DATA SHEET

NEC

MOS INTEGRATED CIRCUIT

μPD77533

GPS SIGNAL PROCESSOR LSI

DESCRIPTION

The μPD77533 is a GPS (Global Positioning System) signal processor LSI for GPS receivers that realizes the Wireless Assisted GPS of SnapTrack Inc.

The position of an STI-system GPS receiver is determined by means of positioning functions on the Wireless Assisted GPS receiver side such as pseudo range calculation and multi-path detection, via communication with a network location server.

FEATURES

- Wireless Assisted GPS positioning function
    - Pseudo range calculation
    - Multi-path detection
    - Precise, responsive, and fast positioning via communication with server
- On-chip SRAM (Mobile specified RAM), host serial, A/D control, and RF control interfaces
- Power supply voltage
    - Internal system power supply: 1.425 to 1.65 V
    - I/O pin power supply: 2.7 to 3.6 V

ORDERING INFORMATION

Part Number	Package
μPD77533S1-YHC	108-pin plastic fine-pitch BGA (11 × 11)

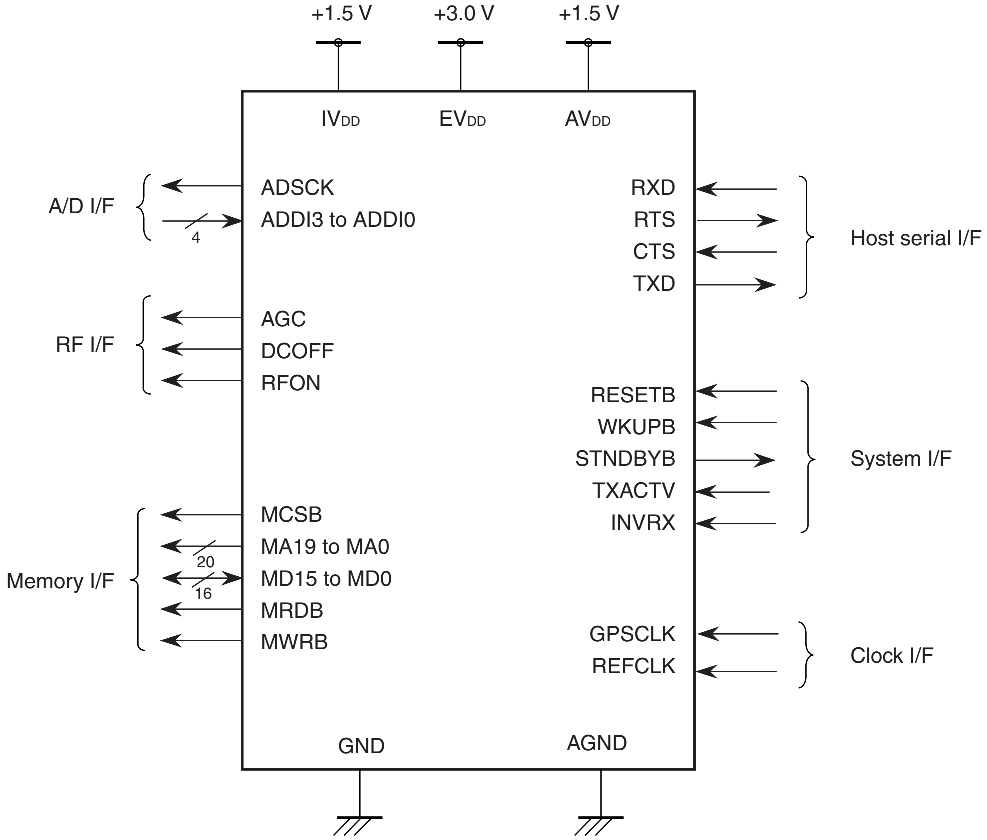
The information in this document is subject to change without notice. Before using this document, please confirm that this is the latest version.
Not all devices/types available in every country. Please check with local NEC representative for availability and additional information.

Document No. S15671EJ4V0DS00 (4th edition)
Date Published July 2002 N CP(K)
Printed in Japan

The mark ★ shows major revised points.

© NEC Corporation 2001

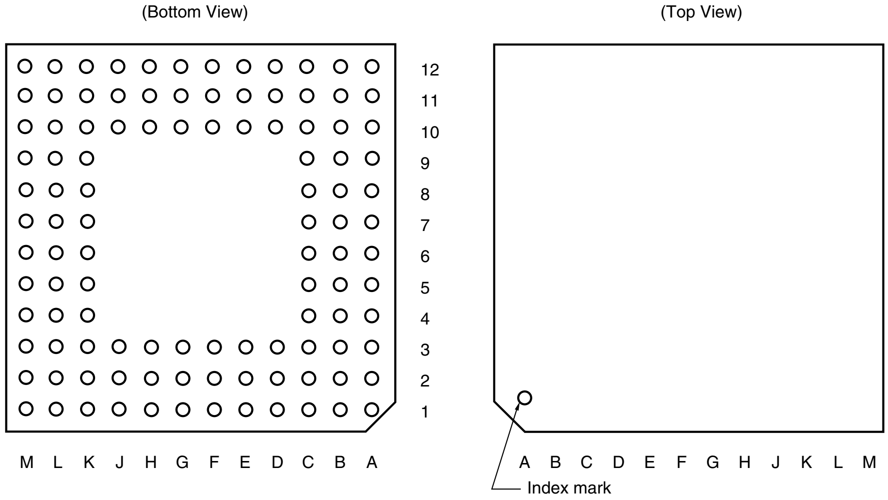
PIN CONFIGURATION


+1.5 V
+3.0 V
+1.5 V
IV$_{DD}$
EV$_{DD}$
AV$_{DD}$
A/D I/F
ADSCK
ADDI3 to ADDI0
4
RF I/F
AGC
DCOFF
RFON
Memory I/F
MCSB
MA19 to MA0
20
MD15 to MD0
16
MRDB
MWRB
RXD
RTS
CTS
TXD
Host serial I/F
RESETB
WKUPB
STNDBYB
TXACTV
INVRX
System I/F
GPSCLK
REFCLK
Clock I/F
GND
AGND

PIN CONFIGURATION (TOP VIEW)

μPD77533S1-YHC

- 108-pin plastic fine-pitch BGA (11 × 11)



Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
A1	NC	C4	GND	G1	MA4	K10	ADSCK
A2	GND	C5	MWRB	G2	MA3	K11	EV$_{DD}$
A3	MA17	C6	I.C.	G3	EV$_{DD}$	K12	ADDI3
A4	MA19	C7	MRDB	G10	I.C.	L1	NC
A5	EV$_{DD}$	C8	STNDBYB	G11	EV$_{DD}$	L2	MD13
A6	IV$_{DD}$	C9	WKUPB	G12	NC	L3	MD11
A7	INVRX	C10	GND	H1	MA2	L4	MD9
A8	RESETB	C11	GND	H2	MA0	L5	MD6
A9	GND	C12	REFCLK	H3	MA1	L6	MD3
A10	AGND	D1	MA11	H10	RFON	L7	MD0
A11	NC	D2	MA12	H11	AGC	L8	RXD
A12	NC	D3	MA10	H12	DCOFF	L9	CTS
B1	GND	D10	GND	J1	MD15	L10	TXACTV
B2	MA15	D11	GPSCLK	J2	EV$_{DD}$	L11	GND
B3	MA16	D12	I.C.	J3	MD12	L12	GND
B4	MA18	E1	MA8	J10	ADDI0	M1	NC
B5	NC	E2	MA9	J11	ADDI2	M2	GND
B6	GND	E3	MA7	J12	ADDI1	M3	MD10
B7	MCSB	E10	I.C.	K1	GND	M4	MD8
B8	NC	E11	I.C.	K2	MD14	M5	MD5
B9	IV$_{DD}$	E12	I.C.	K3	GND	M6	GND
B10	AV$_{DD}$	F1	MA6	K4	MD7	M7	MD1
B11	EV$_{DD}$	F2	GND	K5	EV$_{DD}$	M8	IV$_{DD}$
B12	NC	F3	MA5	K6	MD2	M9	RTS
C1	MA13	F10	GND	K7	MD4	M10	TXD
C2	EV$_{DD}$	F11	I.C. -RL	K8	GND	M11	NC
C3	MA14	F12	I.C.	K9	GND	M12	NC

PIN IDENTIFICATION

ADDI3 to ADDI0:	AD data bus
ADSCK:	AD converter clock output
AGC:	Auto gain control
AGND:	Ground for PLL analog system
AV_{DD}:	Power supply for PLL analog system
CTS:	Clear to send
DCOFF:	DC OFF (DC trimming)
EV_{DD}:	Power supply for I/O pins
GND:	Ground
GPSCLK:	GPS clock
I.C., I.C.-RL:	Internal connection
INVRX:	RX invert
IV_{DD}:	Power supply for internal system
MA19 to MA0:	SnapShot memory address bus
MCSB:	SnapShot memory chip select
MD15 to MD0:	SnapShot memory bus
MRDB:	SnapShot memory read strobe
MWRB:	SnapShot memory write strobe
NC:	Non-connection
REFCLK:	Reference clock
RESETB:	Reset
RFON:	RF power ON
RTS:	Request to send
RXD:	Host serial data input
STNDBYB:	Standby
TXACTV:	Tx active
TXD:	Host serial data output
WKUPB:	Wakeup

CONTENTS

1. Pin Functions **9**

1.1 Explanation of Pin Functions **9**

1.2 Handling of Unused Pins **11**

1.2.1 Function pins 11

1.2.2 Non-function pins 11

1.2.3 Pin I/O circuits 12

2. OVERVIEW OF FUNCTIONS **13**

2.1 List of Functions and Commands **13**

2.2 Host Serial Commands **14**

2.2.1 Command format 14

2.2.2 Asynchronous serial 14

3. RESET **14**

3.1 Reset (Hardware Reset via RESETB Pin) **14**

3.1.1 Status of pins at hardware reset 15

3.1.2 Internal parameter initialization 16

3.2 Reset (Software Reset by Reset Command) **17**

3.3 Processing Required at Startup **17**

4. COMMUNICATION ERROR CONTROL **18**

4.1 Format Error, Undefined CMD-ID Error **18**

4.1.1 When command from host CPU is server command 18

4.1.2 When command from host CPU is local command 20

4.2 Timeout Error (Command Packet Interruption) **20**

4.2.1 Character timeout setting 21

4.2.2 Character timeout operation 21

4.3 Timeout Error (No Response from μPD77533) **22**

5. POSITIONING CONTROL **24**

5.1 Items That Must Be Set Before Start of Positioning **25**

5.1.1 Reference frequency 25

5.1.2 Frequency adjustment function (FCC) 25

5.1.3 TXACTV control 26

5.1.4 Doppler search range 26

5.1.5 Flow point output 26

5.2 RF Block ON **27**

5.2.1 AGC and DCOFF output signals 27

5.3 GPS Signal Reception (SnapShot Execution) **27**

5.4 RF Block OFF **28**

5.5 Pseudo Range Calculation **28**

5.6 Positioning Result Notification **28**

5.7 Positioning End Notification **28**

5.8 Commands That Cannot Be Sent During Positioning **28**

6. SYSTEM CONTROL 29
6.1 FCC Function 29
    6.1.1 Frequency Calibration Control 29
    6.1.2 FCC Reference Frequency 29
    6.1.3 Doppler Search Range 29
6.2 Baud Rate Setting 30
6.3 RF Power Supply Control 31
6.4 TXACTV Control 31

7. SHIFTING TO AND RESTORING FROM POWER SAVE MODE 32
7.1 Shifting to Power Save Mode via Power Control Command 33
7.2 Shifting to Power Save Mode via Sleep Timer Command 33
    7.2.1 Sleep timer setting 33
    7.2.2 Sleep Timer operation 34
7.3 Restoring from Power Save Mode 34
    7.3.1 Restoring via WKUPB pin input 34
    7.3.2 Restoring from power save mode via command reception 35

8. DEBUG FUNCTION 35

9. OTHER FUNCTIONS 36
9.1 Receiver ID Notification 36

10. OVERVIEW OF COMMAND FUNCTIONS 36

11. COMMAND TYPES AND RESPONSES 37
11.1 Commands 37
11.2 Responses 37

12. PACKET AND COMMAND FORMAT 37
12.1 Packet Format 38
    12.1.1 CMD-ID field 38
    12.1.2 LENGTH field 38
    12.1.3 DATA field 38
    12.1.4 FLAG field 38
12.2 Packet Division 39
12.3 Extension Command Format 39
12.4 Server Command Format 40
    12.4.1 Header 40
    12.4.2 Command ID 40
    12.4.3 Data field 40
    12.4.4 Checksum 41
    12.4.5 End 41
12.5 Packeting and De-packeting Server Commands 41

13. ACK, NACK, ERROR STATUS **41**

13.1 Format of ACK, NACK, Error Status **42**

13.1.1 ACK 42

13.1.2 NACK, error status 42

13.2 Flow of Commands, Reponses, ACK, and NACK **43**

13.2.1 ACK, NACK in case of local command 43

13.2.2 ACK, NACK in case of extension command 44

13.2.3 ACK, NACK in case of server command 45

14. HARDWARE FLOW CONTROL **45**

14.1 When Hardware Flow Control Is Not Performed **45**

14.1.1 Local commands 46

14.1.2 Extension commands 46

14.1.3 Server commands 47

15. SERVER COMMANDS **48**

15.1 Handling of Server Commands by Host CPU **48**

15.1.1 Server command analysis 49

15.1.2 Processing of baud rate change command (“Set Baud Rate”) 50

15.2 Set Baud Rate **50**

15.2.1 $BD 50

15.2.2 !bd 50

15.2.3 List of baud rates 51

16. LOCAL COMMANDS **51**

16.1 Baud Rate Change **52**

16.2 FCC Reference Frequency **53**

16.3 Power Control **53**

16.4 Reset **54**

16.5 RF Power Control **55**

16.6 Frequency Calibration Control **56**

16.7 TXACTV Control **56**

16.8 Sleep Timer **57**

16.9 Character Timeout **58**

16.10 !cb Request **59**

16.11 !ia Request **60**

16.12 SnapShot Memory Check **60**

16.13 Status Request **61**

16.14 D77533 Software Version **62**

16.15 Processing Status Request **63**

16.16 Rest Processing **63**

16.17 Error Status **64**

16.18 CA Parameter Set **65**

16.19 CB Parameter Set **66**

16.20 Doppler Search Range **67**

17. EXTENSION COMMANDS 68
17.1 Flow Point Control 68
17.2 Processing End 69

18. HOST CPU PROCESSING WHEN POSITIONING STARTS FROM RECEIVER SIDE 70

19. ELECTRICAL SPECIFICATIONS 71

20. APPLICATION CIRCUIT EXAMPLE 81

21. PACKAGE DRAWING 82

22. RECOMMENDED SOLDERING CONDITIONS 83

APPENDIX SAMPLE PROGRAM 84

1. PIN FUNCTIONS

1.1 Explanation of Pin Functions

- **Power supply**

Pin Name	Pin No.	I/O	Function
IV_{DD}	A6, B9, M8	–	Power supply for core (1.425 to 1.65 V)
EV_{DD}	A5, B11, C2, G3, G11, J2 ,K5, K11	–	Power supply for I/O pins (2.7 to 3.6 V)
GND	A2, A9, B1, B6, C4, C10, C11, D10, F2, F10, K1, K3, K8, K9, L11, L12, M2, M6	–	Ground
AV_{DD}	B10	–	Power supply for PLL analog (1.425 to 1.65 V)
AGND	A10	–	Ground for PLL analog

- **Clock control**

Pin Name	Pin No.	I/O	Function
GPSCLK	D11	Input	GPS reference clock input (16.384 MHz) Inputs the clock for operating the μPD77533.
REFCLK	C12	Input	Frequency adjustment reference clock (PDC: 14.4 MHz, PHS: 19.2 MHz, etc.)

- **System control**

Pin Name	Pin No.	I/O	Function
RESETB	A8	Input	Internal system reset input Initializes the μPD77533.
WKUPB	C9	Input	STOP mode release input WAKEUP interrupt input: Falling edge detection
STNDBYB	C8	Output	Standby status (STOP or HALT) 0: Standby status 1: Operating
TXACTV	L10	Input	ADDI data mask enable signal 0: Not masked 1: Masked
INVRX	A7	Input	RXD logic inversion setting signal This signal sets inversion of the host serial input RXD logic. 0: Not inverted 1: Inverted

- **Host serial interface**

Pin Name	Pin No.	I/O	Function
RXD	L8	Input	Asynchronous serial input
RTS	M9	Output	Request To Send
CTS	L9	Input	Clear To Send
TXD	M10	Output	Asynchronous serial output

- **RF control interface**

Pin Name	Pin No.	I/O	Function
AGC	H11	Output (3S)	Auto gain control See **Table 5-1 Output Values of AGC and DCOFF**.
DCOFF	H12	Output (3S)	DC offset control See **Table 5-1 Output Values of AGC and DCOFF**.
RFON	H10	Output	RF power control 0: RF off 1: RF on

- **A/D converter control**

Pin Name	Pin No.	I/O	Function
ADSCK	K10	Output	A/D conversion clock output
ADDI3 to ADDI0	K12, J11, J12 , J10	Input	A/D conversion data input (4 bits)

- **Memory interface**

Pin Name	Pin No.	I/O	Function
MCSB	B7	Output	SnapShot memory chip select Active-low signal.
MRDB	C7	Output	SnapShot memory read strobe Active-low signal.
MWRB	C5	Output	SnapShot memory write strobe Active-low signal.
MA19 to MA0	A4, B4, A3, B3, B2, C3, C1, D2, D1, D3, E2, E1, E3, F1, F3, G1, G2, H1, H3, H2	Output	SnapShot memory addresses (20 bits)
MD15 to MD0	J1, K2, L2, J3, L3, M3, L4, M4, K4, L5, M5, K7, L6, K6, M7, L7	I/O (3S)	SnapShot memory data (16 bits)

Remark The MD15 to MD0 pins indicated by “3S” in the I/O column become high impedance when the SnapShot memory (external data memory) is not accessed.

- **Other**

Pin Name	Pin No.	I/O	Function
I.C.	C6, D12, E10, E11, E12, F12, G10	–	Internally connected pins Leave these pins open.
I.C.-RL	F11	–	Internally connected pin Connect to GND via a resistor.
NC	A1, A11, A12, B5, B8, B12, G12, L1, M1, M11, M12	–	Non connect pins Leave these pins open.

Caution If a signal is input to these pins, or if they are read, the operation of the μPD77533 cannot be guaranteed.

1.2 Handling of Unused Pins

1.2.1 Function pins

Handle unused pins when the μPD77533 is mounted according to the table below.

Pin	I/O	Recommended Connection
ADSCK	Output	Leave open
ADDI3 to ADDI0	Input	Connect to GND
AGC	Output	Leave open
DCOFF	Output	Leave open
RFON	Output	Leave open
MCSB	Output	Leave open
MA19 to MA0	Output	Leave open
MD15 to MD0	I/O	Connect to EV_{DD} via a pull-up resistor, or to GND via a pull-down resistor
MRDB	Output	Leave open
MWRB	Output	Leave open
RXD	Input	Connect to GND
RTS	Output	Leave open
CTS	Input	Connect to GND
TXD	Output	Leave open
WKUPB	Input	Connect to EV_{DD} Note
STNDBYB	Output	Leave open
TXACTV	Input	Connect to GND
INVRX	Input	Connect to GND

Note Only when STOP mode is not used.

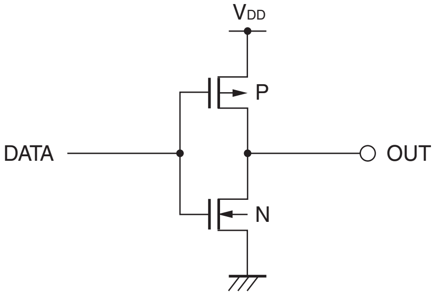
1.2.2 Non-function pins

Pin	I/O	Recommended Connection
I.C.	–	Leave open
I.C.-RL	–	Connect to GND via a resistor
NC	–	Leave open

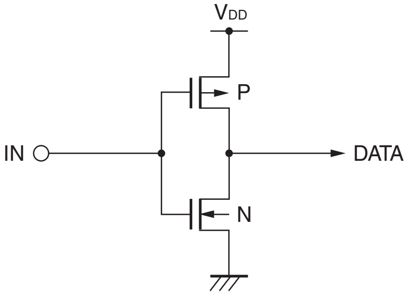
1.2.3 Pin I/O circuits

Pin Name	I/O Circuit Type	Pin Name	I/O Circuit Type
GPSCLK	C	AGC	B
REFCLK	C	DCOFF	B
RESETB	C	RFON	B
WKUPB	C	ADSCK	A
STNDBYB	A	ADDI3 to ADDI0	C
TXACTV	D	MCSB	A
INVRX	C	MRDB	A
RXD	D	MWRB	A
RTS	A	MA19 to MA0	A
CTS	D	MD15 to MD0	B
TXD	A		

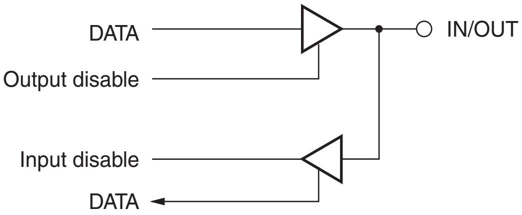
Type A


VDD
P
DATA
OUT
N

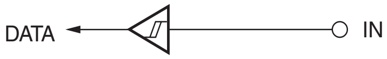
Type C


VDD
P
IN
DATA
N

Type B


DATA
Output disable
IN/OUT
Input disable
DATA

Type D



Schmitt-triggered input with hysteresis characteristics

2. OVERVIEW OF FUNCTIONS

2.1 List of Functions and Commands

The functions of the μPD77533 can be controlled by external pins and commands.

Table 2-1. List of μPD77533 Functions and Related Pins/Commands

Function		Related External Pins (Pin No.)	Related Commands (CMD-ID)
Reset		RESETB (A8)	Reset (0x04) !ia Request (0x21)
Communication error control		–	Error Status (0x3F) Command Timeout (0x13)
Positioning control		ADSCK (K10) RFON (H10) AGC (H11) DCOFF (H12)	Flow Point Control (0x7F) !cb Request (0x20)
System control	FCC	GPSCLK (D11) REFCLK (C12)	Frequency Calibration Control (0x06) FCC Reference Frequency (0x02) Doppler Search Range (0x57)
	Baud rate setting	–	Baud Rate Change (0x01)
	RF power supply control	RFON (H10)	RF Power Control (0x05)
	TXACTV control	TXACTV (L10)	TXACTV Control (0x07)
Shift to/restore from power save mode		WKUPB (C9) STNDBYB (C8)	Power Control (0x03) Sleep Timer (0x12)
Debug function		–	SnapShot Memory Check (0x2D) Status Request (0x30) Processing Status Request (0x33) Rest Processing (0x34) D77533 Software Version (0x31)
Other functions		–	CA Parameter Set (0x40) CB Parameter Set (0x41)

2.2 Host Serial Commands

2.2.1 Command format

The format of the serial commands used for communication between the μPD77533 and the host CPU is shown below. For details of the command ID (CMD-ID), data structure, etc., see **12 PACKET AND COMMAND FORMAT**.

Figure 2-1. Command Format

CMD-ID	LENGTH	DATA	FLAG
8 bits	8 bits	0 to 255 bytes	03H

2.2.2 Asynchronous serial

The asynchronous serial specifications are shown below.

Table 2-2. Asynchronous Serial Specifications

Baud rate	2400, 4800, 9600 (initial value), 19200, 38400, 57600, 115200 bps supported
Start bit	1 bit
Stop bit	1 bit
Data length	8 bits
Parity	None
★ Flow control	RTS (1: Stop transmission, 0: Request transmission) CTS (1: Stop transmission, 0: Request transmission)

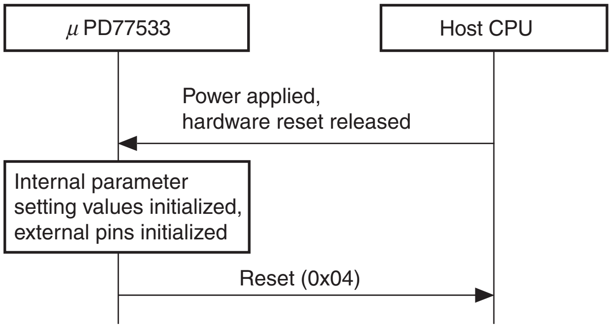
3. RESET

3.1 Reset (Hardware Reset via RESETB Pin)

The hardware of the μPD77533 is initialized by a reset via the RESETB pin.

Caution A hardware reset must be applied at the RESETB pin and a clock input to the GPSCLK pin when the power is turned on.

Figure 3-1. Hardware Reset


μ PD77533
Host CPU
Power applied,
hardware reset released
Internal parameter
setting values initialized,
external pins initialized
Reset (0x04)

3.1.1 Status of pins at hardware reset

The status of the pins when a hardware reset is applied (during RESETB pin input) is shown in Table 3-1 below.

Table 3-1. Status of Pins at Hardware Reset

Pin Name	Pin No.	I/O	Status
STNDBYB	C8	Output	Low-level output
RTS	M9	Output	High-level output
TXD	M10	Output	High-level output
AGC	H11	Output	Low-level output
DCOFF	H12	Output	Low-level output
RFON	H10	Output	Low-level output
ADSCK	K10	Output	Low-level output
MCSB	B7	Output	High-level output
MRDB	C7	Output	High-level output
MWRB	C5	Output	High-level output
MA19 to MA0	A4, B4, A3, B3, B2, C3, C1, D2, D1, D3, E2, E1, E3, F1, F3, G1, G2, H1, H3, H2	Output	Low-level output (all)
MD15 to MD0	J1, K2, L2, J3, L3, M3, L4, M4, K4, L5, M5, K7, L6, K6, M7, L7	I/O	High impedance

3.1.2 Internal parameter initialization

The internal parameters of the μPD77533 and their initial values are shown in Table 3-2 below.

