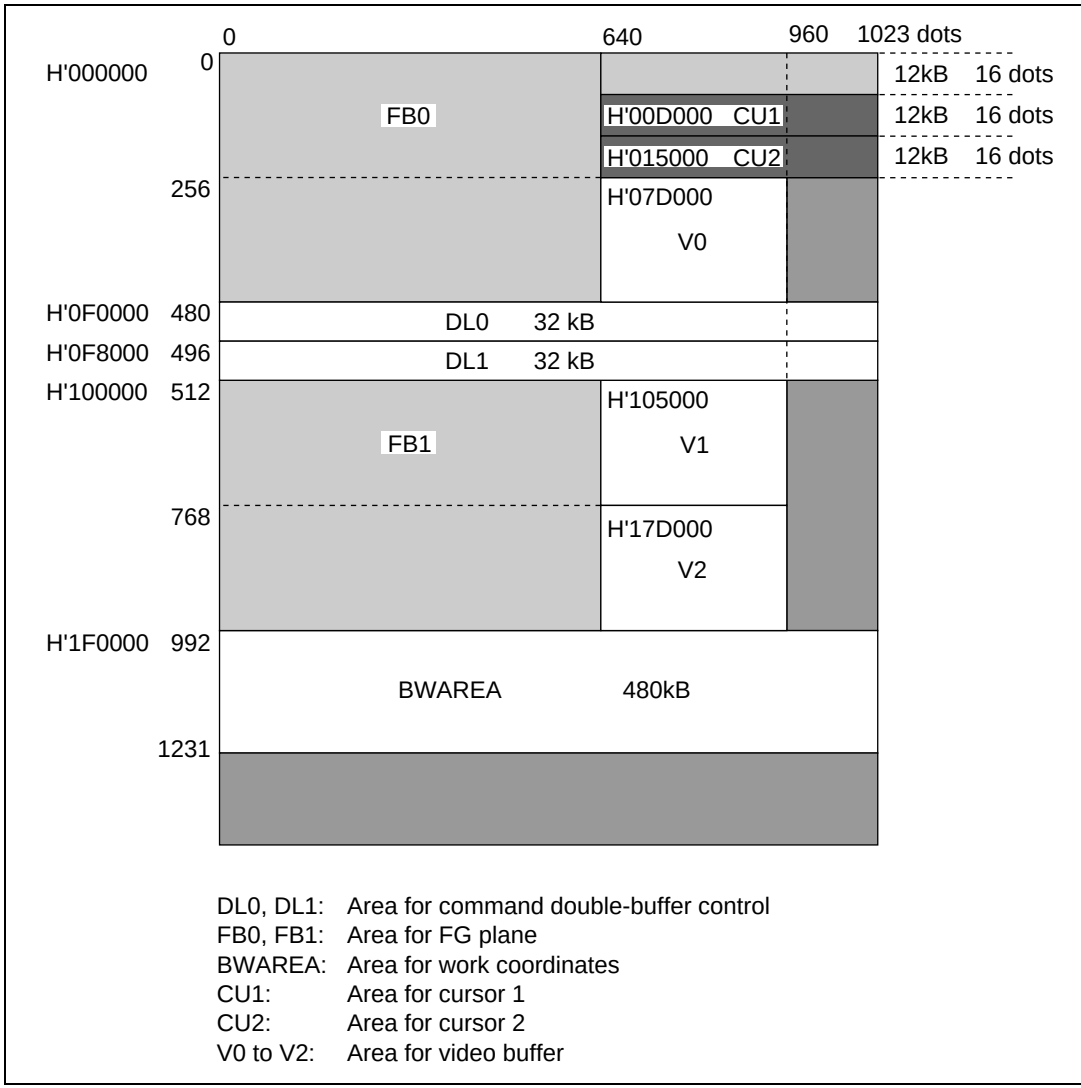


Figure 2.3 UGM Memory Map

2.5.3 Address Seriation in UGM

As shown in the figure below, when UGM is viewed from SuperH, UGM addresses look as a series of tiles arranged by memory unit. Thus, by using more than one memory units that were not used in area assignment of FB0, FB1, etc., the space can be used as a memory space with a series of addresses.

In case of HD64413A, items that can be placed in a memory space with a series of addresses include two-value and multi-value sources and cursor patterns, so normally these are placed in this area.

For example, on the right of FB0, a memory space with a series of addresses having (1024-640) pixels × 16 lines = 6114 pixels from the position X=640, Y=0, or a capacity of 12kB from the relationship 1 pixel = 2B, can be secured. Here CU1 assignment or others is performed.

For details, see 3.2.3 Memory Map in HD64413A Q2SD User's Manual.

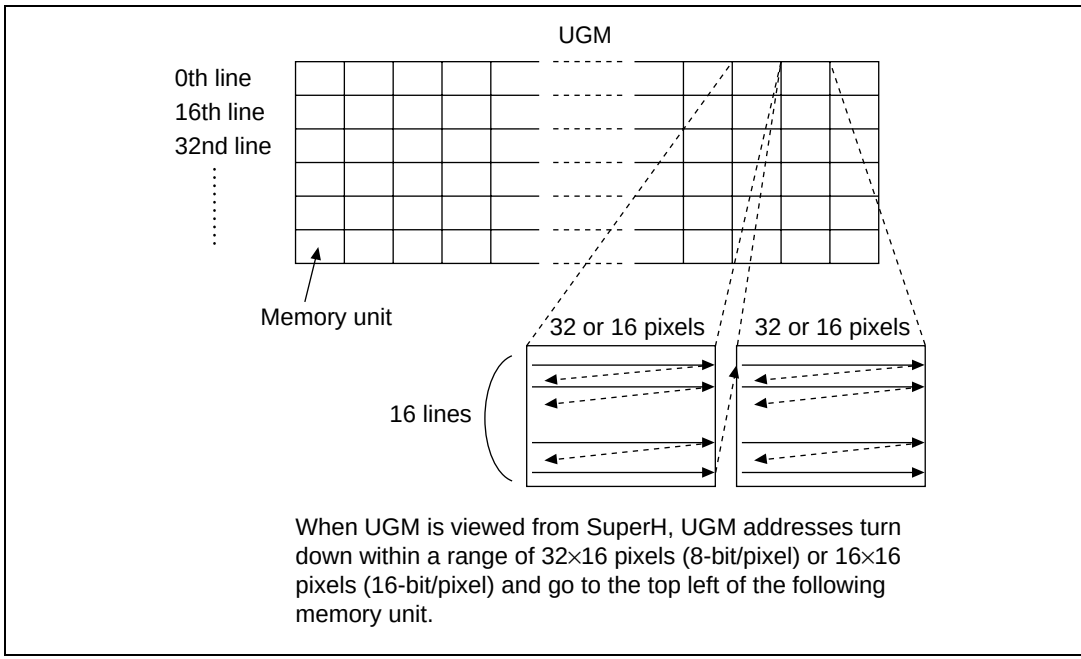


Figure 2.4 UGM Address Transition Overview

2.6 Special Notes on Data Transfer to UGM

To transfer data such as two-value data and display lists by SuperH or the DMA controller to UGM, first do the initial setting to the address-mapped register of HD64413A and start display synchronous operations so as to enable data transfer between SuperH and UGM.

Since an access made by SuperH or the DMA controller to UGM may stop data transfer when display synchronous operations are not performed, do not attempt to make access to UGM when display synchronous operations are not performed.

Additionally, there is only one bus master that can make access to UGM. Accordingly, when the DMA mode in system control register of HD64413A is normal mode, only SuperH can make access to UGM. Likewise, when the DMA mode is DMA transfer mode, only the DMA controller can make access to UGM.

Data transfer is possible even in the midst of drawing processing by HD64413A.

When the DMAC built in SuperH is used, surely check TE (transfer end flag bit) and then the DMF flag of status register of HD64413A before finishing DMA transfer.

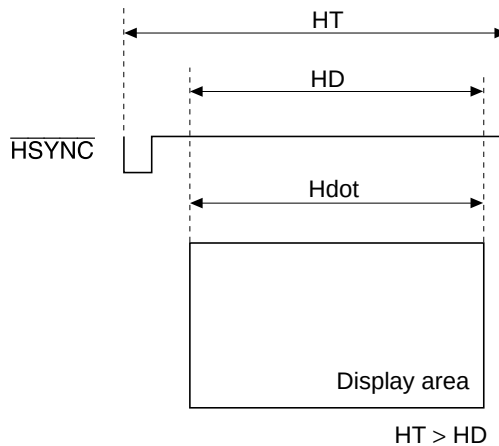
3.1 Determination of Display Size

Display pixels along horizontal direction (Hdot) are required to be values satisfying the following formula. For example, if CLK0=33MHz, N=1, HD=44.7μs, Hdot must be 737 pixels or less.

Also, the frequency of display dot clock (CLK1) must be CLK1=Hdot/HD (Hz).

$$\frac{Hdot}{HD} \leq \frac{CLK0 \times N}{2} = MCLK$$

Where, N is a multiple of Q2SD
CLK0 is a clock (Hz) inputted to the CLK0 pin of Q2SD.



HT: $\overline{\text{HSYNC}}$ cycle (sec)
HD: Display time in $\overline{\text{HSYNC}}$ (sec)
Hdot: Quantity of display pixels along horizontal direction (pixel)

Figure 3.1 Example of Display Timing

3.2 Selection of Display Screen

HD64413A has the following three display screens:

1. Frame screen

   Displayed at the frontmost. It can be displayed with 8 or 16 bit/pixel and is used mainly for realizing dynamic images in drawing processing.

2. Background screen

   Displayed at the rearmost. It can be displayed with 8 or 16 bit/pixel and is used mainly for realizing scroll by pixel.

3. Video window

   Displayed between the frame and background screens. It is used for displaying the stream data taken by the video capture function.

Each display screen can be selected as the frame screen by the FBD bit, the background screen by the BG bit, and the video window by the VWE bit.

3.3 Setting of Synchronous Signal

To enable display control, HD64413A requires synchronous signals to be set in the address-mapped register. An example of register setting of the synchronous signals used for this application note is shown below:

1. Setting example of synchronous signals when the TV synchronous mode is master mode and the scan mode is non-interlace is shown. The display size is 320×240 dots.
   Here, CLK1=(horizontal display pixel)/(xw time) (Hz).

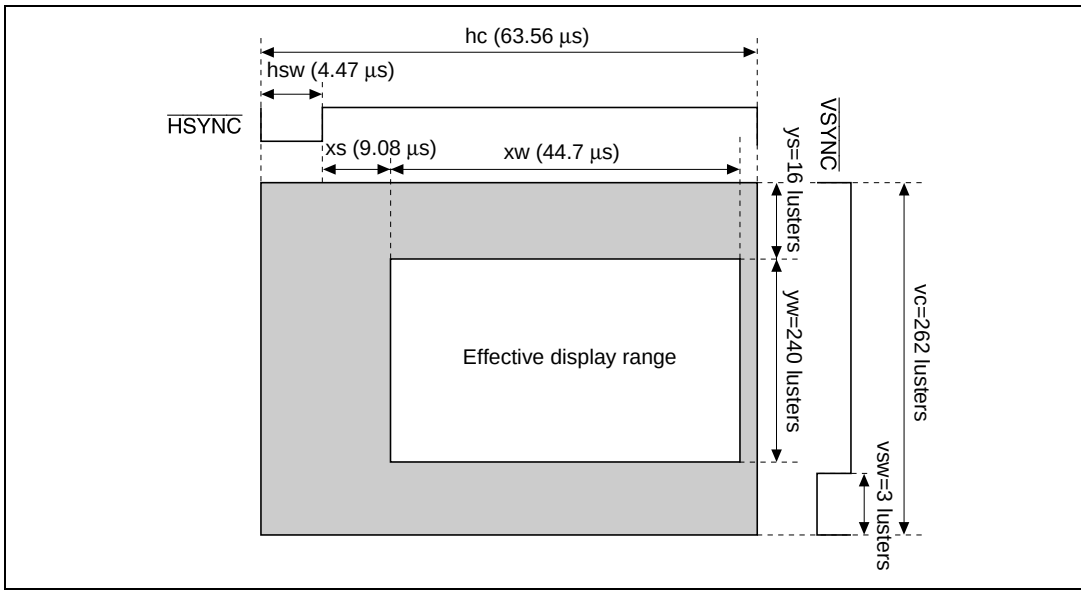


Figure 3.2 Example of Display Timing under Non-interlace Mode

Table 3.2 Setting Example of Variables ((TVM1,0)=(0,0), (SCM1,0)=(0,0))

Variable	Calculation Formula	Value in Display Example
hsw	4.47 µs × CLK1	32
xs	9.08 µs × CLK1	65
xw	44.7 µs × CLK1	320
hc	63.56 µs × CLK1	455

CLK1 = 7.159 MHz

Table 3.3 Register Setting Example ((TVM1,0)=(0,0), (SCM1,0)=(0,0))

Register	Calculation Formula (Master Mode)	Setting Value in Display Example
DSX	xw	320
DSY	yw	240
HDS	hsw + xs – 11	68
HDE	hsw + xs – 11 + xw	406
VDS	ys – 2	14
VDE	ys – 2 + yw	254
HSW	hsw – 1	31
HC	hc – 1	454
VSP	vc – vsw – 1	258
VC	vc – 1	261

2. Setting example of synchronous signals when the TV synchronous mode is master mode and the scan mode is interlace & video mode is shown. The display size is 640×480 dots.
   Here, CLK1=(horizontal display pixel)/(xw time) (Hz).
   As a sample program, see sample7.c.

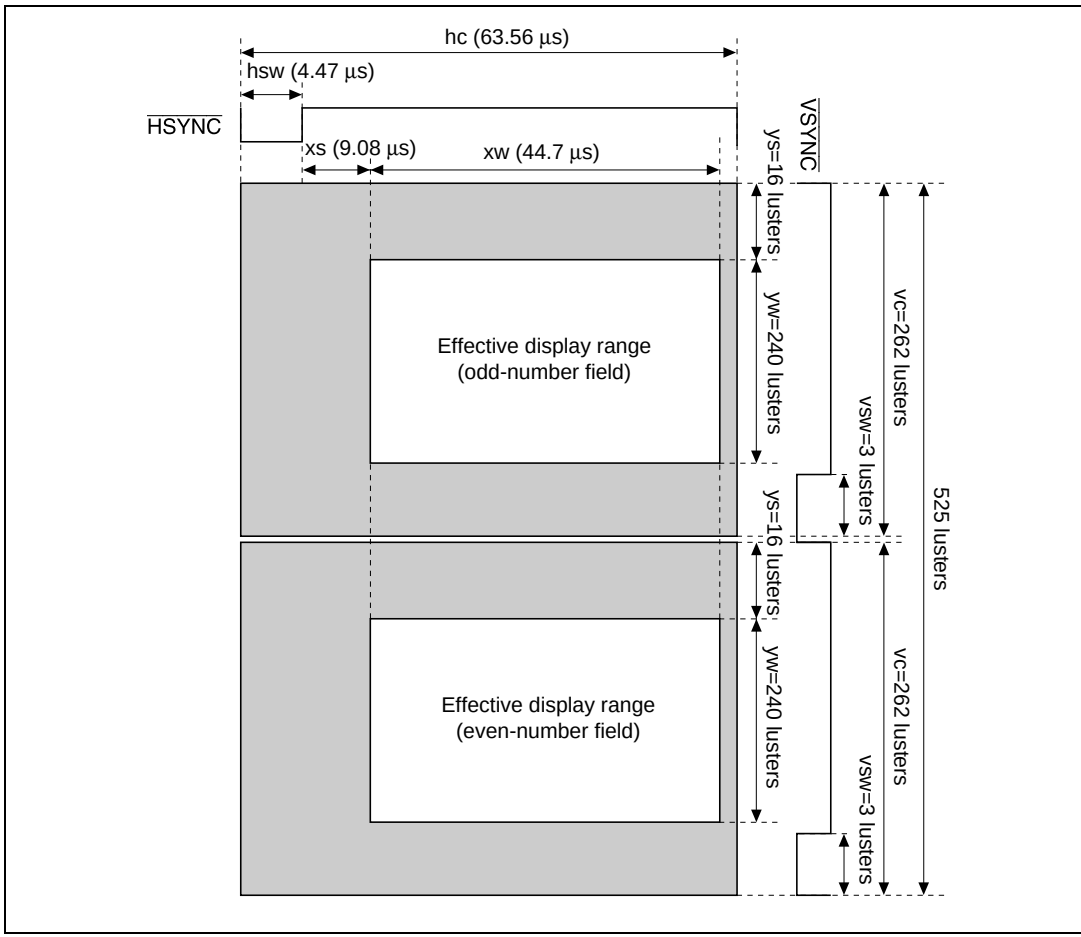


Figure 3.3 Example of Display Timing under Interlace Sink & Video Mode

Table 3.4 Setting Example of Variables ((TVM1,0)=(0,0), (SCM1,0)=(1,1))

Variable	Calculation Formula	Value in Display Example
hsw	4.47 µs × CLK1	64
Xs*	9.08 µs × CLK1	131
xw	44.7 µs × CLK1	640
hc	63.56 µs × CLK1	910

CLK1 = 14.318 MHz

Note: * When using a video encoder, determine xs so that the effective display range does not overlap the color burst.

Table 3.5 Register Setting Example ((TVM1,0)=(0,0), (SCM1,0)=(1,1))

Register	Calculation Formula (Master Mode)	Setting Value in Display Example
DSX	xw	640
DSY	yw	240
HDS	hsw + xs – 11	184
HDE	hsw + xs – 11 + xw	824
VDS	ys – 2	14
VDE	ys – 2 + yw	254
HSW	hsw – 1	63
HC	hc – 1	909
VSP	vc – vsw – 1	258
VC	vc – 1	261

3.4 Setting and Changing Register Values related to Display Control

3.4.1 Setting of Color Palette

The color palette of HD64413A is designed to write or read color palettes by two-word continuous access. So, when setting values to color palettes, surely set registers containing G and B, following a register containing R.

Likewise, when reading values from color palettes, surely read out registers containing G and B, following a register containing R.

3.4.2 Switching Procedure of Synchronous Mode

A change in synchronous mode, from master mode to TV synchronous mode and the like, is performed by way of synchronous switching mode. Switching to synchronous switching mode can be performed by setting TVM=0, TVM0=1.

Also, since HD64413A does not refresh UGM under synchronous switching mode, set DRES=1, DEN=0 and switch the mode to one under which HD64413A refreshes UGM, before entering synchronous switching mode. Procedures are shown below.

Additionally, as HD64413A refreshes UGM while DRES=1, DEN=0 is valid, do not attempt access to UGM by SuperH or DMAC.

Switching Procedure to synchronous switching mode:

1. Set BG=0, VWE=0, CE1=0, CE2=0.
2. Set DRES=1, DEN=0. Now only refreshing to UGM can be performed.
3. Set TVM1=0, TVM0=1. HD64413A switches to synchronous switching mode.

Returning Procedure from synchronous switching mode:

4. Input a clock to the CLK1 pin. For a switch to TV synchronous mode (TVM1=1, TVM0=0), also input signals to $\overline{\text{EXHSYNC}}$, $\overline{\text{EXVSYNV}}$ and $\overline{\text{ODDF}}$ pins.
5. To change the display size, setting values to the address-mapped register of HD64413A.
6. By setting TVM1=0, TVM0=0 or TVM=1, TVM0=0, the input clock from the CLK1 pin is effective. Further, set BG=1, VWE=1, CE1=1, CE2=1, as needed.
7. Set DRES=0, DEN=1. After internal updating, HD64413A starts displaying.

3.5 Use of Video Capture Function

HD64413A stores the video stream data generated by an external video stream decoder when VIE of the video capture mode register (VIMR) is 1, using the video storage area (V0, V1, V2) prepared on UGM in order. The vertical size of video storage area (VSIZEY) depends on the setting values of ODEN1 and ODEN0. ODEN1 and ODEN0 are described and the calculation formula of VSIZEY is shown below:

- ODEN1=0, ODEN0=0

  The timing of specifying the start address of video storage area is in VVS unit, and data is taken in VVS units.

  VSIZEY=(number of effective lines existing in 1VVS signals)×(video capture thinning rate)

- ODEN1=0, ODEN0=1

  The timing of specifying the start address of video storage area is in 2VVS units, and data of both even and odd-number fields are taken.

  VSIZEY=(number of effective lines existing in 2VVS signals)×(video capture thinning rate)

- ODEN1=1

  The timing of specifying the start address of video storage area is in 2VVS units, and data of either even or odd-number field is taken.

  VSIZEY=(number of effective lines existing in 1VVS signals)×(video capture thinning rate)

When storing video stream data to the video storage area, it is also possible to omit the video stream data at HD64413A every 1, 2, 3 and 4 pixel along horizontal direction and every 1, 2, 3, 4 and 6 pixels along vertical direction.

When using the video capture function, set MCLK to 64MHz or higher so that the data bus width of UGM is 32bit. Also, when executing video capturing, capture the video stream data inputted to VID0 to 7 pins at the rising edge of the clock of VQCLK pin, and transfer them to the video storage area. In this application note, Rockwell video stream decoder (Bt829) is used so that VQCLK is generated only with a timing when video stream data exists. See sample program Init_bt.c.

Use of the stored video stream data is described in 1 or 2 below:

1. Use as display data of video window

   When VIE=1 and the VWE bit of display mode register 2 (DSMR2) is 1, the latest video stream data stored in video storage area is displayed realtime.

   See sample program v_wind.c.

   Additionally, when displaying video stream data using the video window, perform a or b of the following:

a. When the RGB bit in VMIR is 1, set 0 to the VWRY in DSMR2.
b. When the RGB bit in VMIR is 0, set 1 to the VWRY in DSMR2.

2. Use as multi-value source

   When VIE=0 is set and video capturing is stopped, the video storage area where the latest video stream data is stored in the VID1,0 bits of VIMR is shown. If video capture function is executed when the RGB bit in VMIR is 1, it is possible to reference the video storage area specified by VID1,0 as a 16-bit/pixel multi-value source.

   Additionally, VID1,0 has no meaning while video capturing is going on (VIE=1). To reference VID1,0, first stop video capturing (VIE=0).

3.6 Cursor Display

The HD64413A can display two cursors, 32 × 32 pixels in size, located in the UGM. One cursor has two shapes, cursor blink shape A and cursor blink shape B, which are displayed alternately at the timing set in BLNKA and BLNKB. A continuous 2 kB address area is therefore required for one cursor.

A continuous 2 kB address area in the UGM can be assigned by using four memory units in the horizontal direction, as shown in figure 3.4. (For details of memory units, see section 2.5.3, Address Seriation in UGM.)

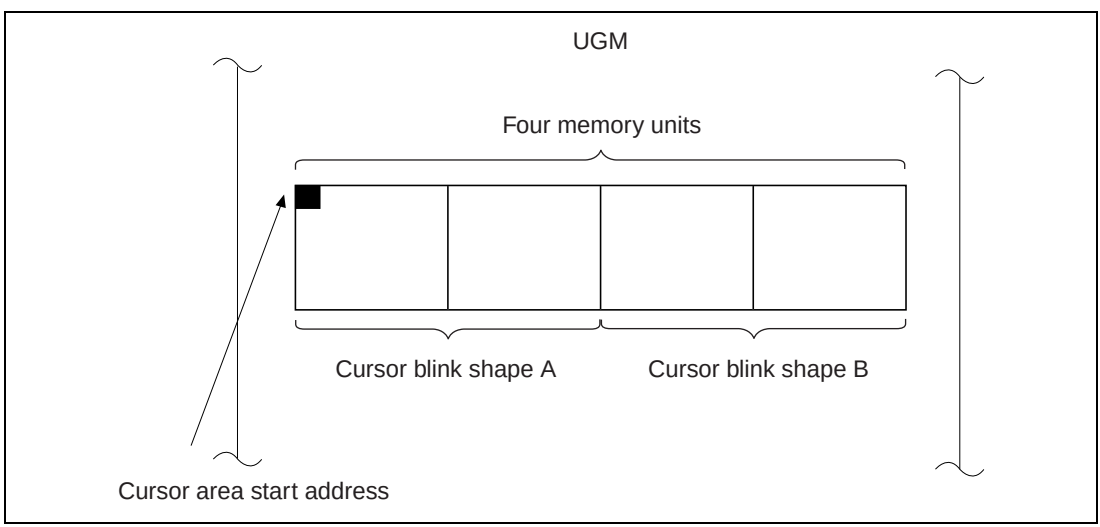


Figure 3.4 Cursor Assignment

When the HD64413A displays a cursor, the cursor shape is read sequentially from the address specified by the cursor area start address register, the color palette is referenced based on the data read, and the cursor is colored and displayed.

Section 4 Drawing

4.1 Starting Drawing

HD64413A performs drawing on rendering coordinates and work coordinates based on command groups called display lists. The drawing procedures are shown below:

1. Using SuperH, place LCOFS and SCLIP commands as a display list to UGM. This display list is intended to set initial values of the local offset and system clip ranges of HD64413A.
2. Following the display list placed in 1, place the VBKEM command to UGM using SuperH to synchronize the frame change timing and draw start timing.
3. Following the display list placed in 2, place the display list using POLYGON4-series commands and the like to UGM using SuperH to let HD64413A perform drawing.
4. Following the display list placed in 3, place the TRAP command to show the end of display listing. At this moment, the display list preparation is finished.
5. After setting the rendering start address, set 1 to RS bit.

   By this register setting, you can let HD64413A perform drawing.

As program examples using POLYGON4-seires commands, sample.c, sample2.c and sample3.c are shown for references. Additionally, item 1 above is to be carried out at the top of a display list generated first, and thereafter LCOFS, SCLIP commands may be used, as needed.

4.2 Frame Change

A frame change can be performed by the following two methods 1 and 2:

1. Frame change following the double buffer control set to DBM
2. Frame change by internal updating.

   In this case frame change is performed by internal updating, while fixing DBM to manual display change mode and controlling display start addresses DSA0 and DSA1 by SuperH.

When using the suspend/resume function of drawing, method 2 by which the draw start address and display start address can be controlled is effective. In this case, first the DBF bit in status register should be checked to judge which of DSA0 or DSA1 is the register that determines the display start address. When DBF=0, DSA0 is the register determining the display start address. Likewise, when DBF=1, DSA1 is the register determining the display start address.

Table 1 shows the relationship of DBF, DSA0 and DSA1.

Table 4.1 Relationship of DBF and Display Screen (FG)

	DSA0	DSA1
DBF = 0	Display screen	Drawing screen
DBF = 1	Drawing screen	Display screen

As an example, control procedures 1 to 4 of DSA0 and DSA1 of a case DBF=0 are shown below:

1. Wait until the internal updating cycle is finished. To confirm the end of internal updating, clear FRM bit and see that the FRM bit becomes 1.
2. Using the WPR command, set in DSA0 the display start address of a position to do displaying in the next internal updating.

   The display start address set to DSA0 is not reflected as an effective value. This setting value becomes effective only passing an internal updating.
3. Using the WPR command, set 1 to RSAE and the draw start address to RSAR.
4. After transferring display lists, set 1 to the RS bit in system control register to start drawing.

By repeating the steps 1 to 4 above, the display start address set to DSA0 becomes effective by internal updating and frame changing is enabled.

