

Direct drive of the LMG7480 from the SH7032 EVB

Introduction

The LMG7480 is a medium resolution monochrome flat panel display with a resolution of 240 X 160 pixels. To satisfy the physical demands of portable equipment such as Mobile Phones / PDA's etc., it is supplied without controller, bezel and bias voltage division circuitry giving a highly compact panel size. The following application note describes how using only the SH7032 EVB (low cost evaluation board) and the power supply circuit, all of the timing waveforms and voltages for the LMG7480 can be generated and a page of data displayed. The total resources used are 4/5 of the 16bit timer unit, 1/4 channels of DMA 1/2 channels of the Timing Pattern Controller and a total of 12 I/O pins. At 20Mhz and a frame rate of 75Hz approximately 20% of the CPU Bandwidth time is used. This can be improved using faster memory or better, 7034 with single state 32bit access internal ROM.

The application note is split into four sections:-

1. How it works
2. SH7032 Code Generation
3. Hardware Description and connection diagram
4. Graphics file conversion code using Borland C++™

*When studying this application note it may be useful to have a copy of the following as reference:- LMG7480 data sheet, EVB7032 Manual, SH7034/7032 User manual.

Section 1 - How it works

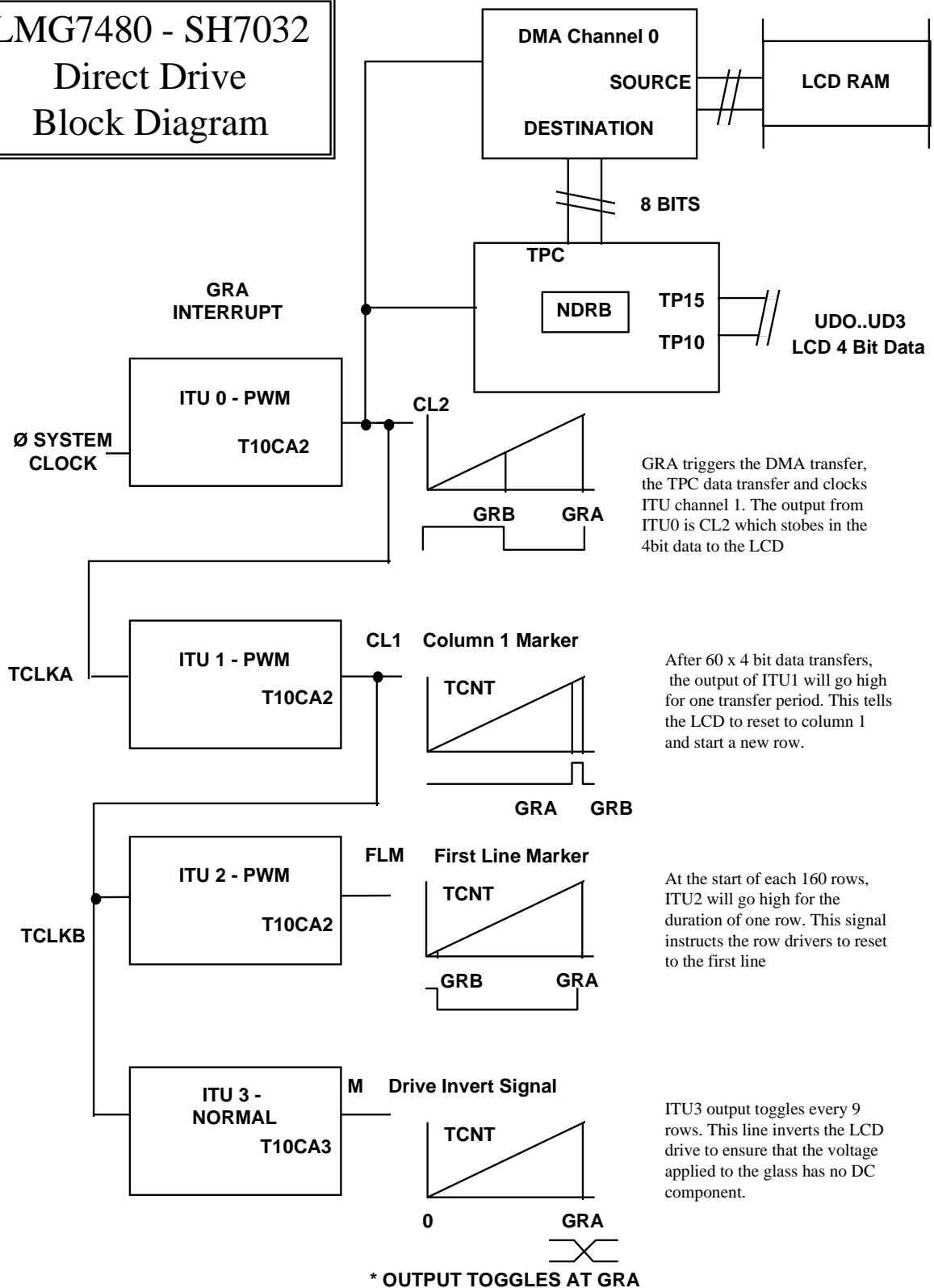
The LMG7480 is controllerless so it requires constant clocked datastream to display even a static frame. The 4bit data needs to be sent around:-

$75\text{Hz}(\text{refresh rate}) \times 160(\text{duty cycle}) \times 60(\text{nibbles of data per row}) = 720,000 \text{ times per second } (\sim 1.4\mu\text{S})$. This obviously would use a considerable amount of CPU time, even for more powerful microcontrollers if implemented in software. It also means that very strict programming techniques should have to be employed to ensure a consistent datastream. It should also be born in mind that if the duty cycle on the M signal for example were to deviate from 50%, a DC level would be applied to the LCD glass which may permanently damage it. The alternative is to use the on board peripherals which can do the same job without minimal CPU intervention giving free CPU time for other tasks.

The Integrated timer unit, Timing Pattern Controller and the Direct Memory Access unit are used to provide the waveforms for the LMG7480 as specified in the datasheet. 4 channels of the ITU unit are used to provide the clock signals and the TPC and DMA are used to supply the 4bit data.

