

H8S Family

Sample Firmware for the M66592 USB ASSP

Introduction

This Application Note consists of two parts: Part I, Connection of the H8S/2218 and the M66592, and Part II, Sample Firmware for the M66592.

This Application Note describes how to use the Renesas generally applicable sample firmware for the M66592 ASSP (hereafter called USB firmware), a sample program for controlling the USB interface of the M66592.

The H8S/2218 is used for control.

Target Device

H8S/2218

Contents

Connection of the H8S/2218 and the M66592

1. Overview	2
2. Development Tools for the USB Firmware.....	4
3. Programming the Internal Flash Memory.....	12
4. Restrictions.....	14

Sample Firmware for the M66592

1. Overview	15
2. Executing the USB Firmware.....	17
3. Data Transfer.....	18
4. Class/Vendor Requests	23
5. User-Defined Information.....	24
6. User-Defined Macro Directives (macusr.h).....	26
7. Pipe Definition (def_ep.h).....	28
8. Descriptor Definitions (descrip.h).....	31
9. Low Power Sleep Function (PCUT).....	33
10. Restrictions.....	34

Connection of the H8S/2218 and the M66592

1. Overview

Related Documents:

1. *H8S, H8/300 Series High-performance Embedded Workshop 3 User's Manual*
2. *H8S, H8/300 Series High-performance Embedded Workshop 3 Tutorial Manual*
3. *H8S, H8/300 Series C/C++ Compiler Package Application Note*
4. *Solution Engine™ H8S/2218 CPU Board Overview*
5. *Universal Serial Bus Revision 2.0 specification*

1.1 Features of the USB Firmware for the H8S/2218

The USB firmware for the H8S/2218 (hereafter referred to as the USB firmware) has the following features:

- Configuration does not include specification of the peripherals; so the user can define them individually.
- The connection can be checked by using the USBCommandVerifier.exe (hereafter called USBCV; it can be downloaded from: <http://www.usb.org/developers/developers/tools/>).
- Sample programs for bulk and interrupt-driven input and output data transfer.
- Files are divided into functional groups (refer to the list under section 1.4, File Configuration).
- The USB firmware runs on the H8S/2218, and evaluation can be conducted by connecting the M66592 to the H8S/2218 CPU board (M2218CP01 Solution Engine® manufactured by Hitachi ULSI Systems, Co., Ltd.) via a signal connection board (created by the user), or the H8S/2218 E10A-USB emulator.

Note: Solution Engine® is a registered trademark of Hitachi ULSI Systems Co., Ltd.

1.2 Objectives of the USB Firmware

The USB firmware was developed with the following objectives:

- to facilitate the development of USB communications programs for the M66592; and
- to give a specific example of M66592 control as a supplementary description.

1.3 Services Provided by the USB Firmware

The services provided by the USB firmware to the upper layer (user-program layer) are:

- initialization of the M66592 (reset, oscillation control, pipe initialization, etc.);
- response to requests (standard requests defined in the USB Revision 2.0 specification);
- handling of data transfer (bulk, interrupt-driven, CPU access);
- status notification (status notification function); and
- request notification (request notification function).

1.4 File Configuration

Files for the USB firmware for the H8S/2218 can be categorized as main files, modified or added files, and the workspace files for the H8S/2218 (generated by using High-performance Embedded Workshop 3, or HEW).

Table 1 File Configuration List

Category	File Name	Description
Main files	changeep.c	User application processing
	classvender.c	Processing of class/vendor requests
	controlrw.c	Control of reading and writing
	dataio.c	Processing to read or write data
	datable.h	Definition of the user buffer for use in transmission and reception
	def592.h	Definitions of M66592 register addresses and bits
	def_ep.h	Data definition for pipe setting
	descrip.h	Descriptor data definition
	extern.h	Definitions of external references
	global.c	Processing of global variables
	Intrn.c	Handlers for the INTR, INTN, and BEMP interrupts
	usbsig.c	USB signal processing
	libassp.c	Processing of USB ASSP register manipulation
	lib592.c	Processing of USB ASSP register manipulation
	libassp.h	Definitions of USB ASSP register manipulation
	macurs.h	Definitions of user macros
	status.c	Processing to manipulate the internal state
	stdreqget.c	Processing of standard requests
	stdreqset.c	Processing of standard requests
	typedef.h	Definitions of variable types
	version.h	Definition of the version information
	Modified files	usbint.c
Main.c		Pseudo user application
defusr.h		Definitions of user settings
Added file	2218S.H	Definitions of H8S/2218 registers
HEW3 workspace files for the H8S/2218	dbstc.c	Settings of sections B and R
	resetprg.c	Reset program
	Sbrk.c	Program of sbrk
	Sbrk.h	Header file for sbrk
	stacksct.h	Setting of stack area
	Fw592_H8S2218.hws(.hbp/.tws)	Workspace file
	Fw592_H8S2218.hwp(.pgs/.tps)	HEW project file
	defaultSession.hsf	Session file
	\debug, \relase	Absolute file generation folders

2. Development Tools for the USB Firmware

The USB firmware for the H8S/2218 was developed and evaluated with the development tools below.

Table 2 Tools Used in Developing the USB Firmware for the H8S/2218

Category	Type No.	Name	Remark
Hardware	M3A-0038G01	M66592 utility board	—
	MS2218CP01	H8S/2218 Solution Engine® (CPU board)	—
	—	Signal connection board	Created by the user
	HS0005KCU01H	E10A-USB	Emulator
Software	—	High-performance Embedded Workshop ver.3.0.06 (release 2)	Development tool
	—	H8S, H8/300H Standard Toolchain (V.6.0.3.0)	Toolchain
	HS0005KCU01SR	HDI	Emulator software

2.1 Hardware Configuration

2.1.1 CPU Board and Utility Board

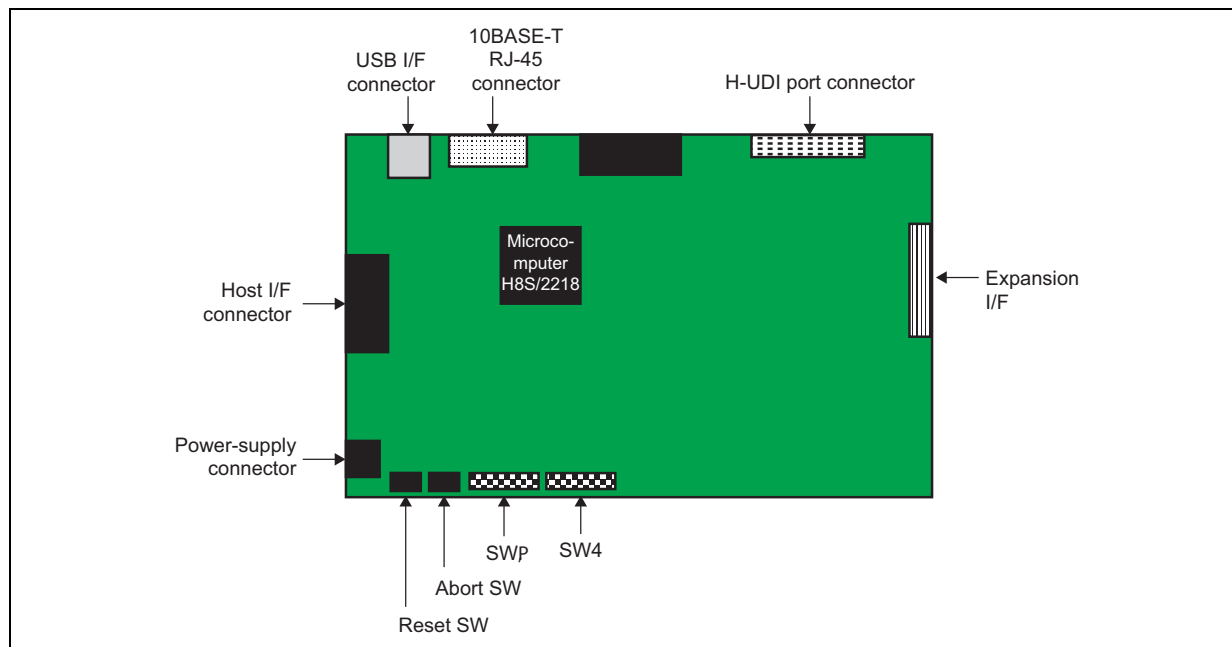


Figure 1 MS2218CPU01 CPU Board

Note: Set the switches only after turning the power supply OFF.

Table 3 MS2218CPU01 SW1 Setting Contents

SW	Name	Function	Setting
SW1-1	MD0	MCU mode switchover (MCU mode: 6)	On
SW1-2	MD1		Off
SW1-3	MD2		Off
SW1-4	FEW	Flash write enable	Off
SW1-5	EMLE	H-UDI function enable	Off
SW1-6	—	Not used	Off
SW1-7	—	Not used	Off
SW1-8	—	Not used	Off

Note: Refer to the *Solution Engine® H8S/2218 CPU Board Overview*.

Table 4 Pin Assignment of Expansion Slot CN12 on the MS2218CP01 CPU Board

Pin	Signal Name	Terminal No.	Pin	Signal Name	Terminal No.
1	GND	—	2	φ	89
3	GND	—	4	D0	64
5	D1	65	6	D2	66
7	D3	67	8	GND	—
9	D4	68	10	D5	69
11	D6	70	12	D7	71
13	GND	—	14	D8	72
15	D9	73	16	D10	74
17	D11	75	18	GND	—
19	D12	76	20	D13	77
21	D14	78	22	D15	79
23	GND	—	24	3.3 V	—
24	3.3 V	—	26	GND	—
27	A0	10	28	A1	11
29	A2	12	30	A3	13
31	GND	—	32	A4	17
33	A5	18	34	A6	19
35	A7	20	36	GND	—
37	A8	37	38	A9	38
39	A10	39	40	A11	40
41	GND	—	42	A12	49
43	A13	50	44	A14	51
45	A15	52	46	GND	—
47	A16	1	48	A17	100
49	nCS1	27	50	nCS3	25
51	GND	—	52	nWAIT	95
53	3.3 V	—	54	nRD	92
55	nIRQ0	6	56	nIRQ1	8
57	nIRQ2	97	58	nRES	58
59	GND	—	60	nHWR	93
61	nLWR	94	62	nAS	91
63	GND	—	64	3.3 V	—

Note: Type No.: 14-5014-064-102-861 (female type) manufactured by Kyocera Elco Ltd.

