RENESAS DIGITAL ASSP

M66591

USB Sample Firmware

Instruction Manual

April. 7, 2003

RENESAS TECHNOLOGY CORPORATION

RENESAS SOLUTIONS CORPORATION

CONTENTS

1 Overview of Manual

1.1 Overview

This document is the instruction manual for the RENESAS DIGITAL ASSP M66591 USB Sample Firmware (hereinafter referred to as USB-FW) that is a sample program for control of USB interface using the M66591.

1.2 Related Documents

[1] M66591 Datasheet
[http://www.renesas.com/eng/products/mpumcu/specific/usb_mcu/index.html]
[2] Basic Information of USB (in Japanese)
[http://www.renesas.com/jpn/products/mpumcu/specific/usb_mcu/outline7.html]
[3] USB Specification Version 1.1 (Chapter 8 Protocol Layer)
[http://www.renesas.com/jpn/products/mpumcu/specific/usb_mcu/outline7.html]
[4] USB Specification Version 1.1 (Chapter 9 Device Framework)
[http://www.renesas.com/jpn/products/mpumcu/specific/usb_mcu/outline7.html]
[5] Universal Serial Bus Revision 2.0 specification
[http://www.usb.org/developers/docs]
[6] USB Application Note
[http://www.renesas.com/eng/products/mpumcu/specific/usb_mcu/index.html]

2 Overview

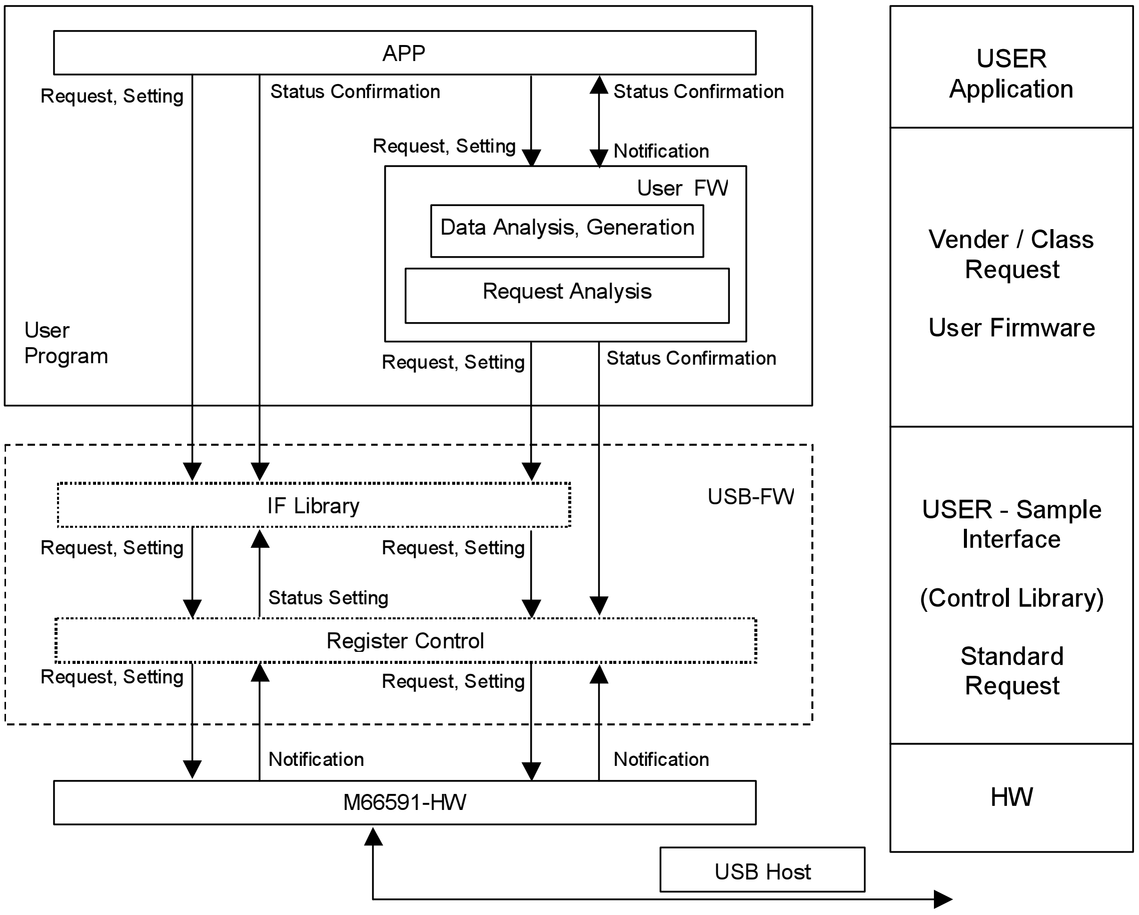
2.1 Features of USB-FW

The USB-FW has the following features:

- ● The control MCU and peripheral are not specified for the configuration (users can define them individually).
- ● The USB-FW can verify the connection using USBCommandVerifier.exe (hereinafter referred to as USBCV). (The USBCV can be downloaded from http://www.usb.org/developers/tools)
- ● A sample program for bulk (IN/OUT)/interrupt (IN) data communication is available.
- ● Files are divided according to function (refer to file configuration list).
- ● The user program has no need to access to the M66591 register directly.

2.2 Layer

The USB-FW consists of the following layers:



2.3 File Configuration List

File Name	Overview
changeep.c	User application processing
classvender.c	Class/vender request processing
controlrw.c	Control read/write processing
dataio.c	Data read/write processing
datatbl.h	User buffer definition for transmit/receive
def591.h	Register address/bit definition for the M66951
def_ep.h	Data definition for PIPE setting
defusr.h	User definition
descrip.h	Descriptor data definition
extern.h	External reference definition
global.c	Global variable processing
intrn.c	INTR,INTN,BEMP interrupt processing
usbsig.c	USB signal processing
libassp.c	USB ASSP register operation processing
Lib591.c	USB ASSP register operation processing
Libassp.h	USB ASSP register operation processing definition
Macusr.h	User macro definition
main.c	Dummy user application
Status.c	Internal status operation processing
Stdreqget.c	Standard request processing
Stdreqset.c	Standard request processing
Typedef.h	Variable type definition
Usbint.c	USB interrupt processing
Version.h	Version information

2.4 Development Purpose

The USB-FW is developed for the following purposes:

- To facilitate the development of the USB communication programs using the M66591.
- To supplement the explanation of the M66591 control with concrete examples.