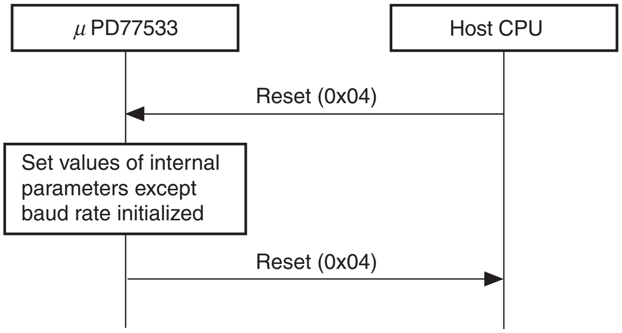
Table 3-2. Initial Values of Internal Parameters

Parameter Name	Related Command	CMD-ID (EXT-ID)	Initial Value	Meaning of Initial Value
Baud rate	Baud Rate Change	0x01	2	9600 bps
Reference frequency	FCC Reference Frequency	0x02	0xE10000	14.4 MHz. The parameter value is set to the reference frequency multiplied by 1.024 (initial value: 14.4 MHz x 1.024 = 0xE10000).
Power save mode	Power Control	0x03	0	No shift to power save mode; RUN mode is entered.
Power supply	RF Power Control	0x05	0	Power off
FCC control	Frequency Calibration Control	0x06	0	No FCC control
TXACTV control	TXACTV Control	0x07	0	No TXACTV control
Sleep mode	Sleep Timer	0x12	0	HALT mode is entered after Sleep Timer timeout
Sleep Timer timer value (time until sleep)			0	Sleep state is not entered
Timer value of character timeout (time until timeout)	Command Timeout	0x13	0	Timeout not detected.
Software version	D77533 Software Version	0x31	1	Ver.1.0
Status (status of μPD77533 signal processing)	Processing Status Request	0x33	–	Idle
Number of processed satellites	Rest Processing	0x34	0	0
Total number of satellites that should be processed			0	0
Current number of times positioning performed			0	0
Total number of times positioning performed			0	0
Error contents	Error Status	0x3F	–	No initial value
Doppler search range	Doppler Search Range	0x57	0x0168	Search range = 360 Hz
Flow points	Flow Point Control	0x7F (0x00)	All 0	Flow point output not controlled

3.2 Reset (Software Reset by Reset Command)

The internal parameters of the μPD77533, except for the baud rate, are initialized by the Reset command (0x04) from the host CPU.

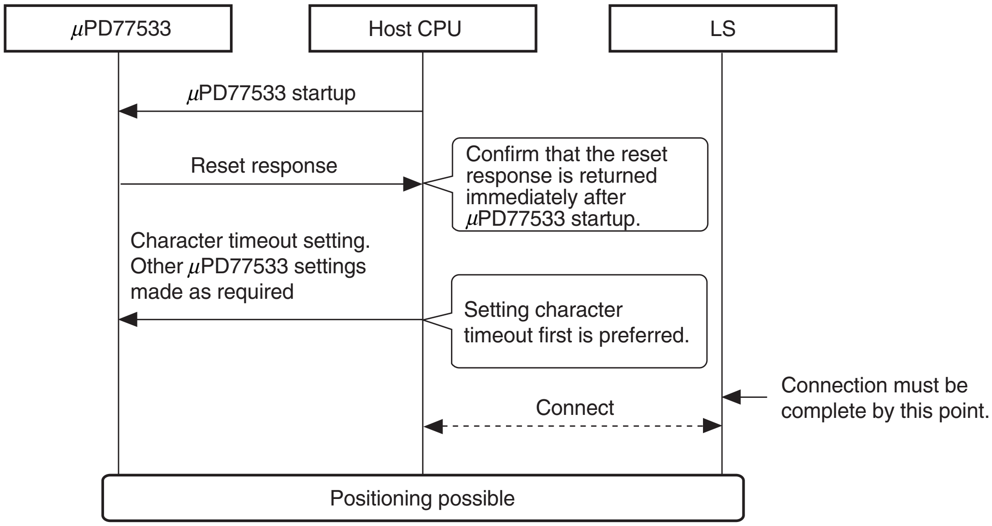
Figure 3-2. Software Reset


μ PD77533
Host CPU
Reset (0x04)
Set values of internal parameters except baud rate initialized
Reset (0x04)

3.3 Processing Required at Startup

When the GPS receiver is activated, the host CPU must perform processing using the procedure shown below.

Figure 3-3. Host CPU Processing Procedure at μPD77533 Startup


μPD77533
Host CPU
LS
μPD77533 startup
Reset response
Confirm that the reset response is returned immediately after μPD77533 startup.
Character timeout setting. Other μPD77533 settings made as required
Setting character timeout first is preferred.
Connection must be complete by this point.
Connect
Positioning possible

4. COMMUNICATION ERROR CONTROL

The following three types of communication errors (command errors hereafter) may occur in communication between the μPD77533 and the host CPU.

One of the following three error types is included in the data field of the Error Status command (CMD-ID: 0x3F).

Table 4-1. Types of Command Errors

Error Type	Error Definition
Format error	The format of the command is incorrect.
Timeout error	The specified time has elapsed while command packet reception is not complete, or the μPD77533 does not respond to a command from the host CPU.
Undefined CMD-ID error	An undefined numeric value has been used as the CMD-ID.

4.1 Format Error, Undefined CMD-ID Error

The control flow used when a format or undefined CMD-ID error occurs is described below.

For timeout error details, see **4.2 Timeout Error (Command Packet Interruption)** and **4.3 Timeout Error (No Response from μPD77533)**.

Caution The control flow used when a format or undefined CMD-ID error occurs during an uplink (μPD77533 → Host CPU) differs depending on whether the command from the host CPU is a server command or a local command.

4.1.1 When command from host CPU is server command

When the command is a server command, packet transmission/reception is confirmed on a packet by packet basis using ACK and NACK. For details of the ACK and NACK formats, see **13 ACK, NACK ERROR STATUS**.

If an error occurs during an uplink (see **Figure 4-1**), the host CPU sends NACK to the μPD77533, which then re-sends the same packet. The μPD77533 can send the next packet after receiving ACK from the host CPU.

If an error occurs during a downlink (host CPU → μPD77533) (see **Figure 4-2**), the μPD77533 sends NACK to the host CPU, which must then re-send the same packet. The host CPU can send the next packet after receiving ACK from the μPD77533.

Figure 4-1. Command Error Control Flow for Server Commands (Uplink Error)

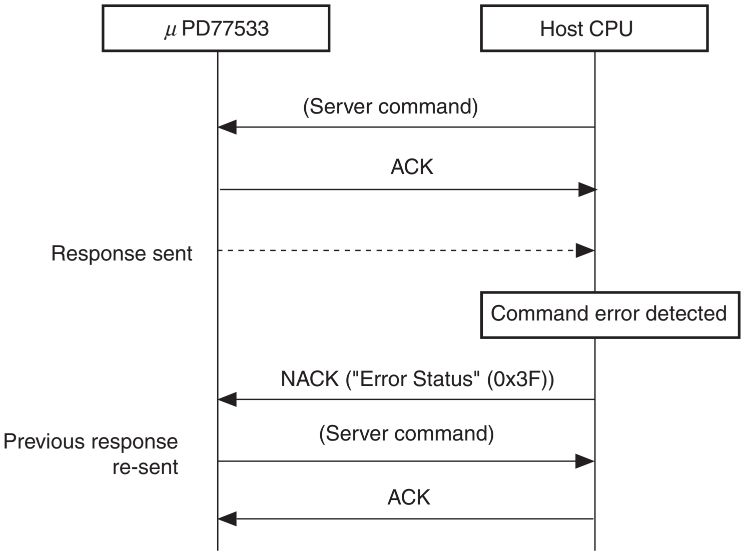
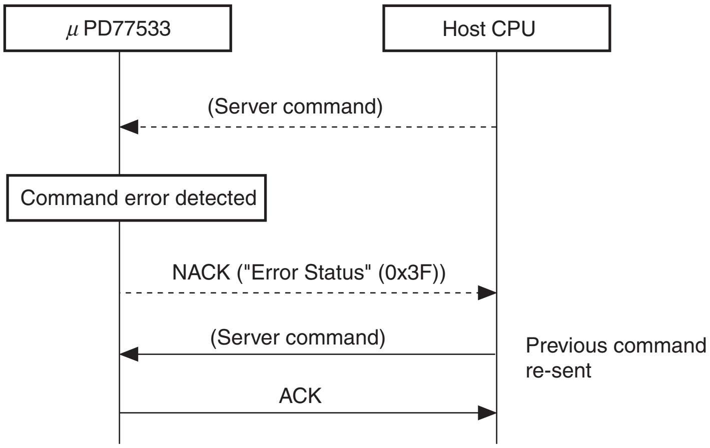

μ PD77533
Host CPU
(Server command)
ACK
Response sent
Command error detected
NACK ("Error Status" (0x3F))
Previous response re-sent
(Server command)
ACK

Figure 4-2. Command Error Control Flow for Server Commands (Downlink Error)


μ PD77533
Host CPU
(Server command)
Command error detected
NACK ("Error Status" (0x3F))
(Server command)
Previous command re-sent
ACK

4.1.2 When command from host CPU is local command

When the command is a local command, packet transmission/reception is not confirmed using ACK and NACK. A response to the command (command execution result) is used as the “ACK” from the μPD77533 to the host CPU. There is no equivalent to ACK for commands that receive no response (baud rate change command).

If an error occurs during an uplink (see **Figure 4-3**), even if the host CPU sends NACK to the μPD77533, the μPD77533 will perform no processing. The host CPU must therefore re-send the same packet without returning NACK.

If an error occurs during a downlink (see **Figure 4-4**), the μPD77533 sends the host CPU the Error Status command (CMD-ID: 0x3F). Once the host CPU receives the Error Status command (CMD-ID: 0x3F), the host CPU must re-send the same packet to the μPD77533.

Figure 4-3. Command Error Control Flow for Local Commands (Uplink Error)

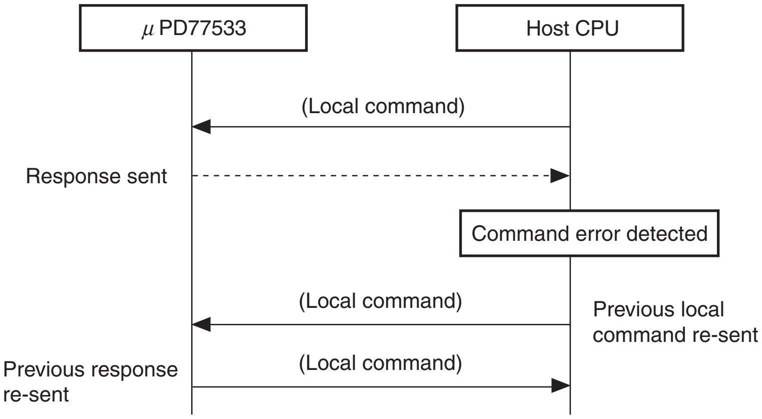
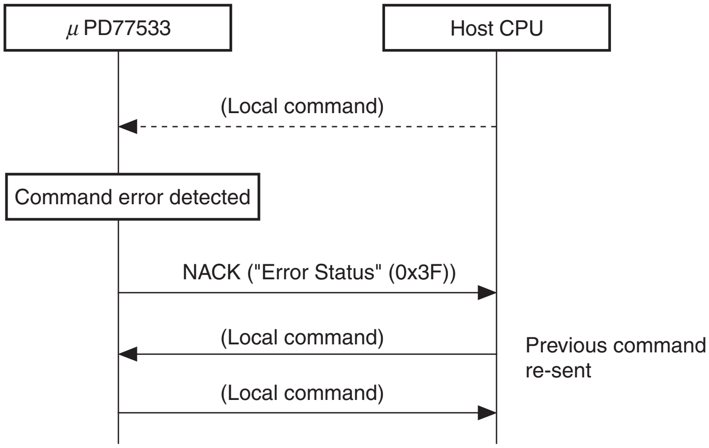

μ PD77533
Host CPU
(Local command)
Response sent
Command error detected
(Local command)
Previous local command re-sent
Previous response re-sent
(Local command)

Figure 4-4. Command Error Control Flow for Local Commands (Downlink Error)


μ PD77533
Host CPU
(Local command)
Command error detected
NACK ("Error Status" (0x3F))
(Local command)
Previous command re-sent
(Local command)

4.2 Timeout Error (Command Packet Interruption)

A timeout occurs in the μPD77533 if the specified time elapses after the last data of a packet is received while packet reception is not complete.

When a timeout occurs, the μPD77533 returns a timeout “Error Status” (CMD-ID: 0x3F), and all packet data received up to that point is discarded.

4.2.1 Character timeout setting

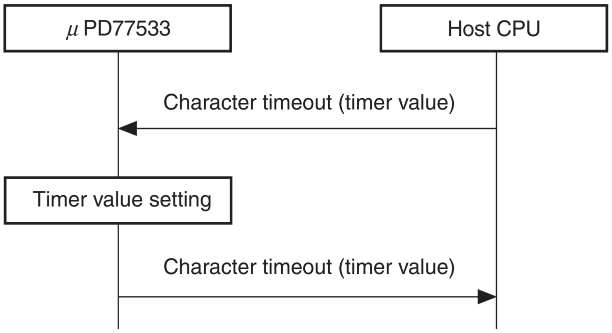
The initial value of the character timeout timer value is 0 (timeout not detected) (see **Table 3-2 Initial Values of Internal Parameters**).

It is therefore recommended to get the host CPU to set this timer to a value other than 0 immediately after the μPD77533 (GPS receiver) is activated (see **Figure 3-3 Host CPU Processing Procedure at μPD77533 Startup**).

The flow for setting the character timeout timer value is shown in Figure 4-5 below.

The timer value can be set to a time within a range of 0 to 16777215 ms. Note that timeout processing is not performed if the timer value is set to 0.

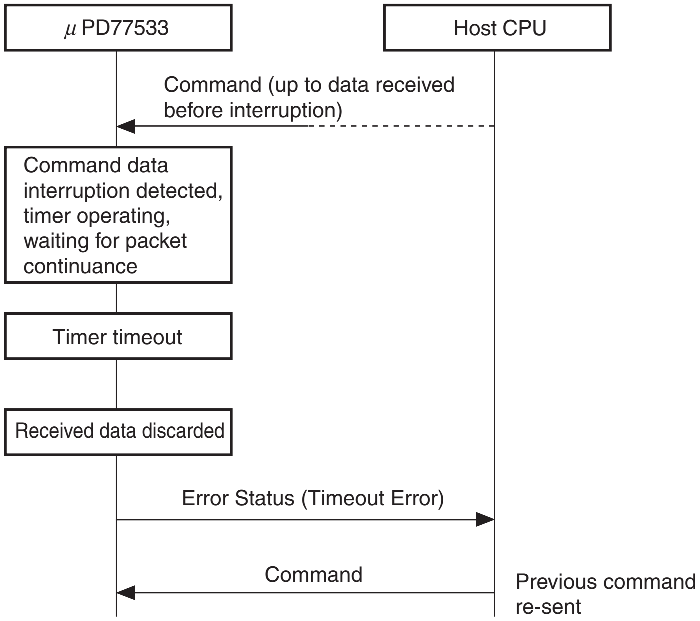
Figure 4-5. Character Timeout Timer Setting


μ PD77533
Host CPU
Character timeout (timer value)
Timer value setting
Character timeout (timer value)

4.2.2 Character timeout operation

The timeout operation that occurs as a result of the character timeout settings in 4.2.1 is shown in Figure 4-6 below. After discarding the command data received up to the interruption, the μPD77533 sends the host CPU the Error Status command (CMD-ID: 0x3F).

Figure 4-6. Character Timeout Occurrence


μ PD77533
Host CPU
Command (up to data received before interruption)
Command data interruption detected, timer operating, waiting for packet continuance
Timer timeout
Received data discarded
Error Status (Timeout Error)
Command
Previous command re-sent

4.3 Timeout Error (No Response from μPD77533)

If there is no response from the μPD77533 to a command from the host CPU, the host CPU must perform the following control processing.

(1) If a character timeout has been set in the μPD77533
A character timeout occurs in the μPD77533 and the host CPU is sent the Error Status command (CMD-ID: 0x3F) from the μPD77533. After receiving the Error Status command (CMD-ID: 0x3F), the host CPU resends the previous command (see **Figure 4-7**).

(2) If a character timeout has not been set in the μPD77533
If there is no response from the μPD77533 for 100 ms or more, the host CPU sends the μPD77533 '3F' in one byte units and waits to receive the Error Status command (CMD-ID: 0x3F) from the μPD77533. After receiving the Error Status command (CMD-ID: 0x3F), the host CPU resends the previous command (see Figure 4-8).

It is recommended to preset a character timeout in the μPD77533 because if not (case (2) above), it often takes a long time for the μPD77533 to be restored.

Figure 4-7. Processing Flow When There Is No Response from μPD77533 (Character Timeout Set)

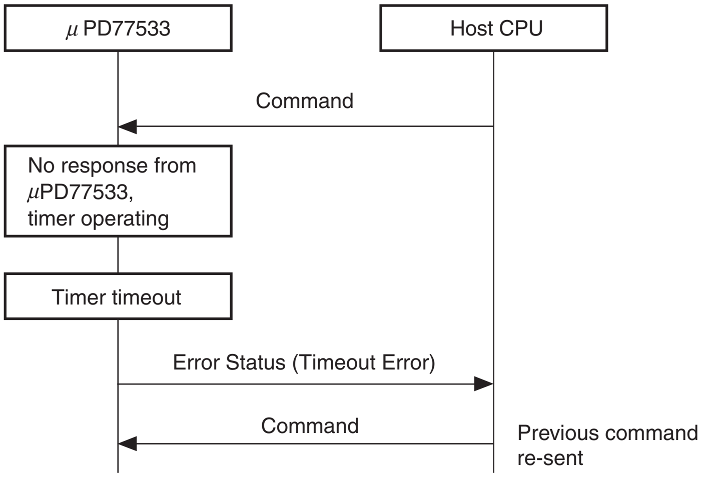
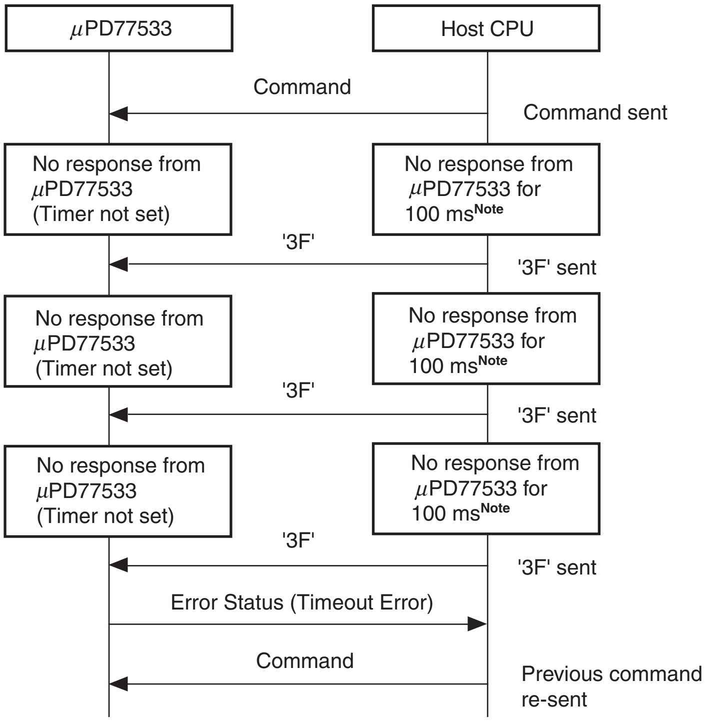

μ PD77533
Host CPU
Command
No response from μPD77533, timer operating
Timer timeout
Error Status (Timeout Error)
Command
Previous command re-sent

Figure 4-8. Processing Flow When There Is No Response from μPD77533 (Character Timeout Not Set)


μPD77533
Host CPU
Command
Command sent
No response from μPD77533 (Timer not set)
No response from μPD77533 for 100 msNote
'3F'
'3F' sent
No response from μPD77533 (Timer not set)
No response from μPD77533 for 100 msNote
'3F'
'3F' sent
No response from μPD77533 (Timer not set)
No response from μPD77533 for 100 msNote
'3F'
'3F' sent
Error Status (Timeout Error)
Command
Previous command re-sent

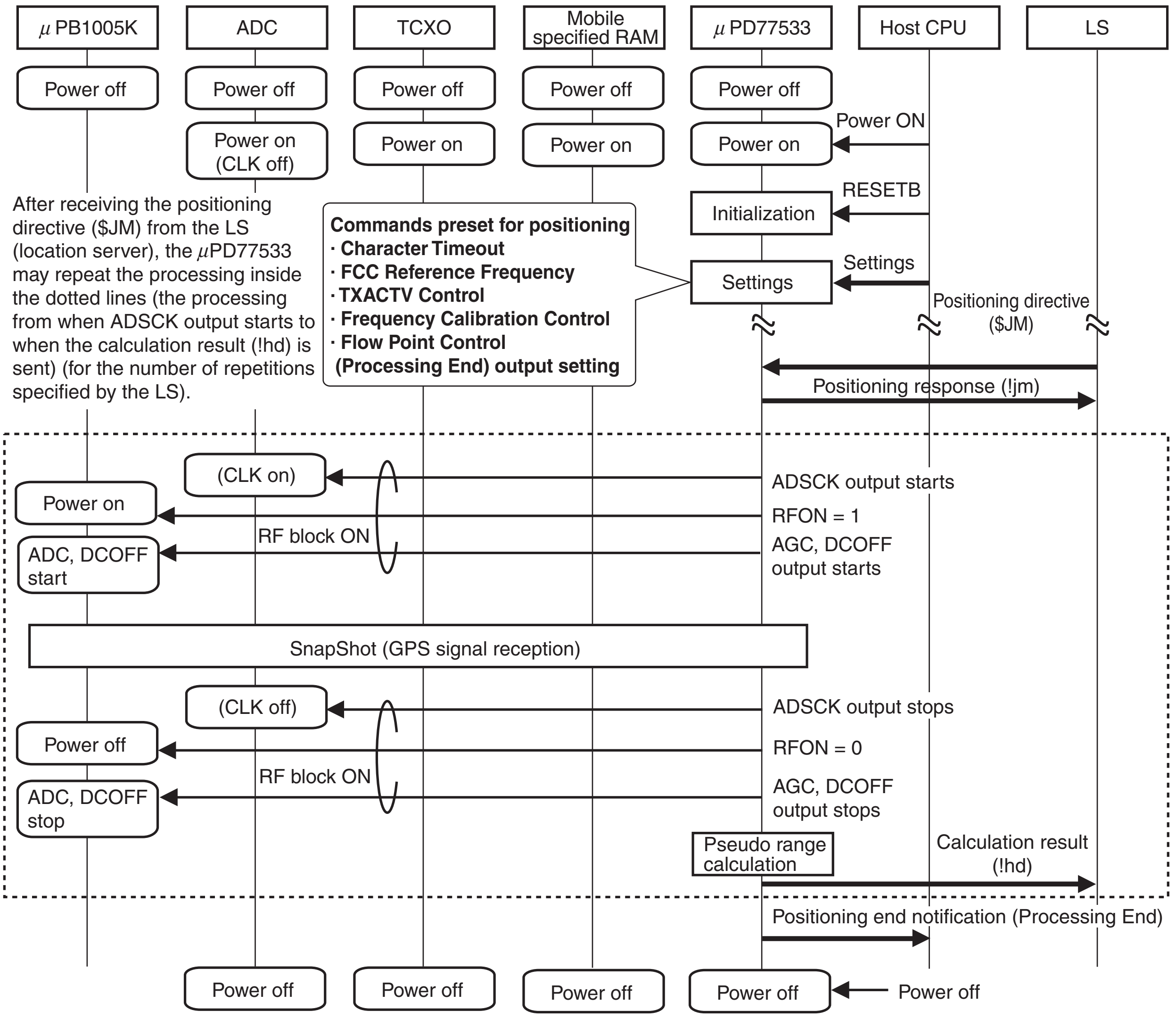
Note When the μPD77533 is performing positioning, “no response” is judged to have occurred after 1 second (if there is no response from the μPD77533 within 1 second, “no response” is judged to have occurred and ‘3F’ is sent). “No response” judgement following transmission of ‘3F’ is made after 100 ms.

5. POSITIONING CONTROL

An outline of the positioning flow is shown in Figure 5-1 below.

Caution ACK and NACK have been omitted from the figure.
The ADC power can also be controlled by the RFON pin, but care must be taken with the control logic.

★ **Figure 5-1. Outline of Positioning Control**


μPB1005K
ADC
TCXO
Mobile specified RAM
μPD77533
Host CPU
LS
Power off
Power off
Power off
Power off
Power off
Power on (CLK off)
Power on
Power on
Power on
Power ON
RESETB
Initialization
Settings
Settings
Positioning directive ($JM)
Positioning response (!jm)
(CLK on)
ADSCK output starts
Power on
RFON = 1
RF block ON
ADC, DCOFF start
AGC, DCOFF output starts
SnapShot (GPS signal reception)
(CLK off)
ADSCK output stops
Power off
RFON = 0
RF block ON
ADC, DCOFF stop
AGC, DCOFF output stops
Pseudo range calculation
Calculation result (!hd)
Positioning end notification (Processing End)
Power off
Power off
Power off
Power off
Power off

After receiving the positioning directive ($JM) from the LS (location server), the μPD77533 may repeat the processing inside the dotted lines (the processing from when ADSCK output starts to when the calculation result (!hd) is sent) (for the number of repetitions specified by the LS).