The system is configured as shown in the figure below:-

LMG7480 - SH7032 Direct Drive Block Diagram



ITU channel 0 is the master clock (CL2) which triggers the DMA transfer, TPC output transfer and toggles the CL2 line to strobe the 4bit data into the LCD driver. The output also serves as the clock source for ITU channel 1 which counts 60 CL2 clocks ($60 \times 4 = 240$ = complete row of pixels). After 60 transfers, ITU1 outputs a 1 on the CL1 line to instruct the drivers to latch the data received from the last row, and start loading data into the next row. The output from CL1 serves as the clock input for ITU channels 2 & 3.

ITU Channel 2 counts up to 160 rows (1 complete frame) and then outputs the FLM (first line marker) to tell the LCD driver to reset to row 1. It also causes a CPU interrupt which resets the DMA source address to the beginning of LCD RAM. This re-starts the display of the frame.

ITU Channel 3 counts up to 9 rows and then toggles the M (drive invert) pin which ensures that the overall drive to the LCD has no DC offset.

Once the ITU, TPC and DMA have been setup, the only task the CPU needs to do is to reset the DMA address and count at the end of each frame. This translates to a small Interrupt service Routine which will be called every 13mS. *Note that the DMA minimum transfer word size is 8 bits. This means only 4 bits of each transfer are valid. Consequently The LCD RAM allocated is twice the size ($240 \times 160 / 8 \times 2 = 9600$ bytes) of the actual data required. The upper 4 bits of each byte are valid and the bottom 4 bits are don't care. The lower 4 bits could be used to save a second frame and a simple "swap" function could be written to swap the nibbles in the LCD RAM to switch between screens.

The rest of the code provides a simple demonstration of the panel and SH7032 working together. By connecting a serial link between SCI0 (User Serial Port) and a host PC running a terminal emulator (19200 bps, 8data bits, no parity) such as Windows™ Terminal, it is possible to download pictures and perform some of simple graphics commands. (note the command letters must be in lower case)

The operation is as follows.

- c - clear screen
- t - draw a spiral. This is done using a for loop, floating point sin & cos and the plot function contained in USERCODE.C
- i - Invert screen. This swaps pixels from 1 to 0 and vice versa
- y - Switch display on. This enables DISP ON and VEEON lines, all of the timers and the DMA transfer of data to the TPC.
- n - Switches the display off. Disables the functions as described as above. This feature is useful for comparing execution time of code with the display on and off
- downloading pictures
Specially converted pictures may be downloaded over the terminal. The format is described in section 4 of this application note (graphics file conversion). To download the picture simply select "SEND TEXT FILE", click on your selected file to download and click OK. The picture takes approximately 5 seconds to download at 19200 baud.

*Remember to set the program counter to the reset vector value (contained at H'A00 0000) before executing the program.

Section 2 - SH1 / 7032 Code generation

To dramatically ease the SH7032 code generation process the following support tools were used:-

- **Hitachi Work Bench.** This is an application development environment which provides the user with a code editing front end and project building facilities. This significantly simplifies and speeds up code generation where many source files are used. It generates all of the necessary command line switches and calls the compiler/assembler/librarian/linker for the relevant source files automatically.
- **Stenkil MakeApp.** With the continuing improvements of on chip peripherals for embedded microcontrollers, it becomes necessary to use additional code generation tools to help set up the associated registers with each peripherals. This utility generates all of the required initialisation code and access functions (eg. interrupt handlers) in C - source format. It also supports rule checking such illegal settings (eg. two functions for one pin) are not accidentally selected.
- **Hitachi/MCS Super H C-Compiler.** The EVB7032 as standard comes supplied with a free copy of the Cygnus/Gnu compiler which is supported by third parties. The compiler used for this application note is the Hitachi/MCS compiler which is sold and supported by Hitachi Europe Ltd. It is an optimising C-compiler/assembler/Librarian and linker code generation tool. It also has the benefit of being compatible with Hitachi Workbench above.
- **Hitachi Debugging Interface.** This is a C-Source level debugger with features such as single stepping, PC breakpoints, variable watches, memory dumps, register windows etc... This version is

designed to operate on the EVB7032 using the HOST serial port (SCI1) to communicate to the host PC running the front end.

All of the above software development tools are designed to work within the Windows™ environment which makes the overall code generation process much more user friendly.

MAKEAPP Configuration

The following section describes the options selected, grouped by peripheral, from the MAKEAPP front end to configure the SH7032 to run as desired. Only relevant options are described. Any other options not described such as DRAM controller settings should be left to their default values. MAKEAPP is intrinsically safe as for each source file generated, all of the associated registers will be set, leaving no registers to their power up default values.

CPU

This section of MAKEAPP configures the CPU and bus controller to work within the hardware configuration of the EVB7032.

- Bus Control
 - ☒ BAS,LBS,WR,HDS Enabled - Allow 16 bit external access of memory
- Wait States
 - ☒ Area 2,0 2 States - For 8 bit EPROM area and 16bit code/data area add 2 extra wait states. (total 3 state access)
- DRAM / TimerUn-altered
- System
 - ☒ BACK
 - ☒ RD - Enable bus control pins
 - ☒ WRH/LDS
 - ☒ WDL/WR

I/O

Here the two I/O pins used in the application are configured as outputs and are pre-loaded with a default value.

- PORT A Low - No change
- PORT A High- PA8 VEE_ON ☐ IN ☒ OUT ☒ "1"

PA.8 is assigned as an output VEE_ON, with default setting HIGH
- PORT B Low - PB7 DISP_OFFn ☐ IN ☒ OUT ☐ "1"

PB.7 is assigned as an output DISP_OFFn, with default value LOW
- PORT B High - No change
- PORT C - No change

INT

Set interrupt priorities and set the level of the interrupt P-mask which will determine the lowest priority of interrupt that the CPU will service. Note SCI1 also has to be set above the CPU P-masks level to enable a break from the user front end.







- Interrupt - Default Settings
- Priority -

ITU 0 Set to		0
ITU 2 Set to		12
SCI1 Set to		12
P- Mask		10

Interrupt mask level set to level 10. ITU 2 GRAB interrupt level set to 12 so the interrupt is enabled. ITU0 GRAB interrupt will also occur but is serviced by DMA channel 0 as it is set to interrupt level 0 which is below the P-mask level

SCI Channel 0

This is set up for communicating with a terminal at 19200 baud, 8 data bits, 1 stop bit and no parity.