2.1.2 Utility Board to be Used

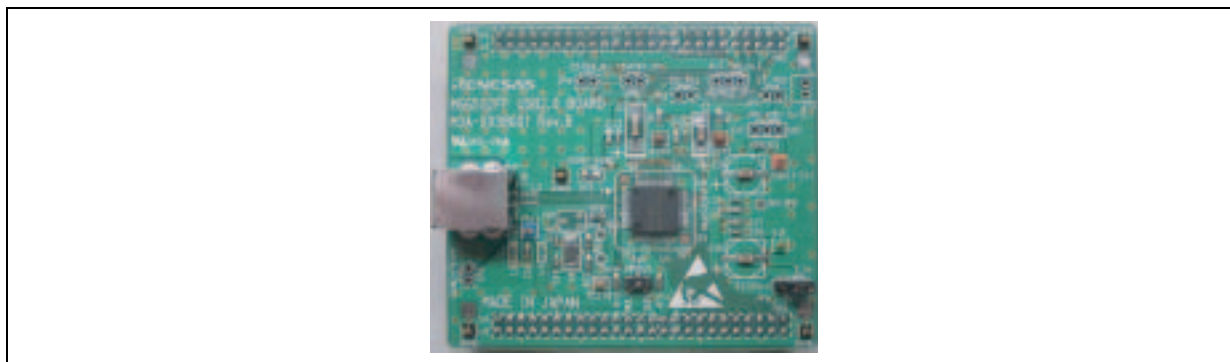


Figure 2 M3A-0038G01 Utility Board

Table 5 Pin Assignment of CN2 on the M3A-0038G01 Utility Board

Pin	16-bit sep.* ¹	16-bit mult.* ²	Pin	16-bit sep.* ¹	16-bit mult.* ²
1	GND	GND	2	D15	D15
3	D14	D14	4	D13	D13
5	D12	D12	6	D11	D11
7	D10	D10	8	D9	D9
9	D8	D8	10	GND	GND
11	D7	D7	12	D6	D6/AD6
13	D5	D5/AD5	14	D4	D4/AD4
15	D3	D3/AD3	16	D2	D2/AD2
17	D1	D1/AD1	18	D0	D0
19	GND	GND	20	GND	GND
21	Not used	Not used	22	Not used	Not used
23	WR1_N	WR1_N	24	VBUS	VBUS
24	EXIOVcc	EXIOVcc	26	EXIOVcc	EXIOVcc
27	—	—	28	—	—
29	GND	GND	30	GND	GND
31	—	—	32	—	—
33	—	—	34	—	—
35	—	—	36	—	—
37	—	—	38	—	—
39	—	—	40	—	—
41	SD7	SD7	42	SD6	SD6
43	SD5	SD5	44	SD4	SD4
45	SD3	SD3	46	SD2	SD2
47	SD1	SD1	48	SD0	SD0
49	GND	GND	50	GND	GND

Note: 16-bit sep.*¹: When a separate 16-bit bus is in use. 16bit-mult.*²: When a 16-bit multiplexed bus is in use.

Table 6 Pin Assignment of CN3 on the M3A-0038G01 Utility Board

Pin	16-bit sep.* ¹	16-bit mult.* ²	Pin	16-bit sep.* ¹	16-bit mult.* ²
1	WR0_N	WR0_N	2	GND	GND
3	RD_N	RD_N	4	GND	GND
5	CS_N	CS_N	6	RST_N	RST_N
7	DREQ_0	DREQ_0	8	DACK0_N	DACK0_N
9	INT_N	INT_N	10	GND	GND
11	GND	GND	12	A1	Not used
13	A2	Not used	14	A3	Not used
15	A4	Not used	16	A5	Not used
17	A6	ALE	18	GND	GND
19	EXVcc	EXVcc	20	EXVcc	EXVcc
21	Not used	(JP7-ALE)	22	—	—
23	—	—	24	SOF_N	SOF_N
24	DACK1_N/DSTB0_N	DACK1_N/DSTB0_N	26	DREQ1_N	DREQ1_N
27	—	—	28	—	—
29	GND	GND	30	GND	GND
31	JP6-EXT (external 1.5-V input)	JP6-EXT (external 1.5-V input)	32	—	—
33	—	—	34	—	—
35	DACK1_N/DSTB0_N	DACK1_N/DSTB0_N	36	DEND0_N	DEND0_N
37	—	—	38	—	—
39	—	—	40	DEND1_N	DEND1_N
41	—	—	42	—	—
43	—	—	44	—	—
45	—	—	46	—	—
47	—	—	48	—	—
49	GND	GND	50	GND	GND

Note: 16-bit sep.*¹: When a separate 16-bit bus is in use. 16bit-mult.*²: When a 16-bit multiplexed bus is in use.

2.1.3 Connecting the Utility Board with the CPU Board

The first step towards developing with and evaluating the USB firmware for the H8S/2218 is to connect the CPU board with the utility board via the signal connection board.

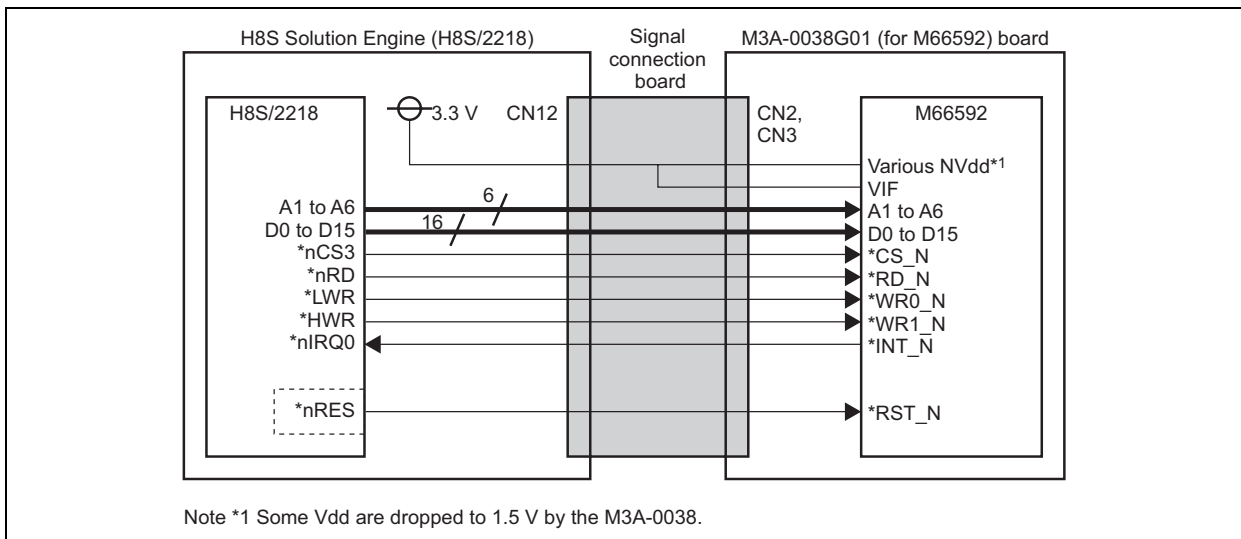


Figure 3 Wiring Connections

2.1.4 Configuring the System

The user system (target system) is connected to the host computer via the E10A-USB emulator.

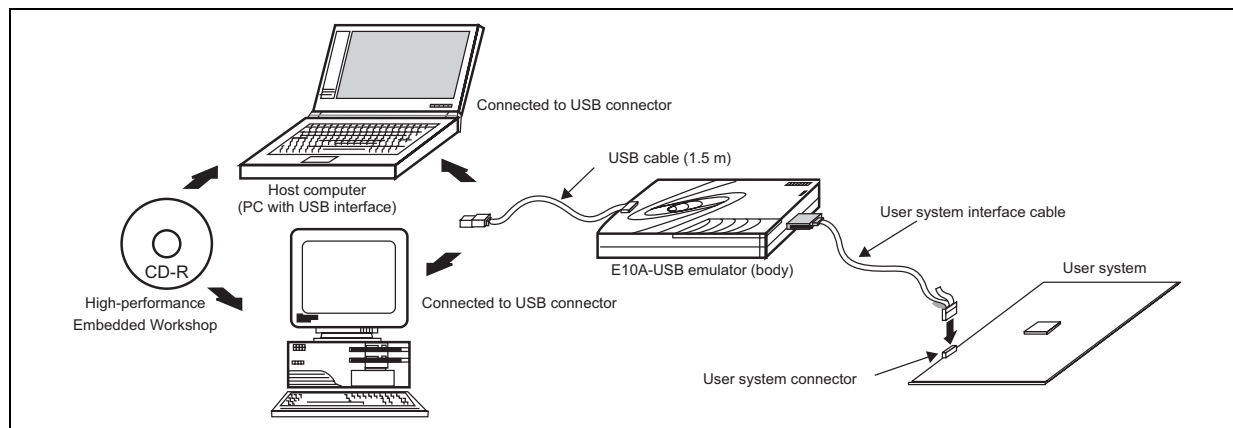


Figure 4 System Configuration

2.2 Software Configuration

The next step towards developing with and evaluating the USB firmware is to create a project with the following settings in the High-performance Embedded Workshop 3.

2.2.1 Settings in the High-performance Embedded Workshop 3

1. Setting the workspace file for the H8S/2218

Generate a workspace file for the H8S/2218 by specifying “Create a new project workspace” in the High-performance Embedded Workshop 3 and make the settings shown below on the displayed input screens to.

Table 7 High-performance Embedded Workshop 3 Settings

Input Screen	Item	Setting	Remarks
	"Application" selected	Application	
	Workspace Name	Fw592_H8S2218	Input
	Project Name	Fw592_H8S2218	Input
	CPU Family	H8S, H8/300	
	Toolchain	Hitachi H8S, H8/300 Standard	
1/9	Toolchain version	6.0.3.0	H8S, H8/300 C/C++ library generator (V.2.00.01) H8S, H8/300 C/C++ compiler (V. 6.00.03) H8S, H8/300 assembler (V.6.01.00) Optimizing linkage editor (V.8.00.07)
	CPU Series	2000	
	CPU Type	Other	2218 is not selectable (when Other is specified for CPU Type, files containing information on the CPU and I/O register definitions must be created)
2/9	Operating Mode	Advanced	
	Address Space	16 Mbyte	
	Merit of Library	Code Size	
	Stack calculation	Medium (2 bytes)	
3/9	Use I/O Library	Not checked	
	Use Heap Memory	Checked	
	[Heap Size]	H'420	
	[Generate main() Function]	None	
	I/O Register Definition Files	Not checked	
4/9	Library	[Disable all] is checked	
5/9	Stack Pointer Address	H'00FFE800	
	Stack Size	H'200	
6/9	Vector Definition Files	Checked	
7/9	Targets	All unchecked	
8/9	Target name	H8S/2218F E10A-USB System (CPU2000)	
	Configuration name	Debug_H8S_2218F_E10A-USB System (CPU2000)	
9/9	The following source files will be generated.		Pressing the Finish button ends the settings.