Commands preset for positioning
- **Character Timeout**
- **FCC Reference Frequency**
- **TXACTV Control**
- **Frequency Calibration Control**
- **Flow Point Control (Processing End) output setting**

Remark ⬅ Command input
← Pin input

The control flow when positioning starts from the receiver side differs depending on the system.

An example of the control flow when positioning starts from the receiver side in a GPS evaluation system is described below.

When positioning starts from the receiver side, the host CPU sends the μPD77533 the !cb Request command (CMD-ID: 0x20). Upon receiving the !cb Request command (CMD-ID: 0x20), the μPD77533 sends the !cb command to the LS (location server) following the procedure shown in 16.10 !cb Request. After receiving "!cb", the LS starts positioning.

If positioning is requested by the LS, this command does not need to be used.

After receiving the positioning command "$JM" from the LS (host CPU), the μPD77533 automatically executes the following processing, as illustrated in Figure 5-1.

1. RF block ON
2. GPS signal reception
3. RF block OFF
4. Pseudo range calculation
5. Calculation result notification
6. Positioning end notification

5.1 Items That Must Be Set Before Start of Positioning

The items that must be set before positioning starts after the GPS module is activated are described below.

5.1.1 Reference frequency

Command: FCC Reference Frequency (CMD-ID: 0x02)

The initial value of the reference clock frequency is 14.4 MHz. To use a reference clock with a frequency of other than 14.4 MHz, change this setting using the FCC Reference Frequency command (CMD-ID: 0x02).

5.1.2 Frequency adjustment function (FCC)

Command: Frequency Calibration Control (CMD-ID: 0x06)

For details of the frequency adjustment function (FCC: Frequency Calibration Control), see **6.1.1 Frequency Calibration Control**.

The initial value is "No FCC".

If the μPD77533 local clock (GPSCLK input clock signal, 16.384 MHz) is not accurate enough, in other words, to use the reference clock (REFCLK input signal) for frequency adjustment, change this setting to "FCC control used" using the Frequency Calibration Control command (CMD-ID: 0x06).

5.1.3 TXACTV control

Command: TXACTV Control (CMD-ID: 0x07)

For details of TXACTV control, see **6.4 TXACTV Control**.

The initial value is “No TXACTV control”.

To avoid degradation of the GPS reception signal due to PDC/PHS RF output noise or noise from another such source, input a signal to the TXACTV input pin indicating that PDC/PHS signals are being transmitted and change this setting to “TXACTV control used” using the TXACTV Control command (CMD-ID: 0x07). By doing this, the A/D converter output value when the TXACTV input signal = 1 (PDC/PHS transmission in progress) is masked by 0x0 in the μPD77533.

5.1.4 Doppler search range

Command: “Doppler Search Range” (CMD-ID: 0x57)

The Doppler search range setting value differs depending on the frequency and accuracy of the reference clock and the local clock used. For details of how to calculate the Doppler search range (calculation formula), see **6.1.3 Doppler Search Range**.

5.1.5 Flow point output

Command: Flow Point Control (CMD-ID: 0x7F)

The initial value of the flow point output is “Flow point output not controlled”.

The output of the Processing End flow point must be set to ON in order for the μPD77533 to notify the host CPU that all positioning is complete.

For flow point details, see **17.1 Flow Point Control**.

5.2 RF Block ON

When positioning starts, the μPD77533 automatically controls the external pins ADSCK, RFON, AGC, and DCOFF as follows, and switches on the RF block. Specifically, the μPD77533 performs the following.

- Start of ADSCK signal output
- RFON = 1 (down converter power on)
- Start of AGC, DCOFF signal output

5.2.1 AGC and DCOFF output signals

The values of the AGC and DCOFF output signals are shown in Table 5-1, together with the corresponding TXACTV input signal values and ADDI3 to ADDI0 input signal values (values input from A/D converter; 4 bits).

DCOFF is a feedback signal to the RF block and is used to adjust the RF offset. AGC is a feedback signal to the RF block and is used to adjust the RF gain.

When TXACTV is 1 (PDC/PHS transmitting), both DCOFF and AGC are high-impedance output, regardless of the value input from ADC.

Table 5-1. Output Values of AGC and DCOFF

TXACTV (Input Signal)	ADDI3 to ADDI0 (Values Input from A/D Converter)	DCOFF (Output Signal)	AGC (Output Signal)
1	All cases	High-impedance	High-impedance
0	0b1111	1	0
0	0b1110	1	0
0	0b1101	1	0
0	0b1100	1	0
0	0b1011	1	0
0	0b1010	1	1
0	0b1001	1	1
0	0b1000	High-impedance	1
0	0b0111	0	1
0	0b0110	0	1
0	0b0101	0	0
0	0b0100	0	0
0	0b0011	0	0
0	0b0010	0	0
0	0b0001	0	0
0	0b0000	0	0

5.3 GPS Signal Reception (SnapShot Execution)

The μPD77533 executes GPS signal reception. Details of this processing are not available to users.

- The SnapShot execution time is set to the time specified by the host CPU or the server command "$JM".
- The signal processing operation during SnapShot execution (the "status" parameter) becomes "SnapShot in progress".
- The signal processing operation at the end of SnapShot execution (the "status" parameter) becomes "Idling".

Remark See **Table 3-2 Initial Values of Internal Parameters** for details of the status parameter.

5.4 RF Block OFF

The μPD77533 switches off the RF block after receiving the GPS signal. Specifically, the μPD77533 performs the following.

- ADSCK signal output stopped
- RFON = 0 (down converter power off)
- AGC, DCOFF signal output stopped

5.5 Pseudo Range Calculation

The μPD77533 calculates the pseudo range using the received GPS signal.

5.6 Positioning Result Notification

The μPD77533 issues the server command “!hd” after calculating the pseudo range and notifies the LS of the result via the host CPU.

5.7 Positioning End Notification

The μPD77533 sends the Processing End command (CMD-ID: 0x7F) after all positioning has finished to inform the host CPU that all positioning is complete.

Note that it is necessary to set flow point output control to ON using the Flow Point Control command (CMD-ID: 0x7F) before starting positioning.

Be aware that the initial value of the Flow Point Control command (CMD-ID: 0x7F) is “Flow point output not controlled”.

After receiving this response, the host CPU can perform processing such as switching off the power of the peripheral modules (such as the Mobile specified RAM).

5.8 Commands That Cannot Be Sent During Positioning

The commands that the host CPU is prohibited from sending while the μPD77533 is performing positioning are shown in Table 5-2.

Table 5-2. Commands That Cannot Be Sent During Positioning

Command Type	CMD-ID	Command
Local command	0x01	Baud Rate Change
	0x05	RF Power Control
	0x06	Frequency Calibration Control
	0x07	TXACTV Control

6. SYSTEM CONTROL

6.1 FCC Function

The FCC (Frequency Calibration Control) function is used to adjust the accuracy of the local clock (clock signal input to GPSCLK input pin; 16.384 MHz) using the higher-accuracy reference clock (clock signal input to REFCLK input pin).

With the μPD77533, the reference clock used by the FCC function must be supplied during positioning (during SnapShot) processing. The accuracy of the local clock adjusted by FCC is up to about ±10 ppm.

The following three commands are related to the FCC function.

- “Frequency Calibration Control” (CMD-ID:0x06)
- “FCC Reference Frequency” (CMD-ID:0x02)
- “Doppler Search Range” (CMD-ID:0x57)

6.1.1 Frequency Calibration Control

The Frequency Calibration Control command (CMD-ID: 0x06) is used to set and check whether the FCC function is enabled or disabled.

For how to set this command, see **16.6 Frequency Calibration Control**.

6.1.2 FCC Reference Frequency

The FCC Reference Frequency command (CMD-ID: 0x02) is used to set and check the reference frequency for the FCC function.

The initial setting for the reference clock is 14.4 MHz.

The value is set as 4-byte data that is the frequency multiplied by 1.024 (therefore the initial value is 0x00E10000 (14.4×1.024)).

For details, see **16.2 FCC Reference Frequency**.

6.1.3 Doppler Search Range

The Doppler Search Range command (CMD-ID: 0x57) is used to set the Doppler search range (search range of receive signal frequency). The initial value is 360 Hz (0x168).

The value to be set as the Doppler search range is determined by the local and reference clocks. The calculation method is shown below.

The minimum value of the Doppler search range parameter (Hz) is calculated as follows.

Doppler search range = $2 \times \left|\, |DIFr| - |DIFi| \,\right|$ (Hz)

When:

$$DIFr = -1.57542G \times \frac{\left[REFCLKr \times 2 \times \dfrac{16.384M \times 1.024}{GPSCLKr} + 1\right] - REFCLKi \times 2 \times 1.024}{REFCLKi \times 2 \times 1.024}$$

$$DIFi = -1.57542G \times \frac{\left[REFCLKi \times 2 \times \dfrac{16.384M \times 1.024}{GPSCLKr}\right] - REFCLKi \times 2 \times 1.024}{REFCLKi \times 2 \times 1.024}$$

REFCLKi:	Logical value of REFCLK
REFCLKr:	Actual value of REFCLK (including error)
GPSCLKr:	Actual value of GPSCLK (including error)

<Example>

REFCLKi:	14.4 MHz
REFCLKr:	14.400002 MHz
GPSCLKr:	16.383900 MHz

$$DIFr = -1.57542G \times \frac{\left[14.400002M \times 2 \times \dfrac{16.384M \times 1.024}{16.383900M} + 1\right] - 14.4M \times 2 \times 1.024}{14.4M \times 2 \times 1.024} = -9882.7$$

$$DIFi = -1.57542G \times \frac{\left[14.4M \times 2 \times \dfrac{16.384M \times 1.024}{16.383900M}\right] - 14.4M \times 2 \times 1.024}{14.4M \times 2 \times 1.024} = -9615.6$$

Therefore:

Doppler Search Range = $2\times \left|\, |-9882.7| - |-9615.6| \,\right| = 534.2$ (Hz)

For details, see **16.20 Doppler Search Range**.

6.2 Baud Rate Setting

The Baud Rate Change command (CMD-ID: 0x01) is used to set the baud rate between the μPD77533 and the host CPU. There are seven settable baud rates: 2400 bps, 4800 bps, 9600 bps (initial value), 19200 bps, 38400 bps, 57600 bps, and 115200 bps.

If the μPD77533 receives the Baud Rate Change command (CMD-ID: 0x01) while another command is being received, the baud rate will be set after reception of that command is complete.

6.3 RF Power Supply Control

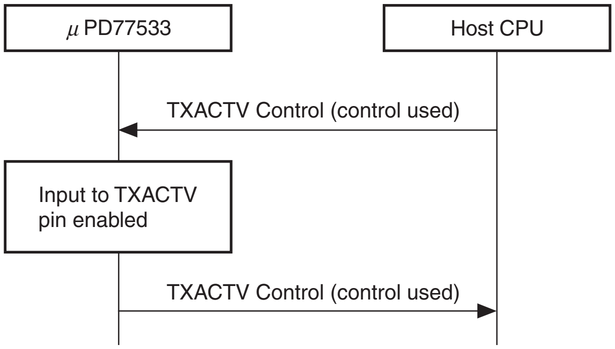
The RF Power Control command (CMD-ID: 0x05) is used to control the power supply of the RF block.

During positioning (while the GPS signal is being received), control from the host CPU via this command is not required because the μPD77533 autonomously controls the RF block at this time.

6.4 TXACTV Control

The TXACTV Control command (CMD-ID: 0x07) is used to set mask control of the A/D conversion value via TXACTV pin input.

Figure 6-1. TXACTV Control


μ PD77533
Host CPU
TXACTV Control (control used)
Input to TXACTV pin enabled
TXACTV Control (control used)

The μPD77533 has a function to mask the A/D converter output value by 0H in cases such as when the GPS receive signal is degraded due to noise at the PDC/PHS RF output, or other such noise. The initial setting is “A/D conversion value unmasked”.

When a signal indicating that PDC/PHS signal transmission is in progress is input to the TXACTV pin and TXACTV control is set to “A/D conversion value masked” (TXACTV input signal = 1; PDC/PHS transmitting), the signal received in the μPD77533 is handled as 0b0000 regardless of the contents of the A/D converter output value (value input to the ADDI3 to ADDI0 pins) (in other words, the A/D conversion value is masked). At the same time, the output of the AGC and DCOFF pins becomes high-impedance.

See **5.2.1 AGC and DCOFF output signals** for the relationship between the TXACTV signal input and AGC/DCOFF pin output.

7. SHIFTING TO AND RESTORING FROM POWER SAVE MODE

The μPD77533 has two power save modes: STOP and HALT.

The μPD77533 is shifted from normal mode (RUN) to STOP or HALT mode via the Power Control command (CMD-ID: 0x03) or Sleep Timer command (CMD-ID: 0x12).

The status is shifted back to the normal mode from the STOP or HALT mode when the WKUPB pin input is 0, or, in the case of the HALT mode, by the reception of any command. An outline of the status transition to and from the power save modes is shown in Figure 7-1.

The HALT and STOP modes differ as follows.

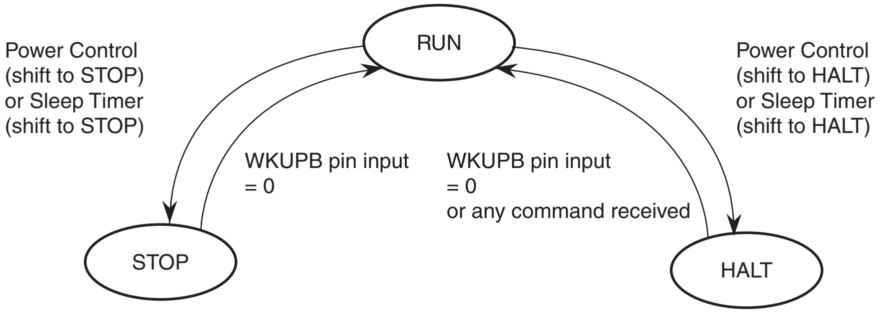
[HALT]

- The PLL circuit in the μPD77533 is not stopped, allowing fast transition back to normal mode.
- Transition back to normal mode (RUN) occurs when the WKUPB pin input is 0 or by the reception of any command.

[STOP]

★ - The PLL circuit in the μPD77533 is stopped, reducing the power consumption to 700 μA (MAX.).
- Transition back to normal mode (RUN) occurs when the WKUPB pin input is 0.

Figure 7-1. Shifting to and Restoring from Power Save Mode


RUN
Power Control
(shift to STOP)
or Sleep Timer
(shift to STOP)
Power Control
(shift to HALT)
or Sleep Timer
(shift to HALT)
WKUPB pin input
= 0
WKUPB pin input
= 0
or any command received
STOP
HALT

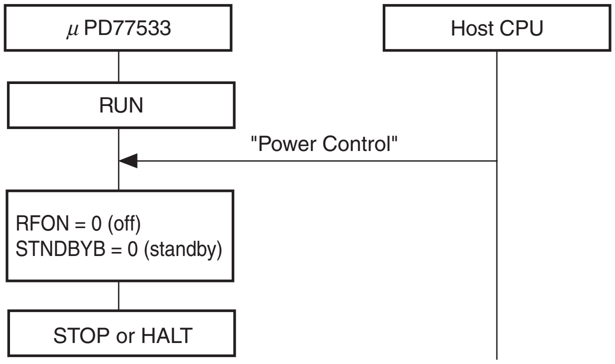
7.1 Shifting to Power Save Mode via Power Control Command

The Power Control command (CMD-ID: 0x03) (with STOP or HALT set in the data field) is used to shift the μPD77533 to a power save mode (STOP or HALT). The flow of this transition is shown in Figure 7-2.

When the μPD77533 receives the Power Control command (CMD-ID: 0x03), it shifts to the specified power save mode after completing the following processing.

- Turning off the power of the RF block (RFON output signal = 0)
- Indicating externally that the status has been shifted to standby mode (STNDBYB output signal = 0: standby mode)

Figure 7-2. Flow of Shifting to Standby Mode



7.2 Shifting to Power Save Mode via Sleep Timer Command

If the idle state (state in which signal processing is stopped and there is no serial command transmission/reception) continues for the timeout period set by the Sleep Timer command (CMD-ID: 0x12), the μPD77533 shifts to a power save mode (STOP or HALT). The mode to be shifted to is preset by the Sleep Timer command (CMD-ID: 0x12).

Sleep timer setting/setting notification and sleep timer operation are described below.

7.2.1 Sleep timer setting

The sleep timer is set by the Sleep Timer command (CMD-ID: 0x12).

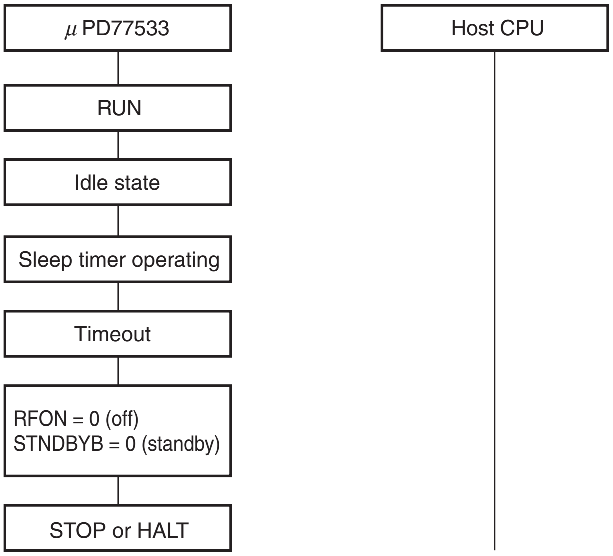
The mode to be shifted to (STOP or HALT) and the timer value at which the shift takes place are included in the data field of the Sleep Timer command.

The Sleep Timer timeout value can be set within a range of 0 to 16777215 (0xFFFFFF) ms. Note, however, that if 0 ms is set, the μPD77533 will not shift to a power save mode.

7.2.2 Sleep Timer operation

The operation that is performed when a timeout occurs in the Sleep Timer set in 7.2.1 is shown in Figure 7-3.

Figure 7-3. Sleep Timer Timeout


μPD77533
Host CPU
RUN
Idle state
Sleep timer operating
Timeout
RFON = 0 (off)
STNDBYB = 0 (standby)
STOP or HALT

Remark Idle state: State in which signal processing is stopped and there is no serial command transmission/reception.

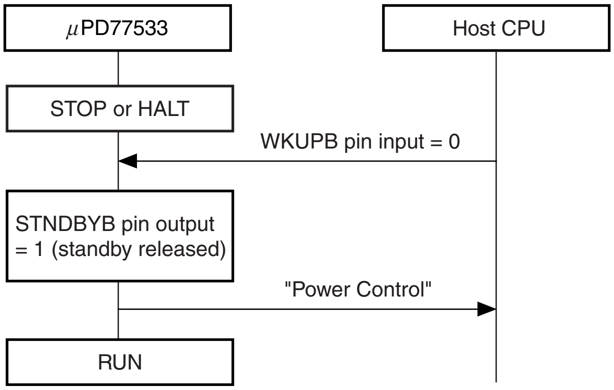
7.3 Restoring from Power Save Mode

The μPD77533 returns from a power save mode (STOP or HALT) to normal mode (RUN) when the WKUPB pin input is 0 or (HALT mode only) when any command is received. The WKUPB pin is active low.

7.3.1 Restoring via WKUPB pin input

The procedure for returning to normal mode from a power save mode via WKUPB pin input is shown in Figure 7-4.

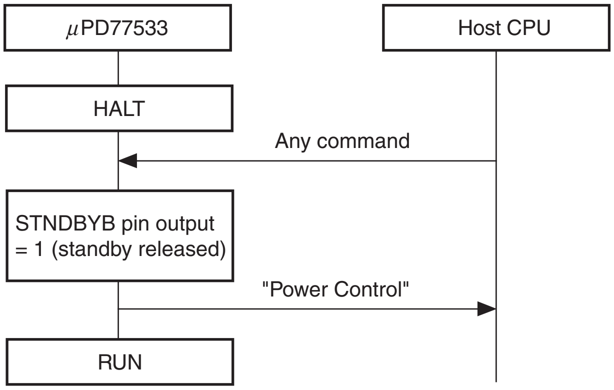
Figure 7-4. Restoring from Power Save Mode (via WKUPB Pin Input)


μPD77533
Host CPU
STOP or HALT
WKUPB pin input = 0
STNDBYB pin output = 1 (standby released)
"Power Control"
RUN

7.3.2 Restoring from power save mode via command reception

The procedure for returning to normal mode from HALT mode via command reception is shown in Figure 7-5. The μPD77533 shifts from HALT mode back to normal mode via WKUPB pin input control or by serial command reception.

Figure 7-5. Restoring from Power Save Mode (HALT) (via Command Reception)



8. DEBUG FUNCTION

The debug function enables the following items to be checked via local commands.
For further details, see the explanation of each command.

Table 8-1. Debug Function

Item	Command	Function
SnapShot memory check	“SnapShot Memory Check” (CMD-ID:0x2D)	Writes values such as 0x0000 and 0xFFFF to the entire SnapShot memory area and checks whether these values can be read correctly.
Serial communication operation check	“Status Request” (CMD-ID:0x30)	Checks the operation of the μPD77533. The μPD77533 simply sends a response to this command.
Signal processing status reference	“Processing Status Request” (CMD-ID:0x33)	References the signal processing status of the μPD77533 (one of the following). • Executing SnapShot • Processing signal • Idling During positioning control, the μPD77533 shifts to the “executing SnapShot” or “processing signal” mode. For the mode transition timing, see **5. POSITIONING CONTROL**.
Positioning progress reference	“Rest Processing” (CMD-ID:0x34)	References the number of satellites remaining to be processed and the number of times positioning has been executed.
μPD77533 version reference	“D77533 Software Version” (CMD-ID:0x31)	References the version of the μPD77533 software (firmware).

9. OTHER FUNCTIONS

9.1 Receiver ID Notification

The following two commands are used to notify the LS (location server) of a receiver's ID number. For details, see **16.18 CA Parameter Set**, and **16.19 CB Parameter Set**.

- CA Parameter Set (CMD-ID: 0x40)
  This command sets or sends notification of the setting of the GPS module's unit ID and software version.

- CB Parameter Set (CMD-ID: 0x41)
  This command sets or sends notification of the setting of the parameters when the positioning start command (!cb) is issued by the μPD77533.

10. OVERVIEW OF COMMAND FUNCTIONS

The configuration of a system using the μPD77533 is shown below. This section will explain the command interface used between the μPD77533 and the host CPU.

Figure 10-1. Diagram of GPS Positioning System Configuration

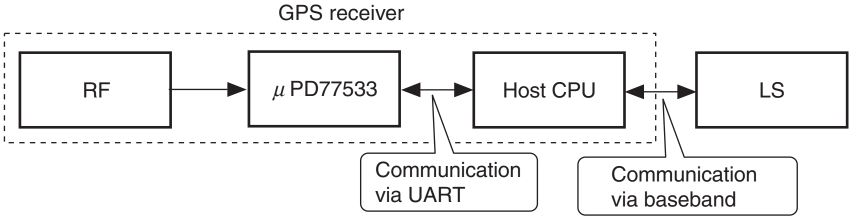


Table 10-1. Function of Each Block

Block	Function
RF	Radio Frequency Receives radio waves from a GPS satellite and transfers them to the μPD77533.
μ PD77533	Consists of a DSP core and logic circuits. Processes the signals sent from the RF block and transfers them to the LS.
Host CPU	Controls the operations of the μPD77533 and is the mediator in communication between the μPD77533 and the LS.
LS	Location Server Determines the position of the GPS receiver based on the results of the signal processing performed by the μPD77533.

11. COMMAND TYPES AND RESPONSES

The μPD77533 executes the commands received from the host CPU or LS and returns a response. Commands issued from the LS are called server commands and those issued by the host CPU are called local commands. The command acknowledgement issued from the μPD77533 to the LS or host CPU is called a response.

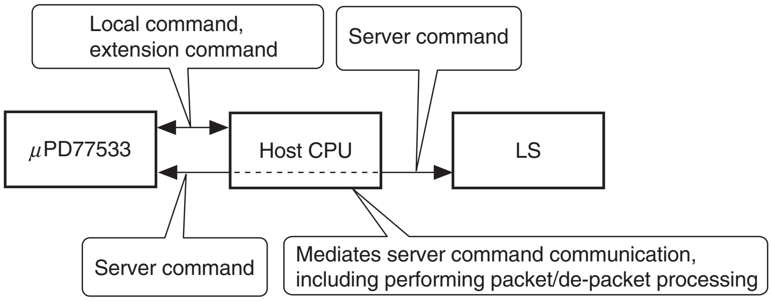
11.1 Commands

The following three types of commands are used.

Table 11-1. Command Types

Type	Issued by	Function
Local command	Host CPU	Used for setting, controlling, and obtaining the current status of the μPD77533. These commands are issued by the host CPU, which also receives the response from the μPD77533. The LS is not involved in this operation.
Extension command	Host CPU	Provided to extend the local commands. These commands are issued by the host CPU, which also receives the response from the μPD77533. The LS is not involved in this operation.
Server command	LS	Used for positioning. These commands are issued by the LS, which also receives the response from the μPD77533. The host CPU converts the data format (into/from packets) and operates as the mediator in the command/response communication between the μPD77533 and the LS. Some server commands must also be processed the host CPU.

Figure 11-1. Relationship Between Commands and Function Blocks


Local command,
extension command
Server command
μPD77533
Host CPU
LS
Server command
Mediates server command communication,
including performing packet/de-packet processing

11.2 Responses

There are two types of responses: those sent in response to a command, and those sent by the μPD77533 spontaneously.

For the former, there is not necessarily one response per command; there are commands for which multiple responses are returned, or for which no response is returned.

For the latter, the response transmission timing is undefined, so the host CPU must always be in a state in which it can receive responses from the μPD77533.