- Mode  Asynchronous
- Operation  Enable TX
-  Enable Rx
- Interrupts - None
- Data Bits  8
- Stop Bits  1
- Parity  None
- Speed Baudrate - 19200
- Clock Generation Clock Source Internal SCK0 Unused

The above settings configure SCI0 (User Serial Port on EVB7032) to Asynchronous operation, Tx and Rx enabled, Polled Operation, 8 data bits, 1 stop bit, no Parity and 19200 baud. The clock source is derived from the internal clock so that the SCK0 clock input pin is available for other uses.




Channel 1

Channel 1 is left unused in the MAKEAPP configuration as it is reserved by HDI debugger. To make sure that HDI can operate after the initialisation, four extra lines are added to USERCODE.C to make sure that the Tx and Rx pins of SCI1 are enabled. Also the SCI1 interrupt level must be set above the P-Mask. See the Interrupt control section of MAKEAPP to do this.

ITU



Channel 0 Used as CL2 Output

This channel needs to produce a 50% duty squarewave which produces an interrupt at the falling edge of each cycle to DMA the 4 bit display data to the Timing Pattern Controller. Hence GRA output compare is set to 1C, causing the output to go high and its associated interrupt is enabled. Note that this interrupt is set to a lower priority than the P-Mask so that it is not serviced by the CPU. GRB is set to 0E and causes the output to go low. The timer is used in PWM mode to ensure that the period and duty are controlled.

- Mode  PWM
- Operation  Independently
- Clock Source - Internal Clock Ø
- Timer Cleared by GRA
- Interrupts  GRA enabled
- I/O Control - 1 Output at GRA Compare Match
0 Output at GRB Compare Match
- General Registers GRA 1.75uS Value H'001C
GRB 0.88uS Value H'000E

Channel 1 Used as CL1 Output



This channel needs to produce a 1/60 duty squarewave clocked by ITU 0. Hence GRB output compare is set to 3A (58 decimal), causing the output to go high. GRA is set to 3B (59 decimal) and causes the output to go low. The timer is used in PWM mode to ensure that the period and duty are controlled. The clock input is set to TCLKA which is fed by ITU0 output.

- Mode  PWM
- Operation  Independently
- Clock Source - TCLKA, count rising edges
- Timer Cleared by GRB
- Interrupts - non enabled
- I/O Control - 1 Output at GRA Compare Match
0 Output at GRB Compare Match
- General Registers

GRA	---	Value	H'003A
GRB	---	Value	H'003B

Channel 2 Used as FLM Output (first line marker)



This channel needs to produce a 1/160 duty squarewave clocked by ITU 1. Hence GRB output compare is set to 01, causing the output to go low. GRA is set to A0 (160 decimal) and causes the output to go high. The timer is used in PWM mode to ensure that the period and duty are controlled. The clock input is set to TCLKB which is fed by ITU1 output.

- Mode  PWM
- Operation  Independently
- Clock Source - TCLKB, count falling edges
- Timer Cleared by GRA
- Interrupts - GRA enabled. The priority of this interrupt is set higher than that of the P-MASK. This means the CPU will service the interrupt. The interrupt effectively marks the end/start of a frame. The interrupt service routine resets the DMA count and source address. See DMA channel 0 configuration.
- I/O Control - 1 Output at GRA Compare Match
No Output at GRB Compare Match
- General Registers

GRA	---	Value	H'00A0
GRB	---	Value	H'0001

Channel 3 Used as M Output (LCD drive invert)







This channel needs to produce a 50% duty signal which toggles every 9 CL1 clocks. Hence ITU3 is clocked by ITU 1. The timer is used in Normal mode and GRB is not used. GRA is set to 09 and causes the output to toggle. Using the timer in toggle mode guarantees a 50% duty cycle which is essential for this signal as it inverts the LCD signals to the glass, ensuring no DC is applied. Applying DC to the glass will cause electrolysis which may permanently damage the glass. The clock input is set to TCLKB which is fed by ITU1 output.

- Mode  Normal
- Operation  Independently
- Clock Source - TCLKB, count falling edges
- Timer Cleared by GRA
- Interrupts - none enabled
- I/O Control - Output toggles at GRA Compare Match
No Output at GRB Compare Match
- General Registers

GRA	---	Value	H'0009
GRB	---	Value	H'FFFF



Channel 4 Unused

TPC


Group 0	Unused		
Group 1	Unused		
Group 2	Unused		
Group 3	Mode		Normal mode (NDRB transferred to output on GRA without output inversion at GRB)
	Triggered by		Triggered by ITU Channel 0. This effectively causes new data from NDRB upper nibble to be output every CL2 clock cycle
	Enable		TP12
			TP13 - Enable output of upper 4 bits of TPC channel B
			TP14
			TP15


DMA




The DMA's role in this application is to transfer data from the LCD ram area to NDRB on each CL2 rising edge using DMA channel 0. The DMA is set to automatically increment up through the LCD RAM on each ITU0 interrupt. Note that because the interrupt P-MASK is set to a level high than the ITU0 interrupt, the CPU will not service the interrupt, so the DMA will do it instead. The DMA's operation is completely automatic apart from intervention by the CPU every 9600 counts which will reset the source address to the beginning of LCD RAM. The destination address remains constant, being NDRB for the Timing Pattern Controller.

- Operation  Dual Address mode
This mode of operation transfers data from one specified address to another specified address.
- Transfer 

Activation	-	IMIA0 Copy from any address to any address on ITU Channel 0 interrupt
Count	-	H'02580

This will transfer H'2580 (9600 decimal) bytes/words on the next 9600 ITU Channel 0 interrupts. Any further interrupts will not trigger a transfer.
- Source 

Increment	-	The source address will increment after each transfer.
Address	-	H'A010000 Start address of transfer= LCD RAM
- Destination 

Fixed	-	This address will remain constant
Address	-	H'5FFFFFF4 = NDRB of Timing Pattern Controller
- Size  Byte - 1 byte of information is transferred
- Bus Control  Cycle Steal - Bus right is given to another bus master after one transfer. If burst mode was selected, the entire 9600 bytes would be transferred on a single ITU0 interrupt. Using cycle steal ensures that the data is passed to the LCD in synchrony with ITU0 output which strobes it into the LCD driver.
- Enable End Interrupt  No end interrupt required. In actual fact this interrupt could be used to trigger an ISR serviced by the CPU that resets the DMA count and Source address. Instead ITU2 GRA is used to perform this function. As they both occur at the same time it does not matter which interrupt is used.