2. Section settings

Section settings required for developing and evaluating the USB firmware for the H8S/2218 are shown below.
Select HEW-[Tools]-[Options]-[H8S, H8/300 Standard Toolchain]-[Optimizing Linkage Editor]-Category: Section.

Table 8 Section Settings

Address	Section Name	Description
0x00000400	PRResetPRG	Reset function
	PIntPRG	Exception processing function
0x00001000	P	Program area
	C	Constants area
	C\$BSEC	For the _INITSCT function
	C\$DSEC	For the _INITSCT function
	ClntPRG	For exception processing
	D	Initialization data area (ROM)
0x00FFC000	B	Non-initialization data area
	R	Initialization data area (RAM)
0x00FFE800	S	Stack area

3. CPU information creation

The CPU information has to be created for use in debugging. Select HEW-[Tools]-[Options]-[H8S, H8/300 Standard Toolchain]-[Optimizing Linkage Editor]-Category: Verification Creation.

Table 9 CPU Information

Device	Start	End
ROM	0x00000000	0x001FFFFFFF
RAM	0x00FFC000	0x00FFFFFFF

For details, please refer to the H8S, H8/300 Series C/C++ Compiler Package Application Note. The USB firmware for the H8S/2218 should be set as shown above since it is to be used with the H8S/2218 Solution Engine.

http://documentation.renesas.com/eng/products/tool/apn/rej05b0464_h8s.pdf

2.2.2 H8S/2218 Settings

Changes have been made to certain files of the Renesas generally applicable. Sample firmware for the M66592 USB ASSP (Version 1.00) and one file has been added to make the set usable with the H8S/2218 Solution Engine by the USB firmware.

Table 10 Additions and Changes to Renesas Generally Applicable Sample Firmware for the M66592 USB ASSP

File Name	Additions and Changes	Remarks
main.c	delay_1ms() , delay_10us() are changed. CPU_init() is set to be used for the H8S/2218.	Time adjustment. Bus and software interrupt settings.
Defusr.h	Changed according to the contents	Details are given in section 2.2.3.
Usbint.c	Method of function declaration was changed. #pragma section IntPRG _interrupt (vect=16) void usbint (void)	Interrupt function is declared together with the vector (set by the compiler function).
2218S.H	H8S/2218 register definition file was added.	

1. Address map

The address map used by the USB firmware for the H8S/2218 is from address H'000000 to H'1FFFFFF, H'780000 to H'78007F, and H'C00000 to H'FFFFFF.

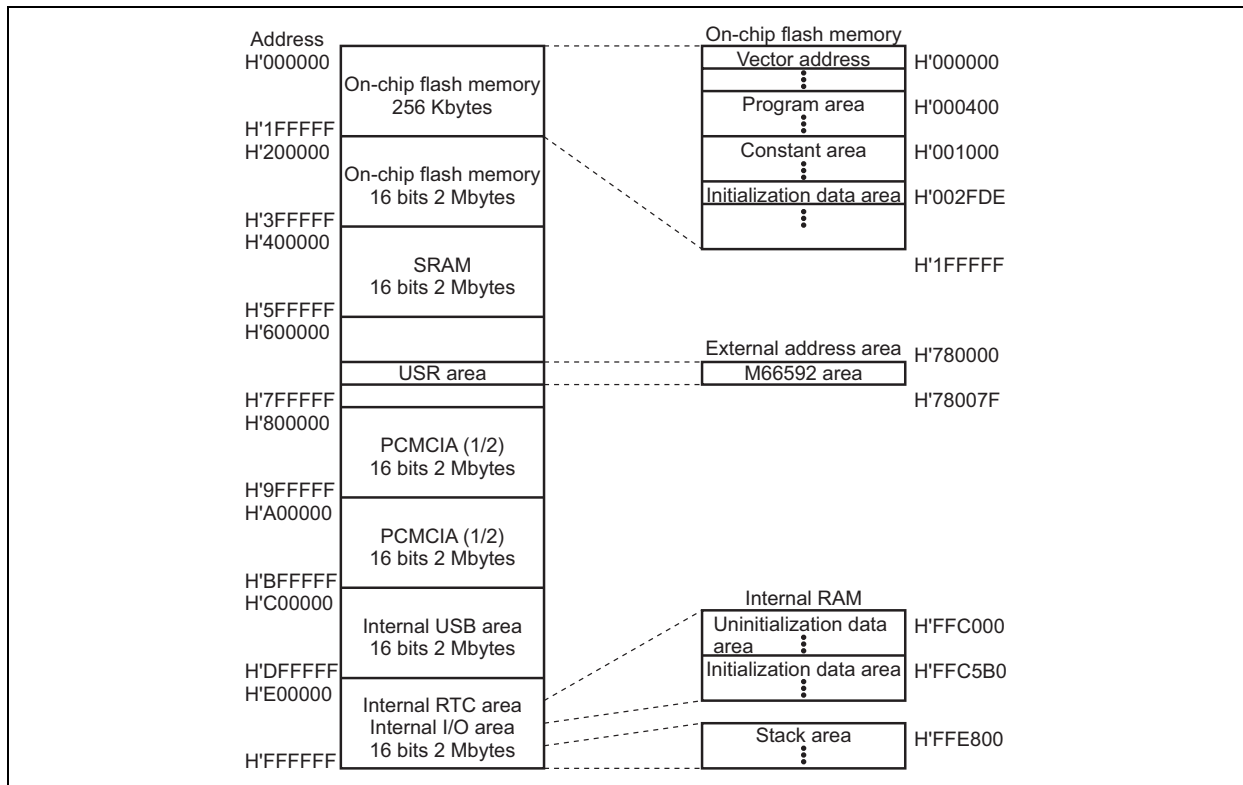


Figure 5 H8S/2218 Solution Engine Address Map

2. Bus width and inserted wait-state settings

Bus width and inserted wait-state settings of the USB firmware for the H8S/2218 for controller MCU is as follows (set in the main.c: CPU_init()).

Table 11 Bus Width and the Number of Wait States

Area No.	Area Name	Bus Width	Access State	Wait State
CS3 (H'780000 to H'78007F)	The M66592 area	16 bits	3 states	1 state

2.2.3 User Defined Items in the USB Firmware for the H8S/2218

The USB firmware for the H8S/2218 includes the following settings for the H8S/2218 Solution Engine.

Table 12 User Defined Items in the USB Firmware for the H8S/2218

Item	Settings	Remarks
Low-power sleep function specification	#define PCUT_MODE PCUT_USE	Brings the low-power sleep function into use
Automatic clock supply function specification	#define ATCKM_MODE ATCKM_USE	-
FIFO endian specification	#define FIFO_ENDIAN BIG_ENDIAN	Big endian specification (required for connection of the H8S/2218 with the M66592)
I/O power supply specification	#define VIF_SET VIF3	3.3 V is used
M66592 address	#define USB_BASE (0x780000)	H8S/2218 area 3 specification (Solution Engine setting)
Type declaration of pointer for M66592 address	typedef volatile U16 REGP;	Note that near or far declaration is not required in the case of the H8S/2218.
Oscillation frequency of connected oscillator	#define XIN XTAL24	24-MHz setting
Remote wakeup specification	#define RWUP_MODE RWUP_NOT_USE	Selects non-usage of remote wakeup

2.3 Precautions

The USB for the H8S/2218 supports only standard requests, and data communications between the USB firmware for the H8S/2218 and pseudo user applications are performed through an informal interface. Accordingly, the user must customize the interface by making required class definitions and responses to vendor-specific requests, to take communications speed, program size, and other factors into account, and to make individual settings for a particular user interface. Furthermore, to pass verification by USBCV, the user needs to set the vendor ID and the product ID in `descrip.h`.

Note: The USB firmware does not guarantee USB communications operations. When applying the firmware in a system, the user must verify operation and confirm the connections between various host controllers.

3. Programming the Internal Flash Memory

This section covers programming of the H8S/2218's internal flash memory with the user program.

3.1 Flash Memory Programming via the Serial Communication Interface

Set the switches and jumpers of the CPU board as shown below. Programming is handled by the attached zip file.

Table 13 Settings of Switches and Jumpers on the MS2218CP01 CPU Board

SW, JP	Name	Function	Setting	Remarks
SW1-1	MD0	Switch operating mode to SCI boot mode	On	
SW1-2	MD1		On	
SW1-3	MD2		Off	
SW1-4	FEW	Flash memory programming is valid	Off	
SW1-5	EMLE	Port function is enabled	On	
JP1	JP1	TxD2 is output from JP4	Short pins 1-2	
JP2	JP2	RxD2 is output from JP5	Short pins 1-2	
JP4	JP4	TxD2 is output from CN3	Short pins 1-2	
JP5	JP5	RxD2 is output from CN3	Short pins 1-2	

Notes:

1. Refer to *Solution Engine® H8S/2218 CPU Board Overview*.
2. Turn the power supply off before setting the switches and jumpers.

3.2 Using the E10A-USB for Flash Memory Programming

Set the switches on the CPU board as shown below, then start up the High-performance Embedded Workshop 3. After initiating the project, connect by selecting “Writing Flash memory”. The flash memory is programmed by downloading the download module.

Table 14 Settings of Switches on the MS2218CP01 CPU Board

SW, JP	Name	Function	Setting	Remarks
SW1-1	MD0	Switch to MCU mode 6	On	
SW1-2	MD1		Off	
SW1-3	MD2		Off	
SW1-4	FEW	Flash memory programming is valid	Off	
SW1-5	EMLE	H-UDI function enabled	Off	

Notes:

1. Refer to *Solution Engine® H8S/2218 CPU Board Overview*.
2. Turn the power supply off before setting the switches or jumpers.

To run the user program, disconnect the CPU board and terminate the High-performance Embedded Workshop 3. Set the switches on the CPU board as follows, supply power to the CPU board, and run the user program.