2.5 Service Overview

The USB-FW provides the following services for the upper layer (user program layer):

- M66591 initialization (Reset, oscillation control, PIPE initialization, etc.)
- Request response (Standard request [USB Revision 2.0 specification])
- Data transfer (Bulk and interrupt transfer: CPU access)
- Status notification (Status notification function)
- Request notification (Request notification function)

2.6 Global Flow

The USB-FW configures interrupt programs consisting of control functions that transmit/receive USB data.
The interrupt event is occurred by external interrupt to the control MCU from the M66591.
The external interrupt program identifies the factor of interrupt and executes relevant processing.

(1) Special signal processing

Vbus interrupt, resume interrupt, SOF detect interrupt*1 and device state transition interrupt
(*1: The USB-FW does not includes processing. The processing should be created if needed.)

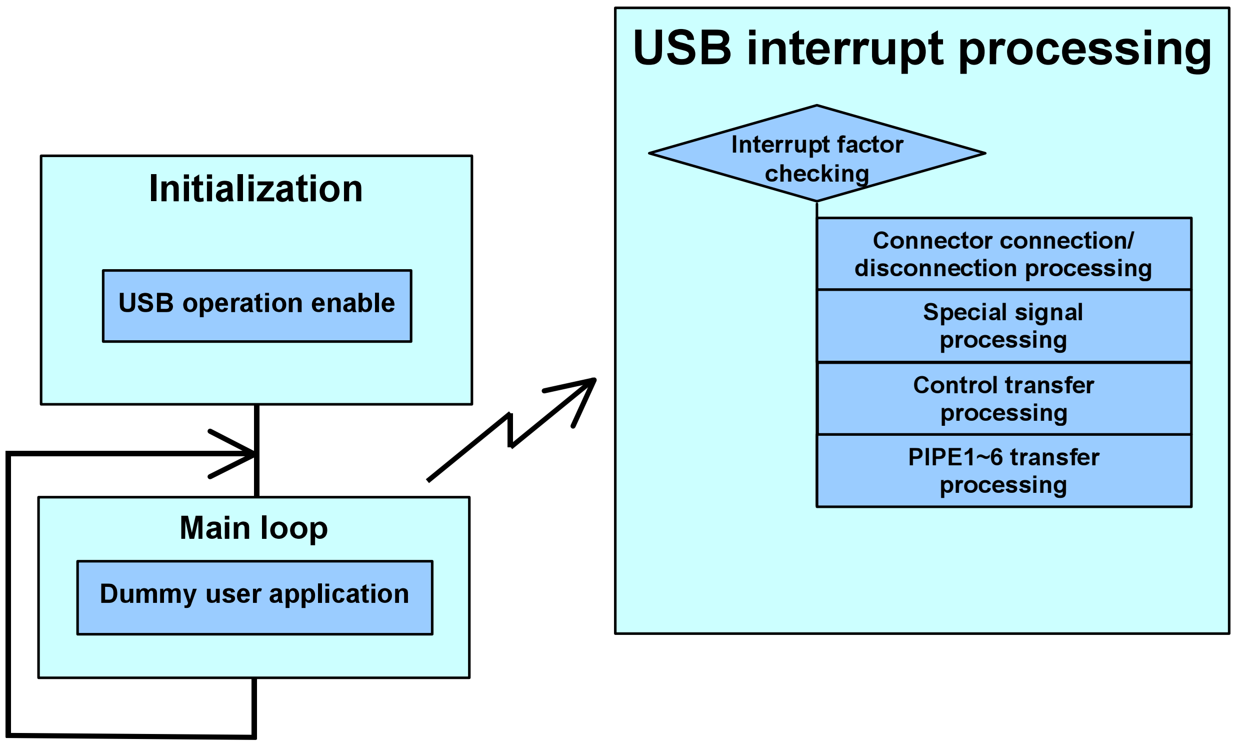
(2) Control transfer processing

Control transfer stage transition interrupt and device state transition interrupt

(3) PIPE transfer processing

Buffer empty/size error interrupt, buffer not ready interrupt and buffer ready interrupt

Since the USB-FW executes USB control processing in an interrupt routine, a permanent loop occurs in the *main* function after the *main* function initializes the control MCU and registers related to the M66591.
The global flow diagram of the USB-FW is as follows.


USB interrupt processing
Interrupt factor checking
Connector connection/ disconnection processing
Special signal processing
Control transfer processing
PIPE1~6 transfer processing
Initialization
USB operation enable
Main loop
Dummy user application

3 Operation

3.1 Overview

The USB-FW is enabled to communicate and execute the USBCV program, when the control MCU and peripheral are initialized (*main.c*), and the user definition information file (*defusr.h*) and user definition macro file (*macusr.h*) are modified.

3.2 Modification of USB-FW

The following program file and header file program have to be modified in order to activate the USB-FW and execute the USBCV.

<u>(1) The following functions in the *main.c* have to be modified.</u>

- ● Initialize the control MCU (*CPUInit* function)
- ● Initialize the peripheral (*PeripheralInit* function)
- ● Enable the control MCU interrupt (*enableINT* function)
- ● Adjust time in functions for waiting for specified time (*delay_1ms* function and *delay_10us* function)
  The specified time is waited by loop processing. Adjust the number of loops to wait for the specified time according to user's system.

<u>(2) The following files have to be customized by users.</u>

- ● Specify vender ID and product ID in the *descrip.h*.
  VenderID and *ProductID*
  (For details, refer to "6. User definition")
- ● Specify endian, I/O voltage, register base address, far area, oscillation constant, and interrupt function in the *defusr.h*.
  FIFO_ENDIAN, *VIF_SET*, *USB_BASE*, *REGP*, *XIN* and *usbint*
  (For details, refer to "6. User definition")

<u>(3) The following processings have to be executed when building</u>

- ● Specify section area
- ● Create start up routine

<u>(4) Others</u>

- ● Special signal processing (Not need at the firmware operation check (execute the USBCV program) level)

3.3 Notes

The USB-FW is a general-purpose firmware not specifying the control MCU and peripheral. It is applicable only to standard requests. It executes data communications using temporary interface between the USB-FW and dummy user application.