The drawing and display timing using the frame change by internal updating are shown below:

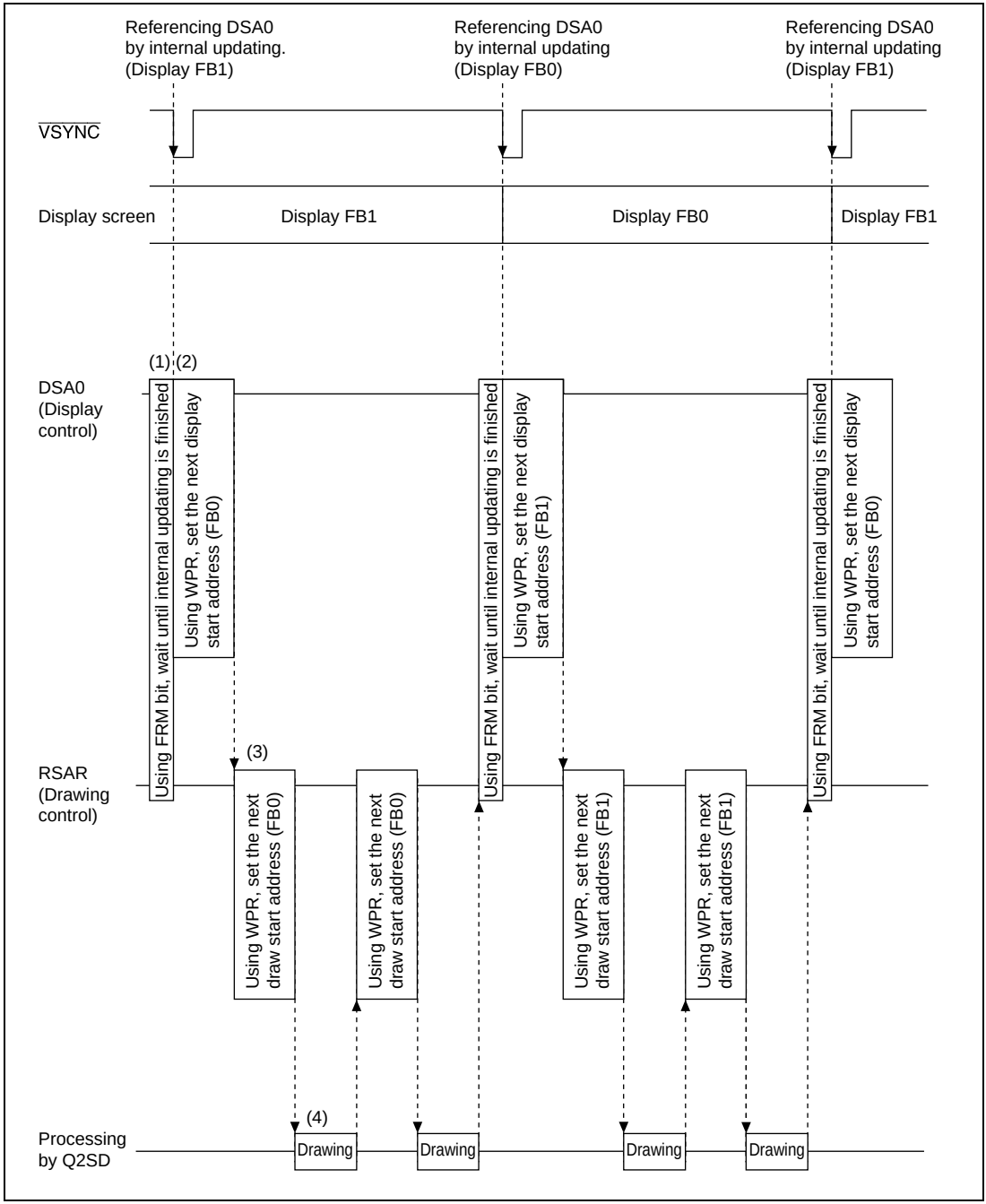


Figure 4.1 Display/Drawing Control Timing Chart (DBF=0)

4.2 Using Example of Draw Commands

4.2.1 Drawing Polygons

Drawing polygons on rendering coordinates using HD64413A can be performed by using work reference, one of rendering attributes, and work coordinates.

Drawing procedures by HD64413A are shown below:

1. To clear the work area, execute the CLRW command.
2. Using the FTRAP command, draw on work coordinates the shape of polygon to draw.
3. Using the POLYGON4C command of which the WORK bit with rendering attribute is set to 1, draw a polygon with the shape drawn on the work coordinates.
4. Using the LINE command, draw an outline of the polygon.

Practically, the display list to perform the above procedures is generated by SuperH, and drawing is performed by HD64413A based on the display list generated.

As a program example, see sample4.c.

4.2.2 Drawing Optional Shapes

To draw an optional shape pattern on rendering coordinates, the following two procedures are available:

- Drawing by partially referencing fixed optional shape pattern and using the shape

  As a program example, see sample9.c. In this program, using a two-value fixed pattern placed on work coordinates (pattern placing 32 16×16-pixel size characters along horizontal direction and 7 characters along vertical direction), drawing is performed by partially referencing the shape.

  Additionally, before placing a two-value pattern on work coordinates, the work coordinates must be zero-cleared. In this sample program, to avoid concurrent drawing on work coordinates by SuperH and HD64413A, the zero-clearing of the work coordinates is performed not by the CLRW command but directly by SuperH.

- Drawing while changing optional shape patterns to use

  As a program example, see sample6.c. In this program, drawing is performed using a 16×16-pixel two-value pattern.

  Since the two-value pattern can be specified for each command, it is possible to change the shape to draw by respecifying the reference pattern when preparing a display list.

4.2.3 Drawing Circles and Ellipses

To draw a circle or an ellipse by HD64413A, the orbit of ellipse is calculated by SuperH, and drawing is performed by the LINE command using the calculated result as parameters. As a sample program, see elsp.c. In this sample program, using Bresenham's circle algorithm, the ellipse orbit is worked out. Drawing a circle is enabled by converting the x, y radiuses of ellipse to same dot quantities.

When drawing is required to be performed in a work area, change the LINE command in use in the _fill_ellipse () function to the LINEW command.

Specification of _fill_ellipse () function:

Prototype: void_fill_ellipse (short xc, short yc, short rx, short ry, short color);

Argument:		
	xc	x coordinates of center point
	yc	y coordinates of center point
	rx	Axial radius in x direction
	ry	Axial radius in y direction
	Color	Color code

Return value: None

4.2.4 Drawing using source data

When using a draw command that references a source by HD64413A, generally it is necessary to judge whether or not source data is stored in UGM on the application soft side. In a certain system, the judgment is difficult or processing requires time. As an example to avoid this, there is one method that relates the draw command and the reference position of source by including source data in the display list.

To embed source data into the display list, place the JUMP command immediately after the draw command and place a display list that skips the source data as shown in figure 4.2.

When a multi-value source is included in the display list, the same method can be applied. In this case, use the POLYGON4A command by setting 1 to the LNi bit with rendering attribute.

Also, a sample program using the POLYGON4B command is shown in sample2.c. In this sample program, drawing is performed by generating a display list containing two-value source data, shown in figure 4.2:

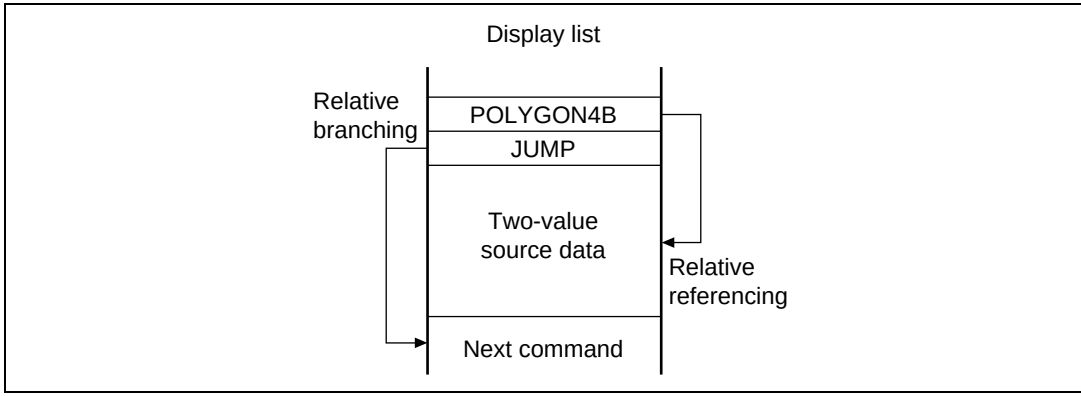


Fig. 4.2 Example of Referencing and Branching

4.2.5 Expressing 3D Space

As a sample program example expressing a 3D space, sample8.c is shown. In this program, two solid graphics formed with 20 rectangles are rotated each. Each of four vertices of each rectangle in the solid graphic has 3D coordinate values X,Y and Z which are determined by an array variable coordindex.

Processing procedures in this sample program are described below:

1. Define the rectangles that are expressed with 3D coordinates values with the set_polygon_sample () function. A set of the rectangles defined with the function forms one solid graphic.

2. Using the rotate_a_rectangle () function, do rotational operation of coordinate values for each rectangle.

3. Using the z_sort function, determine the order of drawing rectangles. Drawing order if from the depth to the front. First the mean value of Z values of each rectangle is determined, and the order is set based on the results. Also, to simplify program descriptions, the bubble sorting is used as a sorting method. Accordingly, if the quantity of elements sorted is n, sorting is performed by approximately n×n times.

   Generally, when a high-speed sorting method like heap sorting is used, sorting can be finished by approximately n×log n times.

4. Using the convert_d3_into_d2 () function, convert 3D coordinate values into 2D coordinate values for each rectangle.

   In this case, conversion is performed by showing the position in which the Z value of each vertex exists in the depth of Z-axis given by variable _z_size, as the ratio of Z value and _z_size, and by multiplying each of 4 vertices by the ratio.

5. Using the order determined in 3 and the 2D coordinate values obtained in 4, drawing is performed on UGM with the POLYGON4C command as a display list.

After finishing the above procedures, the display list is generated on UGM, and a 3D space can be expressed by drawing using HD64413A.

4.3 Special Notes on Using Draw Commands

4.3.1 Notes on the Relationship of Local Offset and Current Pointer

Local offset and the current pointer are given their respective values by the order of command execution. So, place commands while taking into consideration the relationship of local offset and the current pointer. The priority order of command placement is shown below. Draw commands having low priority are to be placed first.

1. lcofs command
   Sets the initial value of local offset.
2. rlcofs command
   Moves local offset by a relative value to the current local offset.
3. move command
   Sets the current pointer by adding the current local offset.
4. remove command
   Moves the current pointer by a relative value to the current pointer currently used.

4.3.2 Notes on Using Relative-series Commands

Commands that control coordinate parameters on relative coordinates are called relative-series commands. When using the relative-series commands, it is necessary to generate a current pointer using the move command and the like beforehand. Also, commands other than relative-series commands use the current pointer as a register for operation and break it. Therefore, when using relative-series commands performing drawing, do not insert other commands between relative-series commands.

4.3.3 Notes on Using Source Data

When HD64413A uses the two-value and multi-value sources placed in UGM, source data is taken to the source buffer existing in HD64413A, and drawing is performed using the accumulated source data. The source buffer has a capacity of 16 words, and HD64413A stores every 32-byte data to the source buffer each time the UGM address passes a 16-word border. Because of this, when using two-value and multi-value sources, it is necessary to let HD64413A perform drawing while taking into consideration to cause source buffer updating. Also, how the source buffer updating is performed is determined by the STYL bit with rendering attribute as described in 1 and 2 below:

1. When STYL with rendering attribute is set to 0

   When STYL=0, consideration must be given so as to cause source buffer updating when the source capacity is within 32 bytes. The following methods can be considered:

   a. Specify different source addresses for each command.

      For example, when referencing two-value sources within 32 bytes by the POLYGON4B command, this method can be performed by specifying different addresses for each of POLYGON4B command parameters SOURCE ADDRESSH and SOURCE ADDRESSL.

   b. Use transparent designation

      By this method, by preparing two-value sources exceeding 32 bytes, only the two-value source of a necessary part is drawn when drawing using the draw command of which transparent designation is enabled.

2. When STYL with rendering attribute is set to 1

   When STYL=1, source referencing is performed repeatedly. Accordingly, when a source referencing is finished at an address within 32 bytes, counting from the source reference start address, consideration must be given so as to cause source buffer updating. The following methods can be considered:

   a. Specify different source addresses for each command.

      For example, when referencing two-value sources within 32 bytes by the POLYGON4B command, this can be performed by specifying different addresses for each of POLYGON4B command parameters SOURCE ADDRESSH and SOURCE ADDRESSL.

4.4 Functions to Support Drawing Processing

4.4.1 Suspension/Resumption of Drawing

Suspension/resumption of drawing is intended to support drawing functions and is supported from HD64412 (Q2i) on. This function is used when doing drawing processing to the frame buffer (FB) during a background screen (BG) drawing or for forcible interruption with drawing processing. Use of suspension/resumption of drawing is described below.

Additionally, the function is available only when 10 is set to the DBM bit of system control register and the double-buffer control is fixed at manual display change.

Suspension of Drawing: "Suspension of drawing" is intended to suspend the drawing that is currently being performed. Suspension of drawing can be performed by setting 1 to the RBRK bit of system control register (SYSR). Setting 1 to the RBRK bit by SuperH, HD64413A sets values set to the register inside LSI (current pointer, local offset, clipping range, return address of GOSUB command) to rendering control register 2 at the top of next command that follows after the execution of current draw command is finished, and suspends the drawing.

To judge suspension of drawing by SuperH, after setting 1 to the RBRK bit, read out the TRA and BRK bits. If the TRA bit is 1, meaning that drawing is finished by executing the TRAP command, and not suspension by RBRK, do not resume drawing thereafter. If the BRK bit is 1, it means suspension of drawing. Thus, suspension of drawing can be judged by confirming BRK=1.

When the BRK bit becomes 1, read values set to the rendering control register 2, draw start address enable (RSAE) of rendering mode register, draw start address register (RSAR) and the command status register (CSTR) by SuperH software processing, and shelter them on the SuperH memory. Sheltered values are used when resuming the suspended drawing.

Thereafter, generate the display list to draw during suspension and execute.

Resumption of Drawing: "Resumption of drawing" is intended to resume a drawing suspended by "suspension of drawing". To resume the drawing, place a display list for resuming drawing to UGM using SuperH, start resumption of drawing to the display list (by setting 1 to the RS bit of system control register) and confirm that the RS bit returns to 0. The composition of display list for resuming drawing is described in 1 to 8 below:

- Order of display list commands used to resume drawing (1 to 8 in this order)

1. WPR command: Sets the draw start address register values sheltered when suspending drawing to the draw start address register (RSAR).
2. WPR command: Sets the draw start address enable values sheltered when suspending drawing to the start address enable (RSAE).

3. WPR command: Sets the return address of GOSUB command sheltered when suspending drawing to the return address register (RTNR).
4. UCLIP command: Returns the UCLIP values sheltered when suspending drawing.
5. SCLIP command: Returns the SCLIP values sheltered when suspending drawing.
6. LCOFS command: Returns the local offset values sheltered when suspending drawing.
7. MOVE command: Returns the current pointer values sheltered when suspending drawing.
8. JUMP command: Returns the command status register values sheltered when suspending drawing.

A program example performing suspension/resumption of drawing is shown in tst_brk.c.

In this program example, first a display list for drawing 256 rectangles is placed to UGM, and drawing processing is suspended by “suspension of drawing” while HD64413A is performing drawing processing based on the display list.

In the interrupt drawing processing after suspending a drawing processing, the whole screen is painted out by the POLYGON4C command.

Further, by “resumption of drawing”, the suspended rectangle drawing is resumed.

When conducting interrupt drawing, like the case of resumption of drawing, set 1 to the RS bit and confirm that the RS bit returns to 0.

The suspension/resumption of drawing in this program consists of five processings shown below. The following 1 to 5 show their overview:

1. Frame change by internal updating (updating of display start position)
2. Suspension of drawing
3. Interrupt drawing processing (paint-out by POLYGON4C command)
4. Resumption of rectangle drawing
5. Register setting for frame change

Section 5 Sample Programs

Sample programs are listed below. sample.c to sample3.c are sample programs using the basic functions of POLYGON4-series commands and perform the extension, contraction and rotation of rectangle based on its four vertex coordinates calculated by SuperH. Additionally, these programs are prepared on Solution Engine (loading SH7709) and the HD64413A (Q2SD) daughter Board by Hitachi ULSI Systems. Accordingly, the main memory of SuperH and the mapping address of Q2SD are on the following preconditions:

SuperH main memory:	H'C000000–H'C3FFFFF
Q2SD UGM:	H'B4000000–
Q2SD address-mapped register:	H'B4800000–

Also, in the sample programs, each display size is set to 640×240 pixels by the Q2SD address-mapped register. (However, only display size of the sample7.c is set to 640×480 dots.)

(Header, library)

Q2SD_REG.h: Definition of Q2SD register address

Q2SD_mac.h: Sample of macro functions of the Q2SD commands used in the sample programs

Q2SDL_inc: Source library of command double-buffer control

MS7709.inc: Setting of bus status controller for SH7709 to make access to Q2SD

(Example of basic programs)

sample.c: Sample program using POLYGON4C; it performs, the extension, contraction, rotation and movement of single-color rectangles.

sample2.c: Sample program using POLYGON4B; it performs, the extension, contraction, rotation and movement of 24×24-dot size characters.

sample3.c: Sample program using POLYGON4A; it performs, the extension, contraction and rotation of color patterns (multi-value).

(Example of application programs)

sample4.c: Sample program for drawing polygons; the sample program draws a hexagon. NET-specified, the display color may differ by the output timing of the NTSC encoder circuit.

sample5.c: Sample program for drawing while changing the draw start address by the WPR command

sample6.c: Sample program that moves 100 balls with different speeds of movement using POLYGON4A. As a precondition, 10 16×16-pixel ball-shaped multi-value patterns be placed in MSAREA.

sample7.c: Sample program for operation under interlace sink & video mode (display size: 640×480 pixels)

sample8.c: Sample program for rotating two solid graphics. Each solid graphic is formed with 20 faces.

sample9.c: Sample program for drawing using a two-value pattern placed on two-value work coordinates

tst_brk.c: Sample program for temporarily suspending a drawing processing, clearing the screen, and resuming the suspended drawing
Additionally, because of the control system, the following functions are not used:

```
draw_start ()
draw_end (DBmode, &first, quick);
```

elps.c: Sample program for drawing ellipses

v_wind.c: Sample program for displaying the video data taken by the video capture function
In video capturing, capturing is performed in field unit while halving only the horizontal direction, and 320×240-pixel video data is stored to UGM.
Note that the object program which can be generated from INIT_BT.C be executed before 640×240-pixel video data is generated per field.

Init_bit.c: Sample program for initializing the Rockewell BT829 loaded in the Q2SD daughter board.
640×480-pixel video data is outputted per frame.

Drawing examples for each sample program are shown in figures 5.1 to 5.11.

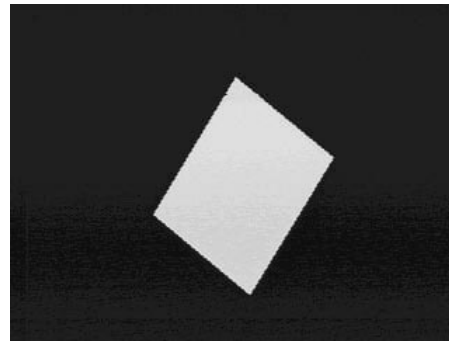


Figure 5.1 Drawing example by sample.c (single-color rectangle)

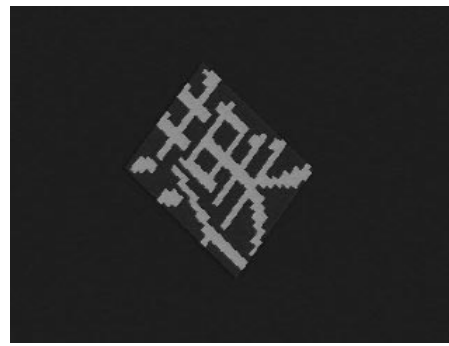


Figure 5.2 Drawing example by sample2.c (two-value character)

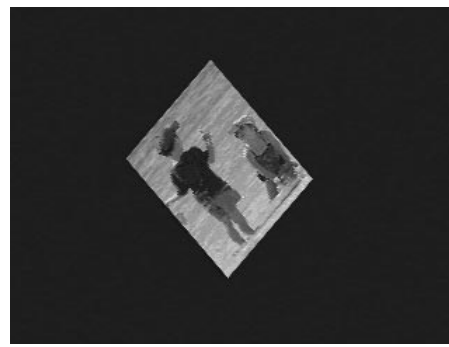


Figure 5.3 Drawing example by sample3.c (natural picture)

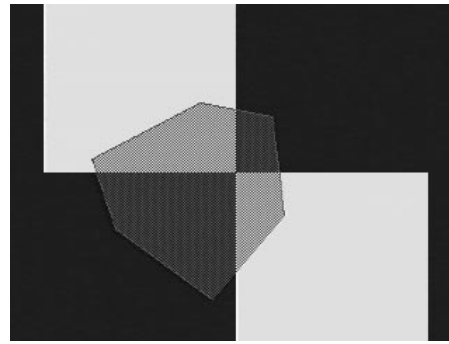


Figure 5.4 Drawing example by sample4.c (polygon)

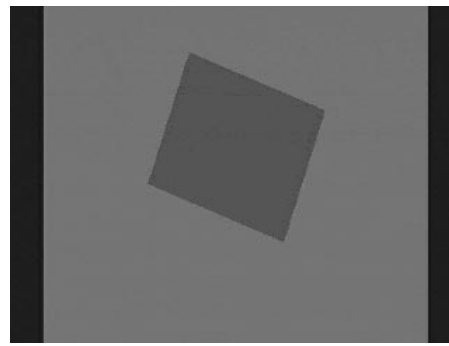


Figure 5.5 Drawing example by sample5.c (changing the draw start position)

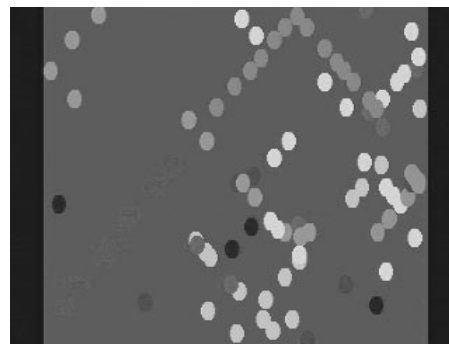


Figure 5.6 Drawing example by sample6.c (optional pattern)

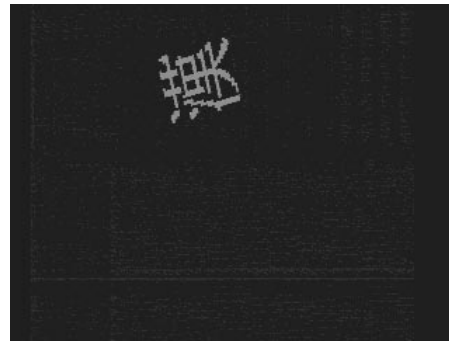


Figure 5.7 Drawing example by sample7.c (display size 640×480)

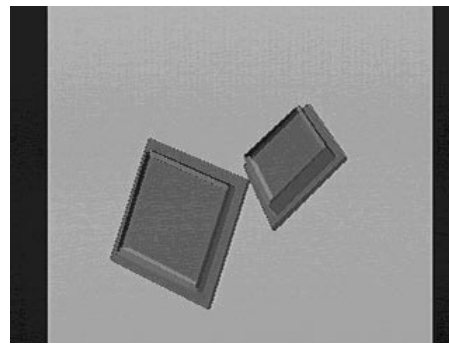


Figure 5.8 Drawing example by sample8.c (expressing a 3D space)

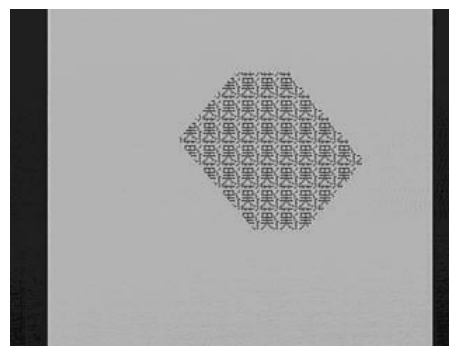


Figure 5.9 Drawing example by sample9.c (drawing with optional pattern)

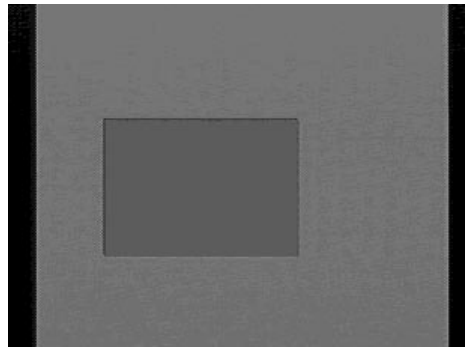


Figure 5.10 Drawing example by tst_brk.c (suspension/resumption of drawing)

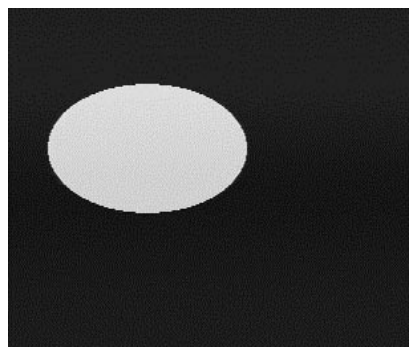


Figure 5.11 Drawing example by elps.c (ellipse)

5.1 Descriptions about Sample Programs

5.1.1 Configuration of Basic Functions

The sample programs shown in this application note use the following functions 1 to 3 as basic functions:

1. Initialize function

   This function is intended to set initial values to the address-mapped register.

   a. ginit () function: Performs initialization using the values determined in “3.3 Setting of Synchronous Signal.”

2. Command double-buffer control function

   These function is used to control the placement of display lists and the start and end of drawing. By alternately using the DL0 and DL1 areas by software control, these functions perform drawing processing and the placement of display lists in parallel.

   a. init_stat () function: Initializes command double-buffer control
   b. draw_stat () function: Starts command double-buffer control
   c. draw_end () function: Stops command double-buffer control and then performs drawing
   d. change_com_buffer () function: Used by draw_end () function to judge end of drawing

   Also, command double-buffer control functions use the following variables:

   a. DrawBuffer: Used to select area to DL0 and DL1 areas
   b. DISPLIST_ptr: Pointer variable to store placement positions of display lists
   c. flag: Used by change_com_buffer () function

3. Input/output function (macro function)

   These function is used to make access to the address-mapped register. Being macro functions, these functions are made suitable for access to the address-mapped register by inhibiting the optimization of variables, using volatile declaration.

   (Normally when making access to an external device, use a volatile-declared variable.)

   a. outport () function: Sets values to address-mapped register
   b. inport () function: Obtains values from address-mapped register

5.1.2 Sample Program Description Rules

Description rules of sample programs are described below. Descriptions are given following the steps 1 to 9.

Also, by setting a span between functions of draw_start and draw_end as a display list generation unit, display lists are placed and rendering (drawing) is performed while using the DL0 and DL1 areas alternately per unit.

Additionally, start addresses of DL0 and DL1 areas are defined respectively as DISPLIST0 and DISPLIST1 in the Q2SD_REG.h file, as shown in the program below:

```
#define   DISPLIST0          0x190000L     /* DL0 area start address */
#define   DISPLIST1          0x198000L     /* DL1 area start address */
```

Macro functions and prototypes of Q2SD commands are described in the Q2SD_mac.h file.