WDT

Not used in this application

A/D

Not used in this application

WorkBench Configuration - “LCD.HWB”

The project was created from the NEW PROJECT option in the pull down menu using the following options

- Project name LCD
- Toolchain HITACHI (MCS)
- CPU Family SH7000 Series
- CPU Type SH/7032
- O/S None
- Use Default Libraries ☒ - Use default library shipped with compiler

From the OPTIONS menu the following configuration was applied:-

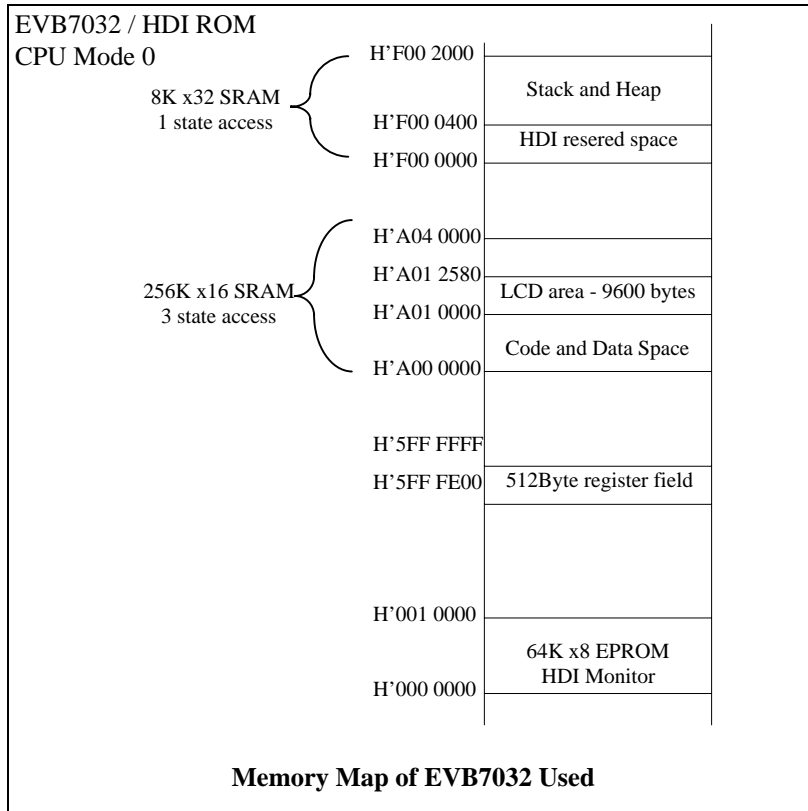
COMPILER - All default settings retained. No optimisation was included to ease debugging. Users may at a later date wish to enable optimisation for code space or speed optimisations.

ASSEMBLER - All default settings retained. The only two files assembled in this project are c0.src and sh_intv.src.

C0.SRC is the C Startup code. It initialises the Stack Pointer and then calls each of the initialisation functions in turn. When main() returns and all tidying up functions have been called, the program sits in an infinite loop. sh_intv.src contains the interrupt vector table. Note to alter entries in this table, use the “Edit Interrupt Vectors” option in “Processor Configuration as described below”.

LINKER - In the “General” category the “Check for Un-defined functions” box was checked. This is a safety net feature which will make sure that all functions are properly declared in your code.

The “Sections” category had to be changed significantly. This section defines exactly where the linker places code, data and stack. Because the default settings place sections from 0x00000000 onwards, the code will not download or run on the EVB7032. Below is the Memory Map of the EVB7032 used.



The following is the section list used in the application.

VECT	-	H'A000000	Interrupt vector table start
P,C,D	-	H'A001000	Program,Constants, Initialise data to copy
B,D	-	H'A008000	Non-Initialised & Initialised data
STACK	-	H'F000400	Start of stack area.

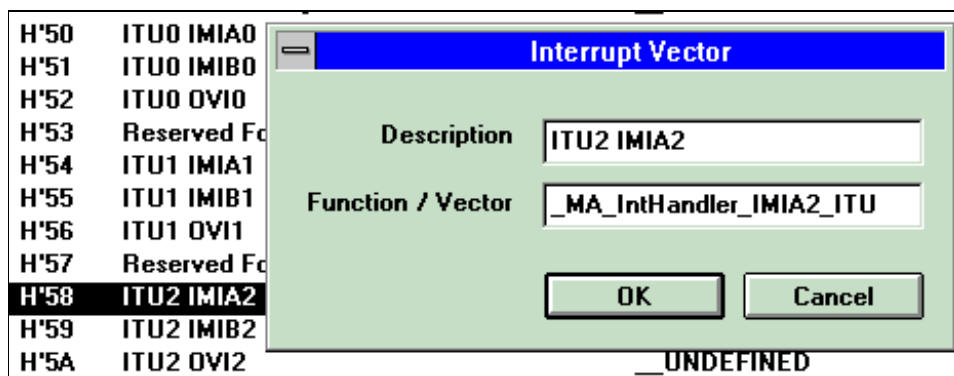
In addition the Align Section and Perform Section Validation were checked (☒) to make sure the sections are at valid addresses.

PROCESSOR CONFIGURATION - Stack size and heap size were both left at 1024, their default value. The two other sections however, Edit Hardware Configuration & Edit Interrupt file, were both altered.

The Hardware Configuration file has been modified to copy the monitor interrupt vectors across to the user vector area (0x0a000000) so that so that when the VBR (vector base register) is changed to the user area, the monitor can still operate. The monitor requires SCI1, RESET, UBC , TRAP etc. User IRQ's and peripheral interrupt sources are not overwritten as they occupy the vector outside of this. See sh_hwcfg.c listing at the end of the application note for details.

The Interrupt File must be edited if you have any interrupt service routines. As MAKEAPP generates your interrupt service routines for you all you need to do is enter the interrupt service routine name eg.

"MA_IntHandler_IMIA2_ITU" in at the relevant vector position. Hitachi workbench will then alter the file sh_intv.src for you. Note it is possible to alter this manually but it is recommended that you use the "Edit Vector Table" utility as it makes the process more user friendly. See figure below.