Therefore, the USB-FW have to be customized the following cases by users:

- When class specification and response to requests unique to a vendor are required
- When the communication speed and program capacity are taken into consideration
- When the user interface is set separately

*** The USB-FW does not guarantee the USB communication operations. When applying it to user's system, users must verify its operations, and confirm its connection on various host controllers.**

4 Data Transfer

4.1 Overview

The USB-FW is capable of simple data communications between the host PC and the device if users prepare the USB driver on the host PC and a data transfer application. (For the device configuration of the USB-FW, refer to the descriptor definition section.)

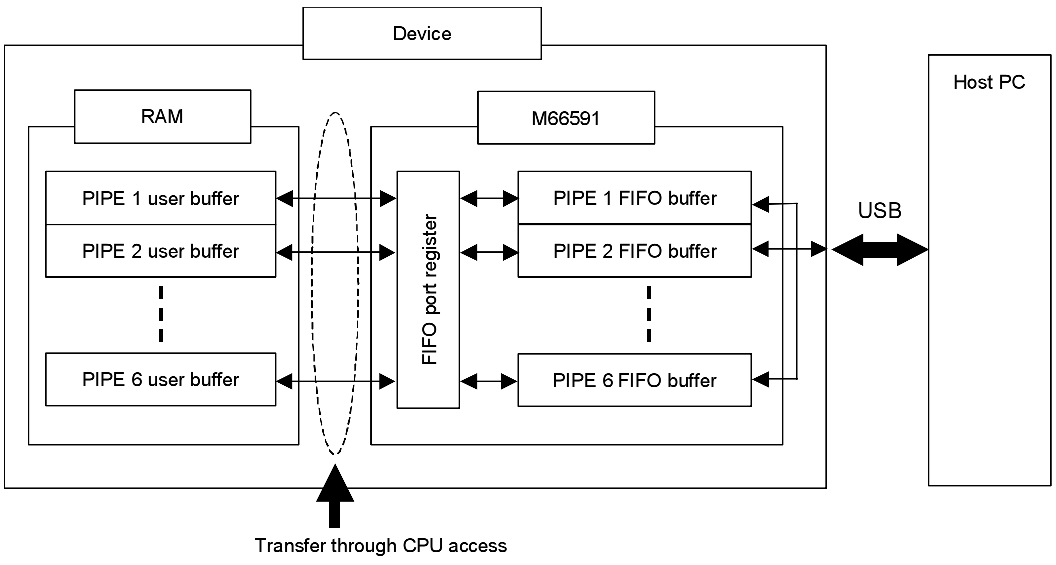
Simple data communications applicable to user's system (system on the host PC) can be realized by modifying information necessary for the device configuration (descriptor definition (*descrip.h*) and PIPE definition (*def_ep.h*)) and the dummy user application (*main.c*).

Since data communication is based on the functional specifications unique to the user, the user should individually modify the transferring method, requests for communication start and end, and buffer configuration.

4.2 Basic Specification of USB-FW

- The USB-FW realizes data transfer between the user buffer and the FIFO port register through CPU access. (The data flow is as follows.)
- The user buffer has secured the 512-byte area to all PIPEs. (Secure the user buffer area larger than the FIFO buffer area.)
- Availability to transfer data larger than PIPE FIFO buffer size by modifying size of the user buffer.
- The transmit data to the host use the fixed data each PIPE.
- The user buffer address notification function (*DI_Start/DO_Start*) is a function common to CPU/DMA accesses.
- For data transmit/receive, a buffer ready interrupt is used. The buffer ready interrupt of each PIPE is enabled when data transmit/receive is started.

Data flow:



4.3 Data Transmit (IN Token) Operation

The USB-FW transmits the data (support IN token) to the host PC as follows:

(1) Check enable/disable of the user buffer use. (Enable when the *Buffer_Write_Data_Flag* is set to the *DATA_NONE*.)

When the user buffer is disabled, no processing is executed, and processing for another PIPE is executed.

(2) Check whether there are the transmit data or not. (When the returned value to the *Create_In_Data* function is other than the *DATA_NONE*, there are the transmit data.)

When there are not the transmit data, no processing is executed, and processing for another PIPE is executed.

(3) The USB-FW sets the user buffer address and the transmit byte count (*dtcnt*), disables the user buffer use (the *Buffer_Write_Data_Flag* is set to the *DATA_WAIT*), and enables the buffer ready interrupt (*DI_Start* function).

(4) Write the data in the user buffer to the FIFO port register when the buffer ready interrupt has been occurred.

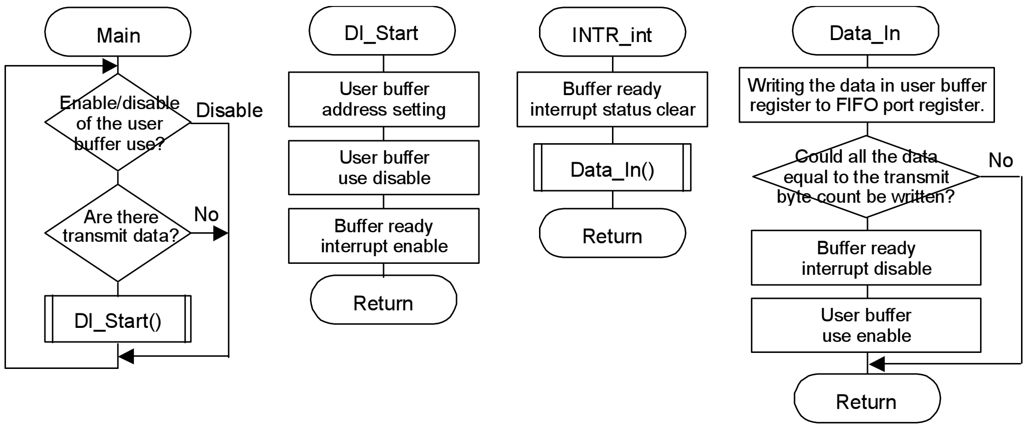
● When transmit byte count (*dtcnt*) > FIFO buffer size

Write the data equal to the FIFO buffer size, and subtract the FIFO buffer size from the transmit byte count. (*dtcnt* = *dtcnt* - FIFO buffer size).