Description rules of sample program are shown below:

```
void main(void)
{
          short array[8];
          short DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
          char  first,quick;

          DBmode = 1;
          first = ON;
          quick = OFF;

/*---------------------------------------------------------------------------
1. Set initial values to SH7709 bus state controller using the init_BSC () function.
---------------------------------------------------------------------------*/
          init_BSC (DBmode);       /* Initialize SH7709 bus state controller */
```

```
/*-------------------------------------------------------------------------------
```

2. Set initial values to the address-mapped register of Q2SD using the ginit () function.

```
-------------------------------------------------------------------------------*/
          ginit(DBmode);           /* Initialize Q2 */

/*-------------------------------------------------------------------------------
```

3. Initialize command double-buffer control using the init_start () function.

```
-------------------------------------------------------------------------------*/
          init_start();           /* Initialize Transfer Procedure */

/*-------------------------------------------------------------------------------
```

4. Starts command double-buffer control using the draw_start () function.
   Further, place the vbkem command as a display list to UGM.

```
-------------------------------------------------------------------------------*/
          draw_start();           /* Start Transfer Display List to UGM */

/*-------------------------------------------------------------------------------
```

5. Describe command macro functions of Q2SD.
   Additionally, surely describe both sclip () macro and lcofs () macro functions first.

```
-------------------------------------------------------------------------------*/
          sclip(0x0000, 319,239);
          lcofs(0x0000,0,0);

/*-------------------------------------------------------------------------------
```

6. Using the draw_end () function, Q2SD performs rendering (drawing) based on the display lists generated by command macro functions of Q2SD.

```
-------------------------------------------------------------------------------*/
          draw_end(DBmode,&first,quick);

/*-------------------------------------------------------------------------------
```

7. Resume command double-buffer control using the draw_start () function. Thereafter describe Q2SD command macro functions.
   Additionally, in the 2nd and the following draw_start () functions, the sclip () macro and lcofs () macro functions may be described, as needed.

```
-------------------------------------------------------------------------------*/
          draw_start();            /* Start Transfer Display List to UGM */
          ...........
          draw_end(DBmode,&first,quick);
```

```
/*--------------------------------------------------------------------------------
```

8. The sample program shows an end of program by describing return.

```
--------------------------------------------------------------------------------*/
          return;                   /* EXIT */
}

/*--------------------------------------------------------------------------------
```

9. Finally, files MS7709.inc and Q2SDL.inc are included.

```
--------------------------------------------------------------------------------*/
#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.1.3 Special Notes on Program Preparation

Points to be noted in preparing programs are described below:

1. Synchronization with VSYNC

   In the drawing system using SuperH and HD64413A, the preparation of programs synchronized with VSYNC is the basic style of program preparation. Thus, do either a or b of the following. The sample programs shown in this application note adopt procedure a.

   a. Using VBKEM as a display list, synchronize the program with VSYNC.
   b. Using VBK or FRM, synchronize the program with VSYNC by IRL interruption.

2. Access using non-cache area

   Of SuperH, some are equipped with cache. So, use a non-cache area to make access to HD64413A.

   When using the cash area, to secure coherence with cache, place a display list to UGM and then nullify cash rewriting before starting drawing.

3. Using optimization-nullified variables

   When making access to HD64413A, an LSI which is an external device for SuperH, make access using optimization-nullified variables. Also, use a volatile declaration to declare nulllifying optimization.

   (Example of description)

```
#define VU_SHORT (volatile unsigned short * const)
#define outport(add,data) ( *VU_SHORT(add) ) = ( (unsigned short)(data) )
#define inport(add)       ( *VU_SHORT(add) )
#define _Q2SYSR           0x000L + 0xA8800000L
```

   When the above definition is given, compilation substitutes it to a description containing a volatile declaration like a, b below.

   a. Description for value setting

      Before substitution

```
/* _Q2SYSR register is set to 0x2080. */
  outport(_Q2SYSR, 0x2080);
```

      After substitution

```
/* _Q2SYSR register is set to 0x2080. */
  ( *(volatile unsigned short * const)(0x000L + 0xA8800000L) ) = ( (unsigned
short)(0x2080) );
```

b. Description for obtaining values

Before substitution

```
/* Read value from _Q2SYSR register. */
  abc = inport(_Q2SYSR);
```

After substitution

```
/* Read value from _Q2SYSR register. */
  abc = ( *(volatile unsigned short * const)(0x000L + 0xA8800000L) );
```

5.2 Sample Program Source List

5.2.1 Build File BuildB.bat (BuildL.bat)

```
SHC /CPU=SH3 /OPT=1 /SPEED /ENDIAN=BIG /LOOP %1.c
if errorlevel 1 goto end
LNK %1 /OUTPUT=%1 /LIB=c:\SHC\V50\LIB\shc3npb.LIB /FORM=A /START=P(0C000000)
if errorlevel 1 goto end
CNVS %1 %1.txt
DEL %1.obj
DEL %1.abs
:end
```

5.2.2 Source File of MS7709.inc

```
/*
              SH7709 bus state controller for MS7709SE01/MS4413DB01

                        Copyright(c) Hitachi Ltd. 1999
*/

/*=======================     DEFINE TYPES      ==========================*/
#define  BCR2 *(volatile unsigned short *)0xFFFFFF62  /* bus control register 1 */
#define  WCR1 *(volatile unsigned short *)0xFFFFFF64  /* wait state control
register 1 */
#define  WCR2  *(volatile unsigned short *)0xFFFFFF66 /* wait state control
register 2 */

void init_BSC(void)                   /* Initialize SH7709 bus state controller */
{
      BCR2 = (BCR2 & 0x3fc0) | 0x0020; /* set 16bit bus width for CS2(area-2) */
      WCR1 = (WCR1 & 0x3fc3) | 0x0010; /* set 1 idle cycle to CS2(area-2) */
      WCR2 = (WCR2 & 0xffe7) | 0x0010; /* set 2 wait state cycle to CS2(area-2) */
}
```

5.2.3 Source File of Q2SDL.inc

```
/*
                    Command double buffer control for Q2SD

                     ((for 16bit/Pixel 65536 Color mode))
                        Copyright(c) Hitachi Ltd. 1999
*/

void  init_start(void);              /* Initilaze Display List Double Buffering */
short draw_start(void);              /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);       /* Change Display List */

short  DrawBuffer;          /* Display List Double Buffering Destination Number */
short  flag;

/* ------------------------------------------------------------------------ */

void init_start(void)           /* Initialze transfer procedure */
{
     DrawBuffer = 0;
     flag = 0;
}



short draw_start(void)         /* Initialize Display List Address Pointer */
{
  long  CMD_StartAddress;
     if(DrawBuffer == 0)
                             /* Display List Start Address */
          CMD_StartAddress = (long)(DISPLIST1+UGMBASE);     /* Execute Buffer0 ->
Transfer Display List to Buffer1 */
     else
          CMD_StartAddress = (long)(DISPLIST0+UGMBASE);     /* Execute Buffer1 ->
Transfer Display List to Buffer0 */

     DISPLIST_ptr = (unsigned short *)CMD_StartAddress;

     vbkem(0x0000,0x0000,0x0000);   /* Add 'VBKEM commnad'. */
```

```
     return( 0 );
}


short draw_end(short DBmode,char *first,char quick)     /* Execute Display List */
{
     trap(0);                      /* Add 'trap' command */

     change_com_buffer();

     switch(DBmode) {

          case 2:                /* Manual Change */
               /* Renddering start */
               outport(_Q2SYSR,0x2180);

               if ( quick == OFF ) {
                    outport(_Q2SYSR,0x2280);  /* Frame change     */
                    /* Wait Display Change Bit '1' -> '0' */
                    while( ( inport(_Q2SYSR) & 0x0200 ) != 0 );
               }

               break;

          case 1:                  /* Auto Renderring */
               /* Renddering start */
               outport(_Q2SYSR,0x2140);
               break;

          default:                 /* Auto Change */
               outport(_Q2SRCR,0x0800);                    /* Clear VBK bit */
               while( ( inport(_Q2SR) & 0x0800 ) == 0);    /* Wait VBK */

               /* Renddering start */
               outport(_Q2SYSR,0x2100);
     }
     return( 0 );
}
```

```
void change_com_buffer(void)
{
  unsigned long CMD_StartAddress;
  unsigned short st;

     if (flag) {
               while(1){
                    st=inport(_Q2SR);
                    if ((st & 0x0400)!=0) break;      /* Check TRA bit */
                    if ((st & 0x0200)!=0) {
                         outport(_Q2SRCR,0x0200);     /* Clear CSF bit */
                         break; /* Check CSF bit */
                    }
               }
          }
          else {
               flag = 1;
          }

          outport(_Q2SRCR,0x0400);                   /* Clear TRA bit  */

          if(DrawBuffer == 0) {
               DrawBuffer = 1;
               CMD_StartAddress = DISPLIST1;
          }
          else {
               DrawBuffer = 0;
               CMD_StartAddress = DISPLIST0;
          }
          outport(_Q2DLSAH,(CMD_StartAddress >> 16L) & 0xffff); /* Change DLSAR */
          outport(_Q2DLSAL, CMD_StartAddress                );
}
```

5.2.4 Source File of Q2SD_mac.h

```
/*

                        Q2 command macro sample

                     Copyright(c) Hitachi Ltd. 1999
*/



/* ------------- Q2 Display List Functions ---------------- */
void polygon4a(short draw_mode, short txs,short tys,short tdx,short tdy, short
poly[]);
void polygon4b(short draw_mode, short src_h,short src_l,short tdx,short tdy, short
poly[], short color0,short color1);
void polygon4c(short draw_mode, short poly[], short color1);
void line(short draw_mode,short color,short n, short poly[]);
void rline(short draw_mode,short color,short n, short poly[]);
void pline(short draw_mode,short color0,short color1,short src_h,short src_l,short
tdx,short n, short poly[]);
void rpline(short draw_mode,short color0,short color1,short src_h,short
src_l,short tdx,short n, short poly[]);
void linew(short draw_mode,short n, short poly[]);
void rlinew(short draw_mode,short n, short poly[]);
void clrw(short draw_mode,short xmin,short ymin,short xmax,short ymax);
void ftrap(short draw_mode,short n, short dxl, short poly[]);
void rftrap(short draw_mode,short n, short dxl, short poly[]);
void move(short draw_mode,short xc,short yc);
void rmove(short draw_mode,short xc,short yc);
void lcofs(short draw_mode,short xo,short yo);
void rlcofs(short draw_mode,short xo,short yo);
void sclip(short draw_mode,short xmax,short ymax);
void uclip(short draw_mode,short xmin,short ymin,short xmax,short ymax);
void jump(short draw_mode,short adr_h,short adr_l);
void gosub(short draw_mode,short adr_h,short adr_l);
void ret(short draw_mode);
void trap(short draw_mode);
void nop3(short draw_mode,short dummy1,short dummy2);
void wpr(short draw_mode,short rn,short data);
void vbkem(short draw_mode,short dummy1,short dummy2);
```

```
/* ----- Q2 function macro ------------------------------- */
#define polygon4a( draw_mode, txs,tys, tdx,tdy, array )\
{\
  short *_q2_array = array;\
         *DISPLIST_ptr++ = 0x0000 | draw_mode;               /* POLYGON4A */\
         *DISPLIST_ptr++ = txs;           *DISPLIST_ptr++ = tys;\
         *DISPLIST_ptr++ = tdx;           *DISPLIST_ptr++ = tdy;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
}

#define polygon4b( draw_mode, src_h,src_l, tdx,tdy, array, color0,color1 )\
{\
  short *_q2_array = array;\
         *DISPLIST_ptr++ = 0x0800 | draw_mode;               /* POLYGON4B */\
         *DISPLIST_ptr++ = src_h;         *DISPLIST_ptr++ = src_l;\
         *DISPLIST_ptr++ = tdx;           *DISPLIST_ptr++ = tdy;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = color0;\
         *DISPLIST_ptr++ = color1;\
}

#define polygon4c( draw_mode, array, color1 )\
{\
  short *_q2_array = array;\
         *DISPLIST_ptr++ = 0x1000 | draw_mode;                /* POLYGON4C */\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = *_q2_array++;  *DISPLIST_ptr++ = *_q2_array++;\
         *DISPLIST_ptr++ = color1;\
}

#define line( draw_mode, color, n, array )\
```

```
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x6000 | draw_mode;                /* Line */\
         *DISPLIST_ptr++ = color;\
         *DISPLIST_ptr++ = n;\
         for(i = 0; i < (n)*2; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define pline( draw_mode, color0,color1, src_h,src_l, tdx, n, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x7101 | draw_mode;               /* PLINE */\
         *DISPLIST_ptr++ = color0;\
         *DISPLIST_ptr++ = color1;\
         *DISPLIST_ptr++ = src_h;\
         *DISPLIST_ptr++ = src_l;\
         *DISPLIST_ptr++ = tdx;\
         *DISPLIST_ptr++ = n;\
         for(i = 0; i < (n)*2; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define clrw( draw_mode, xmin, ymin, xmax, ymax )\
{\
         *DISPLIST_ptr++ = 0xa000 | draw_mode;              /* CLeaR Work */\
         *DISPLIST_ptr++ = xmin;\
         *DISPLIST_ptr++ = ymin;\
         *DISPLIST_ptr++ = xmax;\
         *DISPLIST_ptr++ = ymax;\
}

#define ftrap( draw_mode, n, dxl, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x4000 | draw_mode;             /* Filled TRAPezoid */\
         *DISPLIST_ptr++ = n;\
         *DISPLIST_ptr++ = dxl;\
         for(i = 0; i < (n)*2; i++) *DISPLIST_ptr++ = *_q2_array++;\
```

```
}

#define linew( draw_mode, n, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x5000 | draw_mode;             /* Line Work */\
         *DISPLIST_ptr++ = n;\
         for(i = 0; i < (n)*2; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define move( draw_mode, xc, yc )\
{\
         *DISPLIST_ptr++ = 0x8000 | draw_mode;             /* MOVE */\
         *DISPLIST_ptr++ = xc;\
         *DISPLIST_ptr++ = yc;\
}

#define lcofs( draw_mode, xo,yo )\
{\
         *DISPLIST_ptr++ = 0x9000 | draw_mode;             /* LoCal OFfSet */\
         *DISPLIST_ptr++ = xo;\
         *DISPLIST_ptr++ = yo;\
}

#define sclip( draw_mode, xmax,ymax )\
{\
         *DISPLIST_ptr++ = 0xb800 | draw_mode;             /* System CLIP */\
         *DISPLIST_ptr++ = xmax;\
         *DISPLIST_ptr++ = ymax;\
}

#define uclip( draw_mode, xmin, ymin, xmax, ymax )\
{\
         *DISPLIST_ptr++ = 0xa800 | draw_mode;             /* User CLIP */\
         *DISPLIST_ptr++ = xmin;\
         *DISPLIST_ptr++ = ymin;\
         *DISPLIST_ptr++ = xmax;\
         *DISPLIST_ptr++ = ymax;\
}
```

```
#define jump( draw_mode, adr_h, adr_l)\
{\
         *DISPLIST_ptr++ = 0xc000 | draw_mode;             /* Jump */\
         *DISPLIST_ptr++ = adr_h;\
         *DISPLIST_ptr++ = adr_l;\
}

#define gosub( draw_mode, adr_h, adr_l )\
{\
         *DISPLIST_ptr++ = 0xc800 | draw_mode;        /* GO SUBroutine */\
         *DISPLIST_ptr++ = adr_h;\
         *DISPLIST_ptr++ = adr_l;\
}

#define ret( draw_mode )\
{\
         *DISPLIST_ptr++ = 0xd800 | draw_mode;       /* RETurn from subroutine */\
}

#define nop3( draw_mode, dummy1, dummy2 )\
{\
         *DISPLIST_ptr++ = 0xf000 | draw_mode;       /* NOP3 */\
         *DISPLIST_ptr++ = dummy1;\
         *DISPLIST_ptr++ = dummy2;\
}

#define wpr( draw_mode, rn, data )\
{\
         *DISPLIST_ptr++ = 0xb000 | draw_mode;       /* Write PaRameter */\
         *DISPLIST_ptr++ = rn;\
         *DISPLIST_ptr++ = data;\
}

#define vbkem( draw_mode, dummy1, dummy2 )\
{\
         *DISPLIST_ptr++ = 0xd000 | draw_mode; /* Vertical BlanKing Edge Maker */\
         *DISPLIST_ptr++ = dummy1;\
         *DISPLIST_ptr++ = dummy2;\
}
```

```
#define trap( draw_mode )\
{\
         *DISPLIST_ptr++ = 0xf800 | draw_mode;      /* TRAP */\
}



#define rftrap( draw_mode, n, dxl, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x4800 | draw_mode;    /* Relative Filled TRAPezoid */\
         *DISPLIST_ptr++ = n;\
         *DISPLIST_ptr++ = dxl;\
         for(i = 0; i < n; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define rlinew( draw_mode, n, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x5800 | draw_mode;     /* Rerlative Line Work */\
         *DISPLIST_ptr++ = n;\
         for(i = 0; i < n; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define rline( draw_mode, color, n, array )\
{\
  short *_q2_array = array;\
  short i;\
         *DISPLIST_ptr++ = 0x6800 | draw_mode;      /* Relative Line */\
         *DISPLIST_ptr++ = color;\
         *DISPLIST_ptr++ = n;\
         for(i = 0; i < n; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define rpline( draw_mode, color0, color1, src_h,src_l, tdx, n, array )\
{\
  short *_q2_array = array;\
  short i;\
```

```
        *DISPLIST_ptr++ = 0x7901 | draw_mode;      /* Relative PLINE */\
        *DISPLIST_ptr++ = color0;\
        *DISPLIST_ptr++ = color1;\
        *DISPLIST_ptr++ = src_h;\
        *DISPLIST_ptr++ = src_l;\
        *DISPLIST_ptr++ = tdx;\
        *DISPLIST_ptr++ = n;\
        for(i = 0; i < n; i++) *DISPLIST_ptr++ = *_q2_array++;\
}

#define rmove( draw_mode, xc, yc )\
{\
        *DISPLIST_ptr++ = 0x8800 | draw_mode;       /* Relative MOVE */\
        *DISPLIST_ptr++ = ((xc) << 8) | ((yc) & 0xff);\
}

#define rlcofs( draw_mode, xo, yo )\
{\
        *DISPLIST_ptr++ = 0x9800 | draw_mode;       /* Relative LoCal OFfSet */\
        *DISPLIST_ptr++ = ((xo) << 8) | ((yo) & 0xff);\
}
```

5.2.5 Source File of Q2SD_REG.h

```
/*_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/

                    Q2SD Display List / SHC sample program
                                 (Q2SD_REG.h)

                        Copyright(c) Hitachi Ltd. 1999

  _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/*/

#define  Area2   1   /* Q2SD internal register/UGM address area 2 on MS7709SE01 */
#define  Area5   0   /* Q2SD internal register/UGM address area 5 on MS7709SE01 */

#define   AddressArea Area5   /* set Area2 or Area5 */

#define OFF      0
#define ON       1


/*=======================     DEFINE TYPES      ==========================*/
/*----- Define new type -----*/
#define VU_SHORT (volatile unsigned short * const)


/*====================     DEFINE I/O FUNCTIONS     ======================*/
/*----- Define I/O function -----*/
void outport(unsigned long address, unsigned short data);
unsigned short inport(unsigned long address);
#define  outport(add,data) ( *VU_SHORT(add) ) = ( (unsigned short)(data) )
#define  inport(add)       ( *VU_SHORT(add) )


/*====================     DEFINE MAPPING ADDRESS     ======================*/
/*----- Define commnad double buffer start address -----*/
#define DISPLIST0      0x0F0000L        /* DL0 area start address */
#define DISPLIST1      0x0F8000L        /* DL1 area start address */

#if     AddressArea
#define BASE_ADDRESS   0xA8800000L      /* Q2SD internal register base address  */
```

```
                                        /* Byte address (MS7709SE01/MS4413DB01) */

#define UGMBASE         0xA8000000L     /* UGM base address */
                                        /* Byte address (MS7709SE01/MS4413DB01) */
#else
#define BASE_ADDRESS    0xB4800000L     /* Q2SD internal register base address  */
                                        /* Byte address (MS7709SE01/MS4413DB01) */

#define UGMBASE         0xB4000000L     /* UGM base address */
                                        /* Byte address (MS7709SE01/MS4413DB01) */
#endif

#define _Q2SYSR         0x000L + BASE_ADDRESS       /* No.000 */
#define _Q2SR           0x002L + BASE_ADDRESS       /* No.001 */
#define _Q2SRCR         0x004L + BASE_ADDRESS       /* No.002 */
#define _Q2IER          0x006L + BASE_ADDRESS       /* No.003 */
#define _Q2MEMR         0x008L + BASE_ADDRESS       /* No.004 */
#define _Q2DSMR         0x00AL + BASE_ADDRESS       /* No.005 */
#define _Q2REMR         0x00CL + BASE_ADDRESS       /* No.006 */
#define _Q2IEMR         0x00EL + BASE_ADDRESS       /* No.007 */

#define _Q2DSX          0x010L + BASE_ADDRESS       /* No.008 */
#define _Q2DSY          0x012L + BASE_ADDRESS       /* No.009 */
#define _Q2DSA0         0x014L + BASE_ADDRESS       /* No.00A */
#define _Q2DSA1         0x016L + BASE_ADDRESS       /* No.00B */
#define _Q2DLSAH        0x018L + BASE_ADDRESS       /* No.00C */
#define _Q2DLSAL        0x01AL + BASE_ADDRESS       /* No.00D */
#define _Q2SSAR         0x01CL + BASE_ADDRESS       /* No.00E */
#define _Q2WSAR         0x01EL + BASE_ADDRESS       /* No.00F */
#define _Q2DMASH        0x020L + BASE_ADDRESS       /* No.010 */
#define _Q2DMASL        0x022L + BASE_ADDRESS       /* No.011 */
#define _Q2DMAWL        0x024L + BASE_ADDRESS       /* No.012 */

#define _Q2HDS          0x026L + BASE_ADDRESS       /* No.013 */
#define _Q2HDE          0x028L + BASE_ADDRESS       /* No.014 */
#define _Q2VDS          0x02AL + BASE_ADDRESS       /* No.015 */
#define _Q2VDE          0x02CL + BASE_ADDRESS       /* No.016 */
#define _Q2HSW          0x02EL + BASE_ADDRESS       /* No.017 */
#define _Q2HC           0x030L + BASE_ADDRESS       /* No.018 */
#define _Q2VSP          0x032L + BASE_ADDRESS       /* No.019 */
```

```
#define _Q2VC         0x034L + BASE_ADDRESS       /* No.01A */
#define _Q2DOR        0x036L + BASE_ADDRESS       /* No.01B */
#define _Q2DOG_DOB    0x038L + BASE_ADDRESS       /* No.01C */
#define _Q2CDR        0x03AL + BASE_ADDRESS       /* No.01D */
#define _Q2CDG_CDB    0x03CL + BASE_ADDRESS       /* No.01E */
#define _Q2CSTH       0x03EL + BASE_ADDRESS       /* No.01F */
#define _Q2CSTL       0x040L + BASE_ADDRESS       /* No.020 */

#define _Q2ISAH       0x042L + BASE_ADDRESS       /* No.021 */
#define _Q2ISAL       0x044L + BASE_ADDRESS       /* No.022 */
#define _Q2IDSX       0x046L + BASE_ADDRESS       /* No.023 */
#define _Q2IDSY       0x048L + BASE_ADDRESS       /* No.024 */
#define _Q2IDE        0x04AL + BASE_ADDRESS       /* No.025 */

#define _Q2BGSX       0x04CL + BASE_ADDRESS       /* No.026 */
#define _Q2BGSY       0x04EL + BASE_ADDRESS       /* No.027 */
#define _Q2DMAWH      0x050L + BASE_ADDRESS       /* No.028 */

#define _Q2EQW        0x052L + BASE_ADDRESS       /* No.029 */
#define _Q2SPW        0x054L + BASE_ADDRESS       /* No.02A */
#define _Q2DSMR2      0x056L + BASE_ADDRESS       /* No.02B */
#define _Q2HVP        0x058L + BASE_ADDRESS       /* No.02C */
#define _Q2VVP        0x05AL + BASE_ADDRESS       /* No.02D */

#define _Q2VSAH0      0x062L + BASE_ADDRESS       /* No.031 */
#define _Q2VSAL0      0x064L + BASE_ADDRESS       /* No.032 */
#define _Q2VSAH1      0x066L + BASE_ADDRESS       /* No.033 */
#define _Q2VSAL1      0x068L + BASE_ADDRESS       /* No.034 */
#define _Q2VSAH2      0x06AL + BASE_ADDRESS       /* No.035 */
#define _Q2VSAL2      0x06CL + BASE_ADDRESS       /* No.036 */
#define _Q2VSIZEX     0x06EL + BASE_ADDRESS       /* No.037 */
#define _Q2VSIZEY     0x070L + BASE_ADDRESS       /* No.038 */
#define _Q2VIMR       0x072L + BASE_ADDRESS       /* No.039 */
#define _Q2HCSR1      0x074L + BASE_ADDRESS       /* No.03A */
#define _Q2VCSR1      0x076L + BASE_ADDRESS       /* No.03B */
#define _Q2HCS2       0x078L + BASE_ADDRESS       /* No.03C */
#define _Q2VCS2       0x07AL + BASE_ADDRESS       /* No.03D */
#define _Q2CSAR1      0x07CL + BASE_ADDRESS       /* No.03E */
#define _Q2CSAR2      0x07EL + BASE_ADDRESS       /* No.03F */
```

```
#define _Q2XC         0x080L + BASE_ADDRESS       /* No.040 */
#define _Q2YC         0x082L + BASE_ADDRESS       /* No.041 */
#define _Q2XO         0x084L + BASE_ADDRESS       /* No.042 */
#define _Q2YO         0x086L + BASE_ADDRESS       /* No.043 */
#define _Q2UXMIN      0x088L + BASE_ADDRESS       /* No.044 */
#define _Q2UYMIN      0x08AL + BASE_ADDRESS       /* No.045 */
#define _Q2UXMAX      0x08CL + BASE_ADDRESS       /* No.046 */
#define _Q2UYMAX      0x08EL + BASE_ADDRESS       /* No.047 */
#define _Q2SXMAX      0x090L + BASE_ADDRESS       /* No.048 */
#define _Q2SYMAX      0x092L + BASE_ADDRESS       /* No.049 */
#define _Q2RTNH       0x094L + BASE_ADDRESS       /* No.04A */
#define _Q2RTNL       0x096L + BASE_ADDRESS       /* No.04B */
#define _Q2RSAR       0x098L + BASE_ADDRESS       /* No.04C */
```

5.2.6 Source File of sample.c

```
/*
                    Q2 Display List / SHC sample program (1)
                   (Zoom/Shrink/Rotate/Move 'Solid' Polygon)

                         Copyright(c) Hitachi Ltd. 1999
*/


#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define   PI         3.14159

void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------- */

void  clrscrn(void);
void  ginit(short DBmode);
           /* Graphics System Open  (Q2SD Initialize) */


void main(void)
{
     long   count;
     short  array[8];
     short  d, i, h, j, k, del;
     double rad;
```

```
     short  tx,ty;
     short  DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();                  /* Initialize SH7709 bus state controller */

     ginit(DBmode);               /* Initialize Q2 */
     init_start();                /* Initialize Transfer Procedure */
     draw_start();                /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     del = 4;

     for( count = 0 ; count < 2 ; count++ ){

/* Zoom ------------------------------------- */
          array[0] =  150;    array[1] =  100;
          array[2] =  165;    array[3] =  100;
          array[4] =  165;    array[5] =  115;
          array[6] =  150;    array[7] =  115;

          for(i=0;i<=80;i++) {
               clrscrn();

               array[0] -=  del;      array[1] -=  del;
               array[2] +=  del;      array[3] -=  del;
               array[4] +=  del;      array[5] +=  del;
               array[6] -=  del;      array[7] +=  del;
               polygon4c(0x0000,array,0x001F);              /* POLYGON4C */
```

```
                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Shrink ------------------------------------ */
           for(i=0;i<=80;i++) {
                clrscrn();

                array[0] +=  del;      array[1] +=  del;
                array[2] -=  del;      array[3] +=  del;
                array[4] -=  del;      array[5] -=  del;
                array[6] +=  del;      array[7] -=  del;
                polygon4c(0x0000,array,0x001F);              /* POLYGON4C */

                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Rotate ------------------------------------ */
           for(d=0;d<=360;d+=5) {
                clrscrn();

                array[0] =   -60;        array[1] =   -70;
                array[2] =    40;        array[3] =   -70;
                array[4] =    40;        array[5] =    50;
                array[6] =   -60;        array[7] =    50;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                array[i]   =  tx;
                array[i+1] =  ty;
           }

           lcofs(0x0000, 160, 115);
           polygon4c(0x0000,array,0x001F);        /* POLYGON4C */
           lcofs(0x0000, 0, 0);
           draw_end(DBmode,&first,quick);
           draw_start();
```

```
     }

/* Move ------------------------------------ */
          h=260;
          j=110;
          for(d=0;d<=360;d+=5) {
               clrscrn();

               array[0] =   -50;       array[1] =   -60;
               array[2] =    50;       array[3] =   -60;
               array[4] =    50;       array[5] =    60;
               array[6] =   -50;       array[7] =    60;

               rad = PI * d / 180;
               for(i=0;i<=6;i+=2) {
                    tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                    ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                    array[i]   =  tx;
                    array[i+1] =  ty;
               }

               lcofs(0x0000, h, j);
               polygon4c(0x0000,array,0x001F);     /* POLYGON4C */
               lcofs(0x0000, 0, 0);
               h-=5;
               draw_end(DBmode,&first,quick);
               draw_start();
          }
     }
     return;         /* EXIT */
}


void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ----------------------------------------------------- */
     array[0] =    0; array[1] =    0;
     array[2] =  639; array[3] =    0;
```