Table 15 Settings of Switches on the MS2218CP01 CPU Board

SW, JP	Name	Function	Setting	Remarks
SW1-1	MD0	Switch to MCU mode 6	On	
SW1-2	MD1		Off	
SW1-3	MD2		Off	
SW1-4	FEW	Flash memory programming is valid	Off	
SW1-5	EMLE	Port function enabled	On	

Notes:

1. Refer to *Solution Engine® H8S/2218 CPU Board Overview*.
2. Turn the power supply off before setting the switches or jumpers.

4. Restrictions

The following restrictions apply to the USB firmware for the H8S/2218.

1. The isochronous transfer may not operate correctly (in certain operating states, CPU processing becomes a bottleneck and packets are lost).
2. Operation with a split bus is not supported.
3. Multiple instances of the same endpoint number cannot be used simultaneously. For example, normal operation is not guaranteed when endpoint 1 is specified for BULK IN at pipe 1 and endpoint 1 is specified for BULK OUT at pipe 2.
4. DMA transfer is not supported.

Sample Firmware for the M66592

1. Overview

Related Documents:

1. Universal Serial Bus Revision 2.0 specification
<http://www.usb.org/developers/docs>

1.1 Features of the USB Firmware

The USB firmware has the following features:

- Configuration in which the controller MCU and the peripherals are not specified (individually defined by the user).
- The connection can be checked by using USBCommandVerifier.exe (hereafter called USBCV; this program is available for downloading from: <http://www.usb.org/developers/docs>)
- Sample program demonstrating bulk and interrupt-driven input and output data transfer
- Files are divided into functional groups (refer to the file configuration list).
- The firmware eliminates the need for the user program to directly access the registers of the M66592.

1.2 Layers

The USB firmware includes the layers shown below.

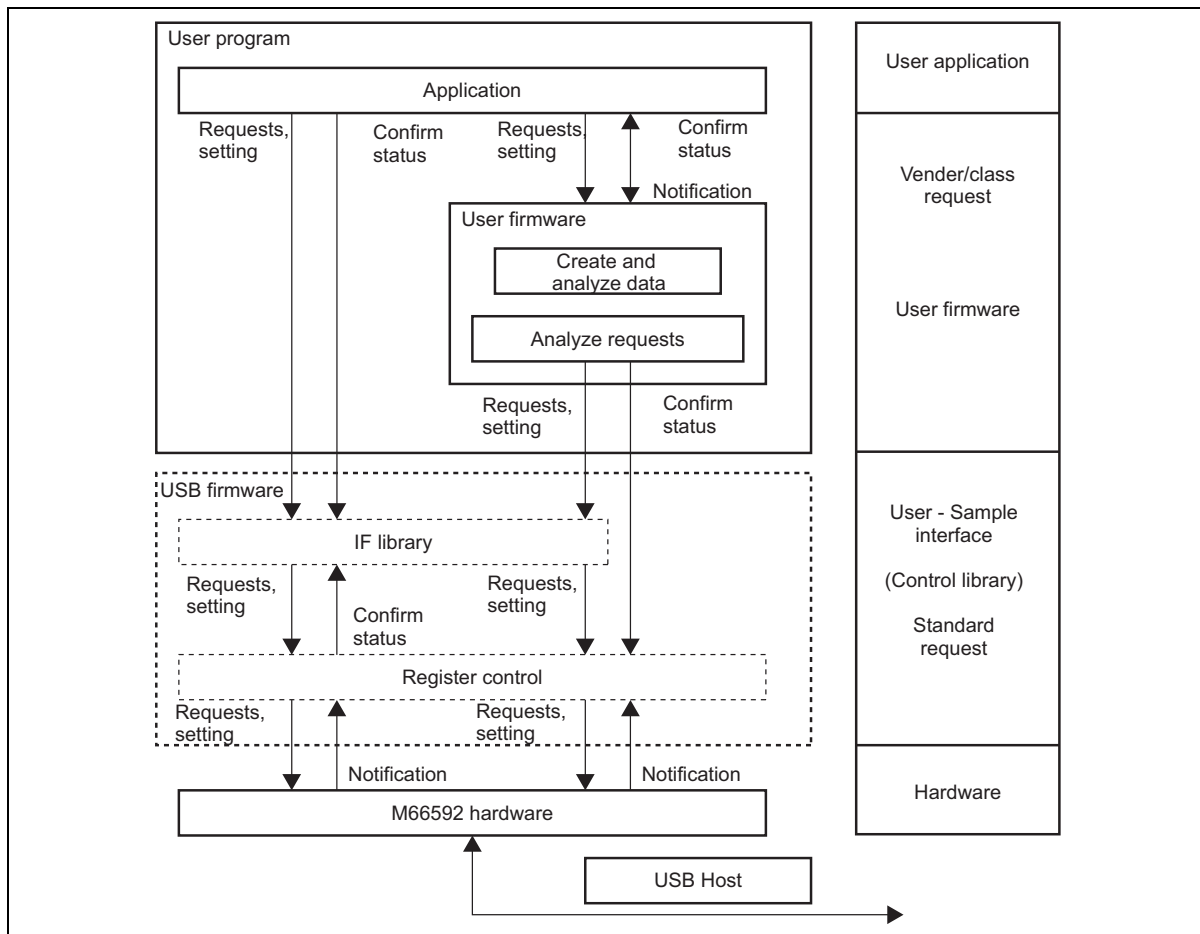


Figure 1 USB Firmware Configuration

1.3 File Configuration List

Table 1 File Configuration List

File Name	Description
changeep.c	User application processing
classvender.c	Processing of class/vendor requests
controlrw.c	Control of reading and writing
dataio.c	Processing to read or write data
datable.h	Definition of the user buffer for use in transmission and reception
def592.h	Definitions of M66592 register addresses and bits
def_ep.h	Data definition for pipe setting
defusr.h	User setting definition
descrip.h	Descriptor data definition
extern.h	Definitions of external references
global.c	Processing of global variables
intrn.c	Handlers for the INTR, INTN, and BEMP interrupts
usbsig.c	USB signal processing
libassp.c	Processing of USB ASSP register manipulation
lib592.c	Processing of USB ASSP register manipulation
libassp.h	Definitions of USB ASSP register manipulation
macusr.h	Definitions of user macros
main.c	Pseudo user application
status.c	Processing to manipulate the internal state
stdreqget.c	Processing of standard requests
stdreqset.c	Processing of standard requests
typedef.h	Definitions of variable types
usbint.c	Handler for USB interrupts
version.h	Definition of the version information

1.4 Objectives in Developing the USB Firmware

The USB firmware was developed with the following objectives:

- to facilitate the development of USB communications programs for the M66592; and
- to give a specific example of M66592 control as a supplementary description.

1.5 Service Outline

The USB firmware provides the following services to the upper layer (user program layer):

- initialization of the M66592 (reset, oscillation control, pipe initialization, etc.);
- response to requests (standard requests defined in the USB Revision 2.0 specification);
- handling of data transfer (bulk, interrupt-driven, CPU access);
- status notification (status notification function); and
- request notification (request notification function).

1.6 Processing Flow in Outline

In the USB firmware, the control functions for sending and receiving USB data are implemented in a set of interrupt handlers. The interrupt events take the form of external interrupts generated by the M66592 for the controller MCU. The interrupt source is determined by the external interrupt handler and the corresponding processing is performed.

Special Signal Processing: Vbus interrupts, resume interrupts, SOF detection interrupts*, device state transition interrupts.

Note: * This processing is not implemented in the USB firmware. The user needs to create code to handle this processing as required.

Control Transfer Processing: Control-transfer stage-transition interrupts and device state-transition interrupts provide the triggers for data transfer.

Pipe Transfer Processing: Buffer-empty/size-error interrupts, buffer-not-ready interrupts, and buffer-ready interrupts provide the triggers for data transfer.

Thus, USB control processing by the USB firmware is handled by a set of USB interrupt routines. The main function performs the initial settings for the controller MCU and the M66592-related registers, then places the MCU in an endless loop.

An outline of the flow of the USB firmware is given below.

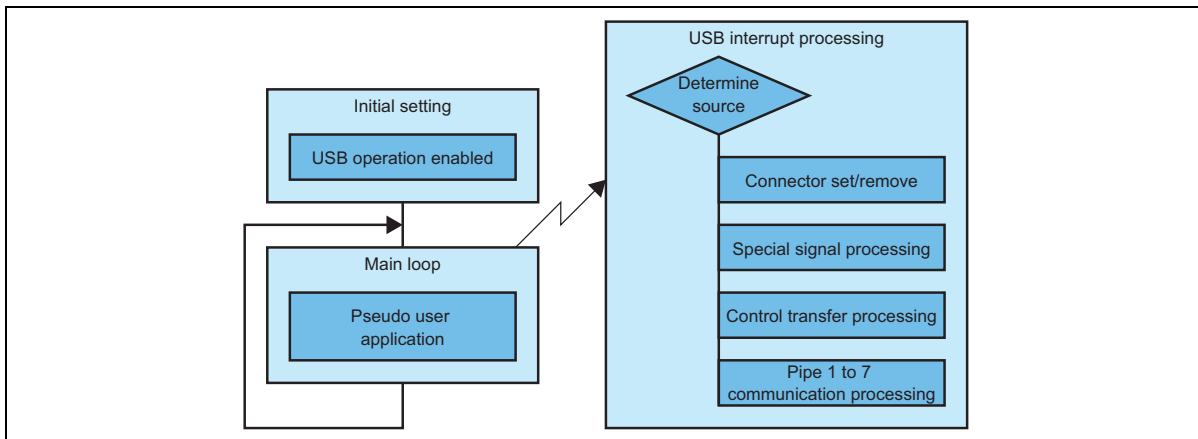


Figure 2 Flow Outline

2. Executing the USB Firmware

Initialization of The controller MCU and peripherals (in main.c) must be initialized, and the user definition information file (defusr.h) and user definition macro file (macusr.h) must be modified to enable transfer by the USB firmware, and is capable of passing verification by the USBCV program.

2.1 Changing the USB Firmware

The following program files and header files need to be changed so that the USB firmware is operable and the setup is capable of passing verification by USBCV.