Note that the Function/Vector entry requires the "_" character pre-fixing it to enable the entry to be recognised as a symbol. The actual function can be written in the associated declaration generated by MAKEAPP (for this vector it can be found in MA_ITU.C). In this application note the interrupt service routines have been written in usercode.c and are called by the default MAKEAPP call. eg. "MA_ITU_IMIA2_USERCODE".

Note for each interrupt entry this must call must be defined in "MASH1.H", for example

```
/*--- User code for ITU module ---*/
#define MA_ITU_IMIA0_USERCODE
#define MA_ITU_IMIA1_USERCODE
#define MA_ITU_IMIA2_USERCODE start_frame();
void start_frame(void);
#define MA_ITU_IMIA3_USERCODE
#define MA_ITU_IMIA4_USERCODE
```

From the PROJECT pull down menu it is then necessary to add the source files which are to be compiled in the application. Dependencies such as include files are automatically pulled in by Hitachi Workbench and added to the project file list. The source files added in this application are as follows:-

Project Files

```
c:\ed\sh_stuff\dma_bits\sh_hwcfg.c
c:\ed\sh_stuff\dma_bits\sh_intv.src
c:\ed\sh_stuff\dma_bits\sbrk.c
c:\ed\sh_stuff\dma_bits\c0.src
c:\ed\sh_stuff\dma_bits\ma_cpu.c
```

```
c:\ed\sh_stuff\dma_bits\ma_dma.c
c:\ed\sh_stuff\dma_bits\ma_int.c
c:\ed\sh_stuff\dma_bits\ma_io.c
c:\ed\sh_stuff\dma_bits\ma_itu.c
c:\ed\sh_stuff\dma_bits\ma_tpc.c
c:\ed\sh_stuff\dma_bits\usercode.c
c:\ed\sh_stuff\dma_bits\ma_sci.c
```

Note how the majority of these files have been generated by MAKEAPP which saves a considerable amount of time writing and more importantly, debugging the peripheral drivers.

This is the only setting up required by Hitachi Workbench for this project and choosing the “build” option will generate the .ABS file suitable for downloading into HDI.

USERCODE.C

This file represents the majority of the code written manually for this application note. In essence it calls the peripheral initialisation routines generated by MAKEAPP, switches the LCD on using the ports and then provides the interrupt service routine which resets the DMA to the start of the LCD display RAM, at the end of each display frame. It also initialises SCIO (user) and depending on which character is received the code will perform a graphics function or accept a frame of graphics data. Refer to the end of the note for the USERCODE.C listing.

Section 3 - Hardware description

The hardware used to create this application note can be split into three items

1. SH1 - EVB7032 Evaluation board
 - This board contains a SH7032 running at 16mhz with 8K of 32 bit wide single state access SRAM built in.
 - 2 serial ports terminated with 9way D connectors connected to SCIO,1
 - 64K 8bit wide 120nS boot memory (for monitor)
 - 256K 16 bit wide 120nS SRAM for code/data
 - *Note this has been upgraded from the standard 64K. The application will work fine with the 64K option as long as the linker sections are amended and the LCD RAM start address is changed in USERCODE.C
 - All ports and bus available on 0.1” connectors.
 - Requires single 5 Volt supply
2. LMG7480 Compact monochrome flat panel
 - 3Volt logic
 - Reflective mode (no backlight power requirements)
 - 1/160 duty cycle
 - -20V driver circuit power supply voltage required
 - External resistor chain / intermediate voltage level supply
 - 9 logic signals; CL1,CL2,UD0..UD3,FLM,M,DOFFn
3. Bias voltage generation circuit and Level shifting block

The functionality of this circuit is as follows:-

 - Generate a 3Volt power supply for level shifter and LCD
 - Generate a -20V variable supply for the LCD bias (VEE). Note that this supply must be able to be switched on and off via a logic signal in order to provide the correct power up / power down sequence as specified in the LMG7480 datasheet
 - Provide 9 channels of 5V⇒3V logic conversion to enable the 5V EVB7032 to drive the 3V LCD display
 - Delay FLM by at least 300nS

Items 1 and 2 are described in the EVB7032 and LMG7480 datasheet respectively. This section of the note will focus on item 3.

The circuit diagram for the Level shift/Voltage generation block can be seen in the circuit diagram. The 3 volt supply is achieved simply by dropping ~1.8V using 3 silicon diodes in series. As the total supply requirement for the LCD/Circuit is less than 5mA this method is perfectly acceptable. Note that this supply is decoupled using a 22uF capacitor and a 0.1uF Capacitor.

The -20V supply is obtained from a 5V \Rightarrow $\pm 12V$ 1Watt DC-DC converter. The +12V side is connected to 0V so the -12V side effectively provides -24V. Note there is a 1uF capacitor included to decouple this supply. This voltage is then switched via the NPN & PNP transistor pair to provide a gated -24V supply. The NPN transistor provides the actual supply switching, acting as an open collector output. The PNP also acts as an open collector output and hence enables control of the NPN transistor from the 5Volt supply rail. Note that the control signal (VEE-ON) is active low. This means when initialising the SH7032 ports, the pin used to control this signal must be pre-loaded with a '1' so that the negative supply is not accidentally switched on.