```
    array[4] =  639; array[5] =  239;
    array[6] =    0; array[7] =  239;
    polygon4c(0x0000, array, 0x0000);
}

void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

    outport(_Q2SYSR, 0x4080);     /* Initilaize Draw/Display */
    outport(_Q2SRCR, 0xfe80);     /* Clear SR register       */

    /* Set InterFace Control Registers */
    outport(_Q2IER ,  0x0000);
    outport(_Q2MEMR,  0x0031);
    outport(_Q2DSMR,  0x0005);
    outport(_Q2DSMR2, 0x0000);
    outport(_Q2REMR,  0x0041);
    outport(_Q2IEMR,  0x0000);

    /* Set Memory Control Regisers */
    outport(_Q2DSX,       xw);    /* Display size of x                    */
    outport(_Q2DSY,       yw);    /* Display size of y                    */
    outport(_Q2DSA0,  0x0000);    /* Frame buffer 0 area start address(H) */
    outport(_Q2DSA1,  0x0010);    /* Frame buffer 1 area start address(H) */
    outport(_Q2DLSAH, 0x0019);    /* Display list area start address(H)   */
    outport(_Q2DLSAL, 0x0000);    /* Display list area start address(L)   */
    outport(_Q2SSAR,  0x0008);    /* Color area sorce start address(H)    */
    outport(_Q2WSAR,  0x001F);    /* Work area start address(H)           */
    outport(_Q2DMASH, 0x0000);    /* DMA transfer start address(H)        */
    outport(_Q2DMASL, 0x0000);    /* DMA transfer start address(L)        */
    outport(_Q2DMAWL, 0x0000);    /* DAM transfer word                    */
```

```
    /* Display Contral Registers */
    outport(_Q2HDS,     hsw+xs-11   );
    outport(_Q2HDE,     hsw+xs-11+xw);
    outport(_Q2VDS,     ys-2        );
    outport(_Q2VDE,     ys-2+yw     );
    outport(_Q2HSW,     hsw-1       );
    outport(_Q2HC,      hc-1        );
    outport(_Q2VSP,     vc-vsw-1    );
    outport(_Q2VC,      vc-1        );
    outport(_Q2DOR,     0x0000);
    outport(_Q2DOG_DOB, 0x007C);
    outport(_Q2CDR,     0x00FC);
    outport(_Q2CDG_CDB, 0xFCFC);

    /* Input Data Control Registers */
    outport(_Q2ISAH, 0x0000);
    outport(_Q2ISAL, 0x0000);
    outport(_Q2IDSX, 0x0000);
    outport(_Q2IDSY, 0x0000);
    outport(_Q2IDE, 0x0000);

    switch(DBmode) {                    /* Enable Draw/Display */
        case 0:
            outport(_Q2SYSR, 0x2000);  /* Idle,Auto Change mode,No DMA */
            break;
        case 1:
            outport(_Q2SYSR, 0x2040);  /* Idle,Auto Renderring mode,No DMA */
            break;
        default:
            outport(_Q2SYSR, 0x2080);  /* Idle,manual change moe,No DMA */
    }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.7 Source File of sample2.c

```
/*
                    Q2 Display List / SHC sample program (2)
                     (Zoom/Shrink/Rotate/Move 'Text 24×24')

                        Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define  PI              3.14159


void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);

/* ------------------------------------------------------- */

void  clrscrn(void);
void  ginit(short DBmode);    /* Graphics System Open (Q2 Initialize) */



void main()
{
     long  count;
     short array[8];
     short rel_adr_h, rel_adr_l;
```

```
    short d, i, h, j, k, del, lp;
    double rad;
    short tx,ty;
    short txs, tys;
    short DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
  char  first,quick;
  short forecolor = 0x0f0f;       /* charactor  color */
  short backcolor = 0x0101;       /* background color */


/* Define Font Data "漢" ( 24 dots × 24 lines ) ---------------------- */
  unsigned short font_pattern[]={
    /*----------------------------------------------*/
    /* Note : Q2 uses font pattern from LSB to MSB. */
    /*----------------------------------------------*/
    /* 'font_pattern' must be mapped at SuperH's big endian area. */

    /* (MSB  LSB)    (MSB  LSB)   (MSB  LSB)                    */
         0x1c00,       0x0e07,      0x430c,
         0x0c1c,       0xd8e3,      0xffff,
         0x0c00,       0x0003,      0x030c,
         0x0140,       0x4718,      0x3fff,
         0x636e,       0x2c18,      0x1863,
         0x6320,       0x3018,      0x1fff,
         0x6310,       0x1800,      0x1860,
         0xff98,       0x0c3f,      0x0060,
         0x600f,       0xec60,      0xffff,
         0xb00c,       0x0c01,      0x0338,
         0x1c0c,       0x0c0e,      0x3c0e,
         0x038c,       0xecf8,      0x6000
    };

  unsigned font_pattern_word_counter = sizeof(font_pattern)/sizeof(short);
```

```
     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();                    /* Initialize SH7709 bus state controller */

     ginit(DBmode);                 /* Initialize Q2 */


/* ---------------------------------------------------------------------- */

     init_start();                  /* Initialize Transfer Procedure */
     draw_start();                  /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     del = 4;

{
  int jump_address  = (font_pattern_word_counter + 3)*2;
     rel_adr_h = (short)( (jump_address >> 13L) & 0xffffL );
     rel_adr_l = (short)(  jump_address         & 0x1fffL );
}

     for( count = 0 ; count < 2 ; count++ ){

/* Zoom ------------------------------------ */
          array[0] =  150;  array[1] =  100;
          array[2] =  165;  array[3] =  100;
          array[4] =  165;  array[5] =  115;
          array[6] =  150;  array[7] =  115;

          for(i=0;i<=80;i++) {
               clrscrn();

               array[0] -=  del;      array[1] -=  del;
               array[2] +=  del;      array[3] -=  del;
               array[4] +=  del;      array[5] +=  del;
               array[6] -=  del;      array[7] +=  del;
```

```
                polygon4b(0x0040, 0,(15+3)*2, 24,24, array,
                       backcolor,forecolor);                /* polygon4b */

                jump( 0x0040, rel_adr_h, rel_adr_l);        /* jump */

                /* Transfer Font Data "漢" ( 24 dots × 24 lines ) */
                for(lp=0;lp<font_pattern_word_counter;lp++) {
                     *DISPLIST_ptr++ = font_pattern[lp];
                }

                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Shrink ------------------------------------ */
           for(i=0;i<=80;i++) {
                clrscrn();

                array[0] +=  del;      array[1] +=  del;
                array[2] -=  del;      array[3] +=  del;
                array[4] -=  del;      array[5] -=  del;
                array[6] +=  del;      array[7] -=  del;

                polygon4b(0x0040, 0,(15+3)*2, 24,24, array,
                       backcolor,forecolor);                /* polygon4b */

                jump( 0x0040, rel_adr_h, rel_adr_l);        /* jump */

                /* Transfer Font Data "漢" ( 24 dots × 24 lines ) */
                for(lp=0;lp<font_pattern_word_counter;lp++) {
                     *DISPLIST_ptr++ = font_pattern[lp];
                }

                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Rotate ------------------------------------- */
           for(d=0;d<=360;d+=5) {
                clrscrn();
```

```
                array[0] =   -60;       array[1] =   -70;
                array[2] =    40;       array[3] =   -70;
                array[4] =    40;       array[5] =    50;
                array[6] =   -60;       array[7] =    50;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short) (array[i] * cos(rad)- array[i+1] * sin(rad));
                     ty =(short) (array[i] * sin(rad)+ array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
                }

                lcofs(0x0000, 160, 115);
                polygon4b(0x0040, 0,(15+3)*2, 24,24, array,
                       backcolor,forecolor);                /* polygon4b */

                jump( 0x0040, rel_adr_h, rel_adr_l);         /* jump */

                /* Transfer Font Data "漢" ( 24 dots × 24 lines ) */
                for(lp=0;lp<font_pattern_word_counter;lp++) {
                     *DISPLIST_ptr++ = font_pattern[lp];
                }

                lcofs(0x0000, 0, 0);

                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Move ---------------------------------------- */
           h=260;
           j=110;
           for(d=0;d<=360;d+=5) {
                clrscrn();

                array[0] =   -50;          array[1] =   -60;
                array[2] =    50;          array[3] =   -60;
                array[4] =    50;          array[5] =    60;
```

```
                array[6] =   -50;           array[7] =    60;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short) (array[i] * cos(rad)- array[i+1] * sin(rad));
                     ty =(short) (array[i] * sin(rad)+ array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
                }

                lcofs(0x0000, h, j);
                polygon4b(0x0040, 0,(15+3)*2, 24,24, array,
                       backcolor,forecolor);                /* polygon4b */

                jump( 0x0040, rel_adr_h, rel_adr_l);         /* jump */

                /* Transfer Font Data "漢" ( 24 dots × 24 lines ) */
                for(lp=0;lp<font_pattern_word_counter;lp++) {
                     *DISPLIST_ptr++ = font_pattern[lp];
                }

                lcofs(0x0000, 0, 0);

                h-=5;
                draw_end(DBmode,&first,quick);
                draw_start();
           }

     }
     return;         /* EXIT */
}

void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ------------------------------------------------------ */
     array[0] =    0;  array[1] =    0;
     array[2] =  639;  array[3] =    0;
     array[4] =  639;  array[5] =  239;
```

```
    array[6] =    0;  array[7] =  239;
    polygon4c(0x0000, array, 0x0000);
}


void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

    outport(_Q2SYSR, 0x4080);          /* Initilaize Draw/Display */
    outport(_Q2SRCR, 0xfe80);          /* Clear SR register       */

    /* Set InterFace Control Registers */
    outport(_Q2IER ,  0x0000);
    outport(_Q2MEMR,  0x0031);
    outport(_Q2DSMR,  0x0005);
    outport(_Q2DSMR2, 0x0000);
    outport(_Q2REMR,  0x0041);
    outport(_Q2IEMR,  0x0000);

    /* Set Memory Control Regisers */
    outport(_Q2DSX,     xw);         /* Display size of x                    */
    outport(_Q2DSY,     yw);         /* Display size of y                    */
    outport(_Q2DSA0,    0x0000);     /* Frame buffer 0 area start address(H) */
    outport(_Q2DSA1,    0x0010);     /* Frame buffer 1 area start address(H) */
    outport(_Q2DLSAH,   0x0019);     /* Display list area start address(H)   */
    outport(_Q2DLSAL,   0x0000);     /* Display list area start address(L)   */
    outport(_Q2SSAR,    0x0008);     /* Color area sorce start address(H)    */
    outport(_Q2WSAR,    0x001F);     /* Work area start address(H)           */
    outport(_Q2DMASH,   0x0000);     /* DMA transfer start address(H)        */
    outport(_Q2DMASL,   0x0000);     /* DMA transfer start address(L)        */
    outport(_Q2DMAWL,   0x0000);     /* DAM transfer word                    */
```

```
    /* Display Contral Registers */
    outport(_Q2HDS,     hsw+xs-11   );
    outport(_Q2HDE,     hsw+xs-11+xw);
    outport(_Q2VDS,     ys-2        );
    outport(_Q2VDE,     ys-2+yw     );
    outport(_Q2HSW,     hsw-1       );
    outport(_Q2HC,      hc-1        );
    outport(_Q2VSP,     vc-vsw-1    );
    outport(_Q2VC,      vc-1        );
    outport(_Q2DOR,     0x0000);
    outport(_Q2DOG_DOB, 0x007C);
    outport(_Q2CDR,     0x00FC);
    outport(_Q2CDG_CDB, 0xFCFC);

    /* Input Data Control Registers */
    outport(_Q2ISAH,  0x0000);
    outport(_Q2ISAL,  0x0000);
    outport(_Q2IDSX,  0x0000);
    outport(_Q2IDSY,  0x0000);
    outport(_Q2IDE,   0x0000);

    switch(DBmode) {                    /* Enable Draw/Display */
        case 0:
            outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
            break;
        case 1:
            outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
            break;
        default:
            outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
    }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.8 Source File of sample3.c

```
/*
                    Q2 Display List / SHC sample program (1)
                    (Zoom/Shrink/Rotate/Move 'Solid' Polygon)

                        Copyright(c) Hitachi Ltd. 1999
*/


#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define  PI                3.14159

#define _Q2_SSAR_REG_NO 0x00e

void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------ */

void  clrscrn(void);
void  ginit(short DBmode);    /* Graphics System Open  (Q2SD Initialize) */


void main(void)
{
     long   count;
     short  array[8];
     short  d, i, h, j, k, del;
```

```
     double rad;
     short  tx,ty;
     short  DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();              /* Initialize SH7709 bus state controller */

     ginit(DBmode);            /* Initialize Q2 */
     init_start();             /* Initialize Transfer Procedure */
     draw_start();             /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     wpr(0x0000, _Q2_SSAR_REG_NO, 0x10);   /* Sst the SSAR (0x100000) */

     del = 4;

     for( count = 0 ; count < 2 ; count++ ){

       /* POYLGON4A commnad reference to V1 area */
       short txs=640,tys=  0;              /* Reference start point */
       short tdx=320,tdy=240;              /* Reference size */

/* Zoom ---------------------------------------- */
          array[0] =  150;  array[1] =  100;
          array[2] =  165;  array[3] =  100;
          array[4] =  165;  array[5] =  115;
          array[6] =  150;  array[7] =  115;

          for(i=0;i<=80;i++) {
               clrscrn();
```

```
                array[0] -=  del;      array[1] -=  del;
                array[2] +=  del;      array[3] -=  del;
                array[4] +=  del;      array[5] +=  del;
                array[6] -=  del;      array[7] +=  del;
                polygon4a(0x0000, txs,tys, tdx,tdy, array ); /* POLYGON4A */

                draw_end(DBmode,&first,quick);
                draw_start();
          }

/* Shrink ------------------------------------ */
          for(i=0;i<=80;i++) {
                clrscrn();

                array[0] +=  del;      array[1] +=  del;
                array[2] -=  del;      array[3] +=  del;
                array[4] -=  del;      array[5] -=  del;
                array[6] +=  del;      array[7] -=  del;
                polygon4a(0x0000, txs,tys, tdx,tdy, array ); /* POLYGON4A */

                draw_end(DBmode,&first,quick);
                draw_start();
          }

/* Rotate ------------------------------------- */
          for(d=0;d<=360;d+=5) {
                clrscrn();

                array[0] =   -60;          array[1] =   -70;
                array[2] =    40;          array[3] =   -70;
                array[4] =    40;          array[5] =    50;
                array[6] =   -60;          array[7] =    50;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                     ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
```

```
                }

                lcofs(0x0000, 160, 115);
                polygon4a(0x0000, txs,tys, tdx,tdy, array );   /* POLYGON4A */
                lcofs(0x0000, 0, 0);
                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Move ------------------------------------- */
           h=260;
           j=110;
           for(d=0;d<=360;d+=5) {
                clrscrn();

                array[0] =   -50;        array[1] =   -60;
                array[2] =    50;        array[3] =   -60;
                array[4] =    50;        array[5] =    60;
                array[6] =   -50;        array[7] =    60;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                     ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
                }

                lcofs(0x0000, h, j);
                polygon4a(0x0000, txs,tys, tdx,tdy, array );   /* POLYGON4A */
                lcofs(0x0000, 0, 0);
                h-=5;
                draw_end(DBmode,&first,quick);
                draw_start();
           }
     }
     return;        /* EXIT */
}
```

```
void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ------------------------------------------------- */
     array[0] =    0; array[1] =    0;
     array[2] =  639; array[3] =    0;
     array[4] =  639; array[5] =  239;
     array[6] =    0; array[7] =  239;
     polygon4c(0x0000, array, 0x0000);
}

void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

     outport(_Q2SYSR, 0x4080);              /* Initilaize Draw/Display */
     outport(_Q2SRCR, 0xfe80);              /* Clear SR register       */

     /* Set InterFace Control Registers */
     outport(_Q2IER , 0x0000);
     outport(_Q2MEMR, 0x0031);
     outport(_Q2DSMR, 0x0005);
     outport(_Q2DSMR2,0x0000);
     outport(_Q2REMR, 0x0041);
     outport(_Q2IEMR, 0x0000);

     /* Set Memory Control Regisers */
     outport(_Q2DSX,       xw);     /* Display size of x                    */
     outport(_Q2DSY,       yw);     /* Display size of y                    */
     outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
     outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
```

```
    outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
    outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
    outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
    outport(_Q2WSAR,  0x001F);     /* Work area start address(H)           */
    outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)        */
    outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)        */
    outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                    */

    /* Display Contral Registers */
    outport(_Q2HDS,     hsw+xs-11   );
    outport(_Q2HDE,     hsw+xs-11+xw);
    outport(_Q2VDS,     ys-2        );
    outport(_Q2VDE,     ys-2+yw     );
    outport(_Q2HSW,     hsw-1       );
    outport(_Q2HC,      hc-1        );
    outport(_Q2VSP,     vc-vsw-1    );
    outport(_Q2VC,      vc-1        );
    outport(_Q2DOR,     0x0000);
    outport(_Q2DOG_DOB, 0x007C);
    outport(_Q2CDR,     0x00FC);
    outport(_Q2CDG_CDB, 0xFCFC);

    /* Input Data Control Registers */
    outport(_Q2ISAH, 0x0000);
    outport(_Q2ISAL, 0x0000);
    outport(_Q2IDSX, 0x0000);
    outport(_Q2IDSY, 0x0000);
    outport(_Q2IDE,  0x0000);

    switch(DBmode) {                      /* Enable Draw/Display */
         case 0:
              outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
              break;
         case 1:
              outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
              break;
         default:
              outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
    }
}
```

```
#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.9 Source File of sample4.c

```
/*
                    Q2 Display List / SHC sample program
                       (Draw hexagon with no pattern)

                       Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define   MAX_WORD  512

void  init_BSC(void);     /* Initialize SH7709 bus state controller */
void  init_start(void);   /* Initilaze Display List Double Buffering (Only 1st) */
short draw_start(void);   /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------- */

void  ginit(short DBmode);   /* Graphics System Open  (Q2 Initialize) */


void main(void)
{
     short array[MAX_WORD], poly[MAX_WORD];
     short DBmode;
     short xmin,xmax,ymin,ymax;
     short location_of_base_line;
     short fill_color = 0x0404;
     short fill_point = 6;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
```

```
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();               /* Initialize SH7709 bus state controller */

     ginit(DBmode);            /* Initialize Q2 */

     init_start();             /* Initialize Transfer Procedure */
     draw_start();             /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

/* ploygon4c ( clear screen ) ---------------------------------------- */
     array[0] =    0;  array[1] =    0;     /* Left  Up   */
     array[2] =  639;  array[3] =    0;     /* Right Up   */
     array[4] =  639;  array[5] =  239;     /* Left  Down */
     array[6] =    0;  array[7] =  239;     /* Right Down */
     polygon4c(0x0000, array,  0x0000);     /* polygon4c  */

     array[0] =    0;  array[1] =    0;     /* Left  Up   */
     array[2] =  159;  array[3] =    0;     /* Right Up   */
     array[4] =  159;  array[5] =  119;     /* Left  Down */
     array[6] =    0;  array[7] =  119;     /* Right Down */
     polygon4c(0x0000, array,  0xffff);     /* polygon4c  */

     array[0] =  160;  array[1] =  120;     /* Left  Up   */
     array[2] =  319;  array[3] =  120;     /* Right Up   */
     array[4] =  319;  array[5] =  239;     /* Left  Down */
     array[6] =  160;  array[7] =  239;     /* Right Down */
     polygon4c(0x0000, array,  0xffff);     /* polygon4c  */

/* Define hexagon ---------------------------------------------------- */
     array[ 0] =   40; array[ 1] =  110;
     array[ 2] =  130; array[ 3] =   70;
     array[ 4] =  190; array[ 5] =   80;
```

```
    array[ 6] =  200; array[ 7] =  150;
    array[ 8] =  140; array[ 9] =  210;
    array[10] =   60; array[11] =  160;
    array[12] = array[ 0];  array[13] = array[1];

/* Draw hexagon at work screen ---------------------------------------- */
    xmin =  40; /* minimum of x */
    xmax = 200; /* maximum of x */
    ymin =  70; /* minimum of y */
    ymax = 210; /* maximum of y */

    location_of_base_line = xmin;

    /* Clear work screen for ftrap */
    clrw(0x0000, xmin, ymin, xmax, ymax);                    /* clrw */

    /* Draw hexagon at work screen */
    ftrap(0x0000, fill_point+1, location_of_base_line, array);  /* ftrap */

/* Draw hexagon with no pattern at rendering screen referencing to work screen -*/
    poly[ 0] =  xmin;poly[ 1] =  ymin;
    poly[ 2] =  xmax;poly[ 3] =  ymin;
    poly[ 4] =  xmax;poly[ 5] =  ymax;
    poly[ 6] =  xmin;poly[ 7] =  ymax;

    polygon4c(0x0021, poly, fill_color);                     /* polygon4c */

    line(0x0000, fill_color, fill_point+1, array);           /* line */

    draw_end(DBmode,&first,quick);

    return;         /* EXIT */
}


void ginit(short DBmode)
{                   /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
```

```
unsigned short hc  = 910;

unsigned short vsw =   3;
unsigned short ys  =  16;
unsigned short yw  = 240;
unsigned short vc  = 262;

   outport(_Q2SYSR, 0x4080);           /* Initilaize Draw/Display */
   outport(_Q2SRCR, 0xfe80);           /* Clear SR register       */

   /* Set InterFace Control Registers */
   outport(_Q2IER ,  0x0000);
   outport(_Q2MEMR,  0x0031);
   outport(_Q2DSMR,  0x0005);
   outport(_Q2DSMR2, 0x0000);
   outport(_Q2REMR,  0x0041);
   outport(_Q2IEMR,  0x0000);

   /* Set Memory Control Regisers     */
   outport(_Q2DSX,       xw);     /* Display size of x                   */
   outport(_Q2DSY,       yw);     /* Display size of y                   */
   outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
   outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
   outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
   outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
   outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
   outport(_Q2WSAR,  0x001F);     /* Work area start address(H)           */
   outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)        */
   outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)        */
   outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                    */

   /* Display Contral Registers */
   outport(_Q2HDS,     hsw+xs-11   );
   outport(_Q2HDE,     hsw+xs-11+xw);
   outport(_Q2VDS,     ys-2        );
   outport(_Q2VDE,     ys-2+yw     );
   outport(_Q2HSW,     hsw-1       );
   outport(_Q2HC,      hc-1        );
   outport(_Q2VSP,     vc-vsw-1    );
   outport(_Q2VC,      vc-1        );
```

```
      outport(_Q2DOR,     0x0000);
      outport(_Q2DOG_DOB, 0x007C);
      outport(_Q2CDR,     0x00FC);
      outport(_Q2CDG_CDB, 0xFCFC);

      /* Input Data Control Registers */
      outport(_Q2ISAH, 0x0000);
      outport(_Q2ISAL, 0x0000);
      outport(_Q2IDSX, 0x0000);
      outport(_Q2IDSY, 0x0000);
      outport(_Q2IDE,  0x0000);

      switch(DBmode) {                      /* Enable Draw/Display */
           case 0:
                outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
                break;
           case 1:
                outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
                break;
           default:
                outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
      }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.10 Source File of sample5.c

```
/*
                    Q2 Display List / SHC sample program (5)

                           ( Move 'Solid' Polygon )

                         Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define   PI         3.14159

#define _Q2_REMR_REG_NO 0x006
#define _Q2_RSAR_REG_NO 0x04C


void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick); /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------- */

void  ginit(short DBmode);         /* Graphics System Open  (Q2 Initialize) */

void main(void)
{
     long   count;
     short array[8];
     short DBmode;
/*
```

```
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();               /* Initialize SH7709 bus state controller */

     ginit(DBmode);            /* Initialize Q2 */


     init_start();             /* Initialize Transfer Procedure */
     draw_start();             /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     for( count = 0 ; count < 2 ; count++ ){

     short i,d;

     /* MOVE */
          short kj = 50;
          double rad;

          for(d=0;d<=360;d+=5) {

          /* Job No.1 : clear screen */
               array[0] =    0;  array[1] =    0;
               array[2] =  639;  array[3] =    0;
               array[4] =  639;  array[5] =  239;
               array[6] =    0;  array[7] =  239;
               polygon4c(0x0000, array, 0x0606);


          /* Set drawing start address in the RSAR */
               if ( (inport( _Q2SR ) & 0x0100) != 0 ) {
```

```
                wpr( 0x0000, _Q2_RSAR_REG_NO, inport( _Q2DSA0 ) );
            } else {
                wpr( 0x0000, _Q2_RSAR_REG_NO, inport( _Q2DSA1 ) );
            }

        /* Enable the RSAR */
            wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) | 0x8000 );


        /* Job No.2 : draw two solid rectangles */
            array[0] =   -50;          array[1] =   -(60);
            array[2] =    50;          array[3] =   -(60);
            array[4] =    50;          array[5] =   -(-60);
            array[6] =   -50;          array[7] =   -(-60);

            rad = PI * d / 180;
            for(i=0;i<=6;i+=2) {
                short tx =(short)  (array[i] * cos(rad)-  array[i+1] *
sin(rad));
                short ty =(short)  (array[i] * sin(rad)+  array[i+1] *
cos(rad));
                array[i]   =  tx;
                array[i+1] =  ty;
            }

            lcofs(0x0000, kj,100);
            polygon4c(0x0000,array,0xF800);          /* POLYGON4C */

            lcofs(0x0000, 320-kj,100);
            polygon4c(0x0000,array,0xF800);          /* POLYGON4C */

            lcofs(0x0000, 0,0);


        /* Disable the RSAR */
            wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) & 0x7fff );


            kj+=5;

        /* Drawing start */
```

```
                draw_end(DBmode,&first,quick);
                draw_start();

           } /* for */
      }
      return;         /* EXIT */
}


void ginit(short DBmode)
{                     /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

     outport(_Q2SYSR, 0x4080);     /* Initilaize Draw/Display */
     outport(_Q2SRCR, 0xfe80);     /* Clear SR register       */

     /* Set InterFace Control Registers */
     outport(_Q2IER ,  0x0000);
     outport(_Q2MEMR,  0x0031);
     outport(_Q2DSMR,  0x0005);
     outport(_Q2DSMR2, 0x0000);
     outport(_Q2REMR,  0x0041);
     outport(_Q2IEMR,  0x0000);

     /* Set Memory Control Regisers */
     outport(_Q2DSX,       xw);     /* Display size of x                   */
     outport(_Q2DSY,       yw);     /* Display size of y                   */
     outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
     outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
     outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
     outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
     outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
```

```
    outport(_Q2WSAR,  0x001F);     /* Work area start address(H)          */
    outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)       */
    outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)       */
    outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                   */

    /* Display Contral Registers */
    outport(_Q2HDS,     hsw+xs-11   );
    outport(_Q2HDE,     hsw+xs-11+xw);
    outport(_Q2VDS,     ys-2        );
    outport(_Q2VDE,     ys-2+yw     );
    outport(_Q2HSW,     hsw-1       );
    outport(_Q2HC,      hc-1        );
    outport(_Q2VSP,     vc-vsw-1    );
    outport(_Q2VC,      vc-1        );
    outport(_Q2DOR,     0x0000);
    outport(_Q2DOG_DOB, 0x007C);
    outport(_Q2CDR,     0x00FC);
    outport(_Q2CDG_CDB, 0xFCFC);

