ROBO BUILDER

International Patented Technology

wCK

S A M series

User Manual

Ver 1.0

SAM - 5

SAM - 2X

SAM - 210

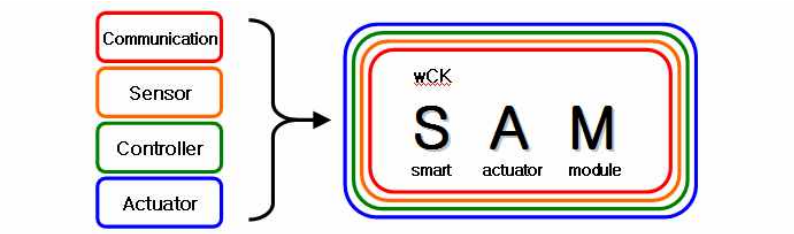
T I T L E

1. SAM Introduction

1-1. Main Features

- **All-in-One, Integrated Robot Module**

A robot module in which the controller, sensor, actuator, and input/output ports are integrated in one case

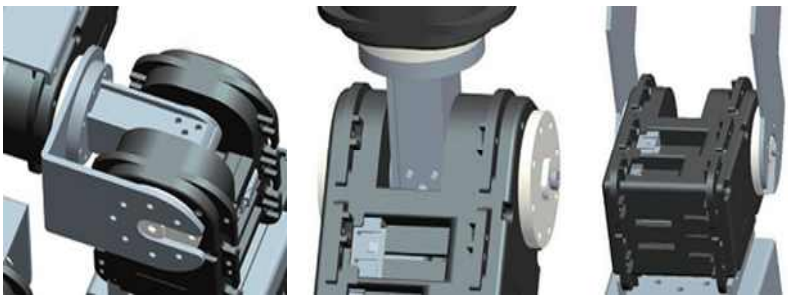


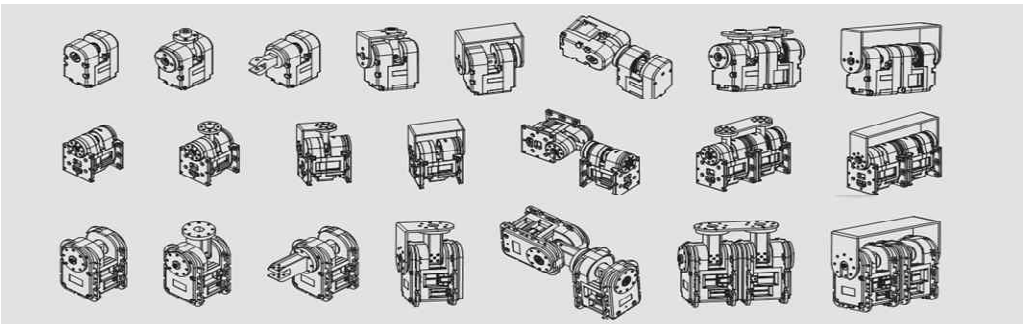
Compact Design

Compact, functional design enhances the efficiency of the robot design and aesthetic design

- **Double Shaft**

Supports **Double Shaft** to enable the designing of assembly structures of various types





• **Precise Control Feature**

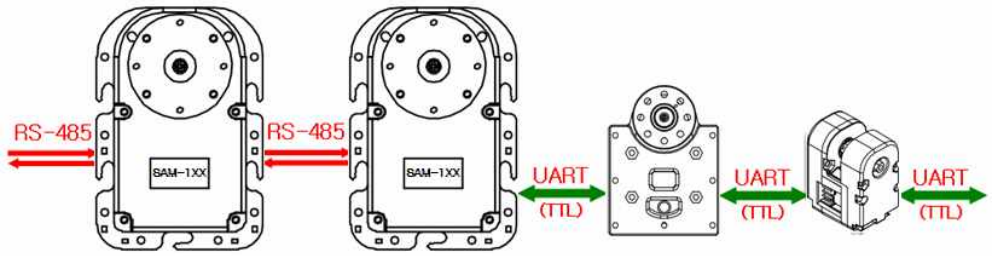
In the standard mode with the precise PID control method,
18 bit at the command of precise position control (control cone 0.001187˚, SAM-210EO200)
15 bit at the command of normal position control (control cone 0.01187˚, SAM-210EO200)

• **Supports Position Control, Speed Control, and Torque Control**

Supports position control, speed control, and torque control to enable the diverse utilization of the control system for each application.