12. PACKET AND COMMAND FORMAT

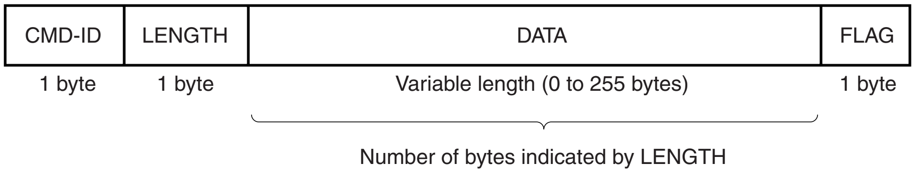
Data is transferred between the μPD77533 and the host CPU in the form of packets.

The data is 8-bit binary data (little endian), and does not necessarily consist only of ASCII characters.

12.1 Packet Format

The packet format is shown below. The maximum packet length is 258 bytes.

Figure 12-1. Packet Format


CMD-ID
LENGTH
DATA
FLAG
1 byte
1 byte
Variable length (0 to 255 bytes)
1 byte
Number of bytes indicated by LENGTH

12.1.1 CMD-ID field

The CMD-ID field indicates the type of command: local, extension, or server.

In the case of local commands, the CMD-ID value indicates the actual command identifier.

Table 12-1. Types of CMD-ID

CMD-ID	Type
0x00 to 0x7E	Local command (CMD-ID value indicates command identifier)
0x7F	Extension command
0x80	Server command

12.1.2 LENGTH field

The LENGTH field indicates the number of bytes in the DATA field. A value in the range of 0 to 255 is specified. The length does not include CMD-ID, LENGTH, and FLAG.

12.1.3 DATA field

The DATA field stores the data and status of the command specified by CMD-ID.

The length of the DATA field is indicated by the value in the LENGTH field. Some commands do not include a DATA field.

When the host CPU packets the server command, if the total length of the server command in an unpacketed state exceeds 255 bytes, the data is divided into multiple packets.

If the size of one data item is 2 or more bytes, the data is stored in the order of lower byte(s) then higher byte(s) (little endian).

12.1.4 FLAG field

The FLAG field contains a flag indicating the end of a packet. There are two types of flags.

Table 12-2. Flag Types

Flag	Meaning
0x03	Single packet or the last of a number of divided packets.
0x02	First packet of a number of divided packets or one of the middle packets.

12.2 Packet Division

The total length of the data in an unpacketed state in a server command or server command response sometimes exceeds 255 bytes. In this case, the host CPU cannot fit all the data in a single packet, and therefore divides the data into multiple packets.

The data in local or extension commands cannot be divided and sent in multiple packets; the command must fit completely into a single packet.

Figure 12-2. Example of Single Packet

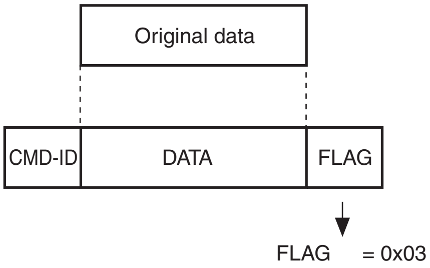
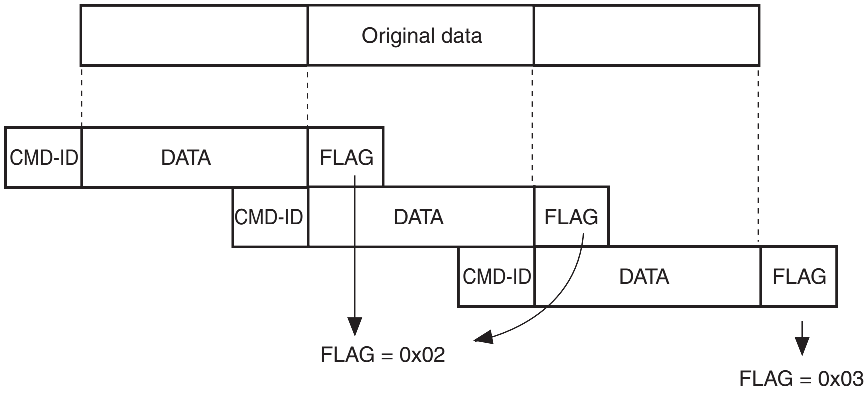

Original data
CMD-ID
DATA
FLAG
FLAG = 0x03

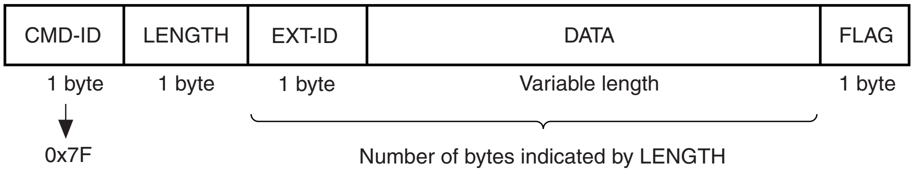
Figure 12-3. Example of Data Divided into Multiple Packets


Original data
CMD-ID
DATA
FLAG
CMD-ID
DATA
FLAG
CMD-ID
DATA
FLAG
FLAG = 0x02
FLAG = 0x03

12.3 Extension Command Format

Extension commands (CMD-ID = 0x7F) contain an additional EXT-ID field, which identifies the command.

Figure 12-4. Extension Command Format

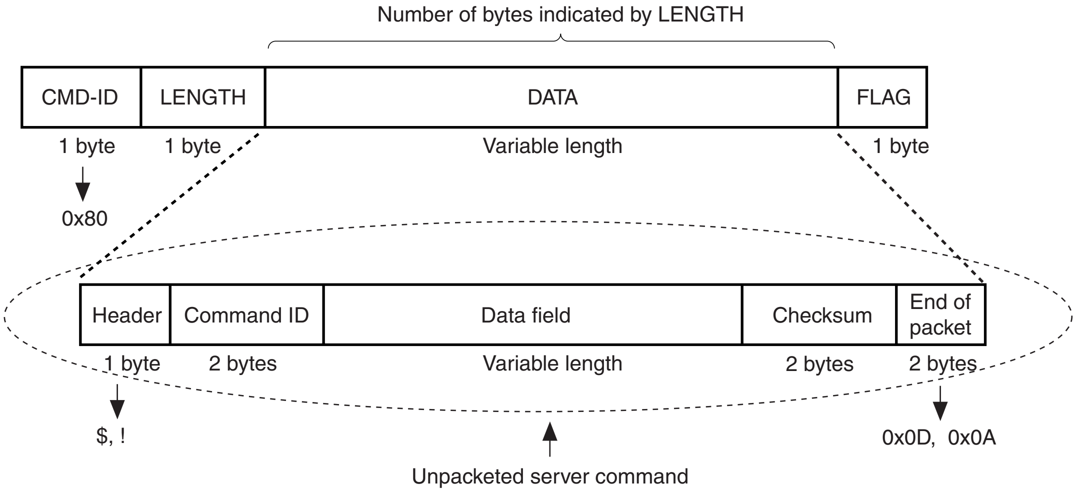

CMD-ID
LENGTH
EXT-ID
DATA
FLAG
1 byte
1 byte
1 byte
Variable length
1 byte
0x7F
Number of bytes indicated by LENGTH

12.4 Server Command Format

Server commands are stored in the DATA field of a packet in the format shown below.

Server commands are not in packet form when they are sent and received by the LS, so they must be "packeted" by the host CPU.

Figure 12-5. Server Command Format


Number of bytes indicated by LENGTH
CMD-ID
LENGTH
DATA
FLAG
1 byte
1 byte
Variable length
1 byte
0x80
Header
Command ID
Data field
Checksum
End of packet
1 byte
2 bytes
Variable length
2 bytes
2 bytes
$, !
0x0D, 0x0A
Unpacketed server command

12.4.1 Header

Commands from and responses to the server have an ASCII character as the header (indicating the start of the command or response).

Table 12-3. Header

Type	ASCII Character
Command	$ (0x24)
Response	! (0x21)

12.4.2 Command ID

The ID that identifies the type of command or response has a 2-byte ASCII character configuration (2 characters).

Table 12-4. Command ID

Type	ASCII Character
Command	Combination of 2 uppercase characters, A to Z
Response	Combination of 2 lowercase characters, a to z

12.4.3 Data field

The data field stores the data and status of the command or response.

12.4.4 Checksum

The lower 16 bits of the sum of all the bytes from the 1-byte CMD-ID to the byte preceding the checksum are used for the checksum.

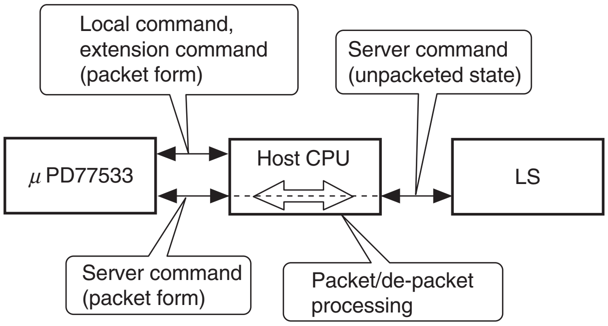
12.4.5 End

All commands and responses end with <CR><LF>.
<CR> = 0x0D, <LF> = 0x0A

12.5 Packeting and De-packeting Server Commands

When issued by the LS, server commands are not in the packet format required for interfacing between the μPD77533 and host CPU, and must therefore be "packeted" by the host CPU. The host CPU must similarly "de-packet" packeted data from the μPD77533 before sending it to the LS.

Figure 12-6. Packeting and De-packeting Server Commands


Local command,
extension command
(packet form)
Server command
(unpacketed state)
μ PD77533
Host CPU
LS
Server command
(packet form)
Packet/de-packet
processing

13. ACK, NACK, ERROR STATUS

In communication between the μPD77533 and the host CPU, the transmission and reception of server and extension commands and responses is confirmed using ACK and NACK.

Table 13-1. ACK and NACK Applications

Type	Description
ACK	Indicates that the command was received.
NACK (error status)	Indicates that the command cannot be received for some reason.

13.1 Format of ACK, NACK, Error Status

13.1.1 ACK

There are two types of ACK formats: ACK format 1 for server commands and ACK format 2 for extension commands.

The μPD77533 handles both these ACK types as “ACK”, regardless of which type is received. The host CPU can use either of these ACK types, irrespective of the circumstances.

The μPD77533 returns ACK to the host CPU in ACK format 1 in response to server commands and ACK format 2 in response to extension commands.

Table 13-2. ACK Format 1 (ACK Returned for Server Commands)

Field	Value	Number of Bytes	Description
CMD-ID	0x80	1	
LENGTH	0x00	1	
FLAG	0x03	1	

Table 13-3. ACK Format 2 (ACK Returned for Extension Commands)

Field	Value	Number of Bytes	Description
CMD-ID	0x3F	1	
LENGTH	0x01	1	
DATA	0x00	1	ACK indicated by 0x00
FLAG	0x03	1	

13.1.2 NACK, error status

Table 13-4. NACK, Error Status Format

Field	Value	Number of Bytes	Description
CMD-ID	0x3F	1	
LENGTH	0x01	1	
DATA	ErrStat	1	0x00: ACK (OK) 0x01: Format error 0x02: Timeout error Undefined CMD-ID error Else: Undefined
FLAG	0x03	1	

13.2 Flow of Commands, Reponses, ACK, and NACK

13.2.1 ACK, NACK in case of local command

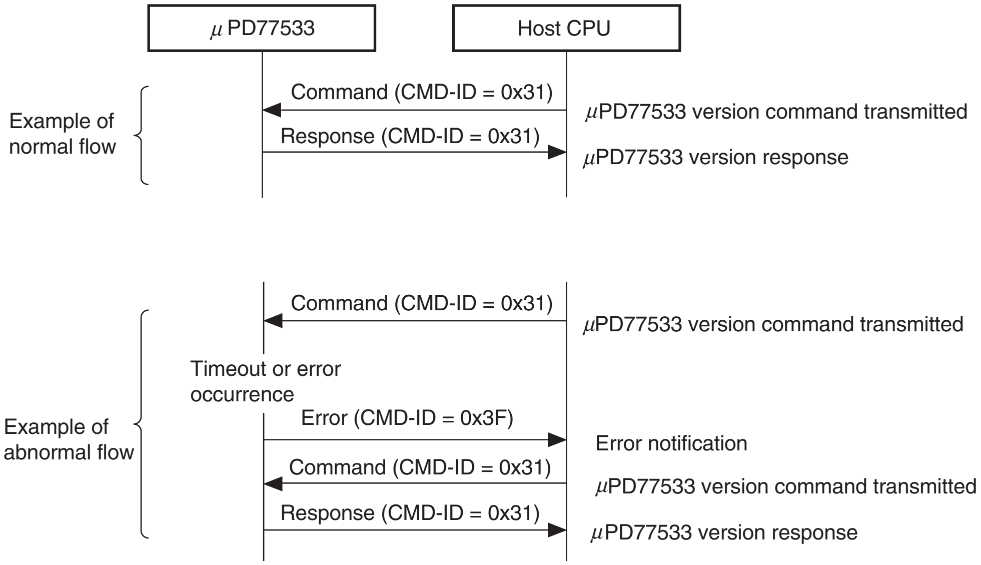
When the command is a local command, ACK and NACK are not used to confirm packet transmission/reception.

A response to the command (command execution result) is sent from the μPD77533 to the host CPU instead of ACK.

There is no equivalent to ACK for a command to which no response is returned (baud rate change command).

Even if NACK is sent from the host CPU to the μPD77533, the latter does not re-send the packet.

Figure 13-1. Local Command Flow


μ PD77533
Host CPU
Example of normal flow
Command (CMD-ID = 0x31)
μPD77533 version command transmitted
Response (CMD-ID = 0x31)
μPD77533 version response
Example of abnormal flow
Command (CMD-ID = 0x31)
μPD77533 version command transmitted
Timeout or error occurrence
Error (CMD-ID = 0x3F)
Error notification
Command (CMD-ID = 0x31)
μPD77533 version command transmitted
Response (CMD-ID = 0x31)
μPD77533 version response

13.2.2 ACK, NACK in case of extension command

Some extension commands use ACK and NACK, and some do not.

If NACK is sent from the host CPU to the μPD77533, the same packet is re-sent. The μPD77533 can send the next packet after receiving ACK from the host CPU.

When a flow point is output from the μPD77533 to the host CPU, the host CPU must send either ACK or NACK. The μPD77533 stops outputting packets until this ACK or NACK is received.

Table 13-5. ACK, NACK for Extension Commands

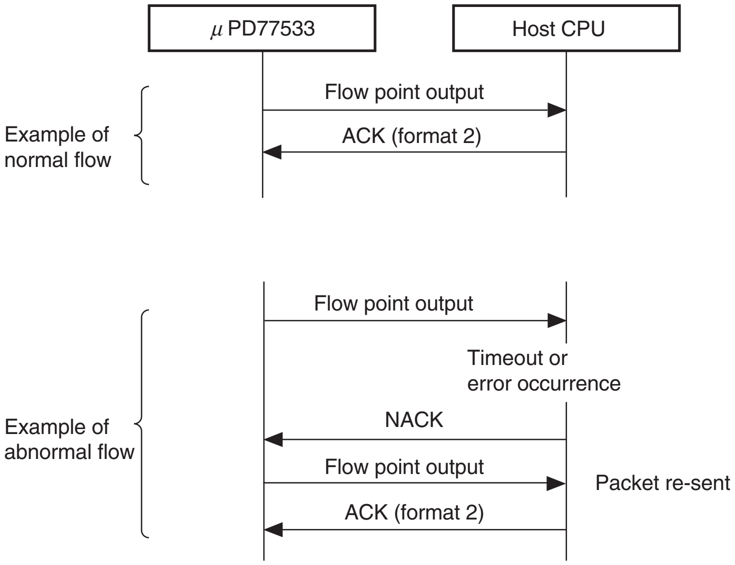
Command type	EXT-ID	ACK, NACK Control
Flow point setting command	0x00	Not controlled
Flow point output	Other than 0x00	Controlled

(1) Flow point setting command

See **Figure 13-1 Local Command Flow**.

(2) Flow point output

Figure 13-2. Flow Point Flow


μ PD77533
Host CPU
Example of normal flow
Flow point output
ACK (format 2)
Example of abnormal flow
Flow point output
Timeout or error occurrence
NACK
Flow point output
Packet re-sent
ACK (format 2)

13.2.3 ACK, NACK in case of server command

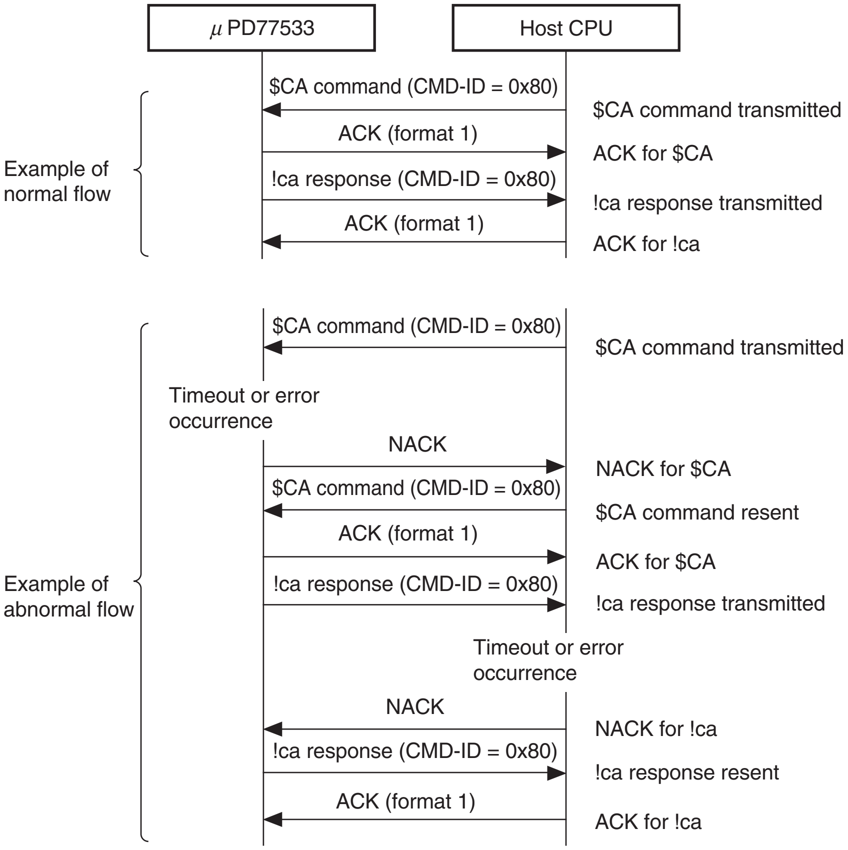
When the command is a server command, ACK and NACK are used to confirm transmission and reception of each packet.

If NACK is sent from the host CPU to the μPD77533, the same packet is re-sent.

The μPD77533 can send the next packet after receiving ACK from the host CPU.

When a response is output from the μPD77533 to the host CPU, the host CPU must send either ACK or NACK. The μPD77533 stops outputting packets until this ACK or NACK is received.

Figure 13-3. Server Command Flow


μ PD77533
Host CPU
Example of normal flow
$CA command (CMD-ID = 0x80)
$CA command transmitted
ACK (format 1)
ACK for $CA
!ca response (CMD-ID = 0x80)
!ca response transmitted
ACK (format 1)
ACK for !ca
Example of abnormal flow
$CA command (CMD-ID = 0x80)
$CA command transmitted
Timeout or error occurrence
NACK
NACK for $CA
$CA command (CMD-ID = 0x80)
$CA command resent
ACK (format 1)
ACK for $CA
!ca response (CMD-ID = 0x80)
!ca response transmitted
Timeout or error occurrence
NACK
NACK for !ca
!ca response (CMD-ID = 0x80)
!ca response resent
ACK (format 1)
ACK for !ca

14. HARDWARE FLOW CONTROL

In communication between the μPD77533 and the host CPU, hardware flow control is carried out using CTS-RTS to prevent the loss of data due to a buffer overflow.

14.1 When Hardware Flow Control Is Not Performed

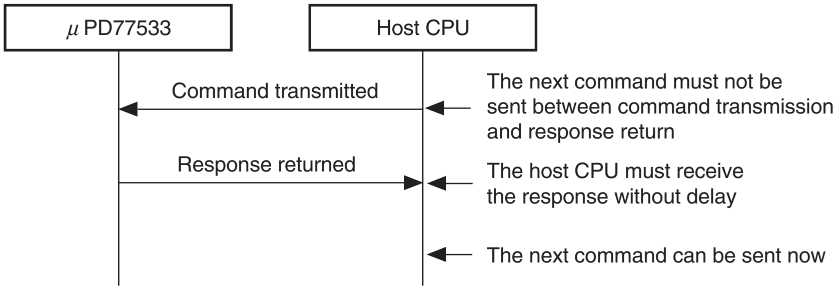
When hardware flow control is not performed, data may be lost due to a buffer overflow. To prevent this, the host CPU must execute the following processing.

14.1.1 Local commands

In the case of a local command, do not send the next command until a command response is returned. The host CPU only has a one-packet buffer, and therefore must receive responses from the μPD77533 without delay.

Do not send local commands to which no response is returned while the μPD77533 is performing signal processing. This is because a little time is required to execute a command received while the μPD77533 is performing signal processing, making it impossible to know when command execution is complete.

Figure 14-1. Local Command Flow (with No Hardware Flow Control)

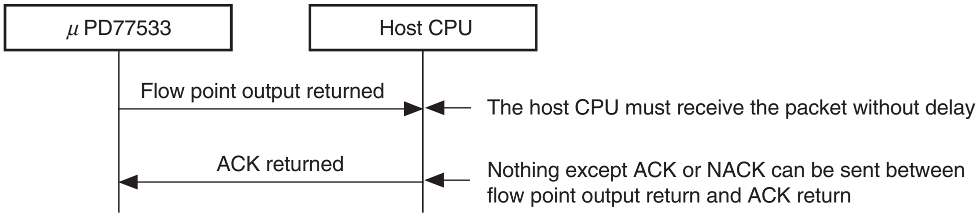

μ PD77533
Host CPU
Command transmitted
Response returned
The next command must not be sent between command transmission and response return
The host CPU must receive the response without delay
The next command can be sent now

14.1.2 Extension commands

Perform processing for the flow point setting command in the same way as for 14.1.1 Local commands.

For flow point output, because packet transmission/reception can be managed via ACK and NACK, there is no problem as long as the packet from the μPD77533 is received without delay in the one-packet buffer provided on the host CPU side.

Figure 14-2. Flow Point Output Flow (with No Hardware Flow Control)


μ PD77533
Host CPU
Flow point output returned
ACK returned
The host CPU must receive the packet without delay
Nothing except ACK or NACK can be sent between flow point output return and ACK return

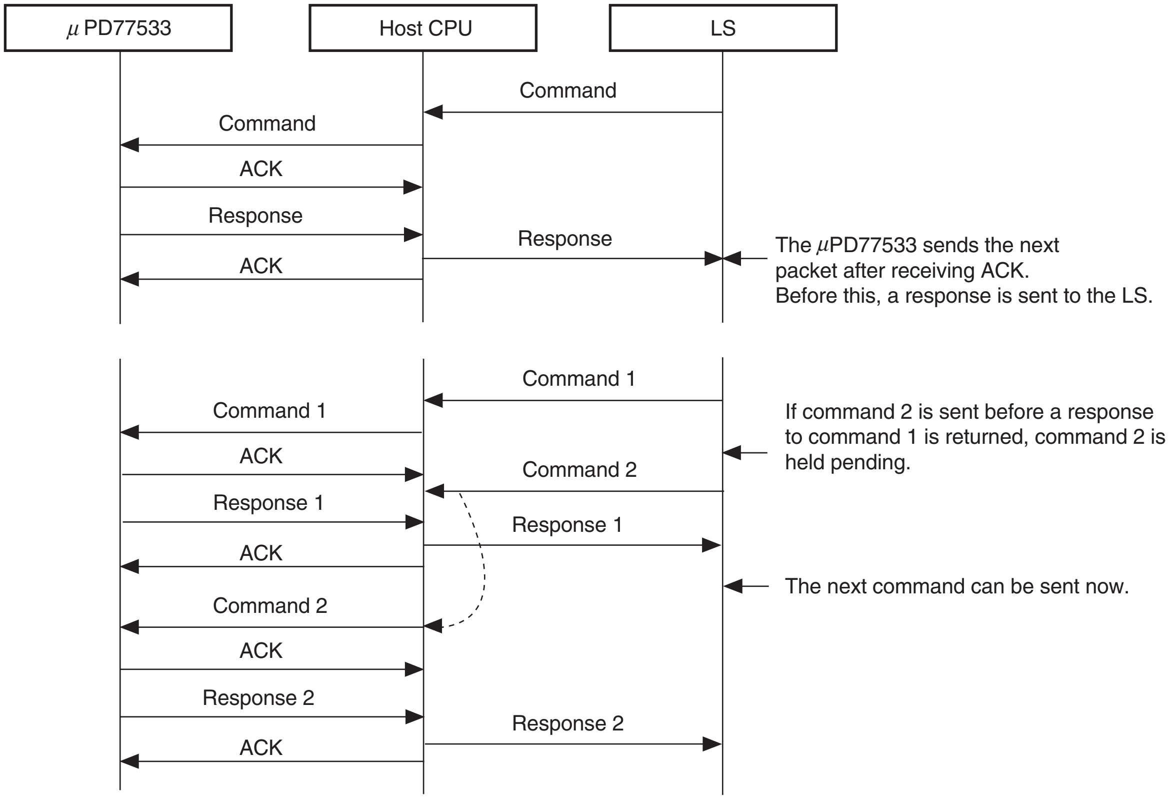
14.1.3 Server commands

When sending server commands to the μPD77533, do not send the next command until a command response is returned.

Multiple commands may be sent consecutively from the LS, but in this case, pool the commands from the LS on the host CPU side and make sure that only one command is sent to the μPD77533 at a time.