    /* Input Data Control Registers */
    outport(_Q2ISAH,  0x0000);
    outport(_Q2ISAL,  0x0000);
    outport(_Q2IDSX,  0x0000);
    outport(_Q2IDSY,  0x0000);
    outport(_Q2IDE,   0x0000);

    switch(DBmode) {                     /* Enable Draw/Display */
         case 0:
              outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
              break;
         case 1:
              outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
              break;
         default:
              outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
    }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.11 Source File of sample6.c

```
/*
                    Q2 Display List / SHC sample program (6)
                                ( Move 'Image')

           Must be load "bolls.bmp" to (txs,tys) = (0,832), before.
           SSAR = 0x80000

                        Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


static short speed[]={2,5,4,8,3,4,6,1,7,5,
                6,9,2,4,3,6,5,4,2,6,
                7,2,4,3,1,5,1,4,9,3,
                4,9,7,2,4,2,3,7,2,4,
                6,4,3,8,2,4,6,8,6,3,
                6,2,7,6,3,5,4,1,6,6,
                2,5,4,6,3,9,6,6,2,7,
                6,4,3,9,5,1,3,7,5,1,
                6,3,8,4,1,1,6,2,5,9,
                3,6,1,5,3,4,5,6,9,6
                };

static unsigned short color_table[]={
                                     (255/8)*2048 | (128/4)*32 | ( 64/8),
                                     (255/8)*2048 | (  0/4)*32 | (128/8),
                                     (128/8)*2048 | (128/4)*32 | (255/8),
                                     (128/8)*2048 | (255/4)*32 | (255/8),
                                     (  0/8)*2048 | (255/4)*32 | (128/8),
                                     (255/8)*2048 | (255/4)*32 | (128/8),
```

```
                                    (255/8)*2048 | (  0/4)*32 | (255/8),
                                    (  0/8)*2048 | (  0/4)*32 | (255/8),
                                    (255/8)*2048 | (  0/4)*32 | (  0/8),
                                    (128/8)*2048 | (128/4)*32 | (128/8)
};

#define OFF          0
#define ON           1

void  init_BSC(void);     /* Initialize SH7709 bus state controller */
void  init_start(void);   /* Initilaze Display List Double Buffering (Only 1st) */
short draw_start(void);   /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick); /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------ */

void  clrscrn(void);
void  ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */


void main(void)
{
  long  count;
  short block_count = 100;
  short lp;
  short dx[100], dy[100];
  short array[100][8];
  short DBmode;
  short ah,al;
  short size_x=16, size_y=16;
  char first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();             /* Initialize SH7709 bus state controller */

     ginit(DBmode);          /* Initialize Q2 */
```

```
/* Transfer Data "●" ( 16 dots × 16 lines ) */
{
  short i;
  unsigned short *address;
  unsigned long SrcAdr;
  unsigned short crcl_pattern[]={
     /* (MSB LSB) */
        0x0000,
        0x07e0,
        0x1ff8,
        0x3ffc,
        0x3ffc,
        0x7ffe,
        0x7ffe,
        0x7ffe,
        0x7ffe,
        0x7ffe,
        0x7ffe,
        0x3ffc,
        0x3ffc,
        0x1ff8,
        0x07e0,
        0x0000
     };

      SrcAdr = 0x006000L + UGMBASE;
      address = (unsigned short *)SrcAdr;
      for(i=0;i<16;i++) {
          *address++ = crcl_pattern[i];
     }

     ah = (short)( ((SrcAdr-UGMBASE) >> 13L) & 0xffffL );
     al = (short)(  (SrcAdr-UGMBASE) & 0x1fffL );
}

for(lp=0; lp<block_count; lp++){
     dx[lp] = speed[lp];
     dy[lp] = speed[lp];
     array[lp][0] = 0  +lp*speed[lp]/2;   array[lp][1] = 0 +lp*speed[lp]/2;
```

```
          array[lp][2] = size_x-1+lp*speed[lp]/2; array[lp][3] = 0+lp*speed[lp]/2;
          array[lp][4] = size_x-1+lp*speed[lp]/2; array[lp][5] = size_y-
1+lp*speed[lp]/2;
          array[lp][6] = 0  +lp*speed[lp]/2;   array[lp][7] = size_y-
1+lp*speed[lp]/2;
     }

     init_start();     /* Initialize Transfer Procedure */
     draw_start();     /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     for( count = 0 ; count < 1000 ; count++ ){
       short tmp_dx;

          clrscrn();
          for(lp=0; lp<block_count; lp++){
               if ( array[lp][0] <   0 ) dx[lp] =  speed[lp];
               if ( array[lp][2] > 319 ) dx[lp] = -speed[lp];
               if ( array[lp][1] <   0 ) dy[lp] =  speed[lp];
               if ( array[lp][7] > 239 ) dy[lp] = -speed[lp];

               array[lp][0] += dx[lp];   array[lp][1] += dy[lp];
               array[lp][2] += dx[lp];   array[lp][3] += dy[lp];
               array[lp][4] += dx[lp];   array[lp][5] += dy[lp];
               array[lp][6] += dx[lp];   array[lp][7] += dy[lp];

               if(dx[lp] < 0)
                    tmp_dx = -dx[lp];
               else
                    tmp_dx = dx[lp];
               polygon4b(0x0200, ah, al, 16, 16, array[lp], 0x0000,
color_table[tmp_dx-1]);
          }

          draw_end(DBmode,&first,quick);
          draw_start();

     }
}
```

```
static void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ------------------------------------------------- */
     array[0] =    0; array[1] =    0;
     array[2] =  639; array[3] =    0;
     array[4] =  639; array[5] =  239;
     array[6] =    0; array[7] =  239;
     polygon4c(0x0000, array, 0x0000);
}



void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

     outport(_Q2SYSR, 0x4080);           /* Initilaize Draw/Display */
     outport(_Q2SRCR, 0xfe80);           /* Clear SR register       */

     /* Set InterFace Control Registers */
     outport(_Q2IER ,  0x0000);
     outport(_Q2MEMR,  0x0031);
     outport(_Q2DSMR,  0x0005);
     outport(_Q2DSMR2, 0x0000);
     outport(_Q2REMR,  0x0041);
     outport(_Q2IEMR,  0x0000);

     /* Set Memory Control Regisers */
     outport(_Q2DSX,       xw);     /* Display size of x                    */
     outport(_Q2DSY,       yw);     /* Display size of y                    */
     outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
```

```
outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
outport(_Q2WSAR,  0x001F);     /* Work area start address(H)           */
outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)        */
outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)        */
outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                    */

/* Display Contral Registers */
outport(_Q2HDS,     hsw+xs-11   );
outport(_Q2HDE,     hsw+xs-11+xw);
outport(_Q2VDS,     ys-2        );
outport(_Q2VDE,     ys-2+yw     );
outport(_Q2HSW,     hsw-1       );
outport(_Q2HC,      hc-1        );
outport(_Q2VSP,     vc-vsw-1    );
outport(_Q2VC,      vc-1        );
outport(_Q2DOR,     0x0000);
outport(_Q2DOG_DOB, 0x007C);
outport(_Q2CDR,     0x00FC);
outport(_Q2CDG_CDB, 0xFCFC);

/* Input Data Control Registers */
outport(_Q2ISAH, 0x0000);
outport(_Q2ISAL, 0x0000);
outport(_Q2IDSX, 0x0000);
outport(_Q2IDSY, 0x0000);
outport(_Q2IDE,  0x0000);

switch(DBmode) {                      /* Enable Draw/Display */
     case 0:
          outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
          break;
     case 1:
          outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
          break;
     default:
          outport(_Q2SYSR, 0x2080);  /* Idle,manual change moe,No DMA */
}
```

```
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.12 Source File of sample7.c

```
/*
                    Q2 Display List / SHC sample program (7)
                     (Zoom/Shrink/Rotate/Move 'Text 24×24')

                          Copyright(c) Hitachi Ltd. 1999

               CLK1 = 14.32 MHz

               Color depth  ; 16bit/pixel
               Display size : 640 pixel X 480 line
               Scan mode    : Interrace sync & video (TVM = 11)

               Frame 0 start addtess          : 0x000000       (  0,   0)
               Frame 1 start address          : 0x100000       (  0, 512)
               Display list 0 start address   : 0x0F0000       (  0, 480)
               Display list 1 start address   : 0x1F0000       (  0, 992)
               Color sorce area start address : Undefine
               Mono pattern area start address: 0x006000       (640,   0)
               Work area start address        : Undefine
*/
#include <machine.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


#undef    DISPLIST0
#undef    DISPLIST1

#define   DISPLIST0  0x0F0000L       /* (0,480) */
#define   DISPLIST1  0x1F0000L       /* (0,992) */


unsigned short *DISPLIST_ptr;

#define   PI                          3.14159
```

```
void  init_BSC(void);     /* Initialize SH7709 bus state controller */
void  init_start(void);   /* Initilaze Display List Double Buffering (Only 1st) */
short draw_start(void);   /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);

/* ------------------------------------------------------ */

void  clrscrn(void);
void  ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */


void main()
{
     long  count;
     short array[8];
     short ah,al;
     short d, i, h, j, k, del;
     double rad;
     short tx,ty;
     short txs, tys;
     short DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char    first,quick;
     int     SrcAdr;         /* Source Address */
     short   forecolor;      /* charactor  color */
     short   backcolor;      /* background color */

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();             /* Initialize SH7709 bus state controller */

     ginit(DBmode);          /* Initialize Q2 */
```

```
/* Transfer Font Data "漢" ( 24 dots × 24 lines ) ---------------------- */
{
  short i;
  unsigned short *address;
  unsigned short font_pattern[]={
          /*-----------------------------------------------*/
          /* Note : Q2 uses font pattern from LSB to MSB. */
          /*-----------------------------------------------*/
          /* 'font_pattern' must be mapped at SuperH's big endian area. */

          /* (MSB  LSB)    (MSB  LSB)   (MSB  LSB)                    */
               0x1c00,       0x0e07,      0x430c,
               0x0c1c,       0xd8e3,      0xffff,
               0x0c00,       0x0003,      0x030c,
               0x0140,       0x4718,      0x3fff,
               0x636e,       0x2c18,      0x1863,
               0x6320,       0x3018,      0x1fff,
               0x6310,       0x1800,      0x1860,
               0xff98,       0x0c3f,      0x0060,
               0x600f,       0xec60,      0xffff,
               0xb00c,       0x0c01,      0x0338,
               0x1c0c,       0x0c0e,      0x3c0e,
               0x038c,       0xecf8,      0x6000
     };

     SrcAdr = 0x6000L + UGMBASE; /* (640,0) */
     address = (unsigned short *)SrcAdr;
     forecolor = 0x0F0F;
     backcolor = 0x0101;
     for(i=0;i<36;i++) {
          *address++ = font_pattern[i];
     }

}

/* ------------------------------------------------------------------------ */

     init_start();            /* Initialize Transfer Procedure */
     draw_start();            /* Start Transfer Display List to UGM */
```

```
      sclip(0x0000, 639,479);
      lcofs(0x0000,0,0);

      del = 4;
      ah = (short)( ((SrcAdr-UGMBASE) >> 13L) & 0xffffL );
      al = (short)(  (SrcAdr-UGMBASE) & 0x1fffL );

      for( count = 0 ; count < 2 ; count++ ){

/* Zoom ---------------------------------------- */
           array[0] =  200;    array[1] =  200;
           array[2] =  215;    array[3] =  200;
           array[4] =  215;    array[5] =  315;
           array[6] =  200;    array[7] =  315;

           for(i=0;i<=45;i++) {
                clrscrn();

                array[0] -=  del;      array[1] -=  del;
                array[2] +=  del;      array[3] -=  del;
                array[4] +=  del;      array[5] +=  del;
                array[6] -=  del;      array[7] +=  del;
                polygon4b(0x0000, ah,al, 24,24, array,
                                       backcolor,forecolor);    /* polygon4b */

                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Shrink -------------------------------------- */
           for(i=0;i<=45;i++) {
                clrscrn();

                array[0] +=  del;      array[1] +=  del;
                array[2] -=  del;      array[3] +=  del;
                array[4] -=  del;      array[5] -=  del;
                array[6] +=  del;      array[7] -=  del;
                polygon4b(0x0000, ah,al, 24,24, array,
                                       backcolor,forecolor);    /* polygon4b */
```

```
                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Rotate ------------------------------------ */
           for(d=0;d<=360;d+=8) {
                clrscrn();

                array[0] =   -60;      array[1] =   -70;
                array[2] =    40;      array[3] =   -70;
                array[4] =    40;      array[5] =    50;
                array[6] =   -60;      array[7] =    50;

                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                     ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
                }

                lcofs(0x0000, 160, 115);
                polygon4b(0x0000, ah,al, 24,24, array,
                                       backcolor,forecolor);     /* polygon4b */
                lcofs(0x0000, 0, 0);
                draw_end(DBmode,&first,quick);
                draw_start();
           }

/* Move -------------------------------------- */
           h=600;
           j=110;
           for(d=0;d<=720;d+=8) {
                clrscrn();

                array[0] =   -50;      array[1] =   -60;
                array[2] =    50;      array[3] =   -60;
                array[4] =    50;      array[5] =    60;
                array[6] =   -50;      array[7] =    60;
```

```
                rad = PI * d / 180;
                for(i=0;i<=6;i+=2) {
                     tx =(short)  (array[i] * cos(rad)-  array[i+1] * sin(rad));
                     ty =(short)  (array[i] * sin(rad)+  array[i+1] * cos(rad));
                     array[i]   =  tx;
                     array[i+1] =  ty;
                }

                lcofs(0x0000, h, j);
                polygon4b(0x0000, ah,al, 24,24, array,
                                        backcolor,forecolor);     /* polygon4b */
                lcofs(0x0000, 0, 0);
                h-=10;
                draw_end(DBmode,&first,quick);
                draw_start();
           }

     }
     return;         /* EXIT */
}

void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ------------------------------------------------------ */
     array[0] =    0; array[1] =    0;
     array[2] =  639; array[3] =    0;
     array[4] =  639; array[5] =  479;
     array[6] =    0; array[7] =  479;
     polygon4c(0x0000, array, 0x0034);
}



void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  42;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;
```

```
unsigned short vsw =   3;
unsigned short ys  =  16;
unsigned short yw  = 240;
unsigned short vc  = 262;

    outport(_Q2SYSR, 0x4080);     /* Initilaize Draw/Display */
    outport(_Q2SRCR, 0xfe00);     /* Clear SR register       */

    /* Set InterFace Control Registers */
    outport(_Q2IER ,  0x0000);
    outport(_Q2MEMR,  0x0031);
    outport(_Q2DSMR,  0x0035);
    outport(_Q2DSMR2, 0x0000);
    outport(_Q2REMR,  0x0041);
    outport(_Q2IEMR,  0x0000);

    /* Set Memory Control Regisers */
    outport(_Q2DSX,       xw);     /* Display size of x                   */
    outport(_Q2DSY,       yw);     /* Display size of y                   */
    outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
    outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
    outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
    outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
    outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
    outport(_Q2WSAR,  0x001F);     /* Work area start address(H)           */
    outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)        */
    outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)        */
    outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                    */

    /* Display Contral Registers */
    outport(_Q2HDS,     hsw+xs-11   );
    outport(_Q2HDE,     hsw+xs-11+xw);
    outport(_Q2VDS,     ys-2        );
    outport(_Q2VDE,     ys-2+yw     );
    outport(_Q2HSW,     hsw-1       );
    outport(_Q2HC,      hc-1        );
    outport(_Q2VSP,     vc-vsw-1    );
    outport(_Q2VC,      vc-1        );
    outport(_Q2DOR,     0x0000);
```

```
     outport(_Q2DOG_DOB, 0x007C);
     outport(_Q2CDR,     0x00FC);
     outport(_Q2CDG_CDB, 0xFCFC);

     /* Input Data Control Registers */
     outport(_Q2ISAH, 0x0000);
     outport(_Q2ISAL, 0x0000);
     outport(_Q2IDSX, 0x0000);
     outport(_Q2IDSY, 0x0000);
     outport(_Q2IDE,  0x0000);

     switch(DBmode) {                       /* Enable Draw/Display */
          case 0:
               outport(_Q2SYSR, 0x2000);    /* Idle,Auto Change mode,No DMA,No
Cache,7709 */
               break;
          case 1:
               outport(_Q2SYSR, 0x2040);    /* Idle,Auto Renderring mode,No
DMA,No Cache,7709 */
               break;
          default:
               outport(_Q2SYSR, 0x2080);    /* Idle,manual change moe,No DMA,No
Cache,7709 */
     }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.13 Source File of sample8.c

```
/*_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/

                   Q2 Display List / SHC sample program (8)
                         ( Rotating 'a solid model' )
                 ( A structure of solid model has 20 surfaces )

                       Copyright(c) Hitachi Ltd. 1999

  _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/*/

#include <machine.h>
#include <stdio.h>
#include <stdlib.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


/* Coordinate3 */
     short point[][3] = {
          0, 360, -2048,
          0, 360, 256,
          0, 448, 256,
          0, 448, -2048,
          2304, 360, -2048,
          2304, 448, -2048,
          2304, 360, 256,
          2304, 448, 256,
          224, 0, -32,
          224, 0, -1792,
          288, 0, -1792,
          288, 0, -32,
          224, 360, -1792,
          288, 360, -1792,
          224, 0, 32,
          224, 360, 32,
          288, 360, -32,
          2016, 0, -32,
          2016, 0, 32,
```

```
            2080, 0, 32,
            2080, 360, 32,
            2016, 360, -32,
            2016, 0, -1760,
            2080, 0, -1760,
            2080, 0, -1824,
            2080, 360, -1824,
            2016, 360, -1760,
            256, 0, -1760,
            256, 0, -1824,
            256, 360, -1824,
            256, 360, -1760
      };

/* IndexedFaceSet */
      short coordIndex[][5] = {
                  0, 1, 2, 3, -1,
                  4, 0, 3, 5, -1,
                  6, 4, 5, 7, -1,
                  1, 6, 7, 2, -1,
                  2, 7, 5, 3, -1,
                  1, 0, 4, 6, -1,
                  8, 9, 10, 11, -1,
                  10, 9, 12, 13, -1,
                  9, 14, 15, 12, -1,
                  11, 10, 13, 16, -1,
                  14, 8, 17, 18, -1,
                  14, 19, 20, 15, -1,
                  17, 11, 16, 21, -1,
                  18, 22, 23, 19, -1,
                  19, 24, 25, 20, -1,
                  22, 17, 21, 26, -1,
                  27, 28, 24, 23, -1,
                  24, 28, 29, 25, -1,
                  27, 22, 26, 30, -1,
                  28, 27, 30, 29, -1
      };

/* X,Y and Z are constance. */
#define X 0
```

```
#define Y 1
#define Z 2

#define _Z1 2
#define _Z2 5
#define _Z3 8
#define _Z4 11

/* N define count of surface. */
#define N 100

#define _DUMMY 0

short sin_table[91]={
   0, 142, 285, 428, 571, 713, 856, 998,1140,1281,
1422,1563,1703,1842,1981,2120,2258,2395,2531,2667,
2801,2935,3068,3200,3331,3462,3591,3719,3845,3971,
4096,4219,4341,4461,4580,4698,4815,4930,5043,5155,
5265,5374,5481,5586,5690,5792,5892,5991,6087,6182,
6275,6366,6455,6542,6627,6710,6791,6870,6947,7021,
7094,7164,7233,7299,7362,7424,7483,7540,7595,7647,
7697,7745,7791,7834,7874,7912,7948,7982,8012,8041,
8067,8091,8112,8130,8147,8160,8172,8180,8187,8190,8192
};


unsigned short *DISPLIST_ptr;



void  init_BSC(void);     /* Initialize SH7709 bus state controller */
void  init_start(void);   /* Initilaze Display List Double Buffering (Only 1st) */
short draw_start(void);   /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick); /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------ */

void clrscrn(void);
void ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */
```

```
/* This is function for rotation. */
void rotate_a_rectangle( short array[12], short angle, short center, short dx,
short dy, short dz );

/* This is function for change from 3D to 2D. */
void convert_d3_into_d2( short array[12], short dx, short dy, short dz, short
_z_size );

/* This is function for Z sorting. */
void z_sort( short array[N][12], short st[N], short n );

void set_polygon_sample(short array[N][12], short point[N][3],
                    short coordIndex[N][5], short _n_point);



void main(void)
{
  long  count;
  char  first = ON;
  char  quick = OFF;
  short _z_size = 1024;

/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
  short DBmode = 1;

     init_BSC();            /* Initialize SH7709 bus state controller */

     ginit(DBmode);         /* Initialize Q2 */
     init_start();          /* Initialize Transfer Procedure */
     draw_start();          /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);

     for( count = 0 ; count < 5 ; count++ ){

          {
            short poly[N][12], st[N];
```

```
  short poly2[N][12], st2[N];
  short color[N];
  short zz = 300;
  short d;
  short n_point;
  short lp;

n_point = sizeof(coordIndex)/sizeof(short)/5;

for(lp=1;lp<=n_point;lp++)
     color[lp] = (((lp%16)<<8) & 0xff00) | (lp%16);

for(d=0;d<360*3;d+=5) {
     clrscrn();

     /* Object No.1 */
     set_polygon_sample(poly, point, coordIndex, n_point );

     /* Object No.2 */
     set_polygon_sample(poly2, point, coordIndex, n_point );

          for(lp=0;lp<n_point;lp++){
            rotate_a_rectangle( poly[lp], d, X, _DUMMY, 20, -100 );
            rotate_a_rectangle( poly[lp], d, Z, 0, 0, _DUMMY );

            rotate_a_rectangle( poly2[lp], d+30, X, _DUMMY, 20, -100 );
            rotate_a_rectangle( poly2[lp], d+30, Z, 0, 0, _DUMMY );
          }

          /* Z sorting */
          z_sort( poly, st, n_point );
          z_sort( poly2, st2, n_point );

          /* Convert 3D data into 2D data */
          for(lp=0;lp<n_point;lp++){
            convert_d3_into_d2( poly[lp], 100, 100, zz, _z_size );
            convert_d3_into_d2( poly2[lp], 100+125, 100-20, zz+200,
_z_size );
          }
```

```
                    /* POLYGON4C */
                    for(lp=0;lp<n_point;lp++){
                      polygon4c(0x0000, poly2[st2[lp]], color[st2[lp]] );
                      polygon4c(0x0000, poly[st[lp]], color[st[lp]] );
                    }
                    draw_end(DBmode,&first,quick);
                    draw_start();
                }
            }

    } /* for */
    return;         /* EXIT */
}

void clrscrn(void)
{
  short array[8];

/* ploygon4c (clear) ------------------------------------------------- */
    array[0] =    0; array[1] =    0;
    array[2] =  639; array[3] =    0;
    array[4] =  639; array[5] =  239;
    array[6] =    0; array[7] =  239;
    polygon4c(0x0000, array, 0x0010);
}

void set_polygon_sample(short array[N][12], short point[N][3],
                                        short coordIndex[N][5], short _n_point)
{

#define _DIV 20

  short i;
  short loc = 0;

    for(i=0; i<_n_point; i++){

      /* X */
      array[i][loc++] = point[ coordIndex[i][0] ][0] / _DIV;
      /* Y */
```

```
        array[i][loc++] = point[ coordIndex[i][0] ][1] / _DIV;
        /* Z */
        array[i][loc++] = point[ coordIndex[i][0] ][2] / _DIV;

        /* X */
        array[i][loc++] = point[ coordIndex[i][1] ][0] / _DIV;
        /* Y */
        array[i][loc++] = point[ coordIndex[i][1] ][1] / _DIV;
        /* Z */
        array[i][loc++] = point[ coordIndex[i][1] ][2] / _DIV;

        /* X */
        array[i][loc++] = point[ coordIndex[i][2] ][0] / _DIV;
        /* Y */
        array[i][loc++] = point[ coordIndex[i][2] ][1] / _DIV;
        /* Z */
        array[i][loc++] = point[ coordIndex[i][2] ][2] / _DIV;

        /* X */
        array[i][loc++] = point[ coordIndex[i][3] ][0] / _DIV;
        /* Y */
        array[i][loc++] = point[ coordIndex[i][3] ][1] / _DIV;
        /* Z */
        array[i][loc++] = point[ coordIndex[i][3] ][2] / _DIV;

        loc = 0;

     }
}


void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
```

```
unsigned short yw  = 240;
unsigned short vc  = 262;

   outport(_Q2SYSR, 0x4080);     /* Initilaize Draw/Display */
   outport(_Q2SRCR, 0xfe80);     /* Clear SR register       */

   /* Set InterFace Control Registers */
   outport(_Q2IER , 0x0000);
   outport(_Q2MEMR, 0x0031);
   outport(_Q2DSMR, 0x0005);
   outport(_Q2DSMR2,0x0000);
   outport(_Q2REMR, 0x0041);
   outport(_Q2IEMR, 0x0000);

   /* Set Memory Control Regisers */
   outport(_Q2DSX,       xw);    /* Display size of x                   */
   outport(_Q2DSY,       yw);    /* Display size of y                   */
   outport(_Q2DSA0,  0x0000);    /* Frame buffer 0 area start address(H) */
   outport(_Q2DSA1,  0x0010);    /* Frame buffer 1 area start address(H) */
   outport(_Q2DLSAH, 0x0019);    /* Display list area start address(H)   */
   outport(_Q2DLSAL, 0x0000);    /* Display list area start address(L)   */
   outport(_Q2SSAR,  0x0008);    /* Color area sorce start address(H)    */
   outport(_Q2WSAR,  0x001F);    /* Work area start address(H)           */
   outport(_Q2DMASH, 0x0000);    /* DMA transfer start address(H)        */
   outport(_Q2DMASL, 0x0000);    /* DMA transfer start address(L)        */
   outport(_Q2DMAWL, 0x0000);    /* DAM transfer word                    */

   /* Display Contral Registers */
   outport(_Q2HDS,     hsw+xs-11   );
   outport(_Q2HDE,     hsw+xs-11+xw);
   outport(_Q2VDS,     ys-2        );
   outport(_Q2VDE,     ys-2+yw     );
   outport(_Q2HSW,     hsw-1       );
   outport(_Q2HC,      hc-1        );
   outport(_Q2VSP,     vc-vsw-1    );
   outport(_Q2VC,      vc-1        );
   outport(_Q2DOR,     0x0000);
   outport(_Q2DOG_DOB, 0x007C);
   outport(_Q2CDR,     0x00FC);
   outport(_Q2CDG_CDB, 0xFCFC);
```

```
     /* Input Data Control Registers */
     outport(_Q2ISAH, 0x0000);
     outport(_Q2ISAL, 0x0000);
     outport(_Q2IDSX, 0x0000);
     outport(_Q2IDSY, 0x0000);
     outport(_Q2IDE,  0x0000);

     switch(DBmode) {                       /* Enable Draw/Display */
          case 0:
               outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
               break;
          case 1:
               outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
               break;
          default:
               outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
     }
}


long sin_t(short d)
{
     d%=360;
     if(d>=  0 && d< 90) return  sin_table[    d];
     if(d>= 90 && d<180) return  sin_table[180-d];
     if(d>=180 && d<270) return -sin_table[d-180];
                         return -sin_table[360-d];
}

long cos_t(short d)
{
     d%=360;
     if(d>=  0 && d< 90) return  sin_table[90 -d];
     if(d>= 90 && d<180) return -sin_table[d -90];
     if(d>=180 && d<270) return -sin_table[270-d];
                         return  sin_table[d-270];
}
```

```
void rotate_a_rectangle( short array[12], short angle, short center, short dx,
short dy, short dz )
{
  short i;
  short tx, ty, tz;
     switch( center ){
       case Z:
          for(i=0;i<12;i+=3) {
               tx =(short)  (( (array[i]-dx) * cos_t(angle)-  (array[i+1]-dy) *
sin_t(angle) )>>13);
               ty =(short)  (( (array[i]-dx) * sin_t(angle)+  (array[i+1]-dy) *
cos_t(angle) )>>13);
               array[i  ] =  tx+dx;
               array[i+1] =  ty+dy;
          }
          break;
       case Y:
          for(i=0;i<12;i+=3) {
               tz =(short)  (( (array[i+2]-dz) * cos_t(angle)-  (array[i]-dx) *
sin_t(angle) )>>13);
               tx =(short)  (( (array[i+2]-dz) * sin_t(angle)+  (array[i]-dx) *
cos_t(angle) )>>13);
               array[i+2] =  tz+dz;
               array[i  ] =  tx+dx;
          }
          break;
       case X:
          for(i=0;i<12;i+=3) {
               ty =(short)  (( (array[i+1]-dy) * cos_t(angle)-  (array[i+2]-dz) *
sin_t(angle) )>>13);
               tz =(short)  (( (array[i+1]-dy) * sin_t(angle)+  (array[i+2]-dz) *
cos_t(angle) )>>13);
               array[i+1] =  ty+dy;
               array[i+2] =  tz+dz;
          }
          break;
     }
}

void convert_d3_into_d2( short array[12], short dx, short dy, short dz, short
_z_size )
{
```

```
  short x0, y0;
  short x1, y1;
  short x2, y2;
  short x3, y3;

     x0 = (short)(array[ 0] * ((long)_z_size - array[ 2] - dz ) / _z_size);
     y0 = (short)(array[ 1] * ((long)_z_size - array[ 2] - dz ) / _z_size);
     x1 = (short)(array[ 3] * ((long)_z_size - array[ 5] - dz ) / _z_size);
     y1 = (short)(array[ 4] * ((long)_z_size - array[ 5] - dz ) / _z_size);
     x2 = (short)(array[ 6] * ((long)_z_size - array[ 8] - dz ) / _z_size);
     y2 = (short)(array[ 7] * ((long)_z_size - array[ 8] - dz ) / _z_size);
     x3 = (short)(array[ 9] * ((long)_z_size - array[11] - dz ) / _z_size);
     y3 = (short)(array[10] * ((long)_z_size - array[11] - dz ) / _z_size);

     array[0] = x0+dx; array[1] = y0+dy;
     array[2] = x1+dx; array[3] = y1+dy;
     array[4] = x2+dx; array[5] = y2+dy;
     array[6] = x3+dx; array[7] = y3+dy;
}

void z_sort( short array[N][12], short st[N], short n )
{
  short i,j,k;
  short max;
  short z[N];

     for(i=0; i<n; i++){
          z[i] = 0;
          for (k=_Z1;k<=_Z4;k+=3){
               z[i] += array[i][k];
          }
          z[i] /= 4;
     }

     st[0] = 0;

     for(i=1; i<n; i++){
          for(j=0; j<i && z[st[j]]>z[i];j++);
          for(k=i; k>j;                 k--) {
               st[k]=st[k-1];
```

```
                                                    }
            st[j]=i;
      }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.14 Source File of sample9.c