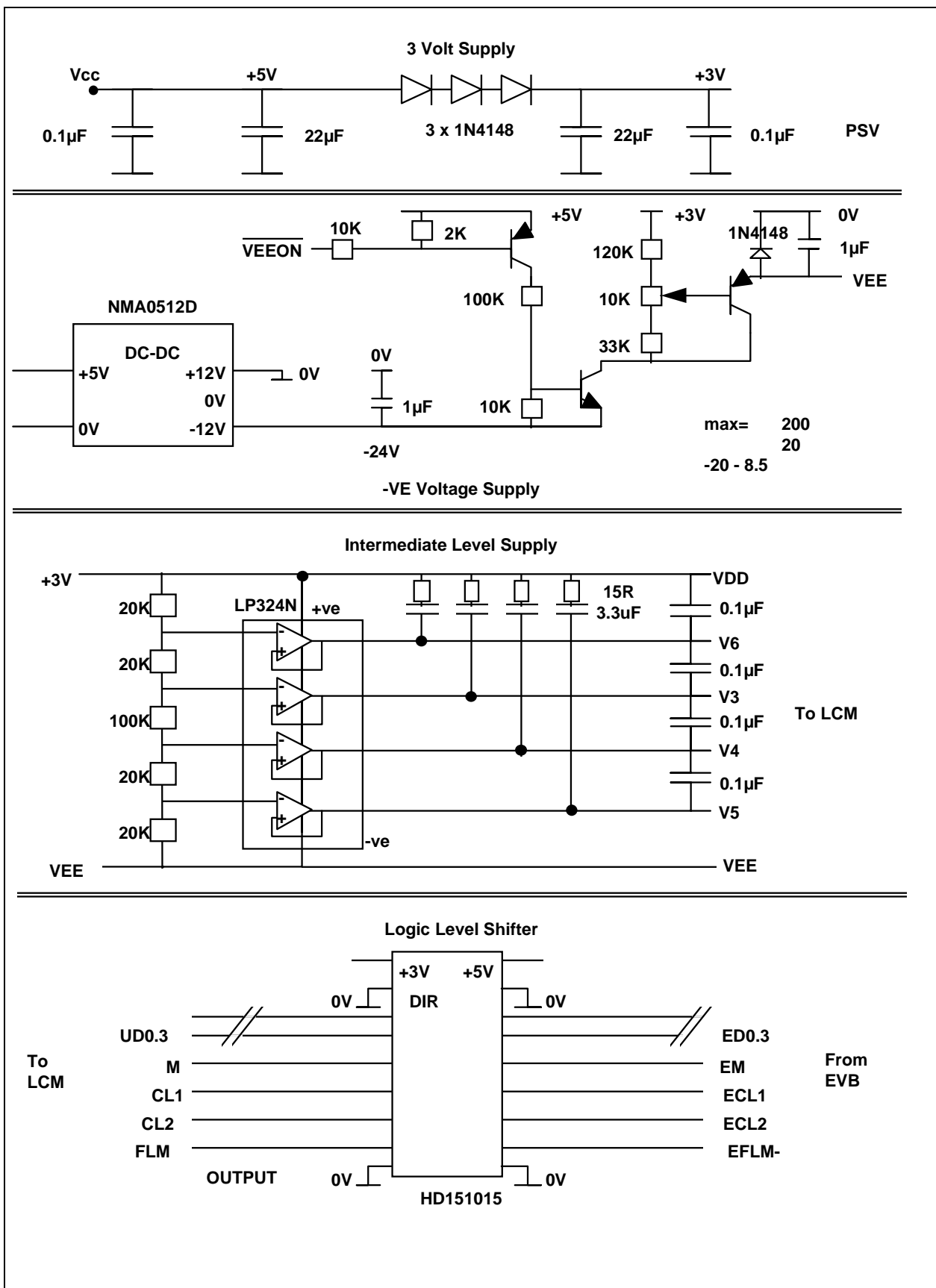
The gated -24V supply is then controlled (variably reduced) using a potential divider and emitter follower which consists of a PNP transistor and three resistors, one variable and two fixed. The LMG7480 can be brought from maximum to minimum contrast by varying VEE from around -15volts to -18volts. Consequently, by adding the fixed resistors, the range of the supply is limited from -10volts to -20volts giving a finer contrast control. The output from this variable supply is again de-coupled using a 1uF capacitor. Fairly low values of de-coupling were chosen to ensure that the fall time of the negative power supply is fast, again to satisfy the requirement of the power up / power down diagram shown in the LMG7480 datasheet.

The non-select voltage levels are obtained from a 5 resistor divider chain, the voltages from which, are buffered using an 4 op-amps as a voltage followers. Again these voltages are de-coupled using 1uF capacitors except here 15ohm resistors are used in series to avoid over-current from the op-amp outputs at power up. The opamp supply is derived from +3volt and the -VEE rails.

The logic is translated from 5volts to 3volts using a 9bit level shifter, HD151015. The data direction and output control are set to make the 5volt side the input and the 3volts side the output. The FLM signal is delayed using a 1nF capacitor and a 1K resistor. This gives a time constant of around 1uS which corresponds to a delay time of about 700nS for CMOS logic.

The connections from the EVB7032 to the user hardware are as follows:-

SH7032	EVB7032	User Hardware	Purpose
-	J4-19	+5V	5 Volt power from EVB
-	J4-20	+5V	as above
-	J4-17	0V	Ground connection to EVB
-	J4-18	0V	as above
PB15/TP15	J4-13	ED3	MSB of 4 bit LCD data
PB14/TP14	J4-14	ED2	Bit 2 of 4 bit LCD data
PB13/TP13	J4-15	ED1	Bit 1 of 4 bit LCD data
PB13/TP13	J4-16	ED0	LSB of 4 bit LCD data
PB7	J4-8	DISP_OFFn	LCD driver on off control, 1 = on
PB2/TIOCA3	J4-3	EM	LCD drive invert
PB0/TIOCA2	J4-1	EFLM	First line marker- resets to row 1
PA13/TCLKB	J5-18	ECL1	ITU2,3 clock source, from O/P of ITU1
PA12/TCLKA	J5-17	ECL2	ITU1 clock source, from O/P of ITU0
PA10/TIOCA1	J5-15	ECL1	CL1, column 1 marker.
PA8	J5-13	VEE_ON	Negative bias voltage supply control line
PA0/TIOCA0	J5-5	ECL2	CL2, 4bit display data strobe



Section 4 - Graphics file Conversion

A simple program was written in Borland C for the PC/DOS which converts graphics files into a format that can be sent down a serial port and displayed by the SH7032/LMG7480. The utility is called from a batch file CONV.BAT which will take a named .RAW format bitmap and output it to a named text file. The files CONV.BAT and CONVERT.EXE should both reside in the same directory as the input .RAW files.

The use of the converter should be as follows:-

> CONV AARDVARK ↵

where "AARDVARK.RAW" is a 2 colour 240 x 160 RAW format bitmap.

The output will be "AARDVARK.TXT" which is suitable for downloading to the evaluation board running the code described in this application note. Graphics packages such as PAINTSHOP PRO™ are capable of resizing bitmaps, reducing colour depth to 1 bit per pixel (2 colour monochrome) and saving them in a RAW file format. The files CONV.BAT and CONVERT.EXE should be placed in the directory which contains the .RAW file.