For command response, because packet transmission/reception can be managed via ACK and NACK, there is no problem as long as the packet from the μPD77533 is received without delay in the one-packet buffer provided on the host CPU side.

Figure 14-3. Server Command Flow (with No Hardware Flow Control)


μ PD77533
Host CPU
LS
Command
Command
ACK
Response
Response
ACK
The μPD77533 sends the next packet after receiving ACK.
Before this, a response is sent to the LS.
Command 1
Command 1
If command 2 is sent before a response to command 1 is returned, command 2 is held pending.
ACK
Command 2
Response 1
Response 1
ACK
The next command can be sent now.
Command 2
ACK
Response 2
Response 2
ACK

15. SERVER COMMANDS

The μPD77533 supports the following server commands.

Table 15-1. List of Server Commands

Command Name	Command ID	Function
Module ID/Version	CA, ca	Obtains the unit ID and software version information
Module Processing Status Request	BB, bb	References the status of the LSI and positioning
Module Reset	IA, ia	Resets the GPS module
Send Processed Result	hd	Sends notification of the positioning result
Multi Fix	JM, jm	Specifies positioning
Ping for Coarse Time Synchronization	PJ, pj	Course time synchronization. First command for positioning session.
Set Adaptive Parameters	AC, ac	Sets various parameters related to positioning
Set SnapShot Duration	DA, da	Command used to verify the functions of the μPD77533^Note 1^ (SnapShot time setting)
Take SnapShot	EA, ea	Command used to verify the functions of the μPD77533^Note 1^ (SnapShot execution)
Send SnapShot Data	FA, FC, fa, fb, fc	Command used to verify the functions of the μPD77533^Note 1^ (SnapShot data (1 ms Frame) acquisition)
Set Baud Rate	BD, bd	Sets connection speed between host CPU and LS^Note 2^
Request for Service	CB, cb	Sends notification of start of positioning at calling receiver

Notes **1.** These commands are not required for normal positioning operations.
2. This command is processed by the host CPU; command processing by the μPD77533 is not required.

15.1 Handling of Server Commands by Host CPU

With a few exceptions, the host CPU simply acts as the mediator when server commands are communicated between the μPD77533 and the LS. Specifically, the host CPU performs the following processing.

- Packets server commands sent from the LS and transfers them to the μPD77533.
- De-packets responses sent from the μPD77533 and transfers them to the LS.
- Changes the baud rate upon recognition of the baud rate change server command and sends a response.
- Performs ACK and NACK processing in communication between the μPD77533 and the host CPU.

In order to perform the above processing, the host CPU must analyze server commands and responses.

15.1.1 Server command analysis

The host CPU analyzes server commands sent by the LS as follows.

(1) Checks whether the command header is the “$” character.
(2) Checks whether the command ID is valid. Starts the checksum calculation.
(3) Calculates the number of bytes in the data field according to “Determination of Number of Bytes in Data Field” in Table 15-2.
(4) Reads data of the number of bytes calculated in (3) and calculates the checksum.
(5) Checks whether the checksum matches the calculated value.
(6) Checks whether the end-of-packet is <CR><LF>.

If all of (1) to (6) are OK, the command is recognized as one command.

If the command is other than the baud rate change command, the command is packeted and transferred to the μPD77533. If an error occurs at any stage, 1 byte is discarded and the operation starts again from (1).

Table 15-2. Determination of Number of Bytes in Data Field

Command ID	Determination of Number of Bytes in Data Field
CA	Fixed to 0 bytes
BB	Fixed to 1 byte
IA	Fixed to 1 byte
JM	Value stored in first 2 bytes of data field minus 7 bytes
PJ	Fixed to 1 byte
AC	Value stored in first 2 bytes of data field minus 7 bytes
DA	Fixed to 5 bytes
EA	Fixed to 0 bytes
FA	Fixed to 4 bytes
FC	Fixed to 0 bytes
BD	Fixed to 1 byte
CB	Fixed to 1 byte
ca	Fixed to 9 bytes
bb	Fixed to 1 byte
ia	Fixed to 1 byte
hd	Value stored in first 2 bytes of data field minus 7 bytes
jm	Fixed to 1 byte
pj	Fixed to 4 bytes
ac	Value stored in first 2 bytes of data field minus 7 bytes
da	Fixed to 1 byte
ea	Fixed to 1 byte
fa	Fixed to 3 bytes
fb	If previous !fa result code is 0: Fixed to 2050 bytes If previous !fa result code is 1: Fixed to 1026 bytes
fc	Fixed to 1 byte
bd	Fixed to 2 bytes
cb	Fixed to 10 bytes

15.1.2 Processing of baud rate change command (“Set Baud Rate”)

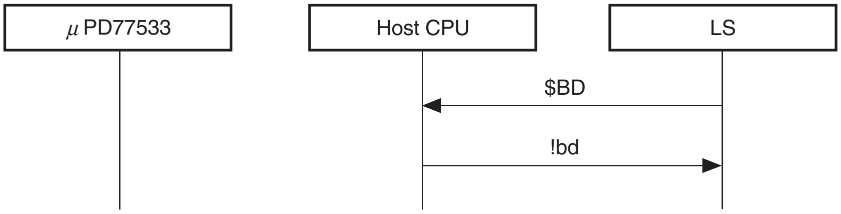
The baud rate change command is processed as follows.

(1) A response to the command is created and sent to the LS.
(2) The system waits until response transmission is complete.
(3) The baud rate is changed in accordance with the command.

15.2 Set Baud Rate

This command is used to change the baud rate between the host CPU and the LS.
This command must be processed by the host CPU; it is not processed by the μPD77533.

Figure 15-1. Set Baud Rate Flow



15.2.1 $BD

Table 15-3. $BD Format (in Unpacketed State)

Section		Number of Bits	Format	Description
Header		8	–	$
Command ID		16	–	BD
Data	Baud rate	8	unsigned	See **Table 15-5 List of Baud Rates**
Checksum		16	–	Checksum
End-of-packet		16	–	<0x0D><0x0A>

15.2.2 !bd

Table 15-4. !bd Format (in Unpacketed State)

Section		Number of Bits	Format	Description
Header		8	–	!
Command ID		16	–	bd
Data	Result code	8	unsigned	0 = OK Else = Error
	Baud rate	8	unsigned	See **Table 15-5 List of Baud Rates**
Checksum		16	–	Checksum
End-of-packet		16	–	<0x0D><0x0A>

★ 15.2.3 List of baud rates

Table 15-5. List of Baud Rates

No.	Baud Rate
0	2400 bps
1	4800 bps
2	9600 bps
3	19200 bps
4	38400 bps
5	57600 bps
Else	115200 bps

Remark The baud rate between the μPD77533 and the host CPU is changed according to **16.1 Baud Rate Change** in **16 LOCAL COMMANDS**.

16. LOCAL COMMANDS

The μPD77533 program supports the following local commands.

Table 16-1. List of Local Commands

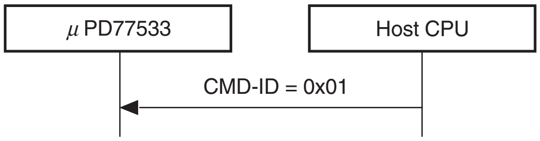
Command	CMD-ID	Function
Baud Rate Change	0x01	Sets the baud rate between the μPD77533 and the host CPU
FCC Reference Frequency	0x02	Sets, references the FCC reference frequency
Power Control	0x03	Sets a power save mode
Reset	0x04	Resets the μPD77533
RF Power Control	0x05	Sets the RF power supply
Frequency Calibration Control	0x06	Sets, references FCC control
TXACTV Control	0x07	Sets, references TXACVT control
Sleep Timer	0x12	Sets, references the sleep timer
Character Timeout	0x13	Sets, references the character timeout
!cb Request	0x20	Requests issuance of the !cb response
!ia Request	0x21	Requests issuance of the !ia response
SnapShot Memory Check	0x2D	Checks the SnapShot memory
Status Request	0x30	Checks the serial communication operation
D77533 Software Version	0x31	References the version of the μPD77533
Processing Status Request	0x33	References the status of signal processing
Rest Processing	0x34	References the progress of positioning
Error Status	0x3F	Sending an error notification
CA Parameter Set	0x40	Sets the !ca response parameter
CB Parameter Set	0x41	Sets the !cb response parameter
Doppler Search Range	0x57	Expands the Doppler search range

16.1 Baud Rate Change

[Function]

Changes the baud rate between the μPD77533 and the host CPU.

Figure 16-1. Baud Rate Change Flow



[Command]

Table 16-2. Format of Baud Rate Change

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x01
LENGTH		1	–	1
DATA	Baud rate	1	unsigned	0 = 2400 bps, 1 = 4800 bps, 2 = 9600 bps (Default), 3 = 19200 bps, 4 = 38400 bps, 5 = 57600 bps, Else = 115200 bps
FLAG		1	–	0x03

[Response]

The μPD77533 does not return a response.

[Remark]

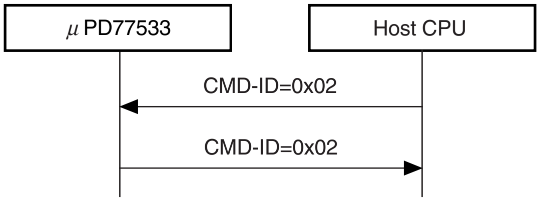
If the μPD77533 can execute this command correctly, no response is returned.

16.2 FCC Reference Frequency

[Function]

Sets and references the FCC reference frequency used for frequency adjustment.

Figure 16-2. FCC Reference Frequency Flow


μ PD77533
Host CPU
CMD-ID=0x02
CMD-ID=0x02

[Command]

Table 16-3. Format of FCC Reference Frequency

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x02
LENGTH		1	–	4 = Sets reference frequency 0 = References current reference frequency
DATA	Reference frequency (when LENGTH = 4)	4	unsigned	Reference frequency The value to be set must be the frequency multiplied by 1.024. Default = 0xE10000 (14.4 MHz x 1.024)
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-3.

[Related commands]

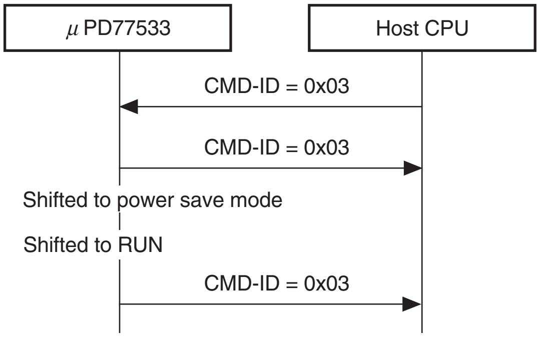
16.6 Frequency Calibration Control

16.3 Power Control

[Function]

Shifts the μPD77533 to a power save mode.

Figure 16-3. Power Control Flow


μ PD77533
Host CPU
CMD-ID = 0x03
CMD-ID = 0x03
Shifted to power save mode
Shifted to RUN
CMD-ID = 0x03

[Command]

Table 16-4. Format of Power Control

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x03
LENGTH		1	–	1
DATA	Mode	1	unsigned	0 = Shifts to RUN (default) 1 = Shifts to HALT Else = Shifts to STOP
FLAG		1	–	0x03

[Response]
The state to be shifted to is returned in the format shown in Table 16-4.

[Remark]
No response is returned when the μPD77533 is automatically shifted to a power save mode by the Sleep Timer.
DATA = 0x00 is returned as the Power Control response even when restoring the μPD77533 from a power save mode.

16.4 Reset

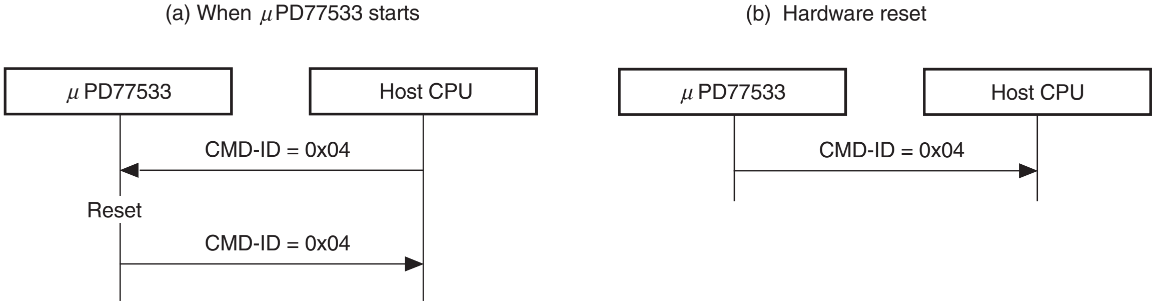
[Function]

Resets the μPD77533.

Initializes all the settings other than the baud rate.

Returns a Reset response even during a hardware reset when the μPD77533 starts.

Figure 16-4. Reset Flow


(a) When μPD77533 starts
μPD77533
Host CPU
CMD-ID = 0x04
Reset
CMD-ID = 0x04
(b) Hardware reset
μPD77533
Host CPU
CMD-ID = 0x04

[Command]

Table 16-5. Format of Reset

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x04
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

The state to be shifted to is returned in the format shown in Table 16-5.

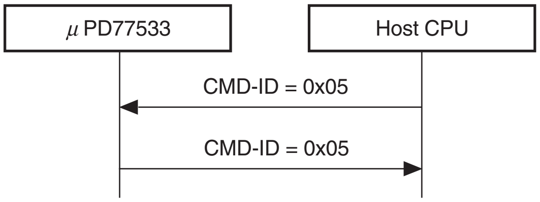
16.5 RF Power Control

[Function]

Switches the power supply to the RF block on and off.

In normal positioning, the logic switches the power supply to the RF block on and off, therefore, control by software is not required.

Figure 16-5. RF Power Control Flow



[Command]

Table 16-6. Format of RF Power Control

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x05
LENGTH		1	–	1
DATA	Power supply	1	unsigned	0 = Power supply off (default) Else = Power supply on
FLAG		1	–	0x03

[Response]

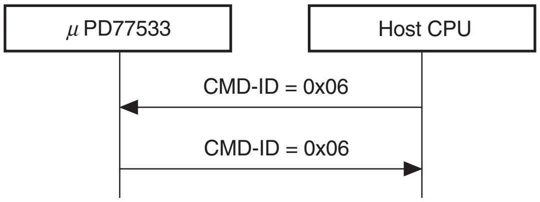
The value set is returned in the format shown in Table 16-6.

16.6 Frequency Calibration Control

[Function]

Sets and references FCC control usage.

Figure 16-6. Frequency Calibration Control Flow


μPD77533
Host CPU
CMD-ID = 0x06
CMD-ID = 0x06

[Command]

Table 16-7. Format of Frequency Calibration Control

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x06
LENGTH		1	–	1 = Sets use of FCC control 0 = References current FCC control usage
DATA	FCC control (when LENGTH = 1)	1	unsigned	0 = FCC control not used (default) Else = FCC control used
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-7.

[Related commands]

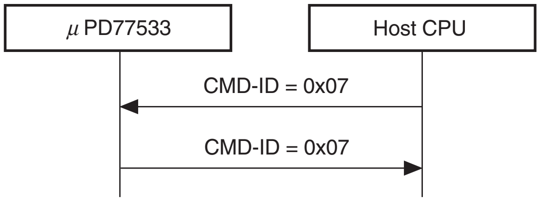
16.2 FCC Reference Frequency

16.7 TXACTV Control

[Function]

Sets and references TXACTV control usage.

Figure 16-7. TXACTV Control Flow


μPD77533
Host CPU
CMD-ID = 0x07
CMD-ID = 0x07

[Command]

Table 16-8. Format of TXACTV Control

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x07
LENGTH		1	–	1 = Sets use of TXACTV control 0 = References current TXACTV control usage
DATA	TXACTV control (when LENGTH = 1)	1	unsigned	0 = TXACTV control not used (default) Else = TXACTV control used
FLAG		1	–	0x03

[Response]

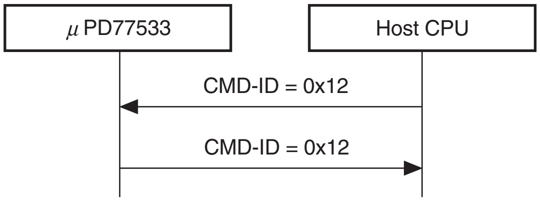
The value set or current value is returned in the format shown in Table 16-8.

16.8 Sleep Timer

[Function]

Sets and references the sleep timer.

Figure 16-8. Sleep Timer Flow



[Command]

Table 16-9. Format of Sleep Timer

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x12
LENGTH		1	–	4 = Sets sleep timer 0 = References current sleep timer setting
DATA	Mode (when LENGTH = 4)	1	unsigned	0 = Shifts to HALT Else = Shifts to STOP
	Time until sleep (when LENGTH = 4)	3	unsigned	Time until the μPD77533 is shifted to sleep mode. Unit = ms. 0 = Not shifted to sleep mode (default)
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-9.

[Remark]

The μPD77533 is shifted to sleep mode after the specified time has elapsed, without performing positioning, command transmission reception, or other such processing.

Triggers such as command reception, the WKUPB signal, a reset, and a timer interrupt are used to restore the μPD77533 from the HALT mode.

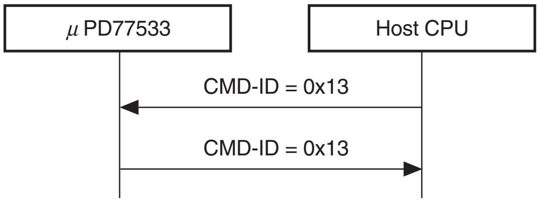
The μPD77533 is restored from STOP mode to RUN mode by the WKUPB signal, and is totally initialized and restarted by a reset.

16.9 Character Timeout

[Function]

Sets and references the character timeout time.

Figure 16-9. Character Timeout Flow



[Command]

Table 16-10. Format of Character Timeout

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x13
LENGTH		1	–	3 = Sets timeout time 0 = References current timeout time
DATA	Time until timeout (when LENGTH = 3)	3	unsigned	Time until timeout occurs. Unit = ms. 0 = Timeout not detected (default)
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-10.

[Remark]

If the specified timeout time elapses after the last data of a packet has been received by the μPD77533 before packet reception is complete, a timeout occurs.

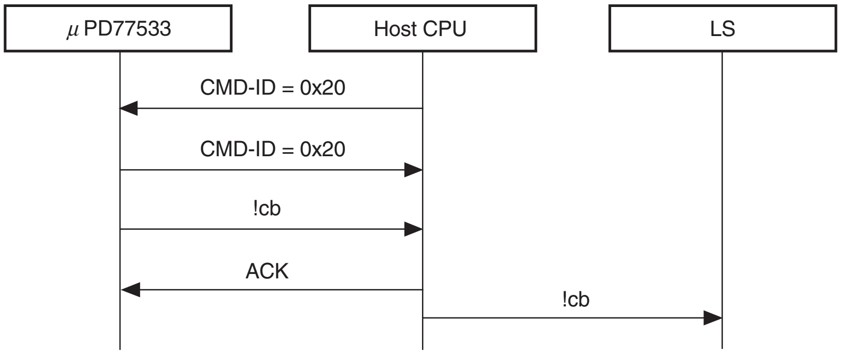
When a timeout occurs, the μPD77533 returns the timeout error status and discards the packet data received until that point.

16.10 !cb Request

[Function]

Requests the !cb response.

Figure 16-10. !cb Request Flow



[Command]

Table 16-11. Format of !cb Request

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x20
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

A response is returned in the format shown in Table 16-11.
The μPD77533 returns !cb following a response.

[Remark]

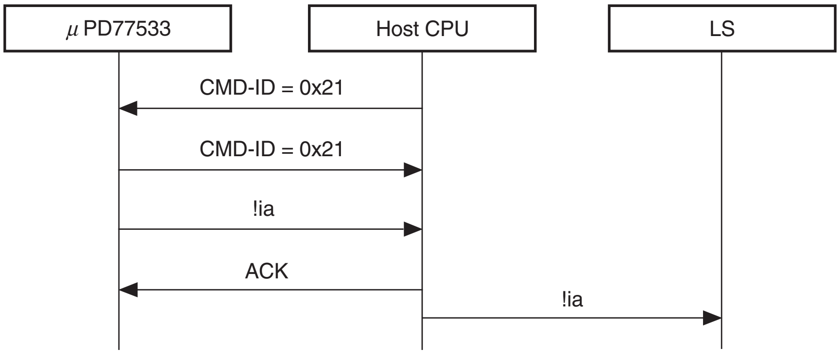
This command can be used when positioning is requested from the receiver side. Upon receiving the !cb Request command, the μPD77533 sends the !cb response to the LS. The LS starts positioning after receiving this response.
This command does not need to be used when positioning is requested from the LS side.
Note that the actual control flow when positioning is started from the receiver side depends on the system.

16.11 !ia Request

[Function]

Requests the !ia response.

Figure 16-11. !ia Request Flow


μ PD77533
Host CPU
LS
CMD-ID = 0x21
CMD-ID = 0x21
!ia
ACK
!ia

[Command]

Table 16-12. Format of !ia Request

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x21
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

A response is returned in the format shown in Table 16-12.

After this response, the !ia response is returned. The !ia result code is 0x09.

[Remark]

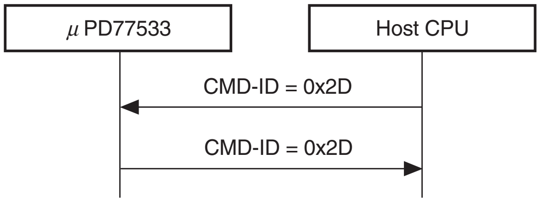
How to use this command depends on the system.

16.12 SnapShot Memory Check

[Function]

Performs a SnapShot memory check.

Figure 16-12. SnapShot Memory Check Flow


μ PD77533
Host CPU
CMD-ID = 0x2D
CMD-ID = 0x2D

[Command]

Table 16-13. Format of SnapShot Memory Check Command

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x2D
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

If there is no SnapShot memory error, a response is returned in the format shown in Table 16-13. If there is an error, the error is returned in the following format.

Table 16-14. Format of SnapShot Memory Check Response (When Error Occurred)

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x2D
LENGTH		1	–	7
DATA	Address	3	unsigned	SnapShot memory address where error occurred (0x0000 to 0xFFFFF)
	Written value	2	–	Value written to SnapShot memory
	Read value	2	–	Value read from SnapShot memory
FLAG		1	–	0x03

[Remark]

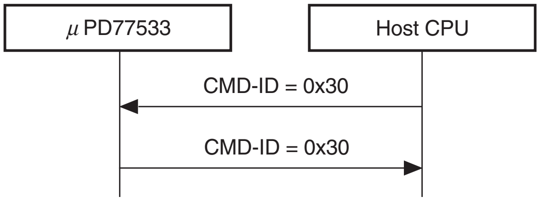
This command is used to check whether the values 0x0000, 0xAAAA, 0xFFFF, and 0x5555 written to all SnapShot memory areas can be read correctly.

16.13 Status Request

[Function]

Checks the operation of the μPD77533.

Figure 16-13. Status Request Flow



[Command]

Table 16-15. Format of Status Request Command

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x30
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

A response is returned in the format shown in Table 16-15.

[Remark]

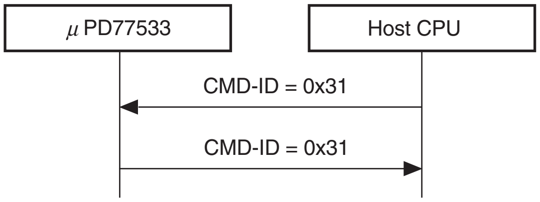
The μPD77533 simply returns a response to this command; no other processing is performed.

16.14 D77533 Software Version

[Function]

References the version of the μPD77533 software.

Figure 16-14. D77533 Software Version Flow



[Command]

Table 16-16. Format of D77533 Software Version Command

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x31
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

Table 16-17. Format of D77533 Software Version Response

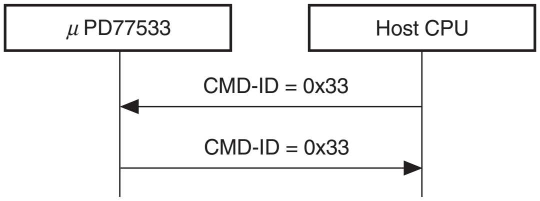
Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x31
LENGTH		1	–	2
DATA	Version	1	–	Higher 4 bits: Digit in the “10” column Lower 4 bits: Digit in the “1” column
		1	–	Higher 4 bits: Digit at the 1st decimal place Lower 4 bits: Digit at the 2nd decimal place
FLAG		1	–	0x03

16.15 Processing Status Request

[Function]

References the status of μPD77533 signal processing.

Figure 16-15. Processing Status Request Flow



[Command]

Table 16-18. Format of Processing Status Request Command

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x33
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

Table 16-19. Format of Processing Status Request Response

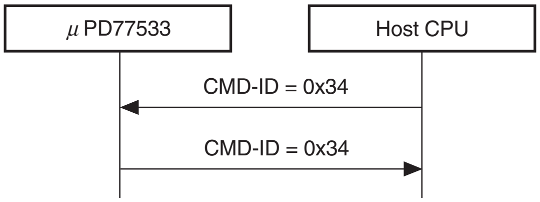
Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x33
LENGTH		1	–	1
DATA	Status	1	unsigned	0x00 = SnapShot under execution 0x10 = Signals being processed Else = Idle
FLAG		1	–	0x03

16.16 Rest Processing

[Function]

References the number of satellites remaining to be processed during positioning and the number of times positioning has been performed.