```
/*
                    Q2 Display List / SHC sample program (9)

           ( Move 'Solid' Polygon with pattern which is in work area )

                        Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;


#define    PI               3.14159


#define X_WIDTH 1024


#define _Q2_REMR_REG_NO 0x006
#define _Q2_RSAR_REG_NO 0x04C


void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------- */

void  ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */


void main(void)
```

```
{
     long  count;
     short array[8];
     short DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 1;
     first  = ON;
     quick  = OFF;

     init_BSC();                    /* Initialize SH7709 bus state controller */

     ginit(DBmode);                 /* Initialize Q2 */


/* Clear work area */
{
  unsigned short x,y;
  unsigned short *wk  = VU_SHORT (0x1F0000L | UGMBASE);
     for(y=1;y<=240;y++){
          for(x=1;x<=X_WIDTH/16;x++){
               *wk = 0x0000;
               ++wk;
          }
     }
}


/* Transfer Font Data "漢" at work area ---------------------------------- */
{
  unsigned short work_area_width = X_WIDTH / 16;
  unsigned long  SrcAdr;                        /* Source Address */
  short i,k,m;
  unsigned short *address;
  unsigned short font_pattern[16]={
```

```
        0x8112, 0x4ffe, 0x2110, 0x0000,
        0x87fc, 0x4444, 0x1444, 0x17fc,
        0x2040, 0x27fc, 0xc040, 0x4ffe,
        0x40a0, 0x4110, 0x4608, 0x5806
            };

    SrcAdr = inport(_Q2WSAR), SrcAdr <<= 16, SrcAdr += UGMBASE;

    SrcAdr += work_area_width * 90;
    address = (unsigned short *)SrcAdr;

    for(m=1;m<8;m++){
        for(i=0;i<16;i++){
            for(k=0;k<work_area_width;k++){
                *address++  = font_pattern[i];
            }
        }
    }
}

    init_start();               /* Initialize Transfer Procedure */
    draw_start();               /* Start Transfer Display List to UGM */
    sclip(0x0000, 639,239);
    lcofs(0x0000,0,0);

    for( count = 0 ; count < 5 ; count++ ){

    /* MOVE */
      short kj = 50;
      double rad;
      short d,i;

        for(d=0;d<=360;d+=5) {

        /* Job No.1 : clear screen */
            array[0] =    0;  array[1] =    0;
            array[2] =  639;  array[3] =    0;
            array[4] =  639;  array[5] =  239;
            array[6] =    0;  array[7] =  239;
            polygon4c(0x0000, array, 0x0606);
```

```
        /* Set drawing start address in the RSAR */
             if ( (inport( _Q2SR ) & 0x0100) != 0 ) {
                  wpr( 0x0000, _Q2_RSAR_REG_NO, inport( _Q2DSA0 ) );
             } else {
                  wpr( 0x0000, _Q2_RSAR_REG_NO, inport( _Q2DSA1 ) );
             }

        /* Enable the RSAR */
             wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) | 0x8000 );


        /* Job No.2 : draw two solid rectangles */
             array[0] =   -50;      array[1] =   -(60);
             array[2] =    50;      array[3] =   -(60);
             array[4] =    50;      array[5] =   -(-60);
             array[6] =   -50;      array[7] =   -(-60);

             rad = PI * d / 180;
             for(i=0;i<=6;i+=2) {
                  short tx =(short)  (array[i] * cos(rad)-  array[i+1] *
sin(rad));
                  short ty =(short)  (array[i] * sin(rad)+  array[i+1] *
cos(rad));
                  array[i]   =  tx;
                  array[i+1] =  ty;
             }

             lcofs(0x0000, kj,100);
             polygon4c(0x0001,array,0xF800);           /* POLYGON4C */

             lcofs(0x0000, 320-kj,100);
             polygon4c(0x0001,array,0xF800);           /* POLYGON4C */

             lcofs(0x0000, 0,0);


        /* Disable the RSAR */
             wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) & 0x7fff );
```

```
                kj+=5;

            /* Drawing start */
                draw_end(DBmode,&first,quick);
                draw_start();

            } /* for */

        } /* for */

        return;         /* EXIT */
}


void ginit(short DBmode)
{                       /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

      outport(_Q2SYSR, 0x4080);      /* Initilaize Draw/Display */
      outport(_Q2SRCR, 0xfe80);      /* Clear SR register       */

      /* Set InterFace Control Registers */
      outport(_Q2IER ,  0x0000);
      outport(_Q2MEMR,  0x0031);
      outport(_Q2DSMR,  0x0005);
      outport(_Q2DSMR2, 0x0000);
      outport(_Q2REMR,  0x0041);
      outport(_Q2IEMR,  0x0000);

      /* Set Memory Control Regisers */
      outport(_Q2DSX,       xw);     /* Display size of x                    */
```

```
outport(_Q2DSY,       yw);     /* Display size of y                    */
outport(_Q2DSA0,  0x0000);     /* Frame buffer 0 area start address(H) */
outport(_Q2DSA1,  0x0010);     /* Frame buffer 1 area start address(H) */
outport(_Q2DLSAH, 0x0019);     /* Display list area start address(H)   */
outport(_Q2DLSAL, 0x0000);     /* Display list area start address(L)   */
outport(_Q2SSAR,  0x0008);     /* Color area sorce start address(H)    */
outport(_Q2WSAR,  0x001F);     /* Work area start address(H)           */
outport(_Q2DMASH, 0x0000);     /* DMA transfer start address(H)        */
outport(_Q2DMASL, 0x0000);     /* DMA transfer start address(L)        */
outport(_Q2DMAWL, 0x0000);     /* DAM transfer word                    */

/* Display Contral Registers */
outport(_Q2HDS,     hsw+xs-11   );
outport(_Q2HDE,     hsw+xs-11+xw);
outport(_Q2VDS,     ys-2        );
outport(_Q2VDE,     ys-2+yw     );
outport(_Q2HSW,     hsw-1       );
outport(_Q2HC,      hc-1        );
outport(_Q2VSP,     vc-vsw-1    );
outport(_Q2VC,      vc-1        );
outport(_Q2DOR,     0x0000);
outport(_Q2DOG_DOB, 0x007C);
outport(_Q2CDR,     0x00FC);
outport(_Q2CDG_CDB, 0xFCFC);

/* Input Data Control Registers */
outport(_Q2ISAH, 0x0000);
outport(_Q2ISAL, 0x0000);
outport(_Q2IDSX, 0x0000);
outport(_Q2IDSY, 0x0000);
outport(_Q2IDE,  0x0000);

switch(DBmode) {                     /* Enable Draw/Display */
     case 0:
          outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
          break;
     case 1:
          outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
          break;
     default:
```

```
                outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
     }
}

#include "Q2SDl.inc"
#include "MS7709.inc"
```

5.2.15 Source File of tst_brk.c

```
/*
                        Q2 Display List / SHC sample program
                            (Draw hexagon with breaking)

                              Only HD64413A
                              Copyright(c) Hitachi Ltd. 1999

            The break drawing is based on 5 operations ( From (1) to (5) ).
            Kind of double buffer control is manual display change.

*/
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"

#define  MAX_WORD  512

#define  DISPLIST2 0x005000L        /* Store dislpay list for Back drawing  */

#define _Q2_REMR_REG_NO 0x006
#define _Q2_RTNH_REG_NO 0x04a
#define _Q2_RTNL_REG_NO 0x04b
#define _Q2_RSAR_REG_NO 0x04c


unsigned short *DISPLIST_ptr;



void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short _draw_start(void); /* Initialize Display List Address pointer */
void  change_com_buffer(void);
void  ginit(short DBmode);             /* Graphics System Open  (Q2 Initialize) */
short _draw_end(short DBmode,char *first,char quick); /* Display List Execution */



void main(void)
```

```
{
     long  count;
     short array[MAX_WORD], poly[MAX_WORD];
     short DBmode;
     short fill_color = 0x0404;
     unsigned long display_list_adr;
     short adr_h, adr_l;
     short color = 0;
     short draw_buffer_area, display_buffer_area;
     unsigned long displying_area_address;
     unsigned long drawing_area_address;
     short st;
     short temp;
     unsigned long  CMD_StartAddress;

     unsigned short move_x,  move_y;
     unsigned short lcofs_x, lcofs_y;
     unsigned short uclip_xmin, uclip_ymin, uclip_xmax, uclip_ymax;
     unsigned short sclip_xmin, sclip_ymin;
     unsigned short rtn_address_h, rtn_address_l;
     unsigned short rsae_bit;

/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;

     DBmode = 2;
     first  = ON;
     quick  = OFF;

     init_BSC();                    /* Initialize SH7709 bus state controller */

     ginit(DBmode);                 /* Initialize Q2 */

     /* Deside displaying area and drawing area */
{
     if( (inport( _Q2SR ) & 0x0100) != 0 ) {
```

```
            displying_area_address = _Q2DSA1;
            drawing_area_address   = _Q2DSA0;
      } else {
            displying_area_address = _Q2DSA0;
            drawing_area_address   = _Q2DSA1;
      }

      draw_buffer_area    = inport( drawing_area_address   );
      display_buffer_area = inport( displying_area_address );
}


/* ===  Create display list for back drawing  ========================== */

      /* Set display list start address */
      DISPLIST_ptr = (volatile unsigned short * const)(DISPLIST2 | UGMBASE);

      vbkem(0x0000, 0,0);              /* vbkem */
      sclip(0x0000, 639,239);          /* sclip */
      lcofs(0x0000, 0,0);              /* lcofs */

      /* Set drawing start address */
      wpr( 0x0000, _Q2_RSAR_REG_NO, display_buffer_area );

      /* Enable the RSAR */
      wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) | 0x8000 );

      for (color=0; color<127; color++){

            lcofs(0x0000, color,color);

            fill_color = ((color & 0xf) << 8) | (color & 0xf);

            poly[ 0] =  40;      poly[ 1] =  10;
            poly[ 2] =  130;     poly[ 3] =  poly[1];
            poly[ 4] =  poly[2]; poly[ 5] =  100;
            poly[ 6] =  poly[0]; poly[ 7] =  poly[5];
            polygon4c(0x0000, poly, fill_color);             /* polygon4c */

            vbkem(0x0000, 0,0);
```

```
    } /* color */

    /* Disable the RSAR */
    wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) & 0x7fff );

    trap(0);                          /* End of display list for back drawing */

    /* Change DLSAR    */
    CMD_StartAddress = DISPLIST2;
    adr_h = (CMD_StartAddress >> 16L) & 0xffff;
    adr_l =  CMD_StartAddress         & 0xffff;
    outport(_Q2DLSAH,adr_h);
    outport(_Q2DLSAL,adr_l);

    /* Transfer command from internal command buffer to UGM */
    outport(_Q2SRCR,0x0400);            /* Clear TRA bit    */
    outport(_Q2SYSR,0x2180);            /* Start drawing    */

/* ===  End of create display list for back drawing  ================= */


/* ===  Start drawing with breaking  ================================== */
    init_start();                       /* Initialize Transfer Procedure */


while(1){

  /*_/_/_/_/_/  (1). The operation of waiting internal update  _/_/_/_/_/*/
    outport( _Q2SRCR, 0x4000 );                       /* Clear FRM bit */
    while( st=inport(_Q2SR), (st&0x4000)==0);         /* FRM = 1 ? */


  /*_/_/_/_/_/  (2). The operation of breaking back drawing  _/_/_/_/_/*/
    outport(_Q2SRCR,0x0080);                         /* Clear BRK bit */
    outport(_Q2SYSR,0x2480);                         /* Stop drawing  */

    if ( st=inport(_Q2SR), (st&0x0400) != 0 ){       /* Check TRA bit */
        break;
    } else {
```

```
        while( st=inport(_Q2SR), (st&0x0080)==0);    /* BRK = 1 ? */

        /* Get current pointer position */
        move_x = inport(_Q2XC);
        move_y = inport(_Q2YC);

        /* Get local offset */
        lcofs_x = inport(_Q2XO);
        lcofs_y = inport(_Q2YO);

        /* Get user clipping renge */
        uclip_xmin = inport(_Q2UXMIN);
        uclip_ymin = inport(_Q2UYMIN);
        uclip_xmax = inport(_Q2UXMAX);
        uclip_ymax = inport(_Q2UYMAX);

        /* Get system clipping renge */
        sclip_xmin = inport(_Q2SXMAX);
        sclip_ymin = inport(_Q2SYMAX);

        /* Get return address */
        rtn_address_h = inport(_Q2RTNH);
        rtn_address_l = inport(_Q2RTNL);

        /* Get RSAE bit */
        rsae_bit = inport(_Q2REMR) & 0x8000;

        /* Get command status */
        display_list_adr = inport(_Q2CSTH);
        display_list_adr <<= 16;
        display_list_adr |= inport(_Q2CSTL);
        adr_h = (short)( (display_list_adr >> 13) & 0x03FFL);
        adr_l = (short)(  display_list_adr        & 0x1FFFL);
    } /* else */


/*_/_/_/_/_/  (3). The oparetion of something drawing _/_/_/_/_/*/
/*_/_/_/_/_/          while breaking back drawing     _/_/_/_/_/*/

    _draw_start();
```

```
    /* Set drawing start address */
    wpr( 0x0000, _Q2_RSAR_REG_NO, display_buffer_area );

    /* Enable the RSAR */
    wpr( 0x0000, _Q2_REMR_REG_NO, inport(_Q2REMR) | 0x8000 );


    lcofs(0x0000, 0,0);

    /* ploygon4c ( clear screen ) */
    array[0] =    0;  array[1] =    0;      /* Left  Up   */
    array[2] =  639;  array[3] =    0;      /* Right Up   */
    array[4] =  639;  array[5] =  239;      /* Left  Down */
    array[6] =    0;  array[7] =  239;      /* Right Down */
    polygon4c(0x0000, array,  0x0000);      /* polygon4c  */

    array[0] =    0;  array[1] =    0;      /* Left  Up   */
    array[2] =  159;  array[3] =    0;      /* Right Up   */
    array[4] =  159;  array[5] =  119;      /* Left  Down */
    array[6] =    0;  array[7] =  119;      /* Right Down */
    polygon4c(0x0000, array,  0xffff);      /* polygon4c  */

    array[0] =  160; array[1] =  120;      /* Left  Up   */
    array[2] =  319; array[3] =  120;      /* Right Up   */
    array[4] =  319; array[5] =  239;      /* Left  Down */
    array[6] =  160; array[7] =  239;      /* Right Down */
    polygon4c(0x0000, array, 0xffff);      /* polygon4c  */


    /* Drawing start with chacking TRA bit */
    _draw_end(DBmode,&first,OFF);


/*_/_/_/_/_/  (4). The operation of continuity back drawing  _/_/_/_/_/*/

    _draw_start();

    /* Store RSAE bit */
    wpr( 0x0000, _Q2_REMR_REG_NO, rsae_bit | (inport(_Q2REMR) & 0x7fff) );
```

```
      /* Store return address */
      wpr( 0x0000, _Q2_RTNH_REG_NO, rtn_address_h );
      wpr( 0x0000, _Q2_RTNL_REG_NO, rtn_address_l );

      /* Store user clipping renge */
      uclip( 0x0000, uclip_xmin,uclip_ymin, uclip_xmax,uclip_ymax );

      /* Store system clipping renge */
      sclip( 0x0000, sclip_xmin,sclip_ymin );

      /* Store local offset before move command */
      lcofs( 0x0000, lcofs_x,lcofs_y );

      /* Store current pointer position */
      move( 0x0000, move_x,move_y );

      /* Go to next display list */
      jump(0x0000, adr_h,adr_l);

      /* Drawing start without chacking TRA bit */
      _draw_end(DBmode,&first,ON);


  /*_/_/_/_/_/  (5). The operation of frame change  _/_/_/_/_/*/

      /* Set display start address */
      outport( displying_area_address, display_buffer_area );

      /* Change display buffer */
      temp                = draw_buffer_area;
      draw_buffer_area    = display_buffer_area;
      display_buffer_area = temp;

} /* while(1) */

/* ===  End of drawing with breaking  ====================================== */

      return;         /* EXIT */
}
```

```
void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

     outport(_Q2SYSR, 0x4080);          /* Initilaize Draw/Display */
     outport(_Q2SRCR, 0xfe80);          /* Clear SR register       */

     /* Set InterFace Control Registers */
     outport(_Q2IER ,  0x0000);
     outport(_Q2MEMR,  0x0031);
     outport(_Q2DSMR,  0x0005);
     outport(_Q2DSMR2, 0x0000);
     outport(_Q2REMR,  0x0041);
     outport(_Q2IEMR,  0x0000);

     /* Set Memory Control Regisers */
     outport(_Q2DSX,       xw);       /* Display size of x                    */
     outport(_Q2DSY,       yw);       /* Display size of y                    */
     outport(_Q2DSA0,  0x0000);       /* Frame buffer 0 area start address(H) */
     outport(_Q2DSA1,  0x0010);       /* Frame buffer 1 area start address(H) */
     outport(_Q2DLSAH, 0x0019);       /* Display list area start address(H)   */
     outport(_Q2DLSAL, 0x0000);       /* Display list area start address(L)   */
     outport(_Q2SSAR,  0x0008);       /* Color area sorce start address(H)    */
     outport(_Q2WSAR,  0x001F);       /* Work area start address(H)           */
     outport(_Q2DMASH, 0x0000);       /* DMA transfer start address(H)        */
     outport(_Q2DMASL, 0x0000);       /* DMA transfer start address(L)        */
     outport(_Q2DMAWL, 0x0000);       /* DAM transfer word                    */

     /* Display Contral Registers */
     outport(_Q2HDS,     hsw+xs-11   );
     outport(_Q2HDE,     hsw+xs-11+xw);
```

```
      outport(_Q2VDS,      ys-2        );
      outport(_Q2VDE,      ys-2+yw     );
      outport(_Q2HSW,      hsw-1       );
      outport(_Q2HC,       hc-1        );
      outport(_Q2VSP,      vc-vsw-1    );
      outport(_Q2VC,       vc-1        );
      outport(_Q2DOR,      0x0000);
      outport(_Q2DOG_DOB, 0x007C);
      outport(_Q2CDR,      0x00FC);
      outport(_Q2CDG_CDB, 0xFCFC);

      /* Input Data Control Registers */
      outport(_Q2ISAH, 0x0000);
      outport(_Q2ISAL, 0x0000);
      outport(_Q2IDSX, 0x0000);
      outport(_Q2IDSY, 0x0000);
      outport(_Q2IDE,  0x0000);

      switch(DBmode) {                      /* Enable Draw/Display */
           case 0:
                outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
                break;
           case 1:
                outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
                break;
           default:
                outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
      }
}

#include "Q2SDl.inc"


short _draw_start(void)             /* Initialize Display List Address Pointer */
{
  long  CMD_StartAddress;
      if(DrawBuffer == 0)           /* Display List Start Address */
           CMD_StartAddress = (long)(DISPLIST1+UGMBASE);     /* Execute Buffer0 ->
Transfer Display List to Buffer1 */
      else
```

```
          CMD_StartAddress = (long)(DISPLIST0+UGMBASE);     /* Execute Buffer1 ->
Transfer Display List to Buffer0 */

     DISPLIST_ptr = (unsigned short *)CMD_StartAddress;

     return( 0 );
}


short _draw_end(short DBmode,char *first,char quick)    /* Execute Display List */
{
  unsigned short st, ah, al;
  unsigned long CMD_StartAddress;

     trap(0);                                   /* Add 'trap' command */

     outport(_Q2SRCR,0x0400);                   /* Clear TRA bit    */

     if(DrawBuffer == 0) {
          DrawBuffer = 1;
          CMD_StartAddress = DISPLIST1;
     }
     else {
          DrawBuffer = 0;
          CMD_StartAddress = DISPLIST0;
     }
     ah = (CMD_StartAddress >> 16L) & 0xffff;
     al =  CMD_StartAddress & 0xffffL;
     outport(_Q2DLSAH,ah);                          /* Change DLSAR */
     outport(_Q2DLSAL,al);

     /* Renddering start */
     outport(_Q2SYSR,0x2180);

     /* Check RS bit */
     while( st=inport(_Q2SYSR), (st&0x0100)!=0 );   /* RS = 0 ? */

     /* Manual Change */
     if(quick==OFF)
          while(1){
```

```
            st=inport(_Q2SR);
            if ((st & 0x0400)!=0) break;             /* Check TRA bit */
            if ((st & 0x0200)!=0) {
                 outport(_Q2SRCR,0x0200);            /* Clear CSF bit    */
                 break; /* CHeck CSF bit */
            }
       }

       return( 0 );
}

#include "MS7709.inc"
```

5.2.16 Source File of elps.c

```
/*
                        Q2 Display List / SHC sample program
                                ( Draw filled ellipse)

                          Copyright(c) Hitachi Ltd. 1999
*/
#include <machine.h>
#include <stdio.h>
#include <math.h>
#include "Q2SD_REG.h"
#include "Q2SD_mac.h"


unsigned short *DISPLIST_ptr;

void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
/* ------------------------------------------------------ */

void  _fill_ellipse( short xc, short yc, short rx, short ry, short color );
void  clrscrn(void);
void  ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */



void main(void)
{
     long  count;
     short DBmode;
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/
     char  first,quick;
```

```
      DBmode = 2;
      first  = ON;
      quick  = OFF;

      init_BSC();                    /* Initialize SH7709 bus state controller */

      ginit(DBmode);                 /* Initialize Q2 */
      init_start();                  /* Initialize Transfer Procedure */
      draw_start();                  /* Start Transfer Display List to UGM */
      sclip(0x0000, 639,239);
      lcofs(0x0000,0,0);

for( count = 1 ; count < 10 ; count++ ){
{
  short lp;
  short rx, ry;
  short x_center, y_center;

/* Define ellipse center */
      x_center = 100, y_center = 100;

      for (lp=1; lp<=100; lp+=count){

           clrscrn();

/* Draw ellipse   ---------------------------- */
           rx = lp;
           ry = lp/2;
           _fill_ellipse( x_center,y_center, rx,ry, 0xFFFF );

           draw_end(DBmode,&first,quick);
           draw_start();
      }
}

{
  short lp;
  short rx, ry;
  short x_center, y_center;
```

```
/* Define ellipse center */
     x_center = 100, y_center = 100;

     for (lp=1; lp<=100; lp+=count){

          clrscrn();

/* Draw ellipse   ---------------------------- */
          rx = lp/2;
          ry = lp;
          _fill_ellipse( x_center,y_center, rx,ry, 0xFFFF );

          draw_end(DBmode,&first,quick);
          draw_start();
     }
}
} /* for */

     return;         /* EXIT */
}


void clrscrn(void)
{
     short array[8];

/* ploygon4c (clear) ------------------------------------------------------ */
     array[0] =    0; array[1] =    0;
     array[2] =  639; array[3] =    0;
     array[4] =  639; array[5] =  239;
     array[6] =    0; array[7] =  239;
     polygon4c(0x0000, array, 0x0000);
}


/* Creates display list to draw ellipse with filling */
void _fill_ellipse( short xc, short yc, short rx, short ry, short color )
{
     if( rx<=0 || ry<=0 ) return;
```

```
if( rx > ry ){
  short x = rx;
  short r = rx;
  short y = 0;
  short poly[4];      /* 1st point  = ( poly[0], poly[1] ) */
                            /* 2nd point  = ( poly[2], poly[3] ) */

     /* Draw ellipse in work area */
     while( x >= y ) {
       short x1=(short)( (long)x * ry / rx );
       short y1=(short)( (long)y * ry / rx );

          poly[0] = xc+x;  poly[2] = xc-x;
          poly[1] = poly[3] = yc+y1;
          line( 0x0000, color, 2, poly );
          poly[1] = poly[3] = yc-y1;
          line( 0x0000, color, 2, poly );

          poly[0] = xc+y;  poly[2] = xc-y;
          poly[1] = poly[3] = yc+x1;;
          line( 0x0000, color, 2, poly );
          poly[1] = poly[3] = yc-x1;;
          line( 0x0000, color, 2, poly );

          if ( (r -= (y++ << 1)-1) < 0) r += (x-- - 1) << 1;
     } /* while */
} /* if */
     else {
       short x = ry;
       short r = ry;
       short y = 0;
       short poly[4];     /* 1st point  = ( poly[0], poly[1] ) */
                                 /* 2nd point  = ( poly[2], poly[3] ) */

          /* Draw ellipse in work area */
          while( x >= y ) {
            short x1=(short)( (long)x * rx / ry );
            short y1=(short)( (long)y * rx / ry );
```

```
                    poly[0] = xc+x1; poly[2] = xc-x1;
                    poly[1] = poly[3] = yc+y;
                    line( 0x0000, color, 2, poly );
                    poly[1] = poly[3] = yc-y;
                    line( 0x0000, color, 2, poly );

                    poly[0] = xc+y1; poly[2] = xc-y1;
                    poly[1] = poly[3] = yc+x;
                    line( 0x0000, color, 2, poly );
                    poly[1] = poly[3] = yc-x;
                    line( 0x0000, color, 2, poly );

                    if ( (r -= (y++ << 1)-1) < 0) r += (x-- - 1) << 1;
                } /* while */
            } /* else */
} /* _fill_ellipse */


void ginit(short DBmode)
{                   /* Q2SD */
  unsigned short hsw =  64;
  unsigned short xs  = 131;
  unsigned short xw  = 640;
  unsigned short hc  = 910;

  unsigned short vsw =   3;
  unsigned short ys  =  16;
  unsigned short yw  = 240;
  unsigned short vc  = 262;

     outport(_Q2SYSR, 0x4080);     /* Initilaize Draw/Display */
     outport(_Q2SRCR, 0xfe80);     /* Clear SR register       */

     /* Set InterFace Control Registers */
     outport(_Q2IER ,  0x0000);
     outport(_Q2MEMR,  0x0031);
     outport(_Q2DSMR,  0x0005);
     outport(_Q2DSMR2, 0x0000);
     outport(_Q2REMR,  0x0041);
     outport(_Q2IEMR,  0x0000);
```