1. The below functions within main.c have to be changed:
 - initialization of the controller MCU (CPUInit function);
 - initialization of the peripherals (PeripheralInit function);
 - enabling of controller MCU interrupts (enableINT function); and
 - adjusting the time setting in the function that waits for a specified time (functions delay_1ms and delay_10us).
 The wait for a specified time is implemented by loop processing, so the number of loops must be adjusted to obtain times that suit the user system.
2. Certain files require user customization.
 - Vendor ID and product ID in descrip.h (for details, refer to section 5, User-Defined Information).
 VenderID and ProductID
 - Endian, I/O voltage, register base address, 'far' declaration if required, constant representing the frequency of oscillation, and interrupt function specifications in defusr.h (for details, refer to section 5, User-Defined Information).
 FIFO_ENDIAN, VIF_SET, USB_BASE, REGP, XIN, and INTERRUPT.
3. Requirements before building
 - Specification of section areas
 - Creation of a startup routine
4. Others
 - Special signal processing (not necessary at the level of confirming firmware operation (passing verification by USBCV))

2.2 Precautions

The USB firmware is generally applicable firmware in that the controller MCU and peripherals are not specified. Only the standard requests are supported. Data communications proceed via a virtual interface between the USB firmware and the user application. Accordingly, the user must customize the interface by making required class definitions and responses to vendor-specific requests, to take communications speed, program size, and other factors into account, and to make individual settings for a particular user interface.

Note: The USB firmware does not guarantee the USB communications operation. Before applying the firmware in a system, the user must verify operation and confirm the connections between various host controllers.

3. Data Transfer

The USB firmware facilitates simple data communications between a host computer and the device. The user must prepare a USB driver for the host computer and a data-transfer application. For details on USB firmware device configuration, refer to section 8, Descriptor Definitions.

By changing the necessary information for device configuration (descriptor definitions (descrip.h) and pipe definitions (def_ep.h), and the pseudo user application (main.c), a simple data-communications application that is specifically for the user system (system on the host computer side) can be achieved.

Note that data communication clearly requires a set of user-specific functions. Accordingly, the user has to modify specifications such as the transfer method, requests for start and end of communications, and buffer configuration.

3.1 Basic Specifications of the USB Firmware

- The USB firmware achieves data transfer between the user buffers and a FIFO port register by CPU access. The flow of data is shown in the figure below.
- An area of at least 512 bytes is reserved for the user buffer of each pipe (the area of the user buffers should be greater than that of the FIFO buffers).
- Transfer through a pipe of larger amounts of data than the size of the FIFO buffer is realized by changing the size of the user buffer.
- Fixed data corresponding to each pipe is used as data for transmission to the host.
- The user-buffer address-notification function (DI_Start/DO_Start) is for common access in both CPU- and DMA-driven transfer.
- The buffer-ready interrupt of each pipe is enabled before the start of data transmission or reception, since these interrupts are used in the transmission and reception of data.

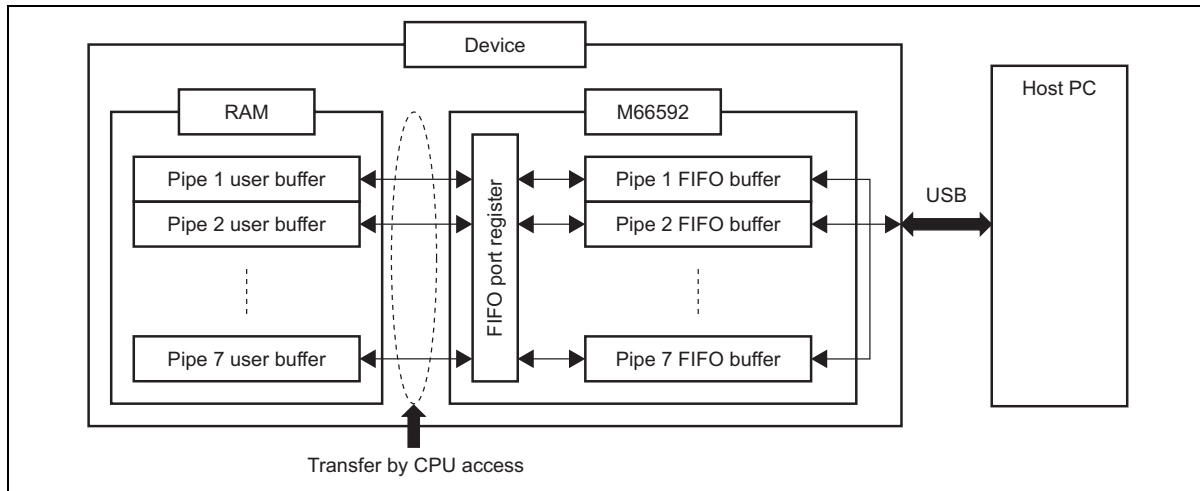


Figure 3 Data Flow

3.2 Data Transmission Operation (with IN token)

The USB firmware follows the below procedure in transmitting data to the host computer (i.e. data with the IN token).

1. Checks whether the user buffer is available for use (the buffer is available when Buffer_Write_Data_Flag is DATA_NONE). If the first user buffer to be tried is not available, the processing is performed on another pipe.
2. Checks whether there is data to be transmitted (a value other than DATA_NONE returned by the Create_In_Data function indicates that there is data to be transmitted).
3. Sets the user buffer address and the number of bytes to be transmitted (dcnt), disables the user buffer (sets DATA_WAIT in Buffer_Write_Data_Flag), and enables buffer-ready interrupts (the DI_Start function).
4. When a buffer-ready interrupt occurs, the data in the user buffer is written to a FIFO port register.
 - When the number of bytes to be transmitted (dcnt) > FIFO buffer size: Data is written to the FIFO buffer until it is full, and the FIFO buffer size is subtracted from the number of bytes to be transmitted (dcnt = dcnt – FIFO buffer size).
 - When the number of bytes to be transmitted (dcnt) ≤ FIFO buffer size: The bytes for transmission are written to the FIFO, buffer ready interrupts are disabled, and the user buffer is re-enabled (the Buffer_Write_Data_Flag is set to DATA_NONE).

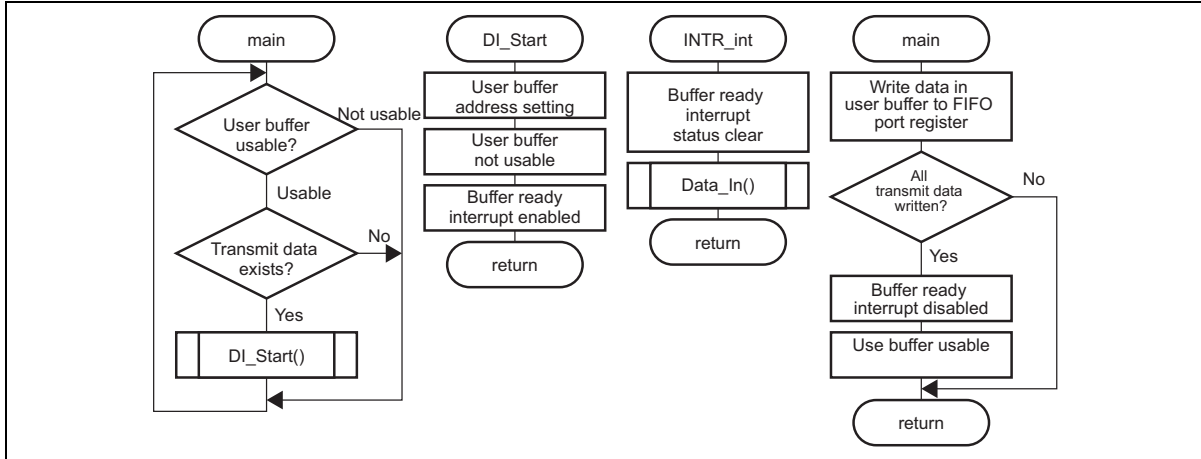


Figure 4 Flow of Control in Data Transmission

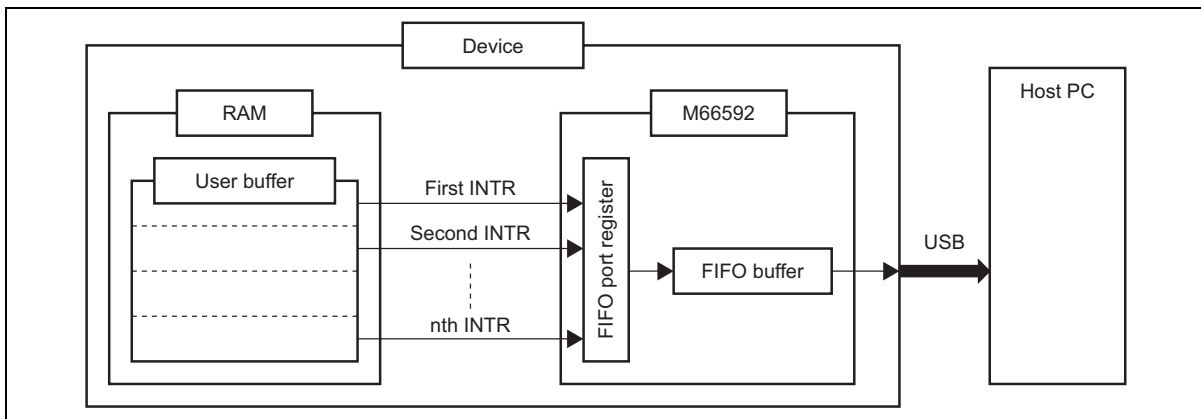


Figure 5 Flow of Data in Data Transmission

3.3 Data Reception Operation (with OUT token)

The USB firmware receives data that has the OUT token from the host computer according to the following procedure.

1. Checks whether the user buffer is available for use (availability is indicated by Buffer_Read_Data_Flag being set to DATA_NONE).
 - If a user buffer is available, checks whether data can be read from the user buffer (enabled when the Buffer_Read_Data_Flag is DATA_OK). When reading from the user buffer is disabled, processing for other pipes proceeds.
 - When the user buffer is enabled, sets the user buffer address and the number of bytes to be received (dtcnt), and enables the user buffer (sets Buffer_Read_Data_Flag to DATA_WAIT), and enables buffer ready interrupts (the DO_Start function).
2. When a buffer ready interrupt occurs, the data is read from a FIFO port register and written to the user buffer.
 - When the number of received bytes (dtcnt) > FIFO buffer size: All data is read from the FIFO buffer, and its size is subtracted from the number of bytes to be received (dtcnt = dtcnt – FIFO buffer size)
 - When the number of received bytes (dtcnt) ≤ FIFO buffer size: All data is read from the FIFO buffer, the buffer-read interrupts are disabled, and the user buffer is enabled (Buffer_Read_Data_Flag is set to DATA_OK).