• **Supports Multi-Drop Full Duplex RS-232(TTL Level) and RS-485 Serial Communication**

With the world's first realization of two-way serial communication at the TTL Level, it is possible to connect directly to the SAM from the external control device without the need for a separate communication device. In addition, SAM supports RS-485 serial communication that allows the control of a distant (maximum 1.2km) module, with wiring in daisy-chain configurations enabling the control of multiple SAM modules with a single bus.



• **Module ID and Communication Speed Reset**

This function allows the reset of the SAM ID and communication speed easily in case of problem happening.

• **Wide Position Control Motion Angle**

Supports a wide position control motion angle range (0˚-275˚ in Quick mode, 0˚- 340˚or 0˚- 358˚ in Standard mode)

• **Various Motion Modes**

Position control mode, 360˚ rotation mode, manual mode, break mode, self-running motion mode, and configuration mode

• **Various Parameter Settings**

Able to set various parameters such as module ID, baud rate, position control, PID position control gain, motion speed, adjustable speed, motion angle, and maximum operating current.

- **Support Sleep Brake Function**

When the brake mode is selected, the dynamic brake runs to enable the sleep function that maintains the assembled form without electricity consumption.

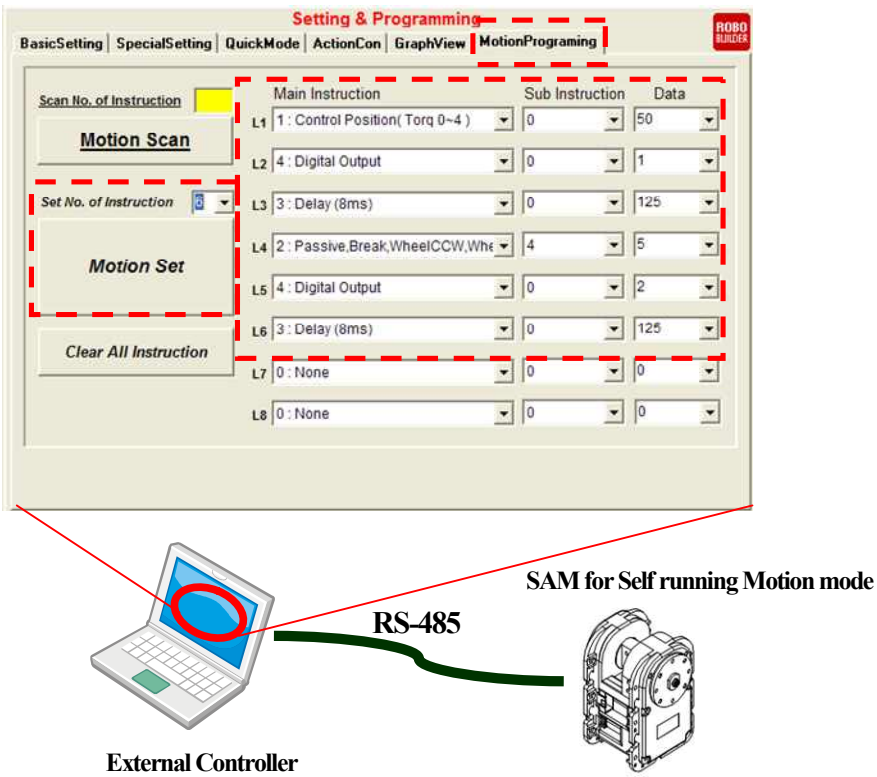
- **Support External I/ O Ports**

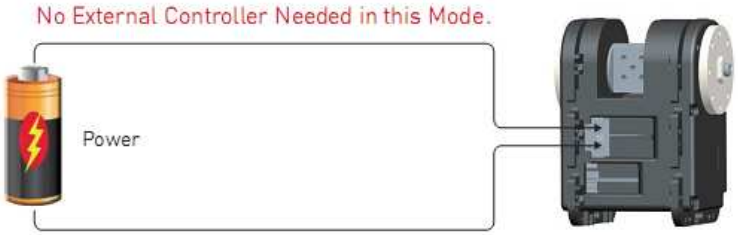
Offers Single-channel A/D input (0V-5.5V), two-channel digital output (3.3V level) to enable the operation of various applications

- **Self running Motion Mode**

Self-running motion programming using an external controller

Allows a SAM to function on its own without an external controller.