```
/* Set Memory Control Regisers */
outport(_Q2DSX,       xw);      /* Display size of x                    */
outport(_Q2DSY,       yw);      /* Display size of y                    */
outport(_Q2DSA0,  0x0000);      /* Frame buffer 0 area start address(H) */
outport(_Q2DSA1,  0x0010);      /* Frame buffer 1 area start address(H) */
outport(_Q2DLSAH, 0x0019);      /* Display list area start address(H)   */
outport(_Q2DLSAL, 0x0000);      /* Display list area start address(L)   */
outport(_Q2SSAR,  0x0008);      /* Color area sorce start address(H)    */
outport(_Q2WSAR,  0x001F);      /* Work area start address(H)           */
outport(_Q2DMASH, 0x0000);      /* DMA transfer start address(H)        */
outport(_Q2DMASL, 0x0000);      /* DMA transfer start address(L)        */
outport(_Q2DMAWL, 0x0000);      /* DAM transfer word                    */

/* Display Contral Registers */
outport(_Q2HDS,     hsw+xs-11   );
outport(_Q2HDE,     hsw+xs-11+xw);
outport(_Q2VDS,     ys-2        );
outport(_Q2VDE,     ys-2+yw     );
outport(_Q2HSW,     hsw-1       );
outport(_Q2HC,      hc-1        );
outport(_Q2VSP,     vc-vsw-1    );
outport(_Q2VC,      vc-1        );
outport(_Q2DOR,     0x0000);
outport(_Q2DOG_DOB, 0x007C);
outport(_Q2CDR,     0x00FC);
outport(_Q2CDG_CDB, 0xFCFC);

/* Input Data Control Registers */
outport(_Q2ISAH, 0x0000);
outport(_Q2ISAL, 0x0000);
outport(_Q2IDSX, 0x0000);
outport(_Q2IDSY, 0x0000);
outport(_Q2IDE,  0x0000);

switch(DBmode) {                    /* Enable Draw/Display */
     case 0:
          outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
          break;
     case 1:
```

```
                outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
                break;
           default:
                outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
     }
}

#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.17 Source File of v_wind.c

```
/*_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/

               Video window ( window size is 320*240 pixel )

                      Copyright(c) Hitachi Ltd. 1999

  _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/*/


#include "Q2SD_REG.h"
#include "Q2SD_mac.h"

unsigned short *DISPLIST_ptr;

void  init_BSC(void);    /* Initialize SH7709 bus state controller */
void  init_start(void);  /* Initialize Display List Double Buffering (Only 1st) */
short draw_start(void);  /* Initialize Display List Address pointer */
short draw_end(short DBmode,char *first,char quick);  /* Display List Execution */
void  change_com_buffer(void);
void  ginit(short DBmode);     /* Graphics System Open  (Q2 Initialize) */

void set_up_video_window( void );
void Stop_the_video_function( void );


#define OFF          0
#define ON           1


static void clrscrn(void);


void main(void)
{
  short array[8];
  long  count_n;
  char  first = ON;
  char  quick = OFF; short  DBmode = 1;
```

```
/*
 DBmode: 0 ... Auto Change
 DBmode: 1 ... Auto Renderring
 DBmode: 2 ... Manual Change
*/

     init_BSC();                  /* Initialize SH7709 bus state controller */

     ginit(DBmode);               /* Initialize Q2 */

     set_up_video_window();       /* Setup Video window */

     init_start();                /* Initialize Transfer Procedure */
     draw_start();                /* Start Transfer Display List to UGM */
     sclip(0x0000, 639,239);
     lcofs(0x0000,0,0);


     for( count_n = 0 ; count_n < 500 ; count_n++ ){

          /* ploygon4c (clear)  */
          array[0] =    0;  array[1] =    0;
          array[2] =  639;  array[3] =    0;
          array[4] =  639;  array[5] =  239;
          array[6] =    0;  array[7] =  239;
          polygon4c(0x0000, array, 0x0000);

          draw_end(1,&first,quick);
          draw_start();
     }

     Stop_the_video_function();

     return;         /* EXIT */
}


void ginit(short DBmode)
{                    /* Q2SD */
  unsigned short hsw =  64;
```

```
unsigned short xs  = 131;
unsigned short xw  = 640;
unsigned short hc  = 910;

unsigned short vsw =   3;
unsigned short ys  =  16;
unsigned short yw  = 240;
unsigned short vc  = 262;

   outport(_Q2SYSR, 0x4080);        /* Initilaize Draw/Display */
   outport(_Q2SRCR, 0xfe80);        /* Clear SR register       */

   /* Set InterFace Control Registers */
   outport(_Q2IER ,  0x0000);
   outport(_Q2MEMR,  0x0031);
   outport(_Q2DSMR,  0x0005);
   outport(_Q2DSMR2, 0x0000);
   outport(_Q2REMR,  0x0041);
   outport(_Q2IEMR,  0x0000);

   /* Set Memory Control Regisers */
   outport(_Q2DSX,       xw);      /* Display size of x                    */
   outport(_Q2DSY,       yw);      /* Display size of y                    */
   outport(_Q2DSA0,  0x0000);      /* Frame buffer 0 area start address(H) */
   outport(_Q2DSA1,  0x0010);      /* Frame buffer 1 area start address(H) */
   outport(_Q2DLSAH, 0x0019);      /* Display list area start address(H)   */
   outport(_Q2DLSAL, 0x0000);      /* Display list area start address(L)   */
   outport(_Q2SSAR,  0x0008);      /* Color area sorce start address(H)    */
   outport(_Q2WSAR,  0x001F);      /* Work area start address(H)           */
   outport(_Q2DMASH, 0x0000);      /* DMA transfer start address(H)        */
   outport(_Q2DMASL, 0x0000);      /* DMA transfer start address(L)        */
   outport(_Q2DMAWL, 0x0000);      /* DAM transfer word                    */

   /* Display Contral Registers */
   outport(_Q2HDS,     hsw+xs-11   );
   outport(_Q2HDE,     hsw+xs-11+xw);
   outport(_Q2VDS,     ys-2        );
   outport(_Q2VDE,     ys-2+yw     );
   outport(_Q2HSW,     hsw-1       );
   outport(_Q2HC,      hc-1        );
```

```
      outport(_Q2VSP,     vc-vsw-1    );
      outport(_Q2VC,      vc-1        );
      outport(_Q2DOR,     0x0000);
      outport(_Q2DOG_DOB, 0x007C);
      outport(_Q2CDR,     0x00FC);
      outport(_Q2CDG_CDB, 0xFCFC);

      /* Input Data Control Registers */
      outport(_Q2ISAH, 0x0000);
      outport(_Q2ISAL, 0x0000);
      outport(_Q2IDSX, 0x0000);
      outport(_Q2IDSY, 0x0000);
      outport(_Q2IDE,  0x0000);

      switch(DBmode) {                    /* Enable Draw/Display */
           case 0:
                outport(_Q2SYSR, 0x2000);   /* Idle,Auto Change mode,No DMA */
                break;
           case 1:
                outport(_Q2SYSR, 0x2040);   /* Idle,Auto Renderring mode,No DMA */
                break;
           default:
                outport(_Q2SYSR, 0x2080);   /* Idle,manual change moe,No DMA */
      }
}


void set_up_video_window( void )
{
  unsigned long Video0_area_start_address = 0x07D000L;
  unsigned long Video1_area_start_address = 0x105000L;
  unsigned long Video2_area_start_address = 0x17D000L;


/* Initialize register for voide function */
{
  unsigned short temp;

      /* Hide the video window */
      temp = inport(_Q2DSMR2);
```

```
    temp &= 0xFFFE;
    outport(_Q2DSMR2, temp);                          /* VWE=0 */

    /* Stop the capture for the video window */
    temp = inport(_Q2VIMR);
    temp &= 0xFFFE;
    outport(_Q2VIMR, temp);                           /* VIE=0 */



    {
      short lp;
      for ( lp=0; lp<2; lp++ ) {
         outport( _Q2SRCR, 0x4000 );                  /* Clear FRM flag */
         while( (inport(_Q2SR) & 0x4000) == 0 );      /* Wait FRM flag */
      }
    }



    outport(_Q2HVP, 320);    /* Video window horizpntal display position (dot) */
    outport(_Q2VVP, 0);      /* Video window vertical display position   (dot) */

    /* Video0 area start address(From A22 to A10 ) */
    outport(_Q2VSAH0, Video0_area_start_address >> 16   );
    outport(_Q2VSAL0, Video0_area_start_address & 0xFC00);

    /* Video1 area start address(From A22 to A10 ) */
    outport(_Q2VSAH1, Video1_area_start_address >> 16   );
    outport(_Q2VSAL1, Video1_area_start_address & 0xFC00);

    /* Video2 area start address(From A22 to A10 ) */
    outport(_Q2VSAH2, Video2_area_start_address >> 16   );
    outport(_Q2VSAL2, Video2_area_start_address & 0xFC00);

    /* Set 'video windows size' and 'voide area size' */
    outport(_Q2VSIZEX, 320);
    outport(_Q2VSIZEY, 240);

    outport(_Q2VIMR, 0x0022); /* (VIZ4,3,2,1,0)=00001,  H = HACTIVE * 1/2, V =
VACTIVE * 1     */
                              /* ( Caution )
```

```
                                    HACTIVE(dot/HSYNC) and VACTIVE(line/VSYNC) are
defined by external caputure LSI.
                                    In this sample program requires HACTIVE=640 and
VACTIVE=240 */
                              /* VINM=0                                          */
                              /* (ODEN1,ODEN0) =(0,0), 1/60 caputuring           */
                              /* RGB=1, If RGB=1, have to set VWRY to 0.         */
                              /* VIE=0                                           */

     /* Set RGB bit */
          temp = inport(_Q2VIMR);
          outport(_Q2VIMR, temp | 0x0002);                 /* RGB =1 */

     /* Start the capture for the video stream */
          temp = inport(_Q2VIMR);
          outport(_Q2VIMR, temp | 0x0001);                 /* VIE=1 */

     /*  Set VWRY and Dispaly the video window */
          temp = inport(_Q2DSMR2);
          outport(_Q2DSMR2, temp | 0x0001);                /* VWRY=0, VWE=1 */
 }
}


void Stop_the_video_function( void )
{
  /* Stop the voide function */
  unsigned short temp;

     /* Hide the video window */
     temp = inport(_Q2DSMR2);
     temp &= 0xFFFE;
     outport(_Q2DSMR2, temp);                              /* VWE=0 */

     /* Stop the capture for the video window */
     temp = inport(_Q2VIMR);
     temp &= 0xFFFE;
     outport(_Q2VIMR, temp);                               /* VIE=0 */
}
```

```
#include "MS7709.inc"
#include "Q2SDl.inc"
```

5.2.18 Source File of init_bt.c

```
/*_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/

                     Copyright(c) Hitachi Ltd. 1999

                          Initialize for Bt829.

  _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/*/


#include "Q2SD_REG.h"
#include "Q2SD_mac.h"

void init_BSC(void);           /* Initialize SH7709 bus state controller */
void init_Bt829B(void);        /* Initialize VideoDecoder Bt829B */


/* -------------- Bt829B register address ----------------- */
#if 0
/* Area2 */
#define II2_Data         0xA8900000L
#define II2_Control      0xA8900002L
#else
/* Area5 */
#define II2_Data         0xB4900000L
#define II2_Control      0xB4900002L
#endif


/* BT829B Register Data ( Video In Size = 640 × 480 ) */
short BTdata[] = {
     0x0a, /* IFORM       (0x01) MUX3 -> MUXOUT, Auto XT, NTSC */
     0x00, /* TDEC        (0x02) */
     0x12, /* CROP        (0x03) */
     0x16, /* VDEALY_LO   (0x04) */
     0xe0, /* VACTIVE_LO  (0x05) */
     0x78, /* HDEALY_LO   (0x06) */
     0x80, /* HACVIVE_LO  (0x07) */
     0x02, /* HSCALE_HI   (0x08) */
```

```
      0xAC, /* HSCALE_LO   (0x09) */
      0xf0, /* BRIGHT      (0x0A) */
      0x20, /* CONTROL     (0x0B) */
      0xd8, /* CONTRAST_LO (0x0C) */
      0xfe, /* SAT_U_LO    (0x0D) */
      0xb4, /* SAT_V_LO    (0x0E) */
      0x10, /* HUE         (0x0F) */

      0x02, /* OFORM       (0x12) 8bit stream */
      0x60, /* VSCALE_HI   (0x13) */
      0x00, /* VSCALE_LO   (0x14) */
      0x01, /* TEST        (0x15) */
      0x00, /* VPOLE       (0x16) */

      0x68, /* ADEALY      (0x18) */
      0x5d, /* BDEALY      (0x19) */
      0x82  /* ADC         (0x1A) */
      };


void main(void)
{
      init_BSC();                /* Initialize SH7709 bus state controller */

      init_Bt829B();             /* Initialize VideoDecoder */
}


void init_Bt829B(void)           /* Initiallize VideoDecoder Bt829A */
{
  short  lp;                     /* loop counter */
  short  regno;                  /* regno counter */

      /* Initiallize IIC-bus controller */

      do{
            outport(II2_Control, 0x80 << 8);

            /* Loads into S0 */
            outport(II2_Data,    0x55 << 8);
```

```
        /* Loads into S1 */
        outport(II2_Control, 0xA0 << 8);

        /* Loads into S2 CLK=8MHz */
        outport(II2_Data,    0x18 << 8);

        outport(II2_Control, 0xC1 << 8);

    }while( ( inport(II2_Control) & 0x8100 ) != 0x8100 );


    /* Set up BT829B Register */
    regno = 0x01;

    for(lp=0; lp<23; regno++, lp++){

        /* Check BB */
        while( ( inport(II2_Control) & 0x0100 ) == 0 );

        /* Chip address( write mode ) */
        outport(II2_Data,  0x88 << 8);

        /* START II2 */
        outport(II2_Control, 0xc5 << 8);

        /* Check PIN */
        while( ( inport(II2_Control) & 0x8000 ) != 0 );
        if( ( inport(II2_Control) & 0x0800 ) != 0 ) {

            /* Stop */
            outport(II2_Control, 0xc3 << 8);
            return;
        }

        /* SUB ADDR */
        if( regno == 0x10 ){
            regno = 0x12;
        }
        if( regno == 0x17 ){
```

```
                regno = 0x18;
            }
            outport(II2_Data, (regno << 8) );

            /* Check PIN */
            while( ( inport(II2_Control) & 0x8000 ) != 0 );
            if( ( inport(II2_Control) & 0x0800 ) != 0 ) {

                /* Stop */
                outport(II2_Control, 0xc3 << 8);
                return;
            }

            /* Data */
            outport(II2_Data, BTdata[ lp ] << 8);

            /* Check PIN */
            while( ( inport(II2_Control) & 0x8000 ) != 0 );
            if( ( inport(II2_Control) & 0x0800 ) != 0 ) {

                /* Stop */
                outport(II2_Control, 0xc3 << 8);
                return;
            }

            /* Stop */
            outport(II2_Control, 0xc3 << 8);

        }
}

#include "MS7709.inc"
```

Figures 6.1 to 6.3 show graphs of HD64413A drawing performance. The graphs show the time required for drawing within the range 320 (H) × 240 (V).

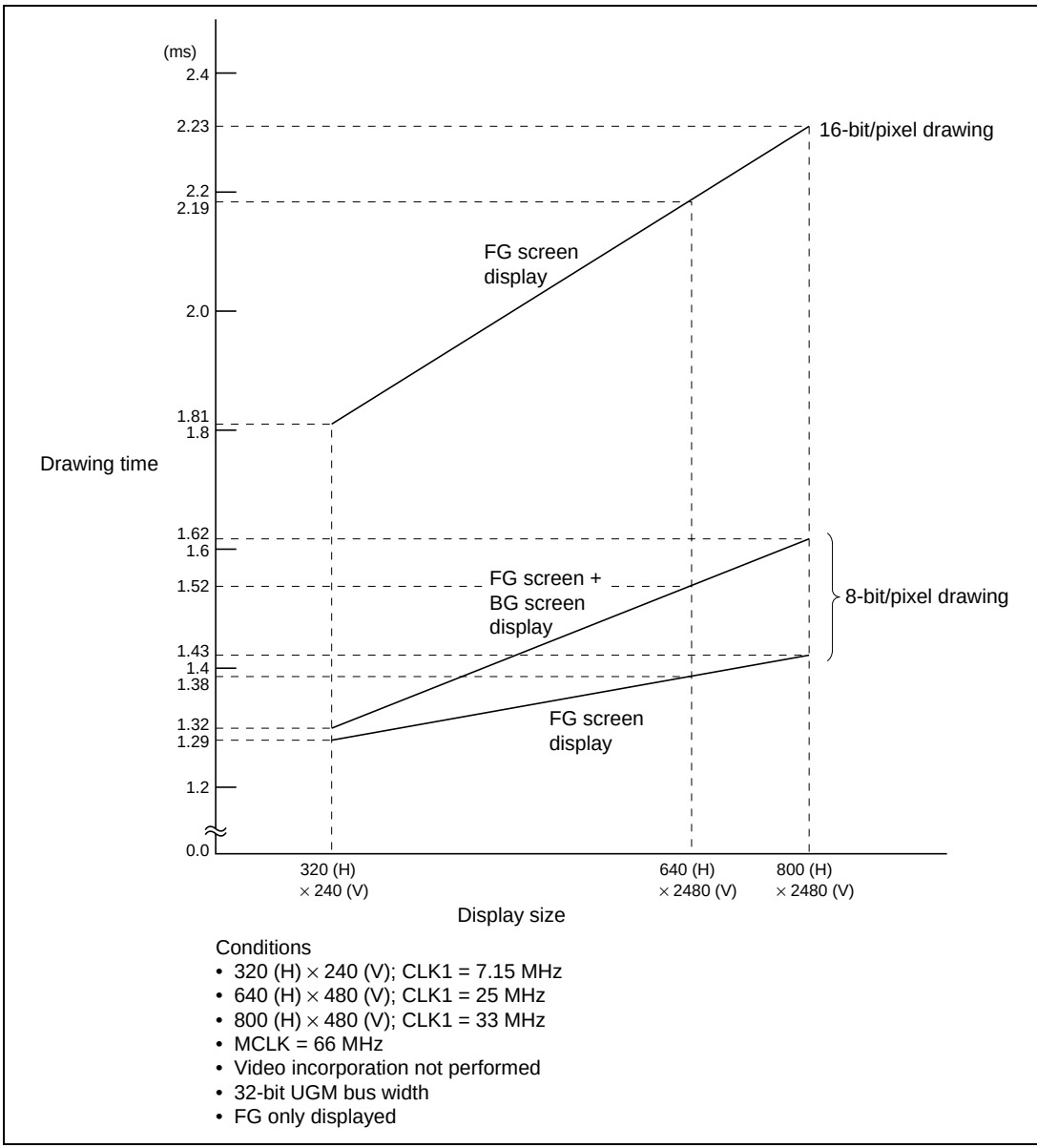


Figure 6.1 POLYGON4C Drawing Performance when FST = 0 (Drawing Range: 320 (H) × 240 (V))

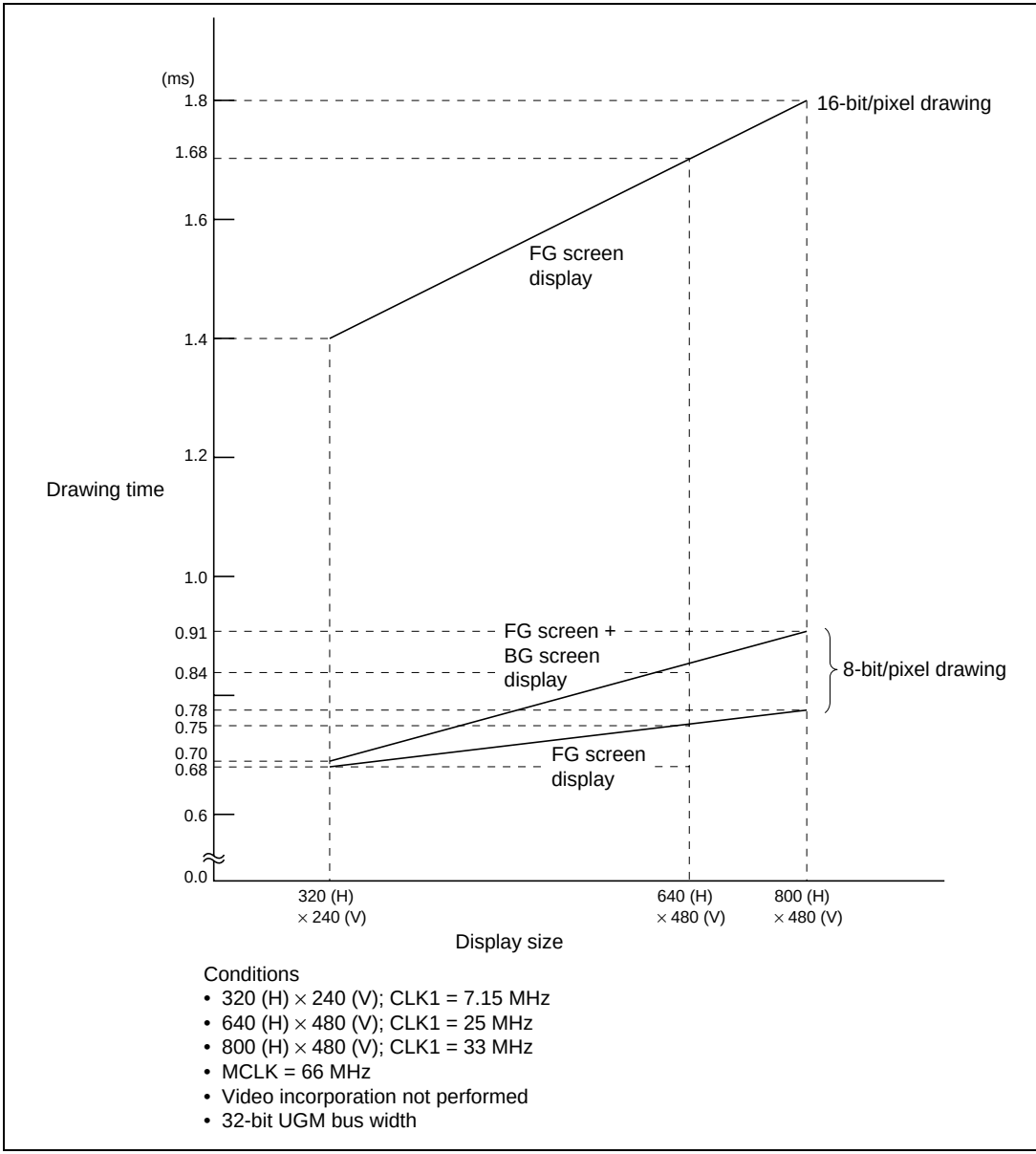
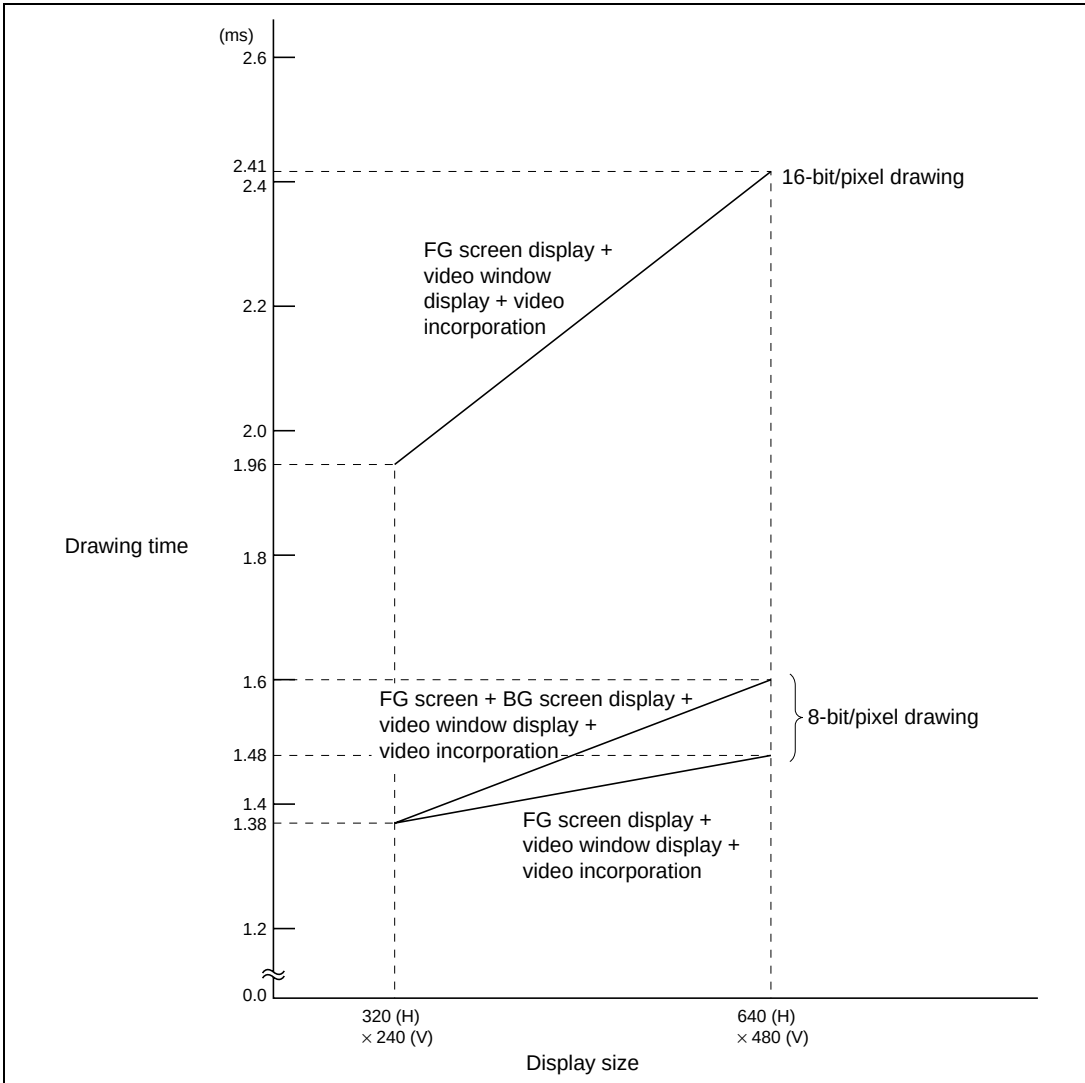


Figure 6.2 POLYGON4C Drawing Performance when FST = 1 (Drawing Range: 320 (H) × 240 (V))



Conditions

- 320 (H) × 240 (V); CLK1 = 7.15 MHz
- 640 (H) × 480 (V); CLK1 = 25 MHz
- 800 (H) × 480 (V); CLK1 = 33 MHz
- MCLK = 66 MHz
- Video window size: 320 (H) × 240 (V)
- Video incorporation size: 320 (H) × 240 (V)
- 32-bit UGM bus width

Figure 6.3 POLYGON4C Drawing Performance when FST = 0 (Drawing Range: 320 (H) × 240 (V))

A. MS4413DB01 Specifications

Table A.1 shows the MS4413DB01 system specifications. In this Application Note, area 5 is used.