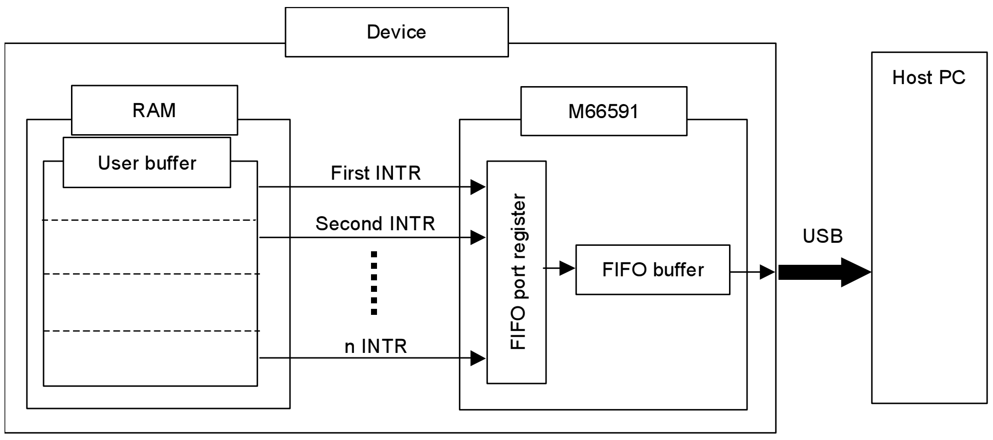
● When transmit byte count (*dtcnt*) ≤ FIFO buffer size

Write the data equal to the transmit byte count, disable the buffer ready interrupt, and enable the user buffer use (the *Buffer_Write_Data_Flag* is set to the *DATA_NONE*).

Control flow:


Main
Enable/disable of the user buffer use?
Disable
Are there transmit data?
No
DI_Start()
DI_Start
User buffer address setting
User buffer use disable
Buffer ready interrupt enable
Return
INTR_int
Buffer ready interrupt status clear
Data_In()
Return
Data_In
Writing the data in user buffer register to FIFO port register.
Could all the data equal to the transmit byte count be written?
No
Buffer ready interrupt disable
User buffer use enable
Return

Data flow:



4.4 Data Receive (IN Token) Operation

The USB-FW receives the data (support OUT token) from the host PC as follows:

(1) Check enable/disable of the user buffer use. (Enable when the *Buffer_Read_Data_Flag* is set to the *DATA_NONE*.)

(2)-1 Check enable/disable of the user buffer read when the user buffer is disabled. (Enable when the *Buffer_Read_Data_Flag* is set to the *DATA_OK*.)

Enable the user buffer use when the user buffer is enabled to read (the *Buffer_Read_Data_Flag* is set to the *DATA_NONE*).

When the user buffer is disabled to read, no processing is executed, and processing for another PIPE is executed.

(2)-2 When the user buffer is enabled, the USB-FW sets the user buffer address and the receive byte count (*dtcnt*), disables the user buffer use (the *Buffer_Read_Data_Flag* is set to *DATA_ WAIT*), and enables the buffer ready interrupt (*DO_Start* function).

(3) Read the data from the FIFO port register and write the data to the user buffer when the buffer ready interrupt has been occurred.

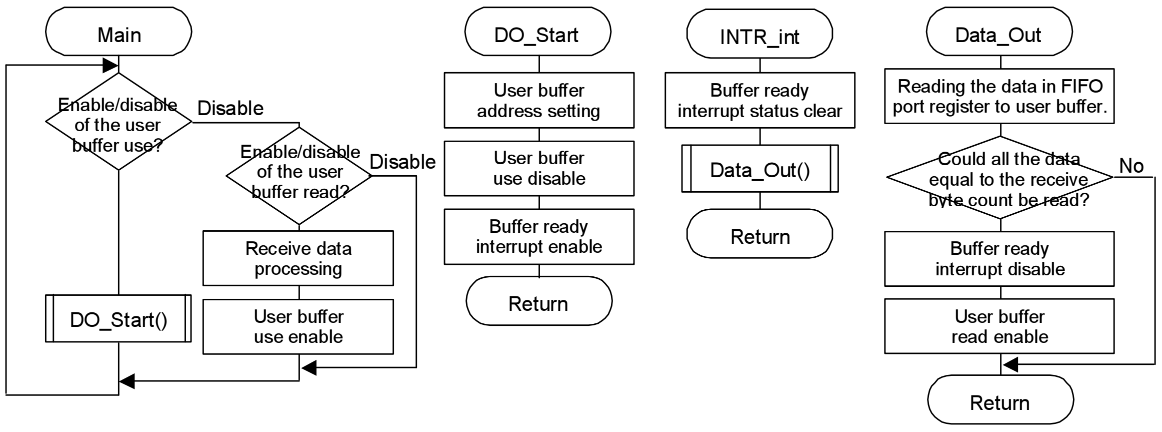
● When receive byte count (*dtcnt*) > FIFO buffer size

Read the data equal to the FIFO buffer size, and subtract the FIFO buffer size from the receive byte count. (*dtcnt* = *dtcnt* - FIFO buffer size).