The code works by reading 4 binary bytes of the .RAW file at a time, and placing them in reverse order in the bottom nibble of byte. This value (0-15) is then added to the character '0' which means all the receiving software has to do is subtract the ASCII value of '0' and multiply by 16 to shift the data into the top byte and place it in LCD RAM. It should be noted that a row of data is stored at a time using the row_data array, and it is written to the output text file in reverse order to ensure the data will be display correctly on the LMG7480.

The batch file CONV.BAT listing is as follows:-

```
@echo Conversion Utility
@echo USE:- CONV <filename>
@echo where filename.raw is input & filename.txt is output
del input.raw
copy %1.raw input.raw
convert
copy output.txt %1.txt
del output.txt
del input.raw
```

The source code listing of the conversion program CONVERT.EXE is as follow:-

```
#include <string.h>
#include <stdio.h>
unsigned char row_data[60];
unsigned char temp,count2,row,column;
unsigned long read_c,write_c,n;
int main(void)
{
    FILE *input,*output;
    printf("\n\r");
    char *byte,*out;
    printf("\n\r240 x 160 graphics conversion utility \n\rInput file= INPUT.RAW , Output file=
OUTPUT.TXT\n\r",read_c);
    output = fopen("output.txt","w+");
    if ((input = fopen("INPUT.RAW", "rb+"))
        == NULL)
    {
        fprintf(stderr,
            "Cannot open input file.\n");
        return 1;
    }
    *out='s';
    fwrite(out,1, 1, output);
```

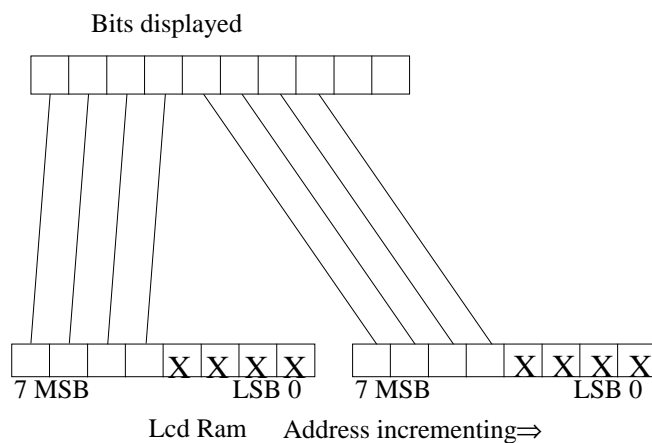
```

fseek(input, SEEK_SET, 0);
for(row=0;row<160;row++)
{
    for(column=0;column<60;column++)
    {
        *out=0x00;
        for(count2=0;count2<4;count2++)
        {
            read_c++;
            fread(byte,1,1,input);
            temp/=2;
            if (*byte) temp=(temp | 0x08);
        }
        *out= ('0'+ temp);
        row_data[59-column]=*out;
    }
    for(column=0;column<60;column++)
    {
        *out=row_data[column];
        fwrite(out,1, 1, output);
        write_c++;
    }
}
*out='f';
fwrite(out,1, 1, output);

fclose(input);
fclose(output);
printf("\n\rbytes read=%u\n\r",read_c);
printf("\n\rbytes written=%u\n\r",write_c);
return 0;
}

```

DATA REPRESENTATION IN LCD_RAM



Conclusions

The system described in this application note provides the correct frame rate when the clock is scaled to 20mhz. At this rate of refresh, using the Whetstones benchmark, it took 27% longer when the LCD screen was being refreshed. This equates to around 20% of lost CPU bandwidth from driving the LCD.

If the compare match values of ITU0 were altered to give a frame rate of around 70-80Hz and a 12.5Mhz system clock, the SH1 can directly talk to the LMG7480 with only a negative voltage supply required. At this CPU speed approximately 31% of the CPU Bandwidth being lost through driving the LCD. As the SH1 family is a single cycle instruction execution range of microcontrollers, this still leaves a considerable amount of processing power available for other applications.

Consequently in a 3.3V handheld system with a controllerless LCD graphics panel, a SH1 microcontroller alone will be able to supply virtually all of the hardware requirements in many applications. This has obvious benefits in terms of cost, size, reliability and EMC emissions.

Further Work

- As described above, it is possible to implement a 3volt only system. 3 volt DC-DC converters are available from analog semiconductor vendors such as MAXIM
- The current system uses 8 bit data memory to supply 4 bit data to the LCD. A more memory efficient approach would be to use an external latch/multiplexer which would accept 8 bit data and switch a 4 bit output between the upper and lower nibbles. There is a tradeoff here however between available memory and hardware complexity.
- Remove ITU 2 from the LCD control, freeing it up for other functions. As mentioned earlier in the note, the ITU 2 interrupt should occur at the same time as the DMA end interrupt request, if enabled. ITU2 really only outputs the FLM (first line marker) to tell the LMG7480 to start at row 1. This signal could be controlled by a delay mechanism (eg. R-C & shmitt or 555) which could be triggered by a port output. The port output would be toggled from the DMA end interrupt service routine.