- **Protection against Inverse Voltage and Overload**

  Built in with protective circuits for inverse voltage and overload to protect the product

- **High-Resolution Load Feed Back**

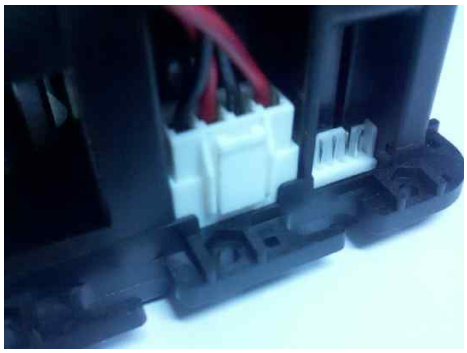
  Allows the monitoring of the load values with a 12-bit high resolution

- **32-bit Controller**

  High-performance 32-bit controller is applied to the internal controller, allowing better performance

- **Connector suitable for High Torque**

  Cable hook function and sufficient allowable current connector



1-2. Main Specification

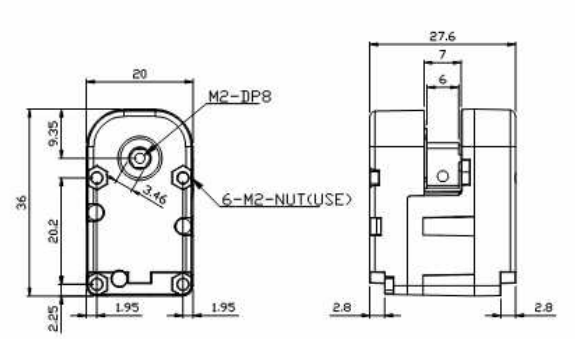
- Specification by Model

Model / Spec	SAM-5	SAM-20	SAM-28	SAM-160E0200	SAM-180E0200	SAM-210E0200
Max. Torque [Kgf.Cm]	5 @ 11V/1.5A	20 @ 19V/2A	28 @ 19V/2A	160 @ 24V/5A	180 @ 24V/5A	210 @ 24V/5A
Max. Speed [RPM]	130	140	90	63	50	42
Gear Material	Full Metal					
Control Resolution [degree]	1.08° (Quick Mode) / 0.083° (Standard Mode)			0.01241° / 0.00177°	0.01138° / 0.00142°	0.0118° / 0.00118°
Communication	UART Serial (Multi Drop TTL Full Duplex)			RS-485 Half Duplex / UART Serial (Multi Drop TTL Full Duplex)		
Operating Voltage [V]	6 ~ 15	9 ~ 24		14 ~ 24		
Position Sensor	Potentiometer [340°]			Optical Encoder [358°] + POT		
Size [mm]	20x36x27.6	38 x 51.64 x 45.9		57 x 82.5 x 62.45		

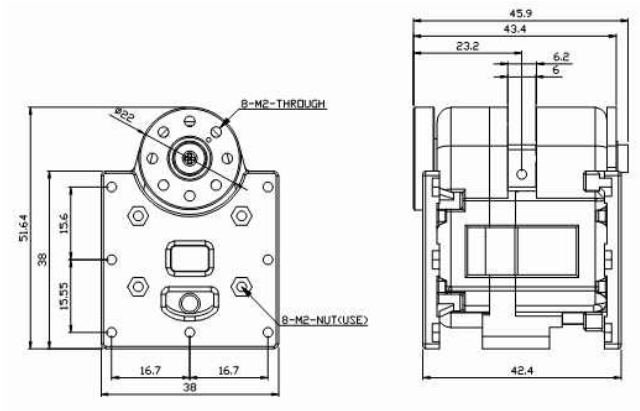
- Communication Mode — Multi drop mode Full Duplex RS232 (TTL) / Half Duplex RS485 serial communication
- Communication Speed — 4,800bps – 2,000,000bps (Select between 0-9, default 1.5 Mbps)
- Module Extension — Connect a maximum of 255 modules per channel (ID 0-254)
- Control Mode — Position control, speed control, torque control
- Case Material — Engineering Plastic