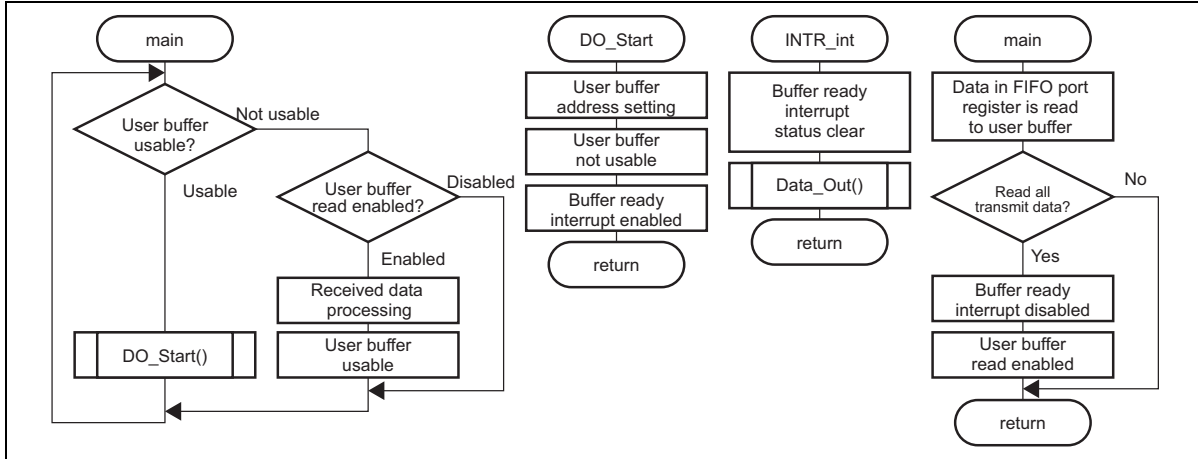


Figure 6 Flow of Control in Data Reception

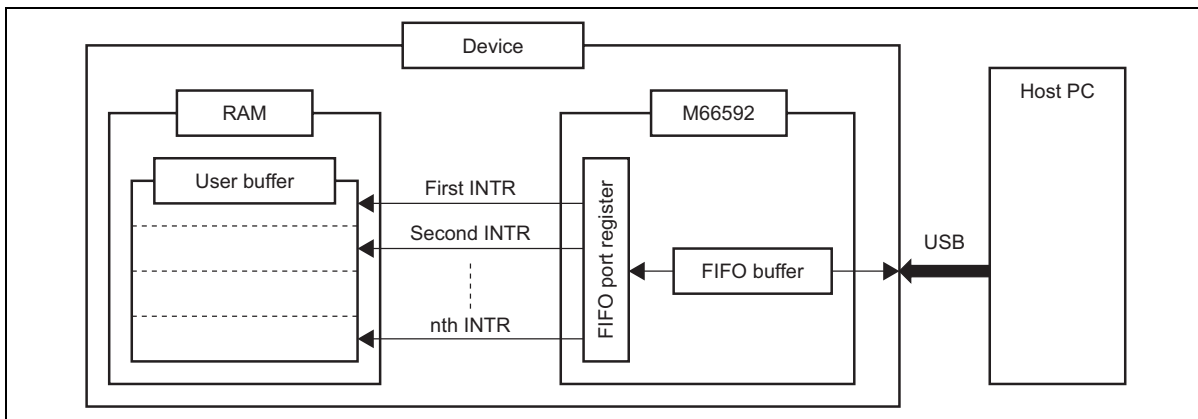


Figure 7 Flow of Data in Data Reception

3.4 Pipe Settings for Data Transfer

Requests such as Set_Configuration and Set_Interface are used when pipe settings must be changed to enable USB communications.

On receiving the above request, the USB firmware uses the descriptor table to find the pipe table index number corresponding to the pipe for which the configuration or interface is to be changed, and then automatically sets the pipe configuration register in conformance with other aspects of the pipe definition.

Pipe Setting: The Esrch function is called with the configuration number, current interface numbers, and replacement settings as parameters as follows.

```
void Esrch (configuration number, interface number, replacement settings);
void resetEP (configuration number).
```

The Esrch function searches for the index numbers of the definitions of all pipes used by the interface of the specified configuration. resetEP makes new settings in the pipe configuration register for all pipes found by Esrch. To change the pipe specifications, pipes on which transfer is currently in progress may have to be disabled in order to avoid the input of invalid data or input errors as the changes are made. Accordingly, the corresponding pipe interrupts are disabled in resetEP.

Descriptor Table: The USB firmware uses virtual descriptor definitions. The user must create user-specific descriptor definitions (descrip.h) that suit the Windows driver and applications on the host side. When changing a descriptor definition, also change the pipe definition (def_ep.h). Moreover, change the pseudo user application (the Change_Config and Change_Interface function) as required.

Note: Define descriptor and pipe definitions in the same order as pipe information is defined. For details, refer to section 7, Pipe Definition.

The Change_Config and Change_Interface Functions: When the Set_Configuration or Set_Interface request is received, the corresponding function is called by the USB firmware. Additional processing may also be required for the user application.

3.5 Precautions in Making Changes to Descriptors, Usage, and Buffer Configuration

1. The user buffer must be larger than the max. packet size (FIFO buffer in continuous transmission and reception).
2. To facilitate the confirmation of communications, a dummy area is allocated in each user buffer.
3. When changing a descriptor definition (descrip.h), be sure to change the corresponding pipe definitions (def_ep.h).
4. When changing the usage or configuration of the user buffer, it may be necessary to change the library functions.

3.6 User Buffer Specifications

An area for counting the number of accesses is also allocated. The user buffers for each pipe are initialized to different sets of values.

```
typedef struct {
    U16 size; /* Buffer size */
    U16 count; /* Buffer access counter */
    U8 Dummy[12]; /* Data area position adjustment */
    U8 buff[EP1_DATA_SIZE]; /* Data buffer area */
} ep_buff1;

ep_buff1 EP_Buff1 = {
    EP1_DATA_SIZE,
    0, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E,
    0x0F,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E,
    0x1F,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E,
    0x2F,

    ep_buff2 EP_Buff2 = {
        EP2_DATA_SIZE,
        0, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
        0x0F, 0x0E, 0x0D, 0x0C, 0x0B, 0x0A, 0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01,
        0xFF,
        0x1F, 0x1E, 0x1D, 0x1C, 0x1B, 0x1A, 0x19, 0x18, 0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11,
        0x10,
        0x2F, 0x2E, 0x2D, 0x2C, 0x2B, 0x2A, 0x29, 0x28, 0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21,
        0x20,

        ep_buff7 EP_Buff7 = {
            EP7_DATA_SIZE,
            0, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00,
            0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
            0x10,
            0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
            0x20,
```

4. Class/Vendor Requests

Responses to class/vendor requests are only possible when the USB firmware is used together with the user-provided Windows driver and application on the host computer, and user firmware on the device side. The USB firmware only provides an entrance when the basic structure of data communications at the data stage is the same as the structure of data communications at an individual pipe.

4.1 Basic Specifications

- The following functions provide for the notification of user-buffer addresses:
CR_Start (control read: user buffer address notification)
CW_Start (control write: user buffer address notification function)
- The following functions provide for the reading and writing of data:
Control_Read (control read: data write function)
Control_Write (control write: data read function)
- The user buffer must be larger than the amounts of data sent in each round of transmission from the data stage (that is, the user buffer must be larger than the amount of data transferred in a single control transfer operation).

4.2 Detailed Specifications

- The USB firmware and the user firmware communicate via the user buffer. The USB firmware handles CPU access to realize data transfer between the user buffer and the FIFO port register. Call the CR_Start or CW_Start function from the user application.
- The direction of transfer in the data stage is determined by the INTR_int and BEMP_int functions.
- During transmission and reception, interrupts drive the continuous transfer of data by the Control_Read and Control_Write functions.
- The user firmware must handle the creation of data for transmission to the host (providing data for Control_Read).
Method 1: Make the data in response to the results of requests within an interrupt handler (similar to a standard request)
Method 2: Create the data within the application (with an interrupt used to notify the user application of reception).
- The user firmware must handle the analysis of data sent from the host (support for Control_Write)
Method 1: Judge requests to receive data from within an interrupt handler (similar to a standard request)
Method 2: Have the application handle reception (with an interrupt used for notification to the user application)

4.3 Example of a User Firmware Interface with Control Read (IN direction)

An example of the approach to class/vendor requests within the user firmware is given below.

1. Control-read data stage (the ClassTrans1 function) calls the CR_Start function.
2. A buffer empty/size over interrupt is generated (the BEMP_int function).

Note: This is an example. Create the actual firmware to suit the user specifications.

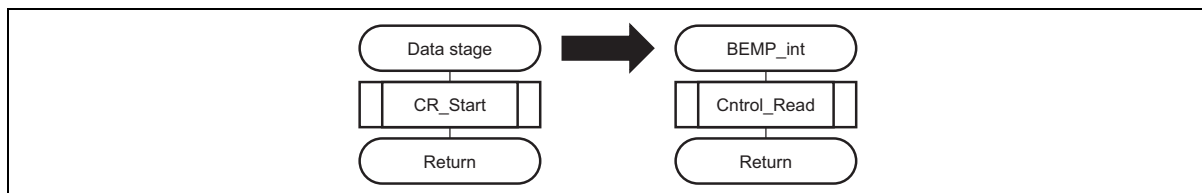


Figure 8 Flow of Transmission

Points to Note:

- After the data for transmission has been created, call the transmission start notification function (the CR_Start function).
- In the same process of notification (the CR_Start function), indicate the user buffer address and the data size.
- As the user buffer, secure a capacity larger than the amount of data (wLength) for transmission in the control read data stage.
- Use a call of the Control_End function to indicate that the control transfer is completed by the control read status stage transition of the USB interrupt function.
- The user firmware needs to store the request type (control-read setup stage information) to indicate that the data for transmission is to be created.

4.4 Example of Control Write (OUT direction) User Firmware Interface

An example of the approach to class/vendor requests within the user firmware is given below.

1. Control write data stage (the ClassTrans2 function) calls the CW_Start function.
2. After data has been received from the host PC, a buffer-ready interrupt is generated (the INTR_int function). The Control_Write function then transfers the received data to the user buffer.
3. The data that has been received into the user buffer is processed by the control write status stage (the ClassTrans5 function).

Note: This is an example. Create the actual firmware to suit the user specifications.