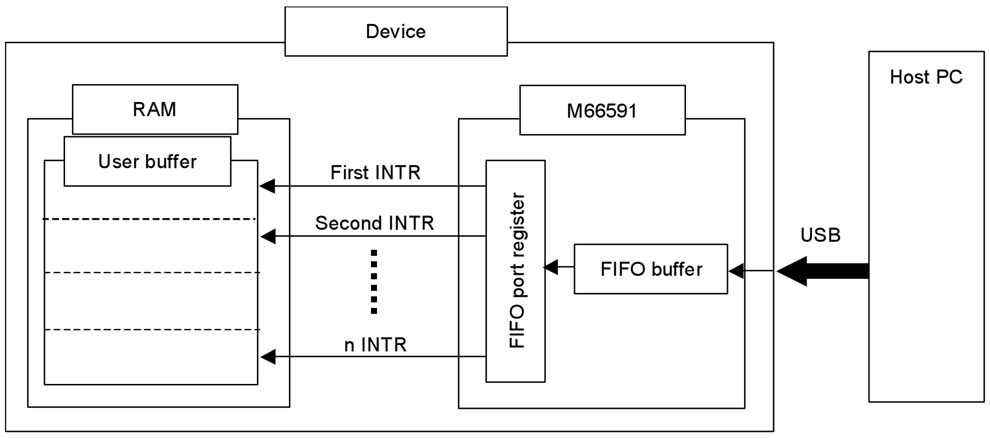
● When receive byte count (*dtcnt*) ≤ FIFO buffer size

Read the data equal to the receive byte count, disable the buffer ready interrupt, and enable the user buffer use (the *Buffer_Read_Data_Flag* is set to the *DATA_OK*).

Control flow:


Main
Enable/disable of the user buffer use?
Disable
Enable/disable of the user buffer read?
Disable
Receive data processing
User buffer use enable
DO_Start()
DO_Start
User buffer address setting
User buffer use disable
Buffer ready interrupt enable
Return
INTR_int
Buffer ready interrupt status clear
Data_Out()
Return
Data_Out
Reading the data in FIFO port register to user buffer.
Could all the data equal to the receive byte count be read?
No
Buffer ready interrupt disable
User buffer read enable
Return

Data flow:



4.5 PIPE Setting for Data Transfer

The USB communication has to modify the PIPE setting for data communications by the request of the *Set_Configuration/Set_Interface* etc.

When receiving the above request, the USB-FW retrieves the PIPE table index number to be used from the descriptor table in the configuration or interface to be modified, and automatically sets the PIPE configuration register according to the PIPE definitions.

(1) PIPE setting

The USB-FW calls the *Esrch* function using the configuration number, interface number and alternate setting as arguments. Then, it calls the *resetEP* function using the configuration number as arguments.

```
void Esrch       (Configuration number, interface number, alternate setting);
void resetEP     (Configuration number);
```

The *Esrch* function retrieves and sets the PIPE definition index numbers of all PIPEs to be used in the interface of the specified configuration.

The *resetEP* re-sets the PIPE configuration registers for all PIPEs retrieved in the *Esrch*. When modifying PIPE specifications, it is necessary to disable existing PIPE communications to avoid invalid data input or input errors during modification. Therefore, the *resetEP* disables relevant PIPE interrupt.

(2) Descriptor table

The USB-FW uses temporary descriptor definitions. Create descriptor definitions unique to user's system (*descrip.h*) according to the windows driver and applications on the host.

Modify PIPE definition (*def_ep.h*) simultaneously when descriptor definition is modified. Also, change the dummy user application (*Change_Config/Change_Interface* function) if it is required.

*** Define the PIPE information in the same order as descriptor definition and PIPE definition. (For details, refer to PIPE definition section.)**

(3) *Change_Config / Change_Interface* function

These functions are called by the USB-FW when the *Set_Configuration* or *Set_Interface* request is received. Add the processing by the USB-FW if user applications require some processing.

4.6 Notes when Modifying

- The user buffer should be larger than max. packet (FIFO buffer when setting to continuous transmit/receive) size.
- A dummy area is provided in each user buffer to facilitate verification of communication.
- Modify PIPE definition (*def_ep.h*) simultaneously when descriptor definition (*descrip.h*) is modified.
- When the usage (or configuration) of any user buffer is modified, it may be required to modify library functions.

4.7 User Buffer Specification

- An area is secured to count the accesses to the user buffers.
- The user buffers define different default data depending on PIPE.

```
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_buff6 EP_Buff6 = {
    EP6_DATA_SIZE,    0,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,
    0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,
    0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,0xF1,
    0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,0xF2,
```

5 Class/Vender Request

5.1 Overview

The USB-FW can respond to class/vender requests if the users prepare the windows driver and applications on the host PC, and a user firmware on the device side. As the basic structure of data communication on the data stage, the USB-FW provides only the entries for the functions designed with the same structure as that of data communication on each PIPE.

5.2 Basic Specification

- The following functions are available as user buffer address notification functions:
  CR_Start (Control read user buffer address notification function)
  CW_Start (Control write user buffer address notification function)
- The following functions are available as data read/write functions:
  Control_Read (Control read data write function)
  Control_ Write (Control write data read function)
- Secure the user buffer area larger than the data capacity that transmit/receive on the data stage:
  (The user buffer size is required larger than the data capacity transferred at one control transfer.)

5.3 Detailed Specification

- A user buffer is provided between the USB-FW and user firmware, and the USB-FW realizes data transfer between the user buffer and the FIFO port register through CPU access.
  -> Use the *CR_Start* or the *CW_Start* function in the user application.
- The transfer direction of the data stage is judged using the *INTR_int* or the *BEMP_int* function.
- The data are continuously transferred using the interrupt when transmitting/receiving.
  -> The data are continuously transferred by the *Control_Read/Control_ Write* function.
- The transmit data to the host (applicable to the *Control_Read*) should be created by the user firmware.
  Proposal 1: Creation according to request in interrupt (same as standard request)
  Proposal 2: Creation by application (notification of receipt to user application from interrupt)
- The receive data from the host (applicable to the *Control_ Write*) should be analyzed by the user firmware.
  Proposal 1: Creation according to request in interrupt (same as standard request)
  Proposal 2: Creation by application (notification of receipt to user application from interrupt)

5.4 Example of Control Read (IN Direction) User Firmware Interface

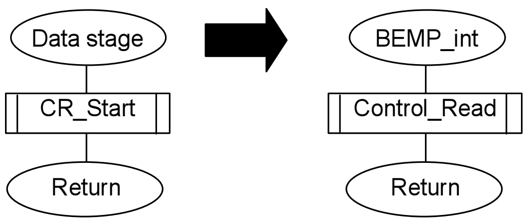
The example of the class/vender request created by the user firmware is as follows:

(1) User calls the *CR_Start* function in the control read data stage (*ClassTrans1* function).