**CODE LISTING
USERCODE.C**

```
/*
**=====
**
**
**   Filename : USERCODE.C
**
**   Abstract : This file calls the initialisation and provides the user
**               interface.
**   Project  : LMG7480/SH7032
**
**=====
*/

#include "MASH1.H"
#include "MA_IO.H"
#include "MAIO7032.H"
#include <machine.h>
#include <mathf.h>
#include "ma_cpu.h"
#include "ma_dma.h"
#include "ma_int.h"
#include "ma_itu.h"
#include "ma_tpc.h"
#include "ma_wdt.h"
#include "ma_sci.h"

unsigned long lcd_ram;
extern void whetstones(void);
void plot(unsigned char,unsigned char,unsigned char);
void display_on(void);
void display_off(void);
void MyApplication (void)
/*=====*/
{
    unsigned short reg;
    float rad;
    unsigned char *byte,*rx,*tx;
    /*output messages for terminal control*/
    const unsigned char *ok="frame received OK!\n\r",*cl="screen cleared!\n\r",
                        *sd="spiral drawn!!\n\r",*ws="whetstones started.\n\r",
                        *wf="whetstones finished!\n\r",*sp="twirly thing drawn!\n\r",
                        *in="image inverted.\n\r";

    lcd_ram=0x0a00ffd1;/*start of lcd ram; note fiddle factor!*/
    /*--- Insert more MakeApp driver initialisations here! ---*/
    MA_Init_IO();
    MA_Init_WDT();
    MA_Init_DMA();
    MA_Init_ITU();
    MA_Init_TPC();
    MA_InitCh0_SCI(NULL,0,NULL,0);
    MA_Init_INT();
```

```
/*this bit just enables sci1 pin to be used for hdi*/
reg = PBCR1;
reg |= 0x00a0; /*set bits 5 and 7 high*/
reg &= 0xffaf; /*clear bits 4 and 6 low*/
PBCR1 = reg;
/*****OK, start code!!*****/
display_on();

/*main loop. Notice no lcd refresh functions need to be placed here*/
while(1)
{
MA_GetCharCh0_SCI(rx,TRUE); /*wait on SCI0 for a character*/
switch (*rx)
{
case 's' : byte=(unsigned char*)0xa010000;
            break; /*start receiving frame from serial port*/
case 'f' : MA_PutStringCh0_SCI(ok);
            byte=(unsigned char*)0x0000000;
            break; /*finish receiving frame from serial port*/
case 'c' : byte=(unsigned char*)0xa010000;
            for(reg=0;reg<9600;reg++)
            {
                *byte&=0x0f;
                byte++;
            }
            MA_PutStringCh0_SCI(cl);
            break; /*clear screen / LCD RAM*/
case 'i' : byte=(unsigned char*)0xa010000;
            for(reg=0;reg<9600;reg++)
            {
                *byte^=0xf0;
                byte++;
            }
            MA_PutStringCh0_SCI(in);
            break; /*invert screen by toggling LCD RAM*/
case 'y' : display_on();
            break;
case 'n' : display_off();
            break;
case 't' : for (rad=0;rad<120;rad+=0.0157)
            {
                plot ((unsigned char)(120+rad*cosf(rad)),(unsigned char)(80+rad*sinf(rad)),1);
            }
            MA_PutStringCh0_SCI(sp);
            break; /*draw spiral*/
default : if ((unsigned long)byte>=0xa010000)
            {
                *byte=(0xff-((*rx-'0')*0x10));
                byte++;
            }
            /*if none of the above assume character is display
            data being downloaded. - Put it in LCD RAM & increment pointer*/
        }
    }
}
```

```
/*interrupt handler function from ITU 2 compare match A*/
/*starts DMA sending data to NDRB for 9600 counts.....*/
void start_frame(void)
{
int test;
test=0;
MA_Set_DMA(MA_CHANNEL_0_DMA,lcd_ram,0x05ffff4,9600);
MA_Start_DMA(MA_CHANNEL_0_DMA);/*start DMA'ing to NDRB on GRAB ITU0*/
}

void display_on(void)
{
const unsigned char *dy="display on.\n\r";
MA_Start_ITU(0);/*CL2*/
MA_Start_ITU(1);/*CL1*/
MA_Start_ITU(2);/*FLM*/
MA_Start_ITU(3);/*M*/
MA_WritePort_IO(0xa,0x0000);/*switch -Vee on-active low*/
MA_WritePort_IO(0xb,0x0080);/*switch display on-active high*/
MA_Set_DMA(MA_CHANNEL_0_DMA,lcd_ram,0x05ffff4,9600);
MA_Start_DMA(MA_CHANNEL_0_DMA);/*start DMA'ing to NDRB on GRAB ITU0*/
MA_PutStringCh0_SCI(dy);
}

void display_off(void)
{
const unsigned char *dn="display off.\n\r";
MA_WritePort_IO(0xb,0x0000);/*switch display off-active high*/
MA_WritePort_IO(0xa,0x0100);/*switch -Vee off-active low*/
MA_Stop_DMA(MA_CHANNEL_0_DMA);/*stop DMA'ing to NDRB on GRAB ITU0*/
MA_Stop_ITU(3);/*M*/
MA_Stop_ITU(2);/*FLM*/
MA_Stop_ITU(1);/*CL1*/
MA_Stop_ITU(0);/*CL2*/
MA_PutStringCh0_SCI(dn);
}

/*function used to plot a pixel on the screen*/
/* use plot ( x position 0 - 239
           y position 0 - 159
           val: 0-clear pixel,1-set pixel,2-toggle pixel)*/
void plot(unsigned char x,unsigned char y,unsigned char val)
{
unsigned char *mem,data;
if ((x<240) && (y<160) && (val<3)) /*make sure valid arguments passed*/
{
mem=(unsigned char*)(0xa010000 + ((x/4)+(60*y)));
switch (val)
{
case 1: *mem|=(0x80>>(x%4)); /*set pixel*/
break;
case 0: *mem&=(0x7f>>(x%4)); /*clear pixel*/
break;
case 2: *mem^=(0x80>>(x%4)); /*toggle pixel (EXOR)*/
break;
}
}
}
```

SH_HWCFG.C

```
#include <machine.h>
#define monitor_vect (unsigned int *)0x00000000
#define user_vect (unsigned int *)0x0a000000
/*
** Do not modify the function name in this source code file! */
/*This function has been modified to copy the monitor interrupt vectors across to
the user vector area (0x0a000000) so that when the VBR (vector base
register) is changed to the user area, the monitor can still operate. The
monitor requires NMI, UBC , TRAP #32, SCI1 .
User IRQ's and peripheral interrupt sources are not overwritten as
they don't occupy the area copied.*/

void hardware_setup(void)
{
const unsigned int vectors[]={ 11,12,32,104,105,106,107};
/*vector numbers of .....nmi,ubc,trap32, sci1 vectors.....*/

unsigned char count;
unsigned int *source, *destination;

/*copy vectors listed in array to allow monitor to take control*/
for (count=0;count<(sizeof(vectors)/4);count++)
{
destination=user_vect+vectors[count];
source=monitor_vect+vectors[count];
*destination=*source;
}

set_vbr(user_vect);    /*set vector base register to start of ram*/
}
```

When using this document, keep the following in mind,

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

Copyright ©Hitachi, Ltd.,1995. All rights reserved