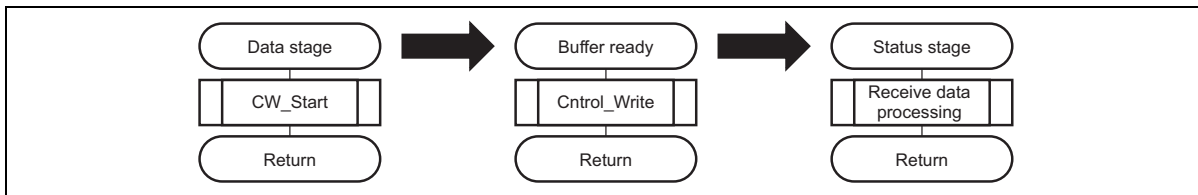


Figure 9 Flow of Reception

Points to Note:

- Indicate the buffer address and data size by calling the function for reception-start notification (the CW_Start function).
- In the same process of notification, indicate the user buffer address and the data size.
- After analysis of the received data, call the Control_End function to notify the USB firmware that the control transfer is complete.

5. User-Defined Information

The USB firmware is intended to be versatile. Accordingly, the user-definition information file is to be altered to create user-specific executable files. Change the following twelve items according to the user system.

Note: Some controller MCUs may require definitions beyond those listed below.

1. Vendor ID
2. Product ID
3. Low-power sleep function (PCUT) specification
4. Automatic clock supply function (ATCKM) specification
5. FIFO endian specification
6. I/O power supply specification
7. Address of the M66592
8. Type declaration of the pointer to the address of the M66592
9. Frequency of oscillation for the oscillator connected to the M66592
10. Declaration of interrupt vectors
11. Enabling of multiple interrupts
12. Specification of remote wakeup

5.1 Vendor ID (descrip.h)

Specify the same vendor ID as that of the user system.

E.g. To set 0x1234:

```
#define Vendor ID 0x1234
```

5.2 Product ID (descrip.h)

Specify the same product ID as that of the user system.

E.g. To set 0x5678:

```
#define Product ID 0x5678
```

5.3 Low-Power Sleep Function (PCUT) Specification

Specify whether or not to use the M66592's low-power sleep function. When PCUT_USE is specified, the automatic clock supply function is used (ATCKM_USE) regardless of the ATCKM_MODE specification.

To specify use of the M66592's low-power sleep function:

```
#define PCUT_MODE PCUT_USE
```

5.4 Automatic Clock Supply Function (ATCKM) Specification

If the low-power sleep function is not to be used, specify whether or not the M66592's automatic clock supply function is to be used.

E.g. To specify use of the M66592's automatic clock supply function:

```
#define ATCKM_MODE ATCKM_USE
```

5.5 FIFO Endian Specification (defusr.h)

Specify the byte endian for accessing the M66592's FIFO port.

E.g. To specify little endian:

```
#define FIFO_ENDIAN LITTLE_ENDIAN
```

5.6 I/O Power Supply Specification (defusr.h)

Specify the voltage on the I/O terminals of the M66592.

E.g. To set 3.3 V:

```
#define VIF_SET VIF3
```

5.7 M66592 Address (defusr.h)

Specify the base address for access to the M66592. The addresses of all registers are specified as offsets from this reference address.

E.g. To set the address 0x8000:

```
#define USB_BASE (0x8000)
```

5.8 Type Declaration of the Pointer for M66592 Register Addresses

Specify the pointer type for access to the M66592. To support MCUs that require near and far declaration, the individual registers of the M66592 are cast with the REGP type. According to the type of the controller MCU, comment out the declaration with whichever of near or far applies.

E.g. When the M66592 is allocated to the near area for an MCU which requires near or far declaration:

```
typedef volatile U16 REGP;  
/* typedef volatile far U16 REGP; */
```

5.9 Frequency of Oscillation for the Connected Oscillator (defusr.h)

Select one of the three possible values for oscillation frequency of the oscillator connected to the M66592.

E.g. To use a 24-MHz oscillator:

```
#define XIN XTAL24
```

5.10 Interrupt Vector Declaration (defusr.h)

For MCUs where the interrupt processing functions must be declared, the relevant statement must also be issued for the interrupts for USB communications provided by the USB firmware. Remove comments as required. This applies to declarations other than the #pragma directive.

E.g. When a #pragma directive is required for the interrupt processing function (usbint):

```
#pragma INTERRUPT usbint
```

5.11 Multiple Interrupts Enabled

To enable multiple interrupts during USB interrupt processing, the instruction for enabling the interrupts must be set. When the M3A-0033 and the KD308 are used, operation of the KD308 may be illegally terminated because the USB interrupt processing is judged to be taking too long. In such cases, multiple interrupts must be enabled.

E.g. To enable multiple interrupts with the M16C/80:

```
#define MULTI_INT_ENABLE( ) asm("fset i")
```

5.12 Remote Wakeup Specification (defusr.h)

Specify whether or not to use the remote wakeup function.

To specify non-usage of the remote wakeup function:

```
#define RWUP_MODE RWUP_NOT_USE
```

6. User-Defined Macro Directives (macusr.h)

The M66592 conforms to the USB Revision 2.0 specification, so registers must be accessed in little endian format. Since the USB firmware is intended to be versatile, we must assume that the endian of the controller MCU may differ from that of the M66592. Since the methods of access to registers and FIFO registers are specified in macro directives, it is possible to create a user-specific executable file by rewriting the user-defined macro header file. Change the macros used to access the registers and FIFO registers to suit the user system.

Nine macros of the following four types are used in register accesses:

- Register and FIFO data register read/write macros
- Register bit set/clear/modify macros
- Status register bit clear macro
- Status register bit set macro

6.1 Register and FIFO Data Register Read/Write Macros

These macros are used for reading or writing the data in the registers and FIFO port registers. The controller MCU and the M66592 must be connected so that DMA transfer between memory and the FIFO port registers is correctly enabled. For specifying the endian of the controller MCU refer to section 5.5, FIFO Endian Specification.

```
#define USBRD_FF(r,v) do{(v)=(r);}while(0)
#define USBWD_FF(r,v) do{(r)=(v);}while(0)
#define USBRD(r,v) do{(v)=(r);} while(0)
#define USBWD(r,v) do{(r)=(v);} while(0)
```

Note: Consider the system configuration when connecting the controller MCU, memory, and the MM66592.

6.2 Register Bit Set/Clear/Modify Macros

These macros are used in setting, clearing, or modifying the bits of registers. The macros have a RMW (read-modify-write) approach, and take advantage of the macros for reading or writing data from and to registers and FIFO registers that were described in the previous section.

- Notes:**
1. These macros will not need to be changed by the user.
 2. Do not use the above register bit-clearing macro to clear bits of the status register; instead, use the status-register bit-clearing macro described in the following section.

```
/* set bit(s) of USB register */
/* r : USB register */
/* v : value to set */
#define USB_SET_PAT( r, v ) do{ register U16 tmp; \
USBRD( r, tmp ); \
tmp |= (v); \
USBWR( r, tmp ); }while(0)
/* reset bit(s) of USB register */
/* r : USB register */
/* m : bit pattern to reset */
#define USB_CLR_PAT( r, m ) do{ register U16 tmp; \
USBRD( r, tmp ); \
tmp &= (~(m)); \
USBWR( r, tmp ); }while(0)
/* modify bit(s) of USB register */
/* r : USB register */
/* v : value to set */
/* m : bit pattern to modify */
#define USB_MDF_PAT( r, v, m ) do{ register U16 tmp; \
USBRD( r, tmp ); \
tmp &= (~(m)); \
tmp |= v; \
USBWR( r, tmp ); }while(0)
```

6.3 Status Register Bit Clear Macro

This macro has a RMW (read-modify-write) approach, and takes advantage of the macro used to write data to the registers and FIFO registers. This macro is used to avoid erasure of status changed during instruction execution by the RMW instruction.

- Notes:**
1. This macro will not need to be changed by the user.
 2. Only use this macro with status registers to which 1 cannot be written.

```
/* reset bit(s) of USB status */
/* r : USB register */
/* m : bit pattern to reset */
#define USB_CLR_STS( r, m ) USBWR( r, (~m) )
```

6.4 Status Register Bit Set Macro

This macro is used to set bits in the status register. This macro takes advantage of the register-write macro described in section 6.1, Register and FIFO Data Register Read/Write Macros. This macro is used to avoid erasure of status changed during instruction execution by the RMW instruction. Use this macro when writing 1 by clearing VBUSINT in the internal-clock-stopped state.

- Notes:**
1. This macro will not need to be changed by the user.
 2. Only use this macro with status registers to which 1 cannot be written.

```
/* set bit(s) of USB status */
/* r : USB register */
/* m : dummy */
#define USB_SET_STS( r, m ) USBWR( r, 0xffff )
```

7. Pipe Definition (def_ep.h)

The configuration registers of the M66592 are used to make various settings for the pipes. Since the USB firmware has been written as generally applicable firmware, the possibility of changes in pipe settings due to changes in configuration and replacement have been taken into account. The pipe information (usage) is configured as a set of data tables in the pipe-definition header file (def_ep.h), and the user creates the user-specific execution file by rewriting the definitions in the file.

The various settings for pipes require settings for speed, one for FULL speed (Eptbl_Full_n) and one for HIGH speed (Eptbl_Hi_n). Change the definitions of pipes to suit the user system.

Note: When the definitions of pipes are changed, descriptor definitions (descrip.h) must be changed accordingly.

The default control-pipe definitions include the following two items (U16 x 2):

- CFIFO port selection (address 0x1E)
- DCP configuration register (address 0x5C)

Definition of pipes 1 to 7 consist of the following five items (U16 x 5):

- Pipe window selection register (address 0x64)
- Pipe configuration register (address 0x64)
- Pipe buffer setting register (address 0x68)
- Pipe maximum packet size register (address 0x6A)
- Pipe period control register (address 0x6C)

7.1 Default Control-Pipe Definition

Default control-pipe definitions are configured of tables. The table for default control pipe information which is given as a sample in USB firmware is shown below.

E.g.

```
const U16 DCPTbl[] = {
/* CFIFOSEL (0x1E) */
    MBW_16,          Defined item 1
/* DCPCFG (0x5C) */
    CNTMD           Defined item 2
};
```

7.1.1 Default Control-Pipe Defined Item 1

Specify a setting for the CFIFO port selection register as follows:

FIFO port access bit width: If port is accessed is in 8-bit units, specify MBW_8; if it is in 16-bit units, specify MBW_16.