Figure 16-16. Rest Processing Flow



[Command]

Table 16-20. Format of Rest Processing Command

Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x34
LENGTH	1	–	0
FLAG	1	–	0x03

[Response]

Table 16-21. Format of Rest Processing Response

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x34
LENGTH		1	–	4
DATA	Number of processed satellites	1	unsigned	1 to 32
	Total number of satellites that should be processed	1	unsigned	1 to 32
	Current number of times positioning performed	1	unsigned	From 0
	Total number of times positioning to be performed	1	unsigned	0 = No control From 1 = Number of times
FLAG		1	–	0x03

16.17 Error Status

[Function]

Sends notification of an error.

[Command]

Table 16-22. Format of Error Status

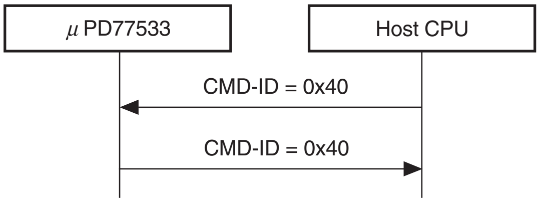
Field	Number of Bytes	Form	Description
CMD-ID	1	–	0x3F
LENGTH	1	–	1
DATA	1	unsigned	0 = OK (ACK) 1 = Format error 2 = Timeout error 4 = Undefined CMD-ID error Else = Undefined
FLAG	1	–	0x03

16.18 CA Parameter Set

[Function]

Sets and references the !ca parameter.

Figure 16-17. CA Parameter Set Flow



[Command]

Table 16-23. Format of CA Parameter Set

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x40
LENGTH		1	–	8 = Sets !ca parameter 0 = References current !ca parameter setting
DATA	Unit ID (when LENGTH = 8)	1	–	Unit ID (7:0)
		1	–	Unit ID (15:8)
		1	–	Unit ID (23:16)
		1	–	Unit ID (31:24)
	Version (when LENGTH = 8)	1	–	Version (7:0)
		1	–	Version (15:8)
		1	–	Version (23:16)
		1	–	Version (31:24)
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-23.

[Remark]

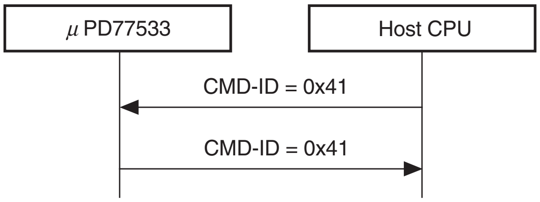
The unit ID and version values depend on the system (customer default).

16.19 CB Parameter Set

[Function]

Sets and references the !cb parameter.

Figure 16-18. CB Parameter Set Flow



[Command]

Table 16-24. Format of CB Parameter Set

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x41
LENGTH		1	–	10 = Sets !cb parameter 0 = References current !cb parameter setting
DATA	Type (when LENGTH = 10)	1	–	Type (7:0)
		1	–	Type (15:8)
	Unit ID (when LENGTH = 10)	1	–	UnitID (7:0)
		1	–	UnitID (15:8)
		1	–	UnitID (23:16)
		1	–	UnitID (31:24)
	Version (when LENGTH = 10)	1	–	Version (7:0)
		1	–	Version (15:8)
		1	–	Version (23:16)
		1	–	Version (31:24)
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-24.

[Remark]

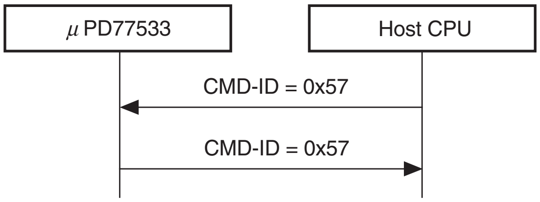
The type, unit ID, and version values depend on the system (customer default).

16.20 Doppler Search Range

[Function]

Sets and references the Doppler search range parameter.

Figure 16-19. Doppler Search Range Flow



[Command]

Table 16-25. Format of Doppler Search Range

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x57
LENGTH		1	–	2 = Sets parameter 0 = References current parameter setting
DATA	Doppler Search Range	2	–	Default Doppler search range Initial value: 0x0168
FLAG		1	–	0x03

[Response]

The value set or current value is returned in the format shown in Table 16-25.

[Remark]

When using a local frequency (GPCLK input clock) with an accuracy of 0.1 ppm, the Doppler search range parameter value is the initial value.

Changing the Doppler search range parameter to a value other than the initial value is explained in 6.1.3 Doppler Search Range.

17. EXTENSION COMMANDS

The μPD77533 supports the following extension commands.

★ **Table 17-1. List of Extension Commands**

Command	EXT-ID	Function
Flow Point Control	0x00	Sets and references flow points
Processing End	0x06	Outputs the “end of all positioning” flow point

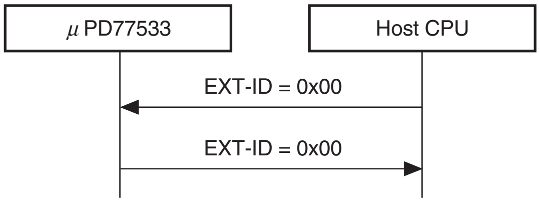
17.1 Flow Point Control

[Function]

Sets whether a flow point is output or not.

ACK is not waited for after this command is executed.

Figure 17-1. Flow Point Control Flow



[Command]

★ **Table 17-2. Format of Flow Point Control Command**

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x7F
LENGTH		1	–	3
DATA	EXT-ID	1	–	0x00
	Flow point	1	–	0x00: Does not output Processing End (default) 0x10: Outputs Processing End
		1	–	0x00
FLAG		1	–	0x03

[Response]

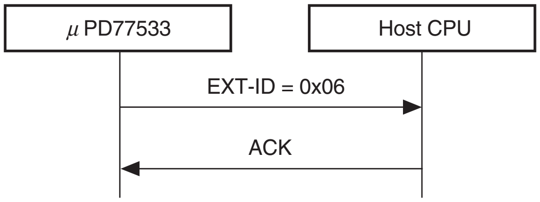
The value set is returned in the format shown in Table 17-2.

17.2 Processing End

[Function]

Sends notification of the end of all positioning operations.

Figure 17-2. Processing End Flow



[Command]

There are no commands for flow point output.

[Response]

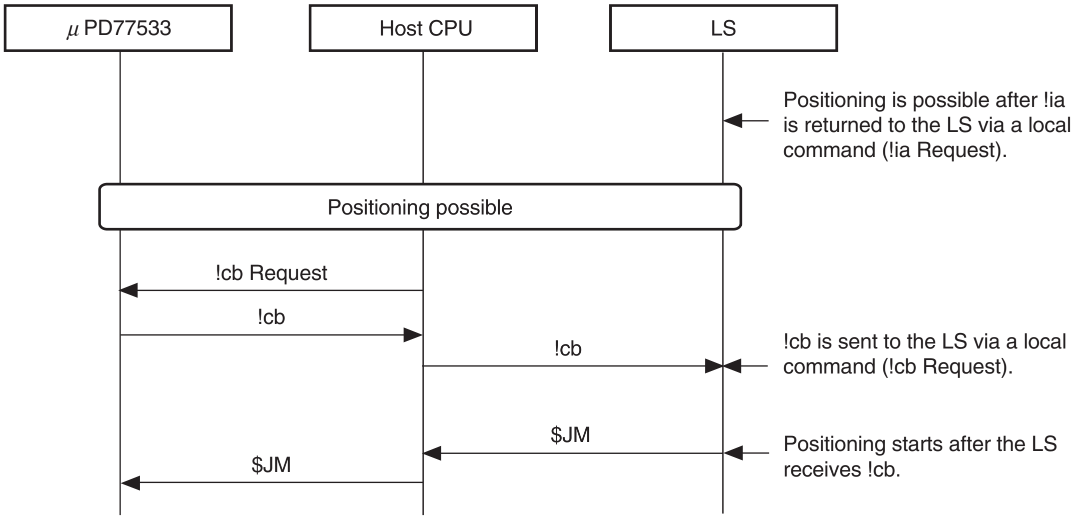
Table 17-3. Format of Processing End Command

Field		Number of Bytes	Form	Description
CMD-ID		1	–	0x7F
LENGTH		1	–	1
DATA	EXT-ID	1	–	0x06
FLAG		1	–	0x03

18. HOST CPU PROCESSING WHEN POSITIONING STARTS FROM RECEIVER SIDE

In system in which the host CPU and LS are directly connedcted, the host CPU must perform processing using the following procedure when positioning starts after a position trigger is executed from the GPS receiver.

Figure 18-1. Host CPU Processing Procedure When Positioning Starts from Receiver


μ PD77533
Host CPU
LS
Positioning is possible after !ia is returned to the LS via a local command (!ia Request).
Positioning possible
!cb Request
!cb
!cb
!cb is sent to the LS via a local command (!cb Request).
$JM
$JM
Positioning starts after the LS receives !cb.

Remark See **3.3 Processing Required at Startup** for details of the processing performed by the host CPU when the GPS receiver is activated.

19. ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings

Parameter		Symbol	Conditions	Rating	Unit
Supply voltage	Internal supply voltage	IV_{DD}	For core	–0.5 to +2.6	V
		AV_{DD}	For PLL analog	–0.5 to +2.6	V
	External supply voltage	EV_{DD}	For I/O pins	–0.5 to +4.6	V
Input/output voltage		V_I /V_O		–0.5 to +4.1, $V_I < V_{DD} +0.5$	V
Operating ambient temperature		T_A		–20 to +85	°C
Storage temperature		T_{stg}		–65 to +150	°C

Caution Product quality may suffer if the absolute maximum rating is exceeded even momentarily for any parameter. That is, the absolute maximum ratings are rated values at which the product is on the verge of suffering physical damage, and therefore the product must be used under conditions that ensure that the absolute maximum ratings are not exceeded.

Recommended Operating Conditions

Parameter		Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Operating voltage	Internal voltage	IV_{DD}	For core	1.425	1.5	1.65	V
		AV_{DD}	For PLL analog	1.425	1.5	1.65	V
	External voltage	EV_{DD}	For I/O pins	2.7	–	3.6	V
Input voltage		V_I		0	–	EV_{DD}	V

Capacitance

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Input capacitance	C_{IN}	f = 1 MHz, Unmeasured pins returned to 0 V	–	–	10	pF
Output capacitance	C_{OUT}		–	–	10	pF
I/O capacitance	C_I /C_O		–	–	10	pF

DC Characteristics (T_A = –20 to +85°C, IV_{DD}, AV_{DD} = 1.425 to 1.65 V, EV_{DD} = 2.7 to 3.6 V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Input voltage, high	V_{IHN}	Other than Schmitt input pins	0.7 EV_{DD}	–	EV_{DD}	V
	V_{IHS}	Schmitt input pins	0.8 EV_{DD}	–	EV_{DD}	V
	V_{IHC}	Clock input pins GPSCLK, REFCLK	0.6 EV_{DD}	–	EV_{DD}	V
Input voltage, low	V_{IL}	Other than clock input pins	0	–	0.2 EV_{DD}	V
	V_{ILC}	Clock input pins GPSCLK, REFCLK	0	–	0.4 EV_{DD}	V
Output voltage, high	V_{OH}	I_{OH} = –2.0 mA	0.7 EV_{DD}	–	–	V
		I_{OH} = –100 μA	0.8 EV_{DD}	–	–	
Output voltage, low	V_{OL}	I_{OL} = 2.0 mA	–	–	0.2 EV_{DD}	V
Input leakage current, high	I_{LH}	V_I = EV_{DD}	0	–	10	μA
Input leakage current, low	I_{LL}	V_I = EV_{DD}	–10	–	0	μA
Pulled-up pin current	I_{PUI}	0 V < V_I < EV_{DD}	–250	–	0	μA
Pulled-down pin current	I_{PDI}	0 V < V_I < EV_{DD}	0	–	250	μA
Normal power supply current	I_{EVDD}	Operation mode, T_{cyc} = 12.2 ns, IV_{DD}, AV_{DD} = 1.5 V, EV_{DD} = 3.0 V	–	2 Note 1	–	mA
	I_{IVDD}		–	34 Note 1	38	mA
	I_{AVDD}		–	1 Note 1	2	mA
Power supply current in HALT mode	I_{EVDDH}	HALT mode, T_{cyc} = 12.2 ns, IV_{DD}, AV_{DD} = 1.5 V, EV_{DD} = 3.0 V	–	–	–	mA
	I_{IVDDH}		–	–	5 Note 2	mA
	I_{AVDDH}		–	–		
Power supply current in STOP mode	I_{EVDDS}	STOP mode, T_A = 25°C, IV_{DD}, AV_{DD} = 1.5 V, EV_{DD} = 3.0 V	–	–	700 Note 3	μA
	I_{IVDDS}		–	–		
	I_{AVDDS}		–	–		

Notes 1. Reference value

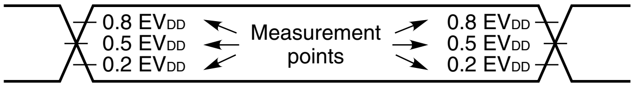
2. Sum of I_{IVDDH} and I_{AVDDH}.

3. Sum of I_{EVDDS}, I_{IVDDS}, and I_{AVDDS}.

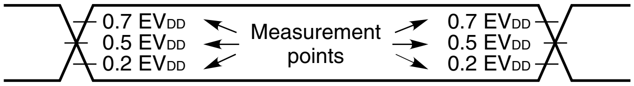
AC Characteristics

- Test waveforms

Schmitt pins



Other than Schmitt pins

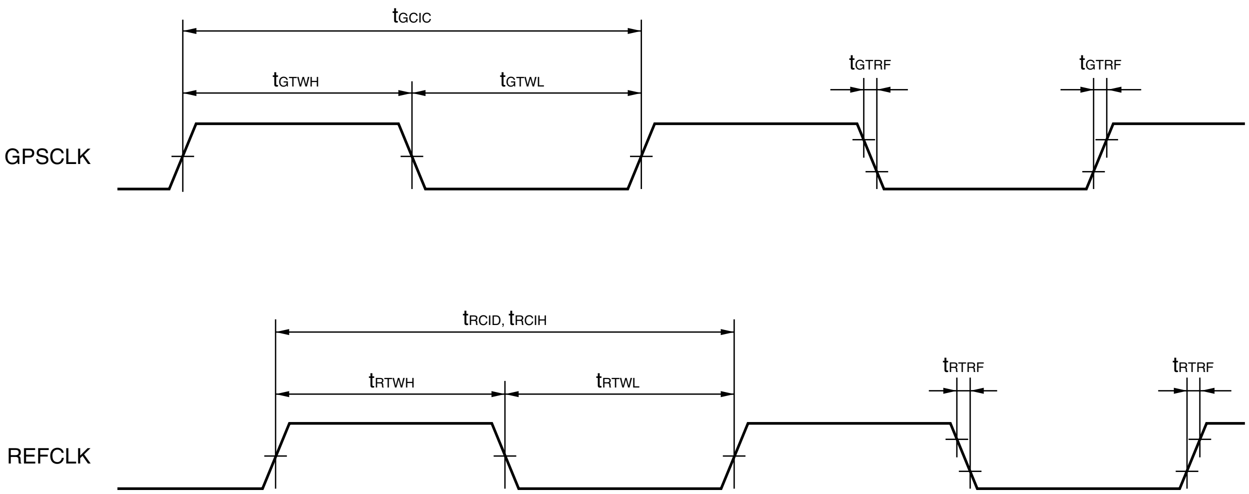


Clock Interface

- Clock

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
GPS clock input cycle	t_{GCIC}	f = 16.384 MHz	61	–	–	ns
GPS clock high-level width	t_{GTWH}		$t_{GCIC} \div 2-3$	–	–	ns
GPS clock low-level width	t_{GTWL}		$t_{GCIC} \div 2-3$	–	–	ns
GPS clock rise/fall time	t_{GTRF}		–	–	5	ns
REF clock input cycle	t_{RCID}	PDC reference	–	14.4	–	MHz
	t_{RCIH}	PHS reference	–	19.2	–	MHz
REF clock high-level width	t_{RTWH}		t_{RCID}/t_{RCIH} $\div 2-3$	–	–	ns
REF clock low-level width	t_{RTWL}		t_{RCID}/t_{RCIH} $\div 2-3$	–	–	ns
REF clock rise/fall time	t_{RTRF}		–	–	5	ns

- Clock timing



t_{GCIC}
t_{GTWH}
t_{GTWL}
t_{GTRF}
t_{GTRF}
GPSCLK
t_{RCID}, t_{RCIH}
t_{RTWH}
t_{RTWL}
t_{RTRF}
t_{RTRF}
REFCLK

System Interface

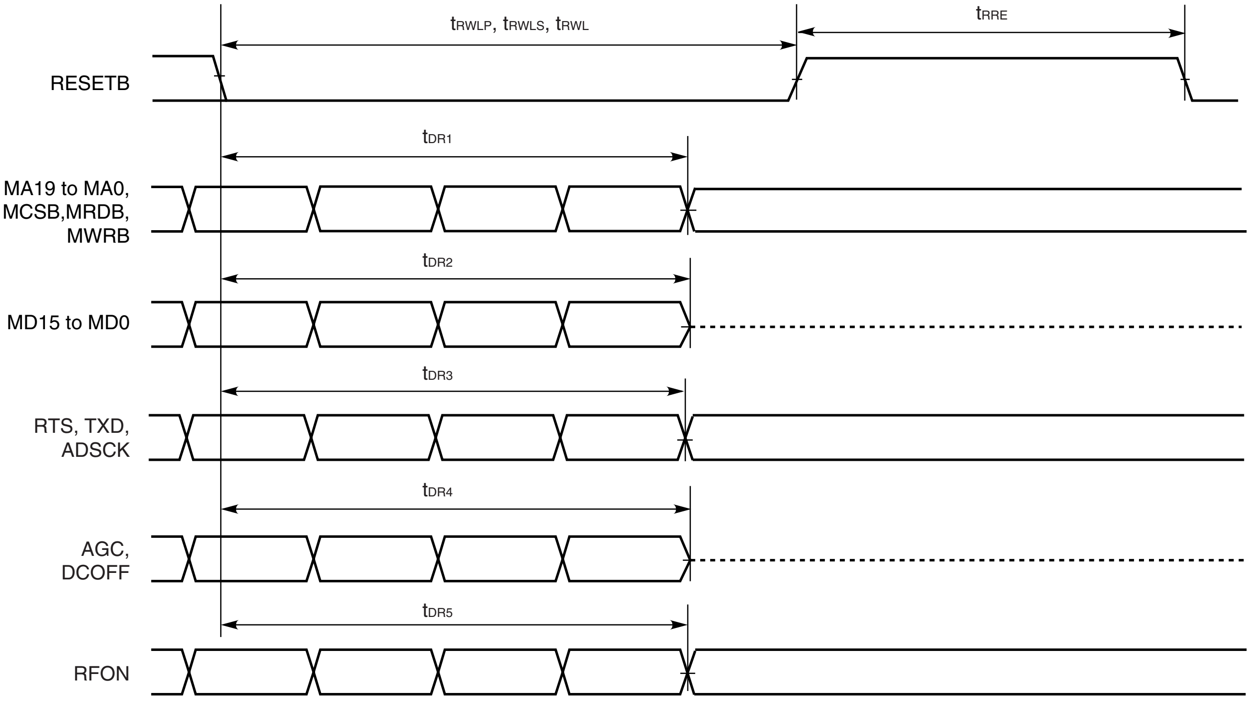
- Reset (RESETB)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
RESETB low-level width	t_{RWLP}	On power application[Note 1]	73.2	–	–	ns
	t_{RWLS}	In STOP mode	73.2	–	–	ns
	t_{RWL}	In normal mode	73.2	–		ns
		In HALT mode	1170			
RESETB recovery time	t_{RRE}	In normal mode	73.2	–	–	ns
		In HALT mode	1170			
Output initialization time 1[Note 2] (from RESETB↓)	t_{DR1}	MA19 to MA0, MCSB, MRDB, MWRB	–	–	t_{GCIC} + 30	ns
Output initialization time 2[Note 2] (from RESETB↓)	t_{DR2}	MD15 to MD0	–	–	t_{GCIC} + 30	ns
Output initialization time 3[Note 2] (from RESETB↓)	t_{DR3}	RTS, TXD, ADSCK	–	–	t_{GCIC} + 30	ns
Output initialization time 4[Note 2] (from RESETB↓)	t_{DR4}	AGC, DCOFF	–	–	t_{GCIC} + 30	ns
Output initialization time 5[Note 2] (from RESETB↓)	t_{DR5}	RFON	–	–	t_{GCIC} + 30	ns

Notes **1.** Measurement of EV_{DD}, IV_{DD}, and AV_{DD} starts at the point they reach the guaranteed operating voltage after power application.

2. Output pin signals are undefined from RESETB input to this output initialization time. Also, when the μPD77533 is restored from the STOP mode at power application, measurement starts from the point at which the internal clock starts oscillating stably.

- Reset timing


tRWLP, tRWLS, tRWL
tRRE
RESETB
tDR1
MA19 to MA0,
MCSB,MRDB,
MWRB
tDR2
MD15 to MD0
tDR3
RTS, TXD,
ADSCK
tDR4
AGC,
DCOFF
tDR5
RFON

Wakeup, TX Active

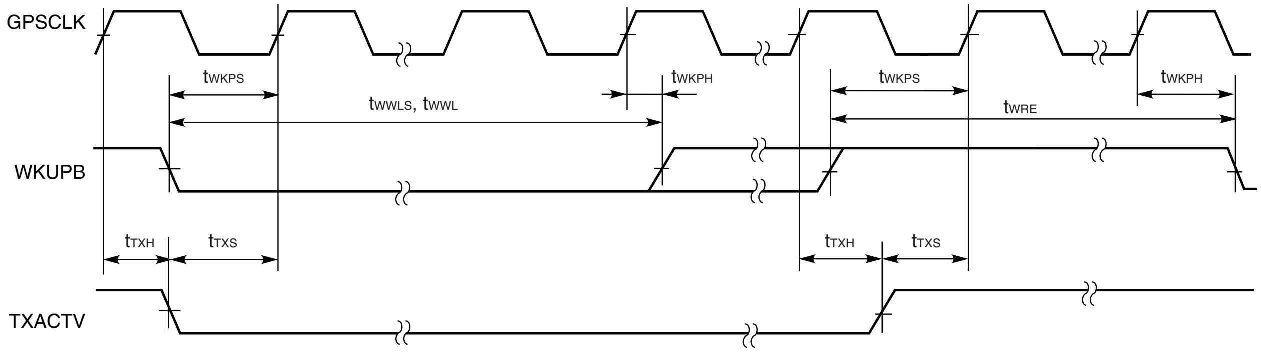
- Wakeup (WKUPB)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
WKUPB setup time (to GPSCLK↑)	t_{WKPS}		0	–	–	ns
WKUPB hold time (from GPSCLK↑)	t_{WKPH}		15.3	–	–	ns
WKUPB low-level width	t_{WWLS}	In STOP mode	73.2	–	–	ns
	t_{WWL}	In normal mode	73.2	–	–	ns
		In HALT mode	1170			
WKUPB recovery time	t_{WRE}	During normal operation	73.2	–	–	ns
		In HALT mode	1170	–	–	

- TX active

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
TXACTV setup time (to GPSCLK↑)	t_{TXS}		0	–	–	ns
TXACTV hold time (from GPSCLK↑)	t_{TXH}		t_{GCIC}	–	–	ns

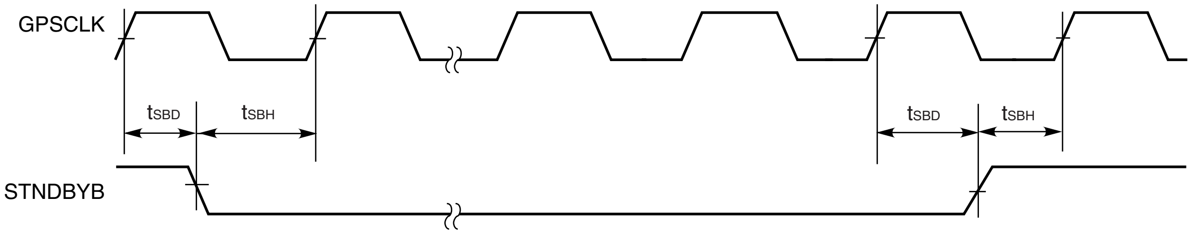
- Wakeup, TX active timing


GPSCLK
WKUPB
TXACTV
tWKPS
tWWLS, tWWL
tWKPH
tWKPS
tWRE
tWKPH
tTXH
tTXS
tTXH
tTXS

- Standby

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
STNDBYB delay time (from GPSCLK↑)	t_{SBD}		–	–	20	ns
STNDBYB hold time (to GPSCLK↑)	t_{SBH}		0	–	–	ns