(2) The buffer empty/size over interrupt is occurred (*BEMP_int* function).
The transmit data transfers to the FIFO buffer by the *Control_Read* function.

*** These are examples for creating the class/vender request. Actually, create the request according to user's specifications.**

Transmit flow:



Notes:

- Call the transmit start notification (*CR_Start* function) after the creation of the transmit data has completed.
- Notify the user buffer address and the data size in the transmit start notification (*CR_Start* function).
- Secure the user buffer capacity larger than the data size (wLength) transmitted in the control read data stage.
- Notify the completion of control transfer (*Control_End* function) at the control read status stage transition of the USB interrupt function.
- The user firmware must store the request type (control read setup stage information) to create the transmit data.

5.5 Example of Control Write (OUT Direction) User Firmware Interface

The example of the class/vender request created by the user firmware is as follows:

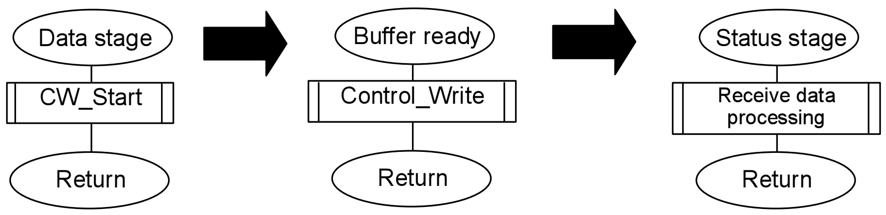
(1) User calls the *CW_Start* function in the control write data stage (*ClassTrans2* function).

(2) The buffer ready interrupt is occurred when the data has been received from the host PC (*INTR_int* function). The receive data transfers to the user buffer by the *Control_Write* function.

(3) The receive data in the user buffer is processed in the control write status stage (*ClassTrans5* function).

*** These are examples for creating the class/vender request. Actually, create the request according to user's specifications.**

Receive flow:



Notes:

- Notify the buffer address and the data size in the receive start notification (*CW_Start* function).
- Secure the user buffer capacity larger than the received data size (wLength) in the control write data stage.
- Notify the completion control transfer to the USB-FW after the receive data analysis has completed (using *Control_End* function).

6 User Definition

6.1 Overview

Since the USB-FW is described as a general-purpose firmware, it is possible to create an execution file unique to the user by rewriting the user definition information file.
Change the following eight items according to user's system.

*** Other definitions may be required depending on control MCU.**

(1) Vender ID
(2) Product ID
(3) FIFO endian
(4) I/O power supply
(5) Address of the M66591
(6) Pointer type for address of the M66591
(7) Oscillation frequency of the oscillator connected to the M66591
(8) Interrupt vector

6.1.1 Vender ID (*descrip.h*)

Specify the vender ID according to user's system.

Example) When 0x1234 is set
```
#define  VenderID  0x1234
```

6.1.2 Product ID (*descrip.h*)

Specify the product ID according to user's system.

Example) When 0x5678 is set
```
#define  ProductID  0x5678
```

6.1.3 FIFO endian (*defusr.h*)

Specify the endian for access to the FIFO port of the M66591.

Example) When the little endian is set
```
#define  FIFO_ENDIAN  LITTLE_ENDIAN
```

6.1.4 I/O power supply (*defusr.h*)

Specify the voltage of the I/O pin in the M66591.

Example) When 3.3V is set
```
#define  VIF_SET  VIF3
```

6.1.5 Address of the M66591 (*defusr.h*)

Specify the standard address for access to the M66591.

All registers are specified the address based on offset from the standard address.

Example) When 0x8000 address is set
```
#define  USB_BASE  (0x8000)
```

6.1.6 Pointer type for address of the M66591 (*defusr.h*)

Specify the pointer type for access to the M66591.

Each register on the M66591 is cast in the REGP type to cope with an MCU requiring declaration of the *near/far*.

Comment out one of them according to the type of control MCU.

Example) When the M66591 is allocated to *near* area with an MCU requiring declaration of the *near/far*

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

6.1.7 Oscillation frequency of the oscillator connected to the M66591 (*defusr.h*)

Select the oscillation frequency of the oscillator to be connected to the M66591 from three oscillation types.

Example) When 24MHz oscillator is used

```
#define XIN Xtal24
```

6.1.8 Interrupt vector (*defusr.h*)

For an MCU requiring declaration of interrupt processing function, it is necessary to declare interrupt for USB communication provided by the USB-FW. Cancel the comment as necessary. When the declaration is not the *#pragma*, change the declaration.

Example) When the *pragma* declaration is required for interrupt processing function (*usbint*)

```
#pragma INTERRUPT  usbint
```

7 User Definition Macro (*macusr.h*)

7.1 Overview

The M66591 has been designed compliant with the USB Revision 2.0 specification. It must access each register in little endian. The USB-FW is described as a general-purpose firmware on the assumption that the endian of control MCU may differ from that of the M66591. Since the register access and FIFO register access are described as macros, it is possible to create an execution file unique to the user by rewriting the user definition macro header file. Change the macro for access to the register and FIFO register according to user's system.

The register access macro consists of the following nine macros classified into four kinds:

(1) Register and FIFO data register read/write macro

(2) Register bit set/clear/modify macro

(3) Status register bit clear macro

(4) Status register bit set macro

7.1.1 Register and FIFO data register read/write macro

These macros are used for reading and writing data in the register and the FIFO data register.

The control MCU and the M66591 must be connected so that DMA transfer between memory and FIFO data register can be executed normally. Refer "6.1.3 FIFO endian (defusr.h)" to set FIFO endian.

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

*** Connect the control MCU, memory and the M66591 in sufficient consideration of the system configuration.**

7.1.2 Register bit set/clear/modify macro

These macros are used for setting, clearing and modifying register bits.

Each macro is described in accordance with RMW (Read Modify Write) command and uses the above-mentioned macro for reading and writing registers and FIFO data register.