E.g. When the FIFO port is accessed with a 16-bit width:

```
MBW_16,
```

7.1.2 Default Control-Pipe Defined Item 2

Specify a setting for the DCP configuration register as follows:

Continuous transmission/reception mode: for continuous reception, specify CNTMD.

E.g. For continuous reception:

```
CNTMD
```

Note: For specification of the max. packet size of the default control pipe, refer to section 8, Descriptor Definitions.

7.2 Definition of Pipes 1 to 7

In a similar way to the descriptor definitions, each pipe definition is configured of tables, and is described in order of related interfaces and replacement settings. The pipe definitions given as a sample in USB firmware are shown below. The items defined for each pipe are described in the following sections.

E.g.

```

/* Configuration 1 */
const U16 EPtbl_Full_1[] = {
/* Interface 1-0-0 */
/* Endpoint 1-0-0-0 */
/* Pipe Window Select Register (0x64) */
PIPE1,
/* Pipe Configuration Register (0x66) */
BULK | DBLB | CNTMD | DIR_IN | EP1,
/* Pipe Buffer Configuration Register (0x68) */
BUF_SIZE(512) | 6,
/* Pipe Maxpacket Size Register (0x6A) */
64,
/* Pipe Cycle Configuration Register (0x6C) */
OFF | 0,
: : :
};
/* Configuration 1 */
const U16 EPtbl_Hi_1[] = {
/* Interface 1-0-0 */
/* Endpoint 1-0-0-0 */
/* Pipe Window Select Register (0x64) */
PIPE1,
/* Pipe Configuration Register (0x66) */
BULK | DBLB | CNTMD | DIR_IN | EP1,
/* Pipe Buffer Configuration Register (0x68) */
BUF_SIZE(512) | 6,
/* Pipe Maxpacket Size Register (0x6A) */
512,
/* Pipe Cycle Configuration Register (0x6C) */
OFF | 0,
: : :
};

```

For full speed

Pipe definition item 1

Pipe definition item 2

Pipe definition item 3

Pipe definition item 4

Pipe definition item 5

For high speed

7.2.1 Pipe Definition Item 1

Specify a value for setting in the pipe window selection register as follows.

Pipe Selection: Specify the pipe to be selected (PIPE1 to PIPE7).

E.g. To specify pipe 1:

```
PIPE1,
```

7.2.2 Pipe Definition Item 2

Specify pipe configuration register settings as follows.

Transfer Type: Specify either BULK, INT, or ISO.

Double Buffer Mode: To specify double-buffering, set DBLB; for a single buffer, set this to OFF.

Continuous Transmission/Reception Mode: For the single transmission/reception mode, set this to OFF; for the continuous transmission/reception mode, set CNTMD.

Transfer Direction: If the pipe is to be an input, set DIR_IN; if it is to be an output, set DIR_OUT.

Endpoint Number: Specify the endpoint number (EP1 to EP15).

E.g. To specify bulk transfer, double-buffered mode, continuous transmission/reception, input, and EP1:

```
BULK | DBLB | CNTMD | DIR_IN | EP1,
```

7.2.3 Pipe Definition Item 3

Settings for the pipe buffer setting register are as follows:

Buffer Size: Specify the PIPE buffer size in 64-byte units.

Buffer Start Number: Specify the start number of the buffer.

E.g. To specify 512 bytes for the buffer size, and the buffer start number is 6:

```
BUF_SIZE(512) | 6
```

7.2.4 Pipe Definition Item 4

Specify pipe maximum packet size register settings as follows:

Max Packet Size: Specify the maximum packet size of the pipe.

E.g. To specify the max. packet size as 64:

```
64
```

7.2.5 Pipe Definition Item 5

Specify the pipe period control register as follows:

Isochronous IN Transfer Buffer Flush: To enable flushing, specify IFIS; to disable flushing, specify OFF.

Interval Error Detection Time: Specify the interval.

E.g. To disable buffer flushing and select an interval value of 0:

```
OFF | 0,
```

8. Descriptor Definitions (descrip.h)

Since the USB firmware is written as generally applicable firmware, the descriptor header file must be edited to set up user-specific descriptor definitions for use in device configuration.

There are four types of descriptor definitions.

- Notes:**
1. For details on the individual descriptors, refer to Chapter 9 of the USB Revision 2.0 specification.
 2. When changing the descriptor definitions, also change the pipe definitions (def_ep.h) to match the descriptor definitions.

1. Standard Device Descriptors
The USB firmware uses the table below for these definitions:
U8 DeviceDescriptor[]
2. Device Qualifier Descriptors
The USB firmware uses the table below for these definitions:
U8 QualifierDescriptor[]
3. Configuration/Other_Speed_Configuration/Interface/Endpoint
The USB firmware uses the tables below for these definitions:
U8 Configuration_Full_1[]
U8 Configuration_Hi_1[]
4. String Descriptors
The USB firmware uses the tables below for these definitions:
U8 StringDescriptor_tbl0[]
U8 StringDescriptor_tbl1[]
U8 StringDescriptor_tbl2[]
U8 StringDescriptor_tbl3[]
U8 StringDescriptor_tbl4[]
U8 StringDescriptor_tbl5[]

8.1 Creating the Descriptors

The USB firmware uses the tables described in items 1 to 3 and, according to the current M66592 operating mode, copies DT_CONFIGURATION or DT_OTHER_SPEED_CONFIGURATION to Configuration_Full_n[1] or Configuration_Hi_n[1], and creates the descriptor in RAM. Parts of the program to be changed are written as SOFTWARE_CHANGE. The flow of creation is shown below.

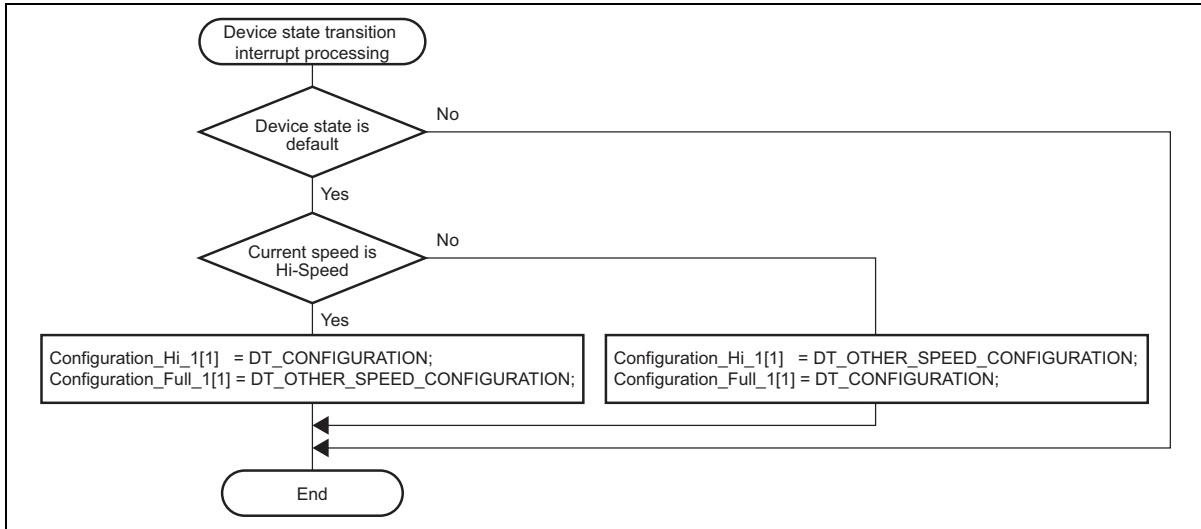


Figure 10 Creating the Descriptors

8.2 Setting the Default Control Pipes

After creating the descriptor on RAM, the USB firmware sets the registers to use the M66592’s pipes as default.

The following M66592 registers have to be set:

1. CFIFO port selection register (address 0x1E)
2. DCP configuration register (address 0x5C)
3. DCP maximum packet size register (address 0x5E)

Of the above items, 1 and 2 involve setting data in the constant table DCPTbl[] in the register. For details, refer to section 7, Pipe Definition. Item 3 involves setting data in the table DeviceDescriptor[7] on RAM in the register, so set the max. packet size to be used.

8.3 USB Firmware Sample Descriptor Configuration

The configuration of descriptors for the USB firmware is shown below.

```

/*
 * |--- Configuration 1
 * | |--- Interface 1-0-0
 * | | |--- Endpoint 1-0-0-0
 * | | |--- Endpoint 1-0-0-1
 * | | |--- Endpoint 1-0-0-2
 * | | |--- Endpoint 1-0-0-3
 * | | |--- Endpoint 1-0-0-4
 * | | |--- Endpoint 1-0-0-5
 * | | |--- Endpoint 1-0-0-6
 */

```

9. Low Power Sleep Function (PCUT)

The USB firmware comprises the following three types of processing samples.

1. When the low power sleep mode and automatic clock supply function are both in use.
2. When the low power sleep mode is not in use but the automatic clock supply function is in use.
3. When neither the low power sleep mode nor the automatic clock supply function is in use.

The PCUT_MODE declaration in defusr.h must be changed as described below according to whether or not the low power sleep function is in use.

9.1 When the Low Power Sleep Function is in Use

The PCUT_MODE declaration in defusr.h should be PCUT_USE. Here, when suspend or detach function is used, the clock stops, and the low power sleep function is activated. The clock is started up by hardware.

9.2 When the Low Power Sleep Function is not in Use (but the Automatic Clock Supply Function is)

The PCUT_MODE declaration in defusr.h should be PCUT_NOT_USE. The ATCKM_MODE declaration in the same file should be ATCKM_USE. Here, when suspend or detach function is used, the clock stops. The clock is started up by hardware.

9.3 When the Low Power Sleep Function is not in Used (but the Automatic Clock Supply Function is)

The PCUT_MODE declaration in defusr.h should be PCUT_NOT_USE. The ATCKM_MODE declaration in defusr.h should be ATCKM_NOT_MODE. Here, when suspend or detach function is used, the clock stops. Restart the clock by software.

10. Restrictions

The following restrictions apply to usage of the USB firmware.

1. The isochronous transfer may not operate correctly (in certain operating states, CPU processing becomes a bottleneck and packets are lost).
2. Operation with a split bus is not supported.
3. Multiple instances of the same endpoint number cannot be used simultaneously. For example, normal operation is not guaranteed when endpoint 1 is specified for BULK IN at pipe 1 and endpoint 1 is specified for BULK OUT at pipe 2.
4. DMA transfer is not supported.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul.13.05	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. 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 Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.