1-3. Dimension by Model

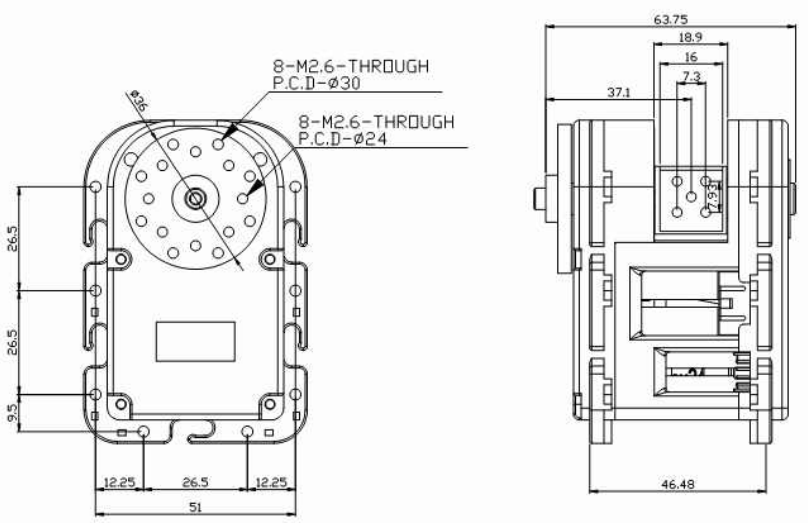
- SAM - 5



- SAM - 2X

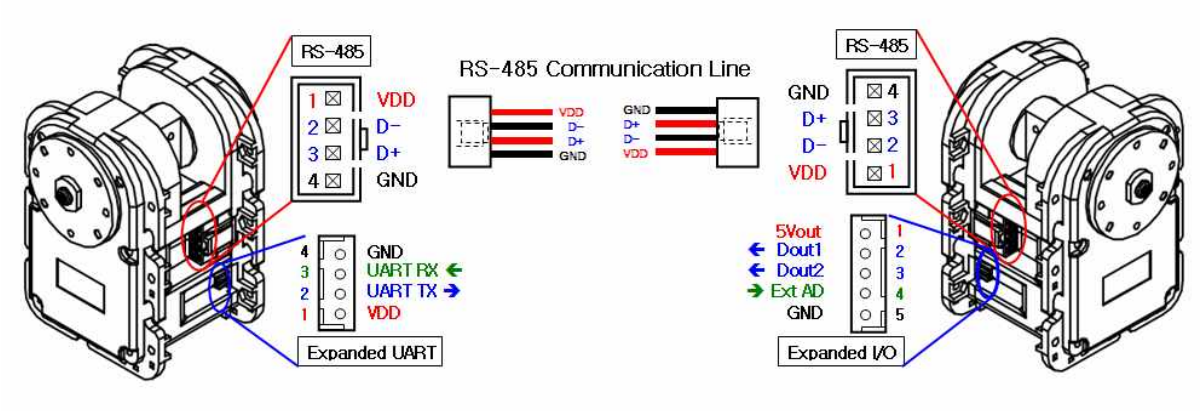


- SAM - 1XX

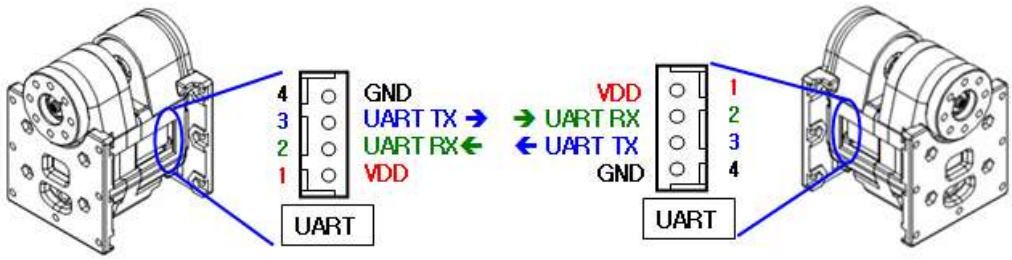


1-4. SAM Series Layout and Wiring

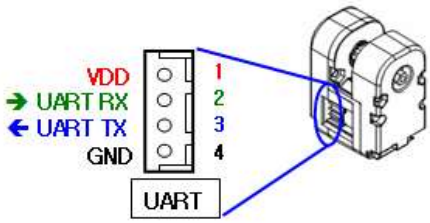
- SAM-xxx



- SAM-2x

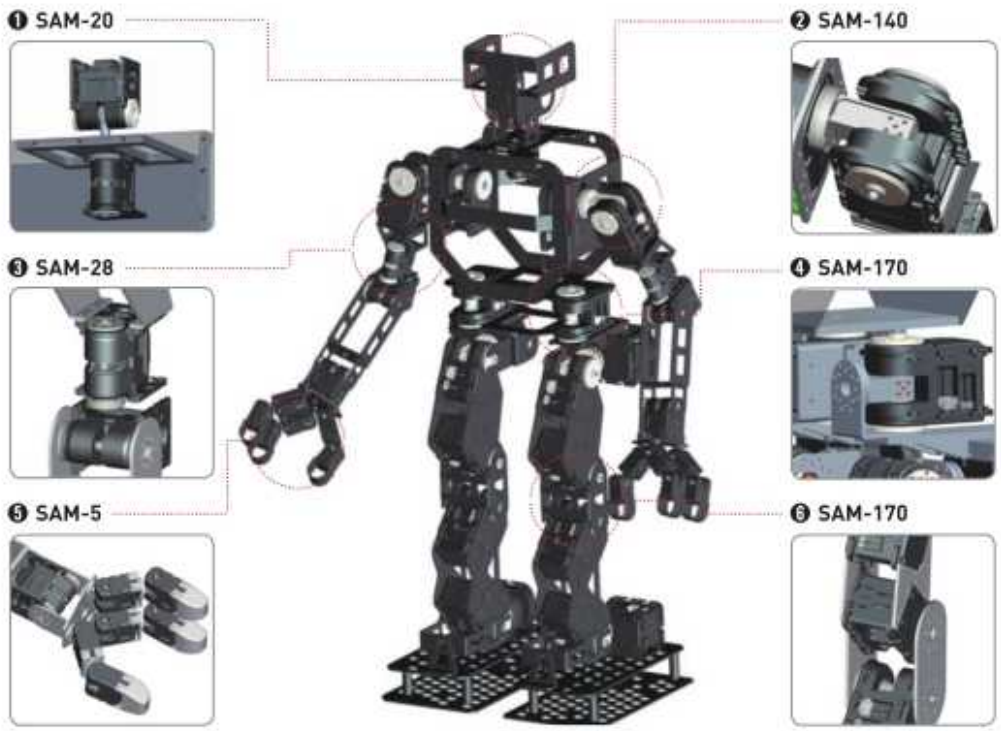


- SAM-5



1-5. SAM Series Utilization

The SAM series provides a wide range of line-ups by torque, precision, and size, allowing the selection of the various mixtures of SAMs when designing a humanoid or other user applications. For example, when designing the lower part of a humanoid that requires a large torque and precise control, the SAM-1xx series is desirable. The SAM-1xx series is perfect for designing parts that need a large torque and fast speed but not precise control, such as shoulders, while the SAM-2X series can be ideal for head and forearms, and the SAM-5 series is best for hands that will perform gripper functions. Therefore the SAM series is efficient in both energy use and design, and capable of swift and precise control of robots with its fast communication speed of a maximum of 2Mbps (1.5Mbps at normal).