*** It is not necessary to change these macros unique to the user.**

*** Do not clear the bit of the status register by this bit clear macro. The bit of the status register is cleared using the status register bit clear macro.**

```
/* 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)
```

7.1.3 Status register bit clear macro

This macro is used for clearing the status register bit.

This macro is described in accordance with RMW (Read Modify Write) command and uses the above-mentioned macro for reading and writing registers and FIFO data register. This macro avoids clearing the status that has changed during execution of a command in accordance with RMW command.

*** It is not necessary to change these macros unique to the user.**

*** This macro is used only for the status register with invalid writing “1”.**

```
/* reset bit(s) of USB status      */
/* r : USB register                */
/* m : bit pattern to reset        */
#define  USB_CLR_STS( r, m )   do{ register U16  tmp; \
                                   tmp = 0; \
                                   tmp |= (~(m)); \
                                   USBWR( r, tmp ); }while(0)
```

7.1.4 Status register bit set macro

This macro is used for setting the status register bit.

This macro uses the above-mentioned macro for writing register. The macro avoids clearing the status that has changed during execution of a command by writing "1" to all bits.

(Use this macro when writing "1" to the bit by the VBUSINT bit clear etc. in the internal clock disable state.)

*** It is not necessary to change these macros unique to the user.**

*** This macro uses only for the status register with invalid writing "1".**

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

8 PIPE Definition (*def_ep.h*)

8.1 Overview

The M66591 sets the conditions for PIPEs by the configuration register. The USB-FW is described as a general-purpose firmware on the assumption that PIPE settings will be changed as the result of modification of configuration and alternative caused by standard requests, such as *Set_Configuration* and *Set_Interface*. It is possible to create an execution file unique to the user by configuring PIPE information (usage) in each state on the header file as a data table and rewriting the PIPE definition header file (*def_ep.h*).
Two parameters for each PIPE must be set; full-speed (*Eptbl_Full_n*) and hi-speed (*Eptbl_Hi_n*).
Change the PIPE definition according to user's system.

*** Change descriptor definition (*descrip.h*) according to PIPE definition when PIPE definition is changed.**
The default control PIPE definition consists of the following two items (U16 x 2):
(1) C_FIFO Port Control Register 0 (0x28 address)
(2) Default Control PIPE Configuration Register 1 (0x82 address)

The PIPE 1 to 6 definitions consist of the following two items (U16 x 2):
(1) C_FIFO Port Control Register 0 (0x28 address)
(2) PIPE Configuration Window Register 0 (0x90 address)

8.2 Default Control PIPE Definition

The default control PIPE definition consists of the table.
The default control PIPE information table described as a sample in the USB-FW is the following:

Example)
```
const U16 DCPtbl[] = {
   /* C_FIFOControlRegister 0 (0x28) */
       RCNT | MBW_16,                                              <- Definition item 1
   /* Default Control PIPE Configuration Register 1 (0x82) */
       CNTMD                                                       <- Definition item 2
```

8.2.1 Default control PIPE definition item 1

This item specifies the value for the C_FIFO Port Control Register 0.
Set as follows:

● Read count mode	: Specify *RCNT* when the ODLN register value is counted down.
	Specify *OFF* when the ODLN register value is not counted down.
● FIFO access bit width	: Specify *MBW_8* when setting to 8-bit width.
	Specify *MBW_16* when setting to 16-bit width.

Example) when the read count mode and the 16-bit width are set
```
RCNT | MBW_16,
```

8.2.2 Default control PIPE definition item 2

This item specifies the value for the Default Control PIPE Configuration Register 1.

Set as follows:

- Continuous transmit/receive mode : Specify *CNTMD* when setting to continuous transmit/receive mode.
  Specify *OFF* when setting to non-continuous transmit/receive mode.

Example) when the continuous transmit/receive mode is set

```
CNTMD
```

*** For the specification method of max packet size of the default control PIPE, refer to "9. Descriptor Definition (*descrip.h*)".**

8.3 PIPE 1 to 6 Definitions

Each PIPE definition consists of respective table of each configuration same as the descriptor definitions. The table is described in order of related interfaces and alternate settings. The PIPE definition described as a sample in the USB-FW is the following.

Each PIPE definition item is described on the following pages.

Example)

```
/* Configuration 1 */
const U16  EPtbl_Full_1[] = {                                  <- for full-speed
    /* Interface 1-0-0 */
        /* Endpoint 1-0-0-0 */

            /* (EP1 BULK) Single/DoubleBuffer CONT/notCONT IN/OUT */
            /* C_FIFO Port Control Register 0 (0x28) */
                PIPE1,                                         <- PIPE definition 1
            /* PIPE Configuration Window Register (0x90) */
                DBLB | CNTMD | DIR_IN,                         <- PIPE definition 2

                 :
                 :
                 :
};

const U16  EPtbl_Hi_1[] = {                                    <- for hi-speed
    /* Interface 1-0-0 */
        /* Endpoint 1-0-0-0 */

            /* (EP1 BULK notCONT) Single/DoubleBuffer IN/OUT */
            /* C_FIFO Port Control Register 0 (0x28) */
                PIPE1,
            /* PIPE Configuration Window Register (0x90) */
                DBLB | DIR_IN,

                 :
                 :
                 :
};
```

8.3.1 PIPE definition item 1

This item specifies the value for the C_FIFO Port Control Register 0.

Set as follows:

● CPU access PIPE:	Specify CPU access PIPE (PIPE1 to PIPE6).

Example) when PIPE 1 is set

```
PIPE1,
```

8.3.2 PIPE definition item 2

This item specifies the value for the PIPE Configuration Window Register 0.