Table A.1 MS4413DB01 Specifications

No.	Function		Specifications
1	Address areas used		MS7709RP01: Area 2 or area 5 (SW switching) UGM area: H'A8000000 to H'A87FFFFF (using area 2) H'B4000000 to H'B47FFFFF (using area 5) Q2I register area: H'A8800000 to H'A88002FF (using area 2) H'B4800000 to H'B48002FF (using area 5) I2C bus controller (PCF8584): H'A9000000 to H'A9000002 (using area 2) H'B5000000 to H'B5000002 (using area 5)
2	CLK0 (drawing) input		OSC3: 66 MHz (max), OSC1: 14.31818 MHz, MS7709RP01: CKIO selectable
3	Internal CLK multiplication function		CLK0: Multiplication off/on (1X, 2X, 4X) control by Q2SD setting (max. 66 MHz)
4	CLK1 (display) input		OSC2: 14.31818 MHz
5	UGM memory		64 MB (×32) SDRAM: µPD4564323G5 (NEC) ×1
6	UGM size		8 MB
7	Display		Q2SD: Analog RGB output → NTSC → video generation (PAL support by changing OSC jumper setting)
		NTSC encoder	CXA2075M (SONY) used
8	Display colors		Simultaneous display of 256 colors out of 262,144 (built-in color palette)
9	Supported displays		NTSC video output TV/analog RGB output TV
		Resolution	Standard 480 × 240 (maximum 640 × 480)
		Interface	Analog RGB (separate SYNC)/NTSC (composite video)
10	Video input block		Video decoder: Bt829B (Rockwell) Control unit I/F: I2C bus I2C bus controller: PCF8584T (Philips) External input CLK: 8 MHz I2C—bus CLK 90 kHz/max
11	CPU I/F		3.3 V level
12	Interrupt output		Solution Engine IRQ1–8 selectable
13	DMA transfer		Ch0 or CH1 selectable in Solution Engine DMAC
14	WAIT output		Solution Engine WAIT0–3 selectable
15	Power supply		+3.3 V, +5 V, A+5 V, supplied by expansion slot
16	Board size		128 × 182 mm

Figure A.1 shows a block diagram of the MS4413DB01.

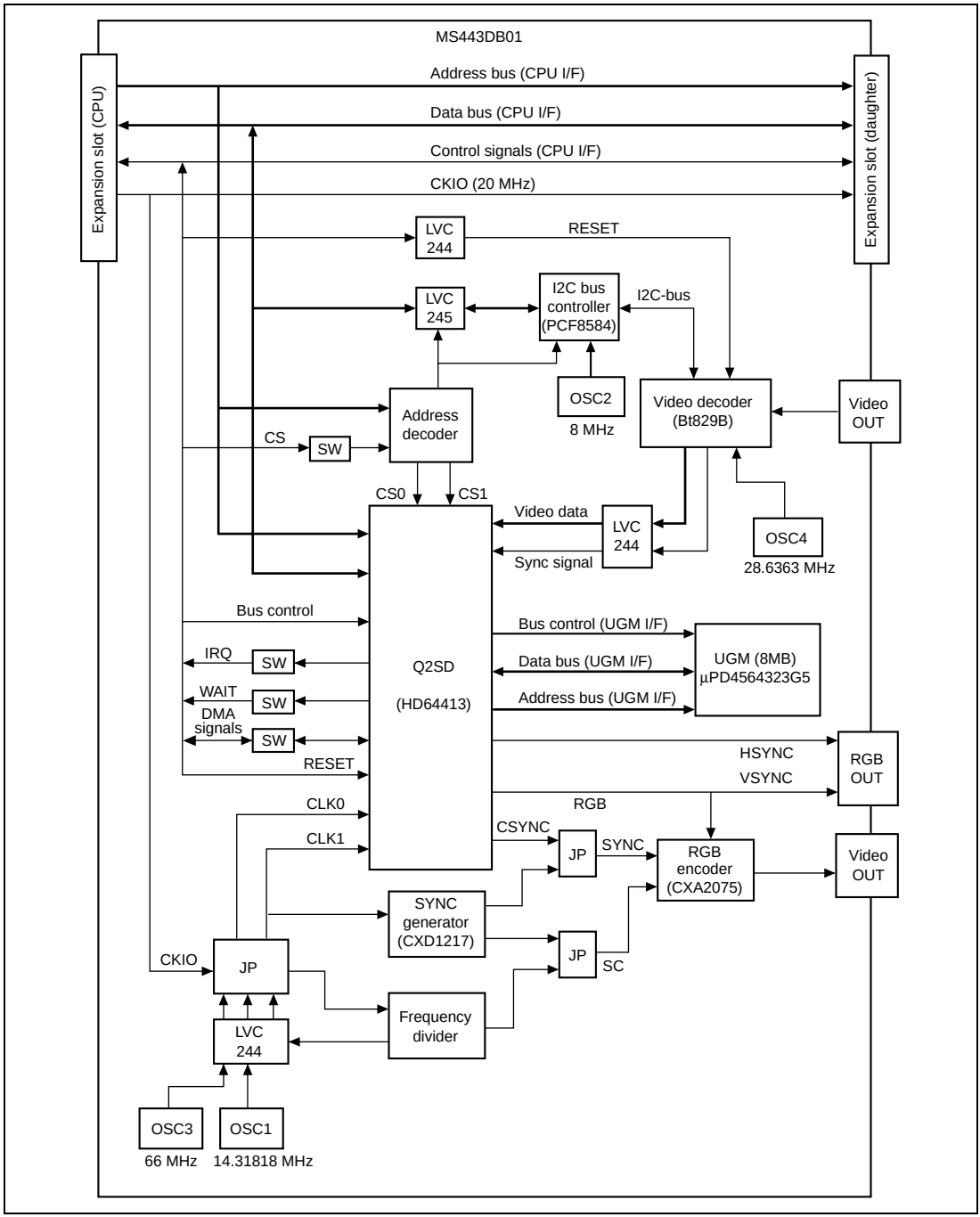


Figure A.1 Block Diagram of MS4413DB01

Figures A.2 to A.7 show MS4413DB01 circuit diagrams.

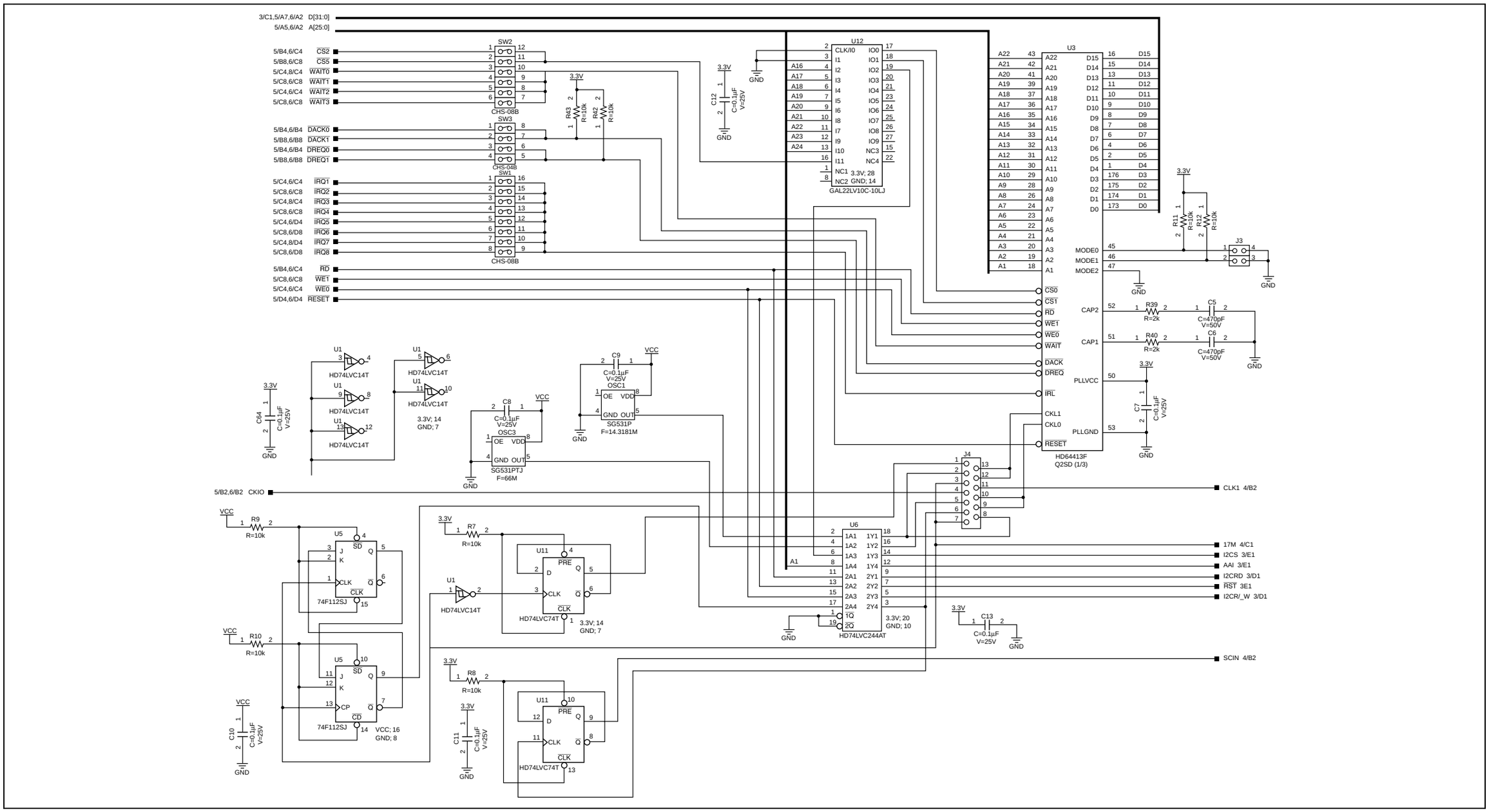


Figure A.2 MS4413DB01 Circuit Diagram (1)

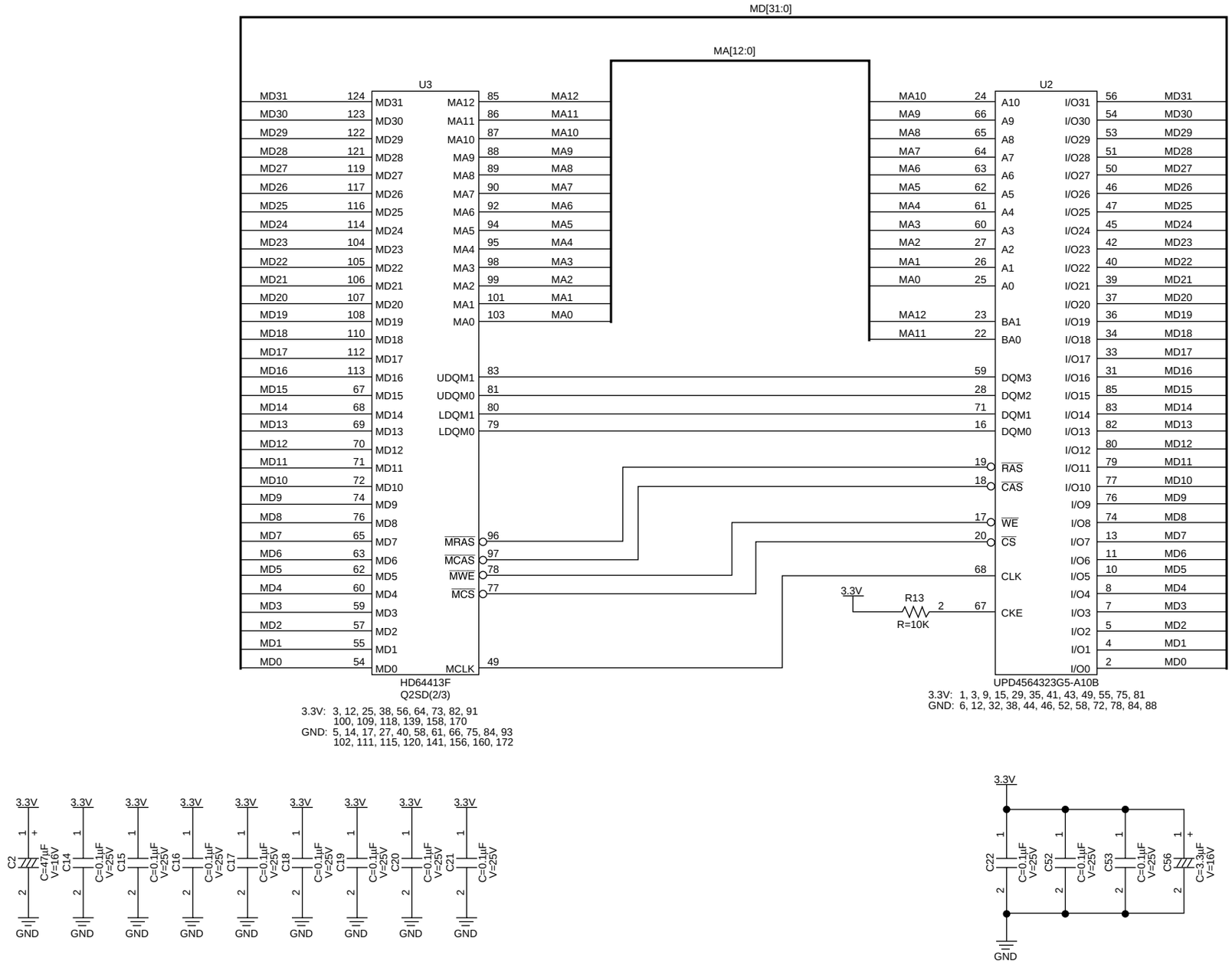


Figure A.3 MS4413DB01 Circuit Diagram (2)

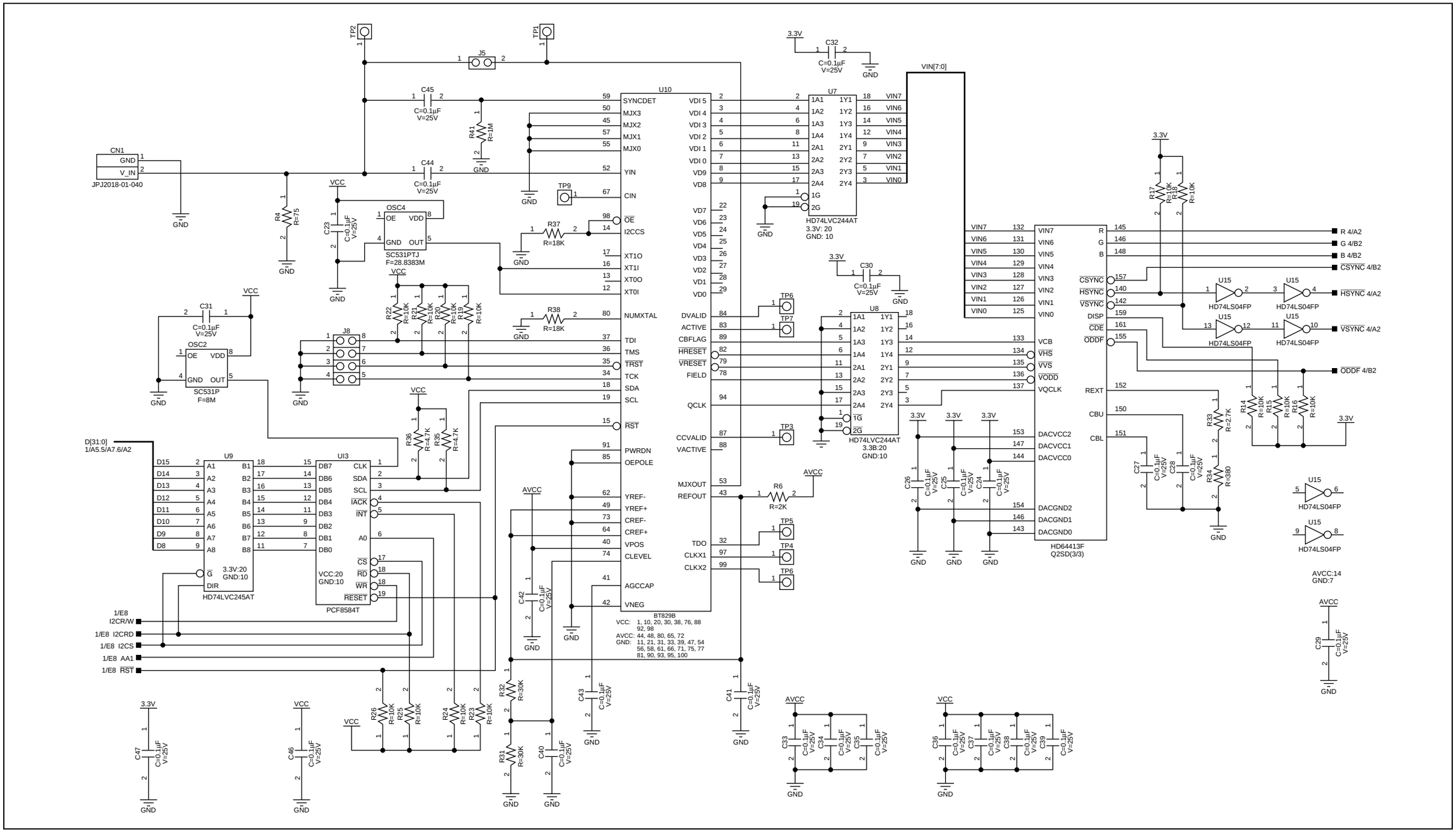


Figure A.4 MS4413DB01 Circuit Diagram (3)

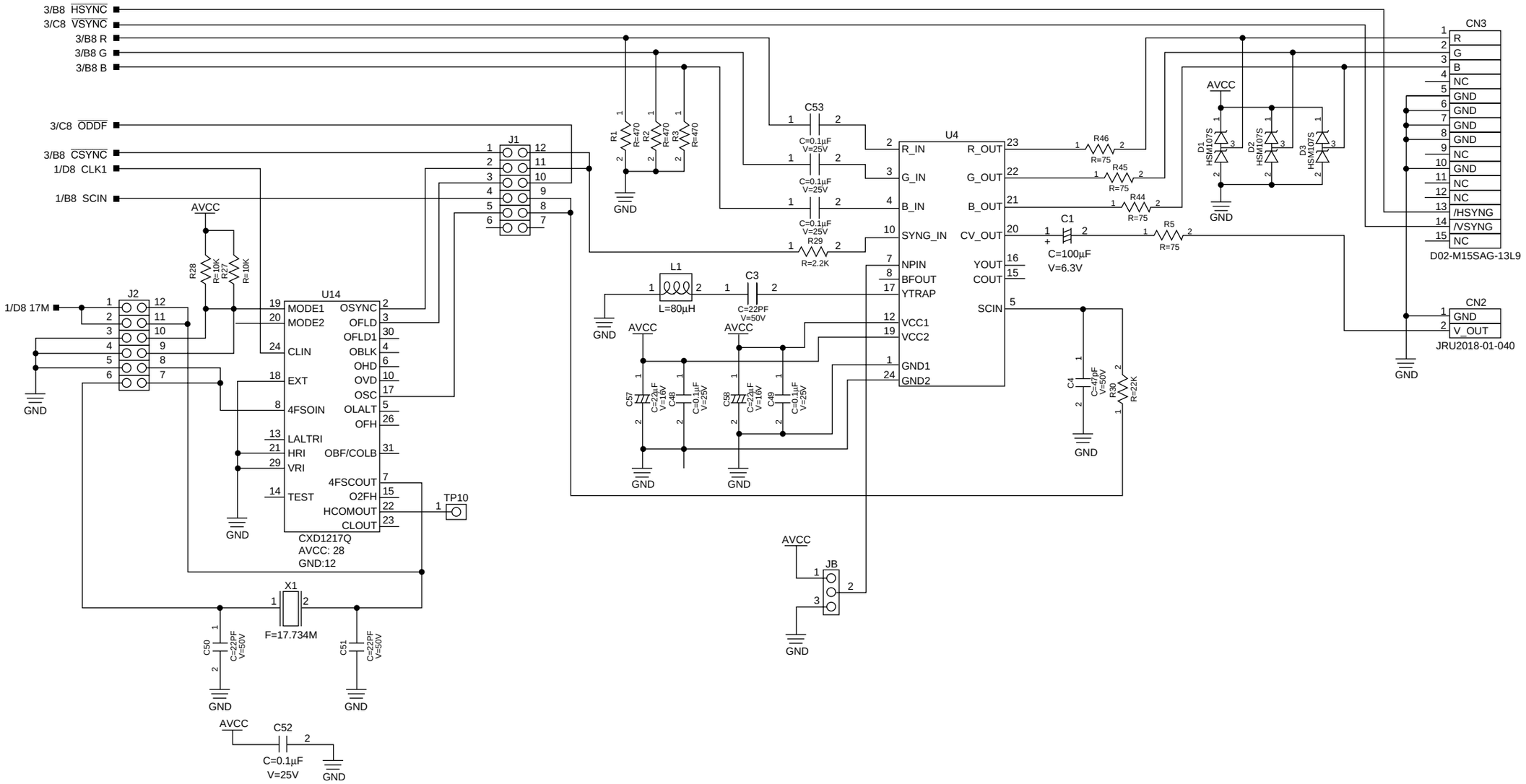


Figure A.5 MS4413DB01 Circuit Diagram (4)

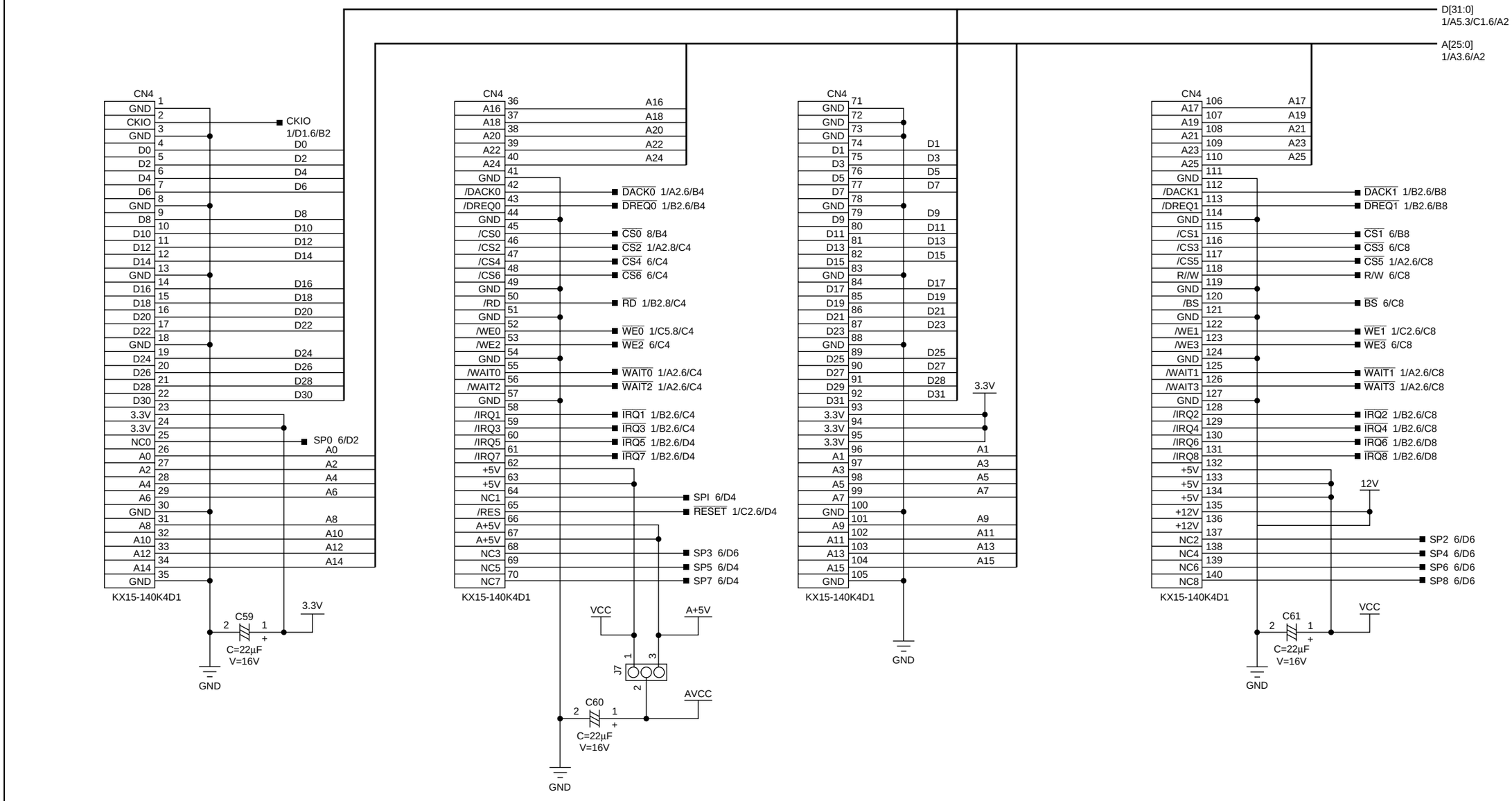


Figure A.6 MS4413DB01 Circuit Diagram (5)

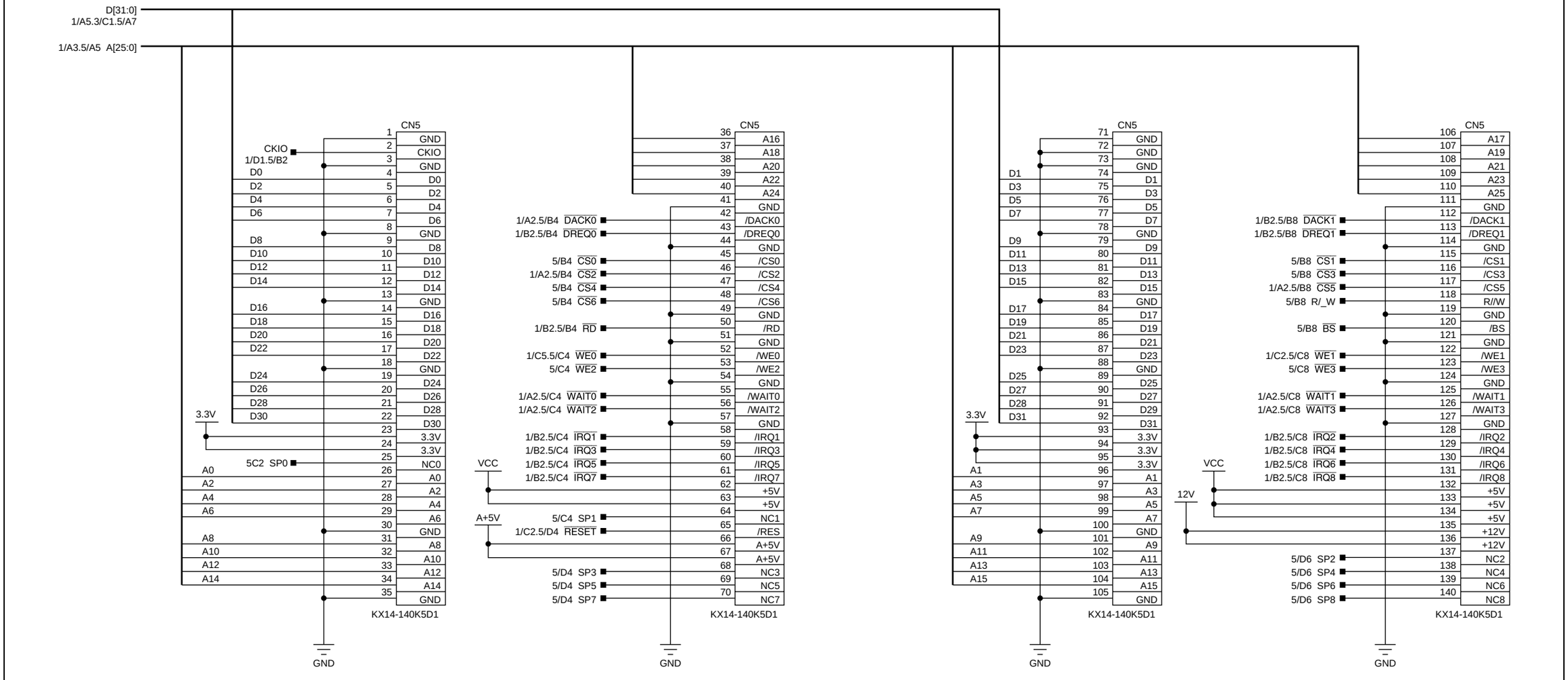


Figure A.7 MS4413DB01 Circuit Diagram (6)

HD64413A Q2SD Application Note

Publication Date: 1st Edition, October 1999
Published by: Electronic Devices Sales & Marketing Group
Semiconductor & Integrated Circuits
Hitachi, Ltd.
Edited by: Technical Documentation Group
UL Media Co., Ltd.