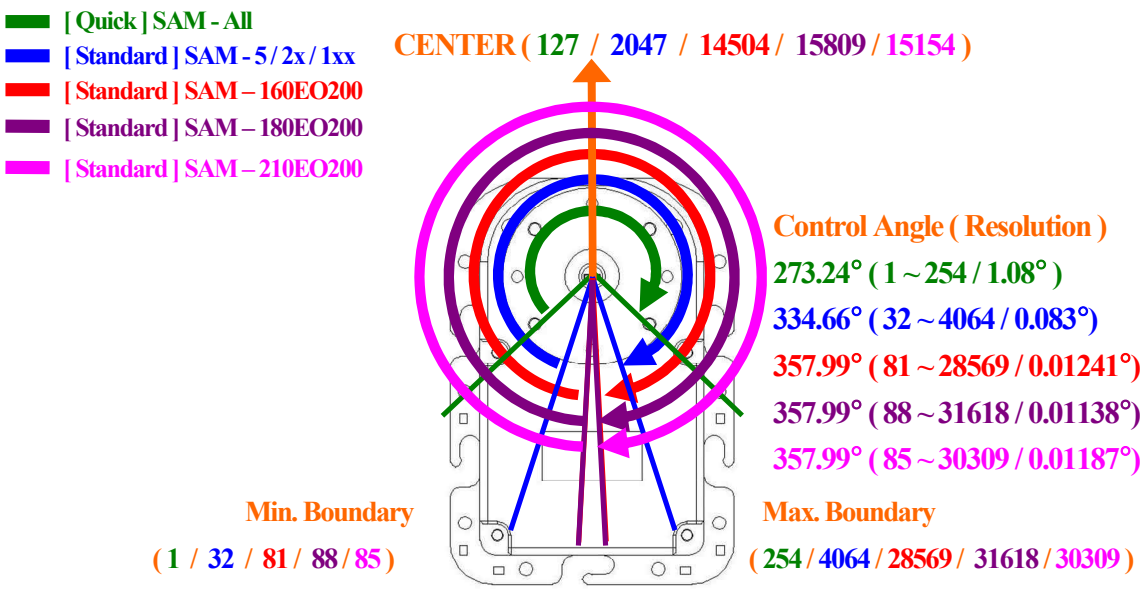


2. SAM Control

2-1. Control Features

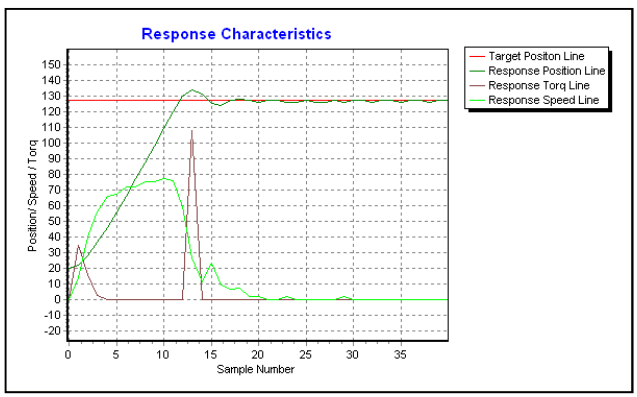
2-1-1. Position Control

The position control function of SAM is a basic function that allows the pivot of the module to move to the position directed by the user. The position of the physical absolute angle differs depending on the mode or model as shown in the picture below. There are two modes: Quick and Standard. Models are divided into SAM-xxxEO200 models and other SAM models depending on the type of the position sensor.

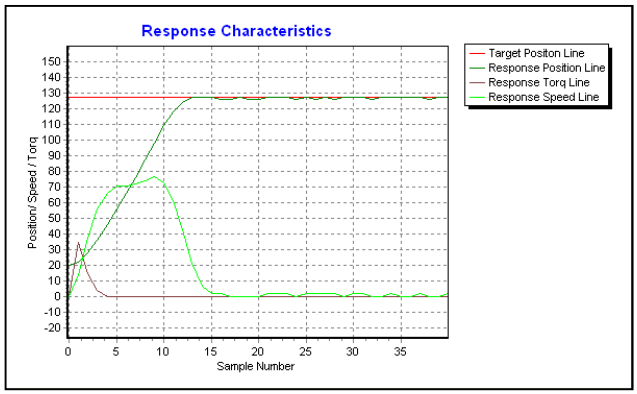


SAM implements precise position control through the PID control mode. The following graph shows the response characteristics for the P, PD, and PID control mode with the position control of the same SAM.

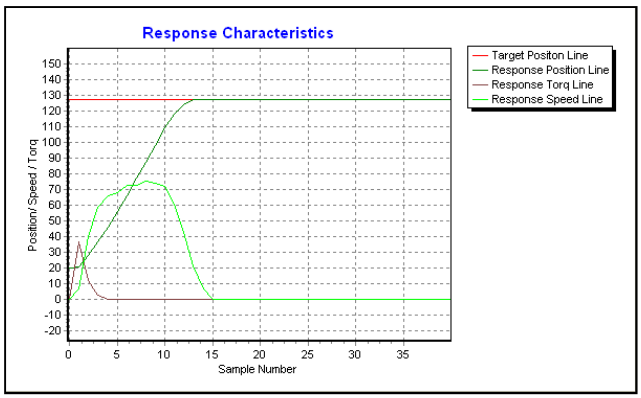
- P Control Mode



- PD Control Mode



- PID Control Mode



The control modes P, PD, and PID is set as the user sets the value with P, I, D gain, and the control characteristic differs depending on the value.

2-1-2. Passive Mode

If SAM is set to Passive Mode, it can