- Standby timing

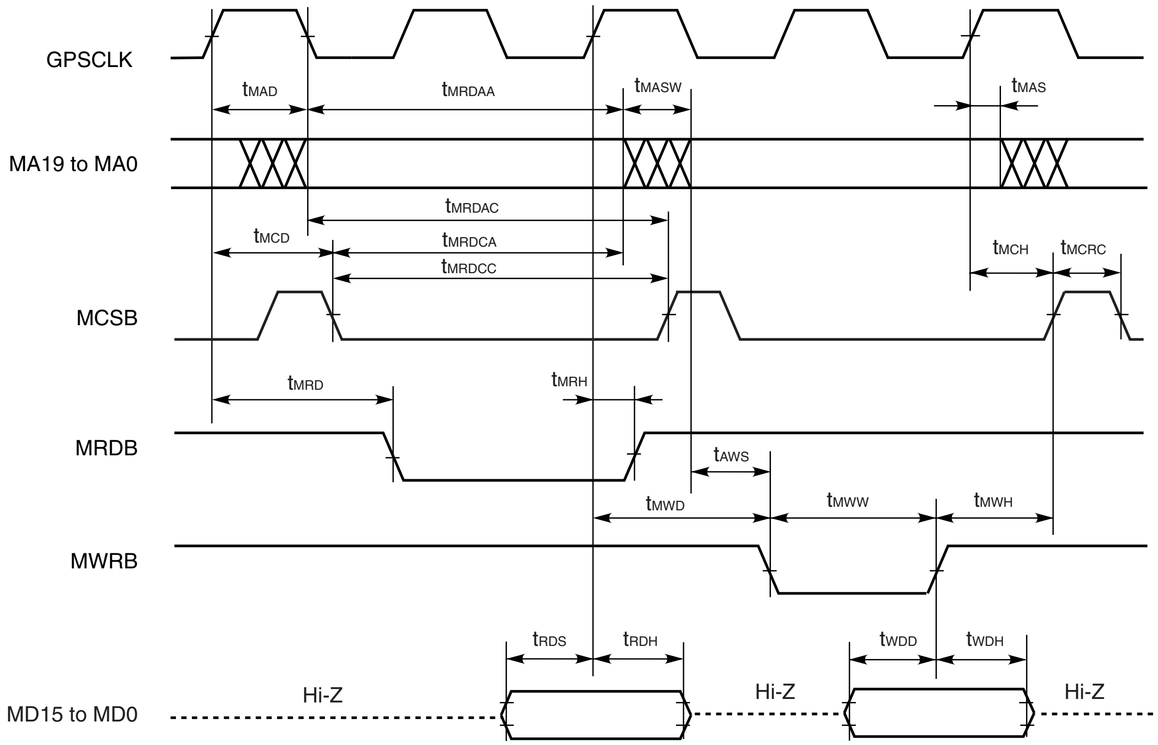

GPSCLK
tSBD
tSBH
tSBD
tSBH
STNDBYB

Memory Interface

- Mobile specified RAM

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Access cycle time (MA to MA)	t_{MRDAA}		122	–	1000	ns
Access cycle time (MA to MCSB↑)	t_{MRDAC}		122	–	1000	ns
Access cycle time (MCSB↓ to MA)	t_{MRDCA}		122	–	1000	ns
Access cycle time (MCSB↓ to MCSB↑)	t_{MRDCC}		122	–	1000	ns
Address (MA) delay time	t_{MAD}		–	–	10	ns
Address (MA) hold time	t_{MAS}		0	–	–	ns
Address (MA) skew time	t_{MASW}		–	–	10	ns
Chip select delay time	t_{MCD}		–	–	10	ns
Chip select hold time	t_{MCH}		0	–	–	ns
Chip select recovery time	t_{MCRC}		10	–	–	ns
Read strobe (MRDB) delay time	t_{MRD}		–	–	10	ns
Read strobe (MRDB) hold time	t_{MRH}		0	–	–	ns
Write strobe (MWRB) delay time	t_{MWD}	t_{AWS} is satisfied	–	–	10	ns
Write strobe (MWRB) hold time	t_{MWH}		10	–	–	ns
Write strobe (MWRB) width	t_{MWW}		30	–	–	ns
Address setup time (to MWRB↓)	t_{AWS}		0	–	–	ns
Read data (MD) setup time	t_{RDS}		20	–	–	ns
Read data (MD) hold time	t_{RDH}		0	–	–	ns
Write data (MD) delay time	t_{WDD}		–	–	20	ns
Write data (MD) hold time	t_{WDH}		0	–	–	ns

- Mobile specified RAM interface timing

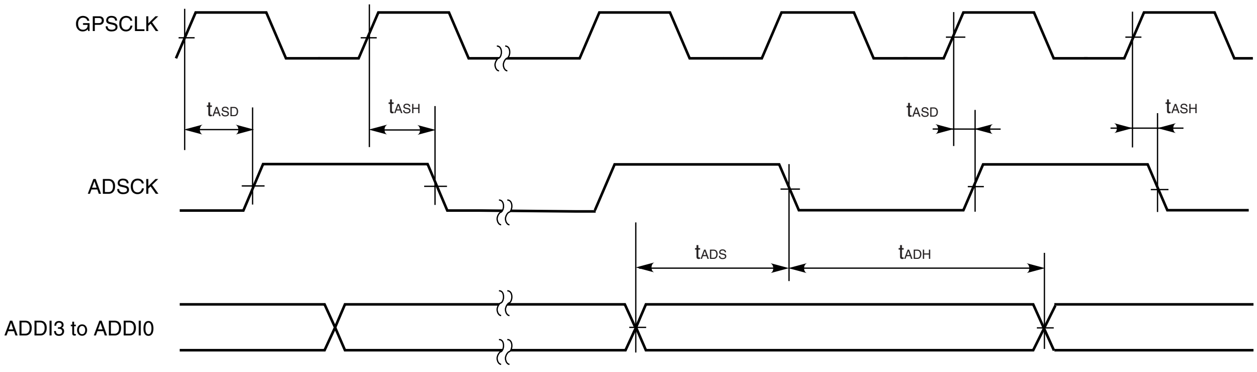

GPSCLK
MA19 to MA0
MCSB
MRDB
MWRB
MD15 to MD0
t_{MAD}
t_{MRDAA}
t_{MASW}
t_{MAS}
t_{MRDAC}
t_{MCD}
t_{MRDCA}
t_{MRDCC}
t_{MCH}
t_{MCRC}
t_{MRD}
t_{MRH}
t_{AWS}
t_{MWD}
t_{MWW}
t_{MWH}
t_{RDS}
t_{RDH}
t_{WDD}
t_{WDH}
Hi-Z
Hi-Z
Hi-Z

A/D Control Interface

- AD clock, AD data

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
ADSCK delay time (from GPSCLK↑)	t_{ASD}		–	–	20	ns
ADSCK hold time (from GPSCLK↑)	t_{ASH}		0	–	–	ns
ADDI setup time (to ADSCK↓)	t_{ADS}		20	–	–	ns
ADSCK hold time (from ADSCK↓)	t_{ADH}		5	–	–	ns

- AD clock, AD data timing

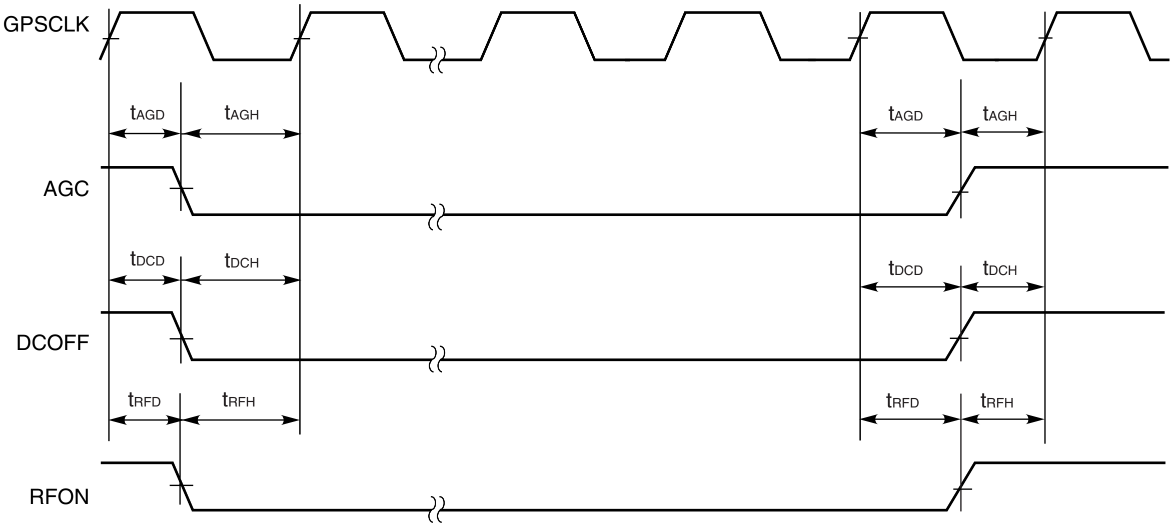

GPSCLK
ADSCK
ADDI3 to ADDI0
t_{ASD}
t_{ASH}
t_{ASD}
t_{ASH}
t_{ADS}
t_{ADH}

RF Control Interface

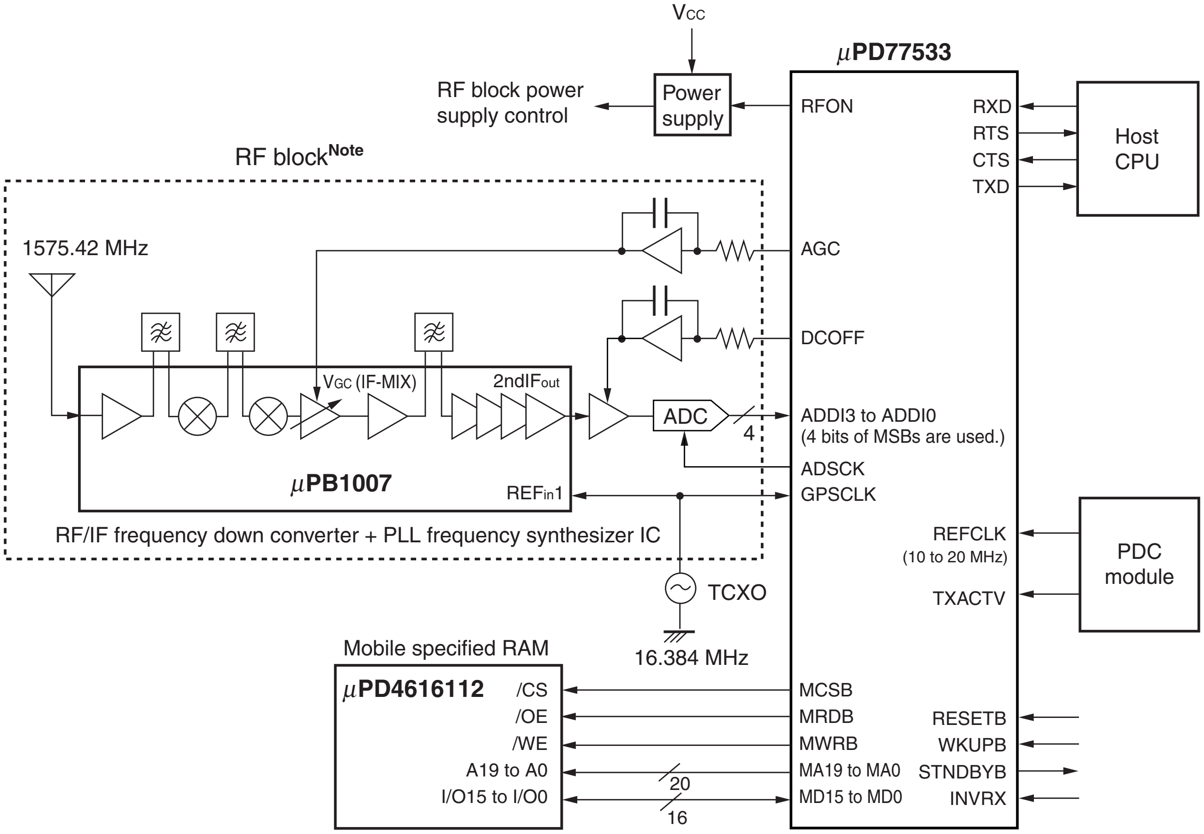
- AGC, DCOFF, RFON

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
AGC delay time (from GPSCLK↑)	t_{AGD}		–	–	20	ns
AGC hold time (to GPSCLK↑)	t_{AGH}		0	–	–	ns
DCOFF delay time (from GPSCLK↑)	t_{DCD}		–	–	20	ns
DCOFF hold time (to GPSCLK↑)	t_{DCH}		0	–	–	ns
RFON delay time (from GPSCLK↑)	t_{RFD}		–	–	50	ns
RFON hold time (to GPSCLK↑)	t_{RFH}		0	–	–	ns

- AGC, DCOFF, RFON timing


GPSCLK
tAGD
tAGH
AGC
tDCD
tDCH
DCOFF
tRFD
tRFH
RFON
tAGD
tAGH
tDCD
tDCH
tRFD
tRFH

★ 20. APPLICATION CIRCUIT EXAMPLE


VCC
μPD77533
RF block power supply control
Power supply
RFON
RXD
RTS
CTS
TXD
Host CPU
RF blockNote
1575.42 MHz
AGC
DCOFF
VGC (IF-MIX)
2ndIFout
ADC
4
ADDI3 to ADDI0
(4 bits of MSBs are used.)
ADSCK
GPSCLK
μPB1007
REFin1
RF/IF frequency down converter + PLL frequency synthesizer IC
REFCLK
(10 to 20 MHz)
TXACTV
PDC module
TCXO
16.384 MHz
Mobile specified RAM
μPD4616112
/CS
/OE
/WE
A19 to A0
I/O15 to I/O0
20
16
MCSB
MRDB
MWRB
MA19 to MA0
MD15 to MD0
RESETB
WKUPB
STNDBYB
INVRX

Caution The system gain of the μPD77533 is 60.1 dB.

Note The following shows the recommended operation environment for the GPS signal input portion (RF block) to the μPD77533.

Parameter	Value	
Local oscillator stability	Accuracy of input clock to REFCLK	< ±0.1 ppm
	Accuracy of input clock to GPSCLK	Not used
Resolution of A/D converter	4 bits or more	
Conversion speed of A/D converter	8.192 MHz	
System NF	< 2 dB	
In-band supurious	In the band of the center frequency of the last IF signal ±1 MHz, the total power of supurious should be less –20 dBc with reference to the IF noise level.	
Average antenna gain	–6 dBi or more (linear)	

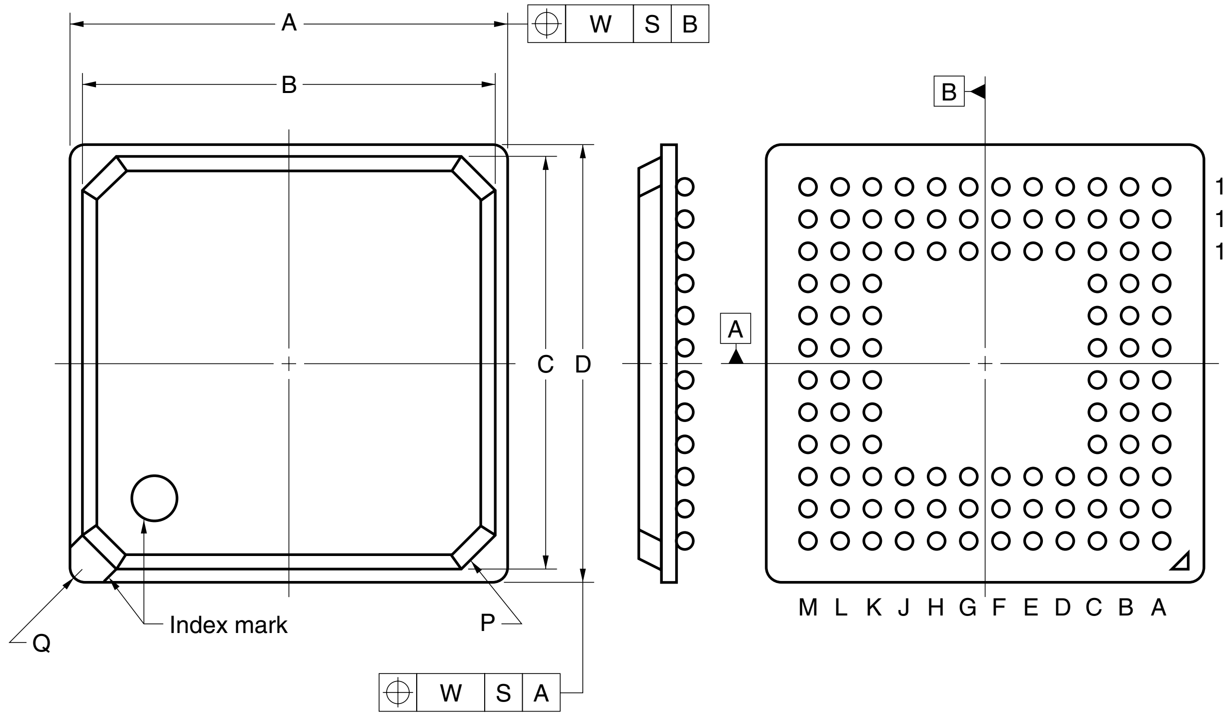
If the recommended accuracy of the local oscillator stability cannot be obtained, it is possible to use the FCC function via REFCLK (refer to **6.1 FCC Function**). In this case, however, the positioning time and sensitivity may be degraded.

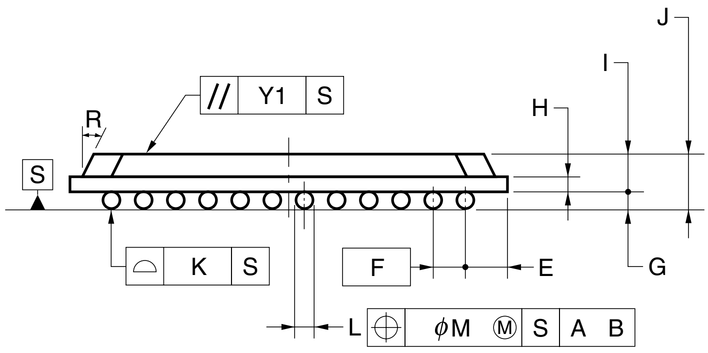
An example is shown below (in this example, the positioning time is approximately 30 seconds max., and the sensitivity is hardly degraded at all).

Accuracy of input clock to REFCLK: < ±0.3 ppm
Accuracy of input clock to GPSCLK: < ±10 ppm

21. PACKAGE DRAWING

108-PIN PLASTIC FBGA (11x11)


A
⊕ W S B
B
C
D
B
A
12
11
10
9
8
7
6
5
4
3
2
1
M L K J H G F E D C B A
Index mark
Q
P
⊕ W S A


J
I
H
// Y1 S
R
S
⌓ K S
F
E
G
L
⊕ ϕM Ⓜ S A B

ITEM	MILLIMETERS
A	11.00±0.10
B	10.40
C	10.40
D	11.00±0.10
E	1.10
F	0.8 (T.P.)
G	0.35±0.1
H	0.36
I	1.16
J	1.51±0.15
K	0.10
L	$\phi 0.50^{+0.05}_{-0.10}$
M	0.08
P	C1.0
Q	R0.3
R	25°
W	0.20
Y1	0.20

S108S1-80-YHC-1

22. RECOMMENDED SOLDERING CONDITIONS

This product should be soldered and mounted under the following recommended conditions.

For details of the recommended soldering conditions, refer to the document **Semiconductor Device Mounting Technology Manual (C10535E)**.

For soldering methods and conditions other than those recommended below, contact an NEC sales representative.

μPD77533S1-YHC: 108-pin plastic BGA (fine pitch) (11 × 11)

Soldering Method	Soldering Conditions	Recommended Condition Symbol
Infrared reflow	Package peak temperature: 260°C, Time: 60 seconds max. (at 220°C or higher), Count: Twice or less, Exposure limit: 3 days[Note] (after that, prebake at 125°C for 10 to 72 hours)	IR30-103-2

Note After opening the dry pack, store it at 25°C or less and 65% RH or less for the allowable storage period.