Set as follows:

● Interrupt toggle mode:	Specify *ITMD* when the function is valid.
	Specify *OFF* when the function is invalid.
	(This specification is valid only PIPE5 and PIPE6)
● Double buffer mode:	Specify *DBLB* when setting to double buffer mode.
	Specify *OFF* when setting to single buffer mode.
	(This specification is valid only PIPE1 and PIPE2)
● Continuous transmit/receive mode:	Specify *OFF* when setting to non-continuous transmit/receive mode.
	Specify *BSIZE* when setting to continuous transmit/receive mode.
	(This specification is valid only PIPE1 to PIPE4 in full-speed)
● Transfer direction:	Specify *DIR_IN* when setting to IN direction.
	Specify *DIR_OUT* when setting to OUT direction.
	(This specification is valid only PIPE1 to PIPE4)

Example) when the double buffer mode and the IN direction are set

```
DBLB | DIR_IN,
```

9 Descriptor Definition (*descrip.h*)

9.1 Overview

Since the USB-FW is described as a general-purpose firmware, the device configuration can be specified in accordance with the descriptor definitions unique to the user by rewriting the descriptor definition header file.
The descriptor definitions consist of the following four kinds:

*** For details of each descriptor, refer to "Chapter 9 of USB Revision 2.0 specification".**

*** Change PIPE definition (*def_ep.h*) according to descriptor definition when descriptor definition is changed.**

(1) Standard Device Descriptor

The USB-FW defines in the following table.

```
U8  DeviceDescriptor[]
```

(2) Device Qualifier Descriptor

The USB-FW defines in the following table.

```
U8  QualifierDescriptor[]
```

(3) Configuration/Other_Speed_Configuration/Interface/Endpoint

The USB-FW defines in the following table.

```
U8  Configuration_Full_1[]
U8  Configuration_Hi_1[]
```

(4) String Descriptor

The USB-FW defines in the following table.

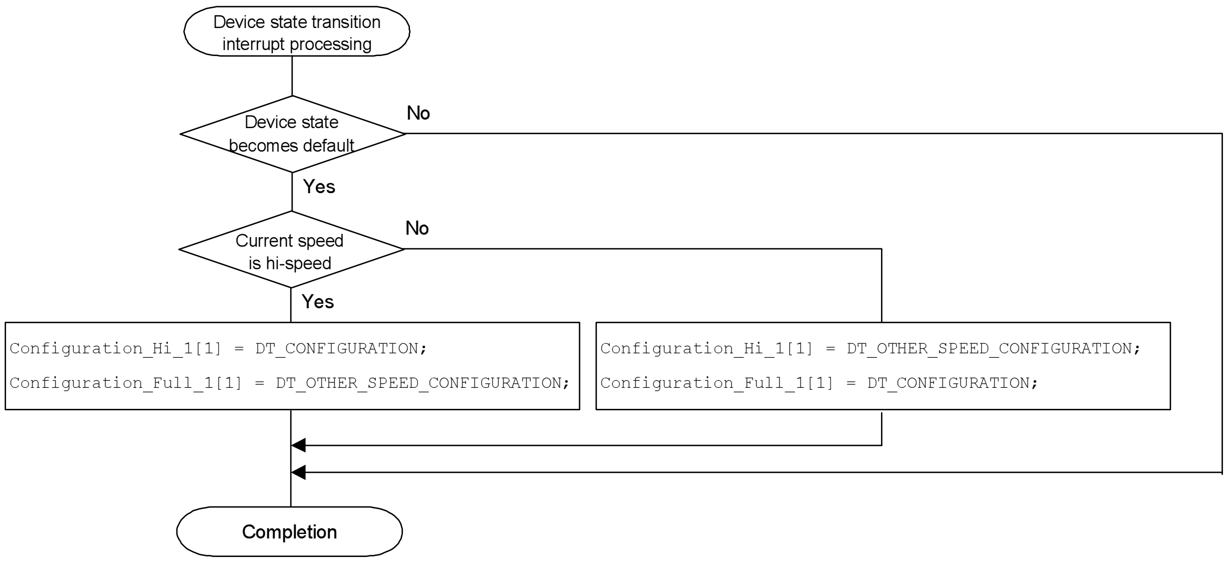
```
U8  StringDescriptor_tbl0[]
U8  StringDescriptor_tbl1[]
U8  StringDescriptor_tbl2[]
U8  StringDescriptor_tbl3[]
U8  StringDescriptor_tbl4[]
U8  StringDescriptor_tbl5[]
```

9.2 Descriptor Creation

The USB-FW copies *DT_CONFIGURATION* or *DT_OTHER_SPEED_CONFIGURATION* to *Configuration_Full_n[1]* and *Configuration_Hi_n[1]* in the program depending on the current operation mode of the M66591 by using the tables stated in above items (1) to (3) and create a descriptor on the RAM.

Parts to be changed in the program are marked *RESERVED*.

Create flow is as follows:


Device state transition interrupt processing
Device state becomes default
No
Yes
Current speed is hi-speed
No
Yes
Configuration_Hi_1[1] = DT_CONFIGURATION;
Configuration_Full_1[1] = DT_OTHER_SPEED_CONFIGURATION;
Configuration_Hi_1[1] = DT_OTHER_SPEED_CONFIGURATION;
Configuration_Full_1[1] = DT_CONFIGURATION;
Completion

9.3 Default Control PIPE Setting

The USB-FW sets the Default Control PIPE Register of the M66591 after creating a descriptor on the RAM.

Users have to set the following registers of the M66591.

(1) C_FIFO Port Control Register 0 (0x28 address)

(2) Default PIPE Configuration Register 1 (0x82 address)

(3) Default PIPE Configuration Register 2 (0x84 address)

The data in the constant table *DCP_tbl[]* is set to the above (1) and (2) registers. For details, refer to "8. PIPE definition".

The data in the table *DeviceDescriptor[7]* created on the RAM is set to the above (3) register. Set the max. packet size to be used.

9.4 USB-FW Sample Descriptor Configuration

The descriptor configuration of the USB-FW is the following definitions:

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

10 Restrictions

The USB-FW has the following restrictions:

(1) The USB-FW does not support operations in the split bus.

(2) The USB-FW does not support the DMA transfer processing.

11 Revision History

Version	Data	Contents
0.70	’02.08.05	Release
0.80	’02.08.29	The clerical error is corrected.
0.90	’02.11.11	7.1 and 8.3 is changed by FW changing.
0.94	’03.04.07	Changed corporation name.

RENESAS DIGITAL ASSP M66591

USB Sample Firmware Instruction Manual