APPENDIX SAMPLE PROGRAM

The following program is a sample provided to enable an understanding of what is processed by the host CPU. The actual operation of the program differs depending on the system.

```
/*=============================================================================
*
*                           Host CPU sample program
*
===============================================================================
*                       Copyright(C) 2001 NEC Corporation.
===============================================================================
* This program describes an example of processing that should be performed by the host CPU.
* To actually operate the program, it must be revised to comply with the system.
=============================================================================*/


/*=====================*/
/*     Constant definition     */
/*=====================*/

/* BOOL value */
#define TRUE                         1       /* TRUE */
#define FALSE                        0       /* FALSE */

/* State values used by state machine */
#define STATE_INIT            0       /* Initialization */
#define STATE_WAIT_RESET      1       /* Reset response wait */
#define STATE_SEND_TIMEOUT    2       /* Character Timeout command transmission */
#define STATE_WAIT_TIMEOUT    3       /* Character Timeout response wait */
#define STATE_SEND_FCC_FREQ   4       /* FCC Frequency command transmission */
#define STATE_WAIT_FCC_FREQ   5       /* FCC Frequency response wait */
#define STATE_SEND_TXACVT     6       /* TXACVT command transmission */
#define STATE_WAIT_TXACVT     7       /* TXACVT response wait */
#define STATE_SEND_FCC_CTRL   8       /* FCC Control command transmission */
#define STATE_WAIT_FCC_CTRL   9       /* FCC Control response wait */
#define STATE_SEND_FLOW_CTRL  10      /* Flow Point Control command transmission */
#define STATE_WAIT_FLOW_CTRL  11      /* Flow Point Control response wait */
#define STATE_INIT_DONE       12      /* Initialization complete */
#define STATE_DONE            13      /* Complete */

/* Serial buffer control value */
#define SERIAL_GM             0       /* GM size of serial line */
#define SERIAL_LS             1       /* LS size of serial line */

#define SERIAL_IN             0       /* Serial input */
#define SERIAL_OUT            1       /* Serial output */

/* Packet */
#define PACKET_DATA_LEN       255     /* Maximum length of data segment of packet */
#define PACKET_INFO_LEN       3       /* Length of information segment of packet */
#define PACKET_MAX_LEN      (PACKET_DATA_LEN+PACKET_INFO_LEN)   /* Maximum length of packet */
#define PACKET_FLG            0x03    /* FLG (not continued) */
#define PACKET_FLG_CONT       0x02    /* FLG (continued) */

/* Buffer size */
#define MAXBLOCK              PACKET_MAX_LEN  /* Block size of serial reception */
```

```
/* LS command */
#define CMD_CA              0x4341  /* "CA" */
#define CMD_BB              0x4242  /* "BB" */
#define CMD_IA              0x4941  /* "IA" */
#define CMD_JM              0x4A4D  /* "JM" */
#define CMD_PJ              0x504A  /* "PJ" */
#define CMD_AC              0x4143  /* "AC" */
#define CMD_DA              0x4441  /* "DA" */
#define CMD_EA              0x4541  /* "EA" */
#define CMD_FA              0x4641  /* "FA" */
#define CMD_FC              0x4643  /* "FC" */
#define CMD_BD              0x4244  /* "BD" */
#define CMD_CB              0x4342  /* "CB" */

#define CMD_ID_LS           0x80    /* LS command */
#define CMD_ID_EXTENTION    0x7F    /* Extension command */

#define LS_CMD_MAX_LEN      330     /* Maximum length of LS command */


/*====================*/
/*    Structure definition    */
/*====================*/

/* COM information */
/* Serial transmission/reception data is stored in COM information work area */
typedef struct {
    unsigned char   BinData[2][PACKET_MAX_LEN]; /* Transmitted/received data (binary) */
    int             CurrentLen[2];    /* Length of data stored in BinData */
    int             EstimateLen[2];   /* Estimated total length of packet */
    int             FixedFlag[2];     /* Packet complete flag */
        /* The above packet complete flag becomes TRUE when the whole */
        /* packet has been received in SERIAL_IN, and in SERIAL_OUT, */
        /* transmission is started immediately after the flag is set to TRUE. */
    int             AckNackWait;      /* ACK/NACK wait flag */
    int             LScmdLen;         /* Length of command from LS */
} COMMINFO;


/* Response information */
/* Response from GS and commands from LS are stored in response */
/* information work area when received                          */
typedef struct {
    /* GM side */
    unsigned int    GmResCnt;         /* Response count on GM side */
    int             GmResFixed;       /* Response reception complete flag on GM side */
    int             GmResLen;         /* Length of response on GM side */
    unsigned char   GmResData[PACKET_MAX_LEN];  /* Response data on GM side */
    /* LS side */
    unsigned int    LsResCnt;          /* Response count on LS side */
    int             LsResFixed;        /* Response reception complete flag on LS side */
    int             LsResLen;          /* Length of response on LS side */
    unsigned char   LsResData[LS_CMD_MAX_LEN];  /* Response data on LS side */
} RESINFO;


/*==================*/
/*   Function prototype   */
/*==================*/

static int CompareData(const unsigned char *Data1, int Len1, const unsigned char *Data2, int Len2);
static int RecvDataFromGm(unsigned char *Buf);
static int RecvDataFromLS(unsigned char *Buf);
static void SendDataToGm(const unsigned char *Data, int Len);
static void SendDataToLs(const unsigned char *Data, int Len);
static void HostCPUThreadLoop(void);
static void GmSerialIn(void);
static void CopyGmRes(void);
static void CheckGmRes(void);
static void DiscardLastPacket(void);
static void ResendLastPacket(void);
static void SendLSAck(void);
static void SendLocalAck(void);
static void SendNack(void);
static void CopyGmToLs(void);
static void GmSerialOut(void);
static void LsSerialIn(void);
static void CopyLsRes(void);
static void CopyLsToGm(void);
static int GetLsCmdLen(unsigned char *Data);
static void LsSerialOut(void);


/*======================*/
/*       Static work area        */
/*======================*/

static COMMINFO GmComm;      /* COM information on GM side */
static COMMINFO LsComm;      /* COM information on LS side */
static RESINFO ResInfo;      /* Response information */

/* Data string of GM command */
/* ACK (LS command) */
static const unsigned char ServerACK[] = { CMD_ID_LS, 0x00, PACKET_FLG };
/* ACK (local command) */
static const unsigned char LocalACK[] = { 0x3F, 0x01, 0x00, PACKET_FLG };
/* NACK */
static const unsigned char BothNACK[] = { 0x3F, 0x01 };
/* NACK (error) */
static const unsigned char ErrorNACK[] = { 0x3F, 0x01, 0x01, PACKET_FLG };
/* Reset response */
static const unsigned char ResetRsp[] = { 0x04, 0x00, PACKET_FLG };
/* Character Timeout command/response */
static const unsigned char TimeoutCmdRsp[] = { 0x13, 0x03, 0xE8, 0x03, 0x00, PACKET_FLG };
/* FCC Frequency command/response */
static const unsigned char FccFreqCmdRsp[] = { 0x02, 0x04, 0x00, 0x00, 0xE1, 0x00, PACKET_FLG };
/* TXACVT command/response */
static const unsigned char TxacvtCmdRsp[] = { 0x07, 0x01, 0x01, PACKET_FLG };
/* FCC Control command/response */
static const unsigned char FccCtrlCmdRsp[] = { 0x06, 0x01, 0x01, PACKET_FLG };
/* Flow Point Control command/response */
static const unsigned char FlowPointCtrlCmdRsp[] = { 0x7F, 0x03, 0x00, 0x10, 0x00, PACKET_FLG };



/*---------------------------------------------------------------------------------
*    Function
*         Main program
*
*    Parameter
*         None
*
```

```
*     Return Value
*           None
---------------------------------------------------------------------------- */
int main(void)
{
    int State;
    unsigned char Buf[LS_CMD_MAX_LEN];
    int Len;

    /* State machine */
    /* State machine to perform a series of processing: initialization – Reset wait – default settings */
    State = STATE_INIT;
    while(State != STATE_DONE)
    {
        switch(State)
        {
            /******************/
            /* Initialization state   */
            /******************/
            case STATE_INIT :
                /* Initialize internal work */
                GmComm.CurrentLen[SERIAL_OUT]        = 0;
                GmComm.CurrentLen[SERIAL_IN]         = 0;
                GmComm.EstimateLen[SERIAL_OUT]       = 0;
                GmComm.EstimateLen[SERIAL_IN]        = 0;
                GmComm.FixedFlag[SERIAL_OUT]         = FALSE;
                GmComm.FixedFlag[SERIAL_IN]          = FALSE;
                GmComm.AckNackWait                   = FALSE;
                GmComm.LScmdLen                      = 0;

                LsComm.CurrentLen[SERIAL_OUT]        = 0;
                LsComm.CurrentLen[SERIAL_IN]         = 0;
                LsComm.EstimateLen[SERIAL_OUT]       = 0;
                LsComm.EstimateLen[SERIAL_IN]        = 0;
                LsComm.FixedFlag[SERIAL_OUT]         = FALSE;
                LsComm.FixedFlag[SERIAL_IN]          = FALSE;
                LsComm.AckNackWait                   = FALSE;
                LsComm.LScmdLen                      = 0;

                ResInfo.GmResCnt      = 0;
                ResInfo.GmResFixed    = FALSE;
                ResInfo.GmResLen      = 0;
                ResInfo.LsResCnt      = 0;
                ResInfo.LsResFixed    = FALSE;
                ResInfo.LsResLen      = 0;

                /* Open serial communication on GM and LS sides */
                OpenSerial(SERIAL_GM);
                OpenSerial(SERIAL_LS);

                /* Start host CPU processing thread */
                CreateThread(HostCPUThreadLoop);

                State = STATE_WAIT_RESET;
                break;

            /*******************************/
            /*      Reset response wait state       */
            /*******************************/
            /* Wait for Reset response and proceed to next processing when received */
            case STATE_WAIT_RESET :
```

```
    Len = RecvDataFromGm(Buf);
    if(CompareData(Buf, Len, ResetRsp, sizeof(ResetRsp)) == 0)
        State = STATE_SEND_TIMEOUT;
    break;

/*******************************************/
/*       Character Timeout transmission state        */
/*******************************************/
/* Send Character Timeout command to GM (timeout set to 1 second) */
case STATE_SEND_TIMEOUT :
    SendDataToGm(TimeoutCmdRsp, sizeof(TimeoutCmdRsp));
    State = STATE_WAIT_TIMEOUT;
    break;

/* Wait for Character Timeout response */
/* Wait for Character Timeout response and proceed to next processing when received */
case STATE_WAIT_TIMEOUT :
    Len = RecvDataFromGm(Buf);
    if(CompareData(Buf, Len, TimeoutCmdRsp, sizeof(TimeoutCmdRsp)) == 0)
        State = STATE_SEND_FCC_FREQ;
    break;

/***************************************/
/*   FCC Frequency command transmission state   */
/***************************************/
/* Send FCC Frequency command to GM (frequency set to 14.4 MHz） */
case STATE_SEND_FCC_FREQ :
    SendDataToGm(FccFreqCmdRsp, sizeof(FccFreqCmdRsp));
    State = STATE_WAIT_FCC_FREQ;
    break;

/* Wait for FCC Frequency response */
/* Wait for FCC Frequency and proceed to next processing when received */
case STATE_WAIT_FCC_FREQ :
    Len = RecvDataFromGm(Buf);
    if(CompareData(Buf, Len, FccFreqCmdRsp, sizeof(FccFreqCmdRsp)) == 0)
        State = STATE_SEND_TXACVT;
    break;

/********************************/
/* TXACVT command transmission state */
/********************************/
/* Send TXACVT command to GM (TXACVT control set to ON) */
case STATE_SEND_TXACVT :
    SendDataToGm(TxacvtCmdRsp, sizeof(TxacvtCmdRsp));
    State = STATE_WAIT_TXACVT;
    break;

/* Wait for TXACVT response */
/* Wait for TXACVT response and proceed to next processing when received */
case STATE_WAIT_TXACVT :
    Len = RecvDataFromGm(Buf);
    if(CompareData(Buf, Len, TxacvtCmdRsp, sizeof(TxacvtCmdRsp)) == 0)
        State = STATE_SEND_FCC_CTRL;
    break;

/*************************************/
/*   FCC Control command transmission state   */
/*************************************/
/* Send FCC Control command to GM (FCC control set to ON) */
case STATE_SEND_FCC_CTRL :
```

```
                SendDataToGm(FccCtrlCmdRsp, sizeof(FccCtrlCmdRsp));
                State = STATE_WAIT_FCC_CTRL;
                break;

            /* Wait for FCC Control response */
            /* Wait for FCC Control response and proceed to next processing when received. */
            case STATE_WAIT_FCC_CTRL :
                Len = RecvDataFromGm(Buf);
                if(CompareData(Buf, Len, FccCtrlCmdRsp, sizeof(FccCtrlCmdRsp)) == 0)
                    State = STATE_SEND_FLOW_CTRL;
                break;

            /********************************************/
            /*     Flow Point Control command transmission state     */
            /********************************************/
            /* Send Flow Point Control command to GM (Processing End output set to ON) */
            case STATE_SEND_FLOW_CTRL :
                SendDataToGm(FlowPointCtrlCmdRsp, sizeof(FlowPointCtrlCmdRsp));
                State = STATE_WAIT_FLOW_CTRL;
                break;

            /* Wait for Flow Point Control response */
            /* Wait for Flow Point Control response and proceed to next processing when received. */
            case STATE_WAIT_FLOW_CTRL :
                Len = RecvDataFromGm(Buf);
                if(CompareData(Buf, Len, FlowPointCtrlCmdRsp, sizeof(FlowPointCtrlCmdRsp)) == 0)
                    State = STATE_INIT_DONE;
                break;

            /****************************************************/
            /* GM initialization complete.  Start processing such as positioning.  */
            /****************************************************/
            case STATE_INIT_DONE :
                /* Perform processing after initialization, if required. */
                break;
        }
    }
    return(0);
}


/*----------------------------------------------------------------------------
*    Function
*        Compare data
*
*    Parameter
*        const unsigned char *Data1: Data 1
*        int Len1: Number of bytes of data 1
*        const unsigned char *Data2: Data 2
*        int Len2: Number of bytes of data 2
*
*    Return Value
*        int: result (0 = match)
-------------------------------------------------------------------------- */
static int CompareData(const unsigned char *Data1, int Len1, const unsigned char *Data2, int Len2)
{
    int Sts;

    Sts = Len1 - Len2;
    if(Sts == 0)
        Sts = memcmp(Data1, Data2, Len1);
    return(Sts);
}


/*--------------------------------------------------------------------------
*    Function
*          Receive data from GM
*
*    Parameter
*          unsigned char *Buf: buffer that receives data
*
*    Return Value
*          int: number of bytes of data (0 = no data)
------------------------------------------------------------------------- */
static int RecvDataFromGm(unsigned char *Buf)
{
    int Len;

    Len = 0;
    if(ResInfo.GmResFixed)        /* If there is a response */
    {
        /* Copy to buffer */
        memcpy(Buf, ResInfo.GmResData, ResInfo.GmResLen);
        Len = ResInfo.GmResLen;
        ResInfo.GmResFixed = FALSE;
    }
    return(Len);
}


/*--------------------------------------------------------------------------
*    Function
*          Receive data from LS
*
*    Parameter
*          unsigned char *Buf: buffer that receives data
*
*    Return Value
*          int: number of bytes of data (0 = no data)
------------------------------------------------------------------------- */
static int RecvDataFromLS(unsigned char *Buf)
{
    int Len;

    Len = 0;
    if(ResInfo.LsResFixed)        /* If there is a response */
    {
         /* Copy to buffer */
        memcpy(Buf, ResInfo.LsResData, ResInfo.LsResLen);
        Len = ResInfo.LsResLen;
        ResInfo.LsResFixed = FALSE;
    }
    return(Len);
}


/*--------------------------------------------------------------------------
*    Function
*          Transmit data to GM
*
*    Parameter
*           unsigned char *Data: data to be transmitted
*           int Len: number of bytes of data
*
*     Return Value
*           None
---------------------------------------------------------------------------- */
static void SendDataToGm(const unsigned char *Data, int Len)
{
    /* Copy data to be transmitted to COM information */
    memcpy(GmComm.BinData[SERIAL_OUT], Data, Len);
    GmComm.CurrentLen[SERIAL_OUT] = Len;
    GmComm.FixedFlag[SERIAL_OUT] = TRUE;     /* This flag starts transmission */
    return;
}


/*----------------------------------------------------------------------------
*     Function
*           Transmit data to LS
*
*     Parameter
*           unsigned char *Data: data to be transmitted
*           int Len: number of bytes of data
*
*     Return Value
*           None
---------------------------------------------------------------------------- */
static void SendDataToLs(const unsigned char *Data, int Len)
{
    /* Copy data to be transmitted to COM information */
    memcpy(LsComm.BinData[SERIAL_OUT], Data, Len);
    LsComm.CurrentLen[SERIAL_OUT] = Len;
    LsComm.FixedFlag[SERIAL_OUT] = TRUE;     /* This flag starts transmission */
    return;
}


/*----------------------------------------------------------------------------
*     Function
*           Thread of host CPU processor
*
*     Parameter
*           None
*
*     Return Value
*           None
*
*     Note
*           This thread is separately processed to prevent it from being
*           affected by other main program processing.
---------------------------------------------------------------------------- */
static void HostCPUThreadLoop(void)
{
    for(;;)
    {
        /* Receive data from GM and store it in GmComm */
        GmSerialIn();

        /* In a state other than waiting for ACK/NACK from GM, */
        /* receive data from LS and store it in LsComm. */
        if(GmComm.AckNackWait == FALSE)
```

```
            LsSerialIn();

        /* Transmit transmission data stored in GmComm. */
        GmSerialOut();

        /* Transmit transmission data stored in LsComm. */
        LsSerialOut();
    }
}


/*--------------------------------------------------------------------------------
*    Function
*        Input by DSP
*
*    Parameter
*        None
*
*    Return Value
*        None
-------------------------------------------------------------------------------- */
static void GmSerialIn(void)
{
    int Len;
    unsigned char Buf[MAXBLOCK];
    int i;

    /* Read response from GM */
    if((Len = RecvFromSerial(SERIAL_GM, (char*)Buf, MAXBLOCK)) > 0)
    {
        /* Analyze contents of response */
        for(i = 0; i < Len; i++)
        {
            GmComm.BinData[SERIAL_IN][GmComm.CurrentLen[SERIAL_IN]] = Buf[i];
            GmComm.CurrentLen[SERIAL_IN] += 1;

            if(GmComm.CurrentLen[SERIAL_IN] == 1)    /* 1st byte? */
            {
                 /* CMD-ID has been read */
                GmComm.EstimateLen[SERIAL_IN] = -1;
            }
            else if(GmComm.CurrentLen[SERIAL_IN] == 2)  /* 2nd byte? */
            {
                /* LENGTH has been read */
                GmComm.EstimateLen[SERIAL_IN] = Buf[i]+PACKET_INFO_LEN;
            }
            else
            {
                if(GmComm.CurrentLen[SERIAL_IN] == GmComm.EstimateLen[SERIAL_IN])
                {
                    /* Whole packet has been read */
                    GmComm.FixedFlag[SERIAL_IN] = TRUE;

                    /* Check response from GM */
                    CopyGmRes();
                    CheckGmRes();

                    GmComm.FixedFlag[SERIAL_IN] = FALSE;
                    GmComm.CurrentLen[SERIAL_IN] = 0;
                }
            }
        }
    }
    return;
}


/*-----------------------------------------------------------------------------
*    Function
*        Copy response from GM to response information
*
*    Parameter
*        None
*
*    Return Value
*        None
----------------------------------------------------------------------------- */
static void CopyGmRes(void)
{
    ResInfo.GmResCnt += 1;
    ResInfo.GmResLen = GmComm.CurrentLen[SERIAL_IN];
    memcpy(ResInfo.GmResData, GmComm.BinData[SERIAL_IN], ResInfo.GmResLen);
    ResInfo.GmResFixed = TRUE;
    return;
}


/*-----------------------------------------------------------------------------
*    Function
*        Check response from GM
*
*    Parameter
*        None
*
*    Return Value
*        None
----------------------------------------------------------------------------- */
static void CheckGmRes(void)
{
    unsigned char Flag;
    int Len;

    Len = GmComm.CurrentLen[SERIAL_IN];
    Flag = GmComm.BinData[SERIAL_IN][Len-1];
    if((Flag == PACKET_FLG) || (Flag == PACKET_FLG_CONT))
    {
        if(((Len == 3) && (memcmp(GmComm.BinData[SERIAL_IN], ServerACK, sizeof(ServerACK)) == 0)) ||
           ((Len == 4) && (memcmp(GmComm.BinData[SERIAL_IN], LocalACK, sizeof(LocalACK)) == 0)))
        {
            /* Response is ACK */
            if(GmComm.AckNackWait)
                DiscardLastPacket();
#if 0
            else
                Unnecessary ACK is received from GM when ACK/NACK is not being waited for
#endif
        }
        else if((Len == 4) && (memcmp(GmComm.BinData[SERIAL_IN], BothNACK, sizeof(BothNACK)) == 0))
        {
            /* Response is NACK */
            if(GmComm.AckNackWait)
                ResendLastPacket();
        }
        else if(GmComm.BinData[SERIAL_IN][0] == CMD_ID_LS)
        {
            /* Response in LS command */
            SendLSAck();
            CopyGmToLs();
        }
        else if((GmComm.BinData[SERIAL_IN][0] == CMD_ID_EXTENTION) &&
                (GmComm.BinData[SERIAL_IN][2] != 0x00))
        {
            /* Response is flow point setting */
            SendLocalAck();
        }
        else
        {
            /* Response is local command */

            /* Add items to be processed, if any */
        }
    }
    else
    {
        /* FLAG incorrect */
        /* Transmit NACK for server command packets to request re-sending */
        if(GmComm.BinData[SERIAL_IN][0] == CMD_ID_LS)
            SendNack();
    }
    return;
}


/*----------------------------------------------------------------------------
*   Function
*           Discard packet transmitted last
*
*   Parameter
*           None
*
*   Return Value
*           None
--------------------------------------------------------------------------- */
static void DiscardLastPacket(void)
{
    GmComm.AckNackWait = FALSE;
    GmComm.CurrentLen[SERIAL_OUT] = 0;
    return;
}


/*----------------------------------------------------------------------------
*   Function
*           Re-send packet transmitted last
*
*   Parameter
*           None
*
*   Return Value
*           None
--------------------------------------------------------------------------- */
static void ResendLastPacket(void)
{
    GmComm.FixedFlag[SERIAL_OUT] = TRUE;
    GmComm.AckNackWait = FALSE;
    GmSerialOut();
}


/*-------------------------------------------------------------------------------
*    Function
*         Transmitted a server command, ACK(80 00 03)
*
*    Parameter
*         None
*
*    Return Value
*         None
------------------------------------------------------------------------------ */
static void SendLSAck(void)
{
    memcpy(GmComm.BinData[SERIAL_OUT], ServerACK, sizeof(ServerACK));
    GmComm.CurrentLen[SERIAL_OUT] = sizeof(ServerACK);
    GmComm.FixedFlag[SERIAL_OUT] = TRUE;
    GmSerialOut();
}


/*-------------------------------------------------------------------------------
*    Function
*         Transmitted a local command, ACK(3f 01 00 03)
*
*    Parameter
*         None
*
*    Return Value
*         None
------------------------------------------------------------------------------ */
static void SendLocalAck(void)
{
    memcpy(GmComm.BinData[SERIAL_OUT], LocalACK, sizeof(LocalACK));
    GmComm.CurrentLen[SERIAL_OUT] = sizeof(LocalACK);
    GmComm.FixedFlag[SERIAL_OUT] = TRUE;
    GmSerialOut();
}


/*-------------------------------------------------------------------------------
*    Function
*         Transmit NACK
*
*    Parameter
*         None
*
*    Return Value
*         None
------------------------------------------------------------------------------ */
static void SendNack(void)
{
    memcpy(GmComm.BinData[SERIAL_OUT], ErrorNACK, sizeof(ErrorNACK));
    GmComm.CurrentLen[SERIAL_OUT] = sizeof(ErrorNACK);
    GmComm.FixedFlag[SERIAL_OUT] = TRUE;
    GmSerialOut();
}
```

```
/*-----------------------------------------------------------------------------
*     Function
*           Transmit response from GM to LS
*
*     Parameter
*           None
*
*     Return Value
*           None
---------------------------------------------------------------------------- */
static void CopyGmToLs(void)
{
    int Len;

    /* Transmit only contents of packet to LS */
    Len = GmComm.CurrentLen[SERIAL_IN] - PACKET_INFO_LEN;
    memcpy(LsComm.BinData[SERIAL_OUT], GmComm.BinData[SERIAL_IN] + 2, Len);
    LsComm.CurrentLen[SERIAL_OUT] = Len;
    LsComm.FixedFlag[SERIAL_OUT] = TRUE;
    LsSerialOut();
    return;
}


/*-----------------------------------------------------------------------------
*     Function
*           Output to GM
*
*     Parameter
*           None
*
*     Return Value
*           None
---------------------------------------------------------------------------- */
static void GmSerialOut(void)
{
    if(GmComm.FixedFlag[SERIAL_OUT])
    {
        /* Output to GM */
        SendToSerial(SERIAL_GM, (char*)GmComm.BinData[SERIAL_OUT], GmComm.CurrentLen[SERIAL_OUT]);
        GmComm.FixedFlag[SERIAL_OUT] = FALSE;

        /* For LS command, set so that ACK/NACK is waited for */
        if((GmComm.BinData[SERIAL_OUT][0] == CMD_ID_LS) &&
           ((GmComm.CurrentLen[SERIAL_OUT] != 3) ||
            (memcmp(GmComm.BinData[SERIAL_OUT], ServerACK, sizeof(ServerACK)) != 0)))
        {
            GmComm.AckNackWait = TRUE;
        }
        else
            GmComm.CurrentLen[SERIAL_OUT] = 0;
    }
    return;
}


/*-----------------------------------------------------------------------------
*     Function
*           Input by LS
```

```
*
*    Parameter
*         None
*
*    Return Value
*         None
----------------------------------------------------------------------------- */
static void LsSerialIn(void)
{
    int Len;

    if(LsComm.EstimateLen[SERIAL_IN] > 0)
    {
        /* Data previously read remains unused.  When multiple commands */
        /* (that need to be divided into multiple packets) are continuously transmitted */
        /* from LS, the first command is immediately used, however, the second and later */
        /* commands stay in packets and are transmitted here.  */
        CopyLsToGm();

        if(LsComm.CurrentLen[SERIAL_IN] == LsComm.EstimateLen[SERIAL_IN])
            LsComm.EstimateLen[SERIAL_IN] = 0;
    }
    else if((Len = RecvFromSerial(SERIAL_LS, (char*)LsComm.BinData[SERIAL_IN], PACKET_DATA_LEN)) > 0)
    {
        LsComm.CurrentLen[SERIAL_IN] = Len;
        LsComm.EstimateLen[SERIAL_IN] = 0;

        CopyLsToGm();

        if(LsComm.CurrentLen[SERIAL_IN] == LsComm.EstimateLen[SERIAL_IN])
            LsComm.EstimateLen[SERIAL_IN] = 0;
    }
    return;
}


/*-----------------------------------------------------------------------------
*    Function
*         Copy command from LS to response buffer
*
*    Parameter
*         None
*
*    Return Value
*         None
----------------------------------------------------------------------------- */
static void CopyLsRes(void)
{
    ResInfo.LsResCnt += 1;
    ResInfo.LsResLen = LsComm.CurrentLen[SERIAL_IN];
    memcpy(ResInfo.LsResData, LsComm.BinData[SERIAL_IN], ResInfo.LsResLen);
    ResInfo.LsResFixed = TRUE;
    return;
}


/*-----------------------------------------------------------------------------
*    Function
*         Output command from LS to GM
*
*    Parameter
*           None
*
*     Return Value
*           None
---------------------------------------------------------------------------- */
static void CopyLsToGm(void)
{
    int i;
    unsigned char Data;
    static unsigned char LScmdData[5];

    for(i = LsComm.EstimateLen[SERIAL_IN]; i < LsComm.CurrentLen[SERIAL_IN]; i++)
    {
        Data = LsComm.BinData[SERIAL_IN][i];

        if(GmComm.CurrentLen[SERIAL_OUT] == 0)
        {
            /* First data */
            /* Put into packet */
            GmComm.BinData[SERIAL_OUT][0] = CMD_ID_LS;   /* CMD-ID = 0x80, Set LENGTH later */
            GmComm.BinData[SERIAL_OUT][2] = Data;
            GmComm.CurrentLen[SERIAL_OUT] = PACKET_INFO_LEN;
        }
        else
        {
            /* Set data */
            GmComm.BinData[SERIAL_OUT][GmComm.CurrentLen[SERIAL_OUT]] = Data;
            GmComm.CurrentLen[SERIAL_OUT] += 1;
        }

        if(LsComm.LScmdLen < 5)
            LScmdData[LsComm.LScmdLen] = Data;
        LsComm.LScmdLen += 1;
        if(LsComm.LScmdLen == 5)
        {
            LsComm.EstimateLen[SERIAL_OUT] = GetLsCmdLen(LScmdData);
            if(LsComm.EstimateLen[SERIAL_OUT] == 0)
                GmComm.CurrentLen[SERIAL_OUT] = 0;
        }

        if(LsComm.LScmdLen == LsComm.EstimateLen[SERIAL_OUT])
        {
            /* Last packet */
            /* Transmit with FLAG = 0x03 */
            GmComm.BinData[SERIAL_OUT][GmComm.CurrentLen[SERIAL_OUT]] = PACKET_FLG;
            GmComm.CurrentLen[SERIAL_OUT] += 1;
            GmComm.BinData[SERIAL_OUT][1] = GmComm.CurrentLen[SERIAL_OUT] - PACKET_INFO_LEN;
            GmComm.FixedFlag[SERIAL_OUT] = TRUE;

            CopyLsRes();

            LsComm.LScmdLen = 0;
            LsComm.EstimateLen[SERIAL_OUT] = 0;
            i++;
            break;
        }
    }

    if(GmComm.FixedFlag[SERIAL_OUT] == FALSE)
    {
        /* Intermediate packet */
```

```
            /* Transmit with FLAG = 0x02 */
            GmComm.BinData[SERIAL_OUT][GmComm.CurrentLen[SERIAL_OUT]] = PACKET_FLG_CONT;
            GmComm.CurrentLen[SERIAL_OUT] += 1;
            GmComm.BinData[SERIAL_OUT][1] = GmComm.CurrentLen[SERIAL_OUT] - PACKET_INFO_LEN;
            GmComm.FixedFlag[SERIAL_OUT] = TRUE;
        }

        GmSerialOut();

        LsComm.EstimateLen[SERIAL_IN] = i;
        return;
    }


    /*------------------------------------------------------------------------------
    *   Function
    *       Find expected value of LS command length
    *
    *   Parameter
    *       unsigned char *Data: data string of LS command
    *
    *   Return Value
    *       int: number of bytes
    ----------------------------------------------------------------------------- */
    static int GetLsCmdLen(unsigned char *Data)
    {
        short Cmd;
        int Len;

        Len = 0;
        Cmd = (*(Data + 1) << 8) | *(Data + 2);
        switch(Cmd)
        {
            case CMD_CA :
            case CMD_EA :
            case CMD_FC :
                Len = 0 + 7;    /* Data field is fixed to 0 bytes */
                break;

            case CMD_BB :
            case CMD_IA :
            case CMD_PJ :
            case CMD_BD :
            case CMD_CB :
                Len = 1 + 7;    /* Data field is fixed to 1 byte */
                break;

            case CMD_FA :
                Len = 4 + 7;    /* Data field is fixed to 4 bytes */
                break;

            case CMD_DA :
                Len = 5 + 7;    /* Data field is fixed to 5 bytes */
                break;

            case CMD_JM :
            case CMD_AC :
                Len = (*(Data + 4) << 8) | *(Data + 3);
                break;
        }
        return(Len);
```

```
}


/*---------------------------------------------------------------------------
*     Function
*           Output to LS
*
*     Parameter
*           None
*
*     Return Value
*           None
------------------------------------------------------------------------ */
static void LsSerialOut(void)
{
    if(LsComm.FixedFlag[SERIAL_OUT])
    {
        SendToSerial(SERIAL_LS, (char*)LsComm.BinData[SERIAL_OUT],
LsComm.CurrentLen[SERIAL_OUT]);
        LsComm.FixedFlag[SERIAL_OUT] = FALSE;
    }
    return;
}
```

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

- **The information in this document is current as of May, 2002. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

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

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

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.

  (Note)

  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.

  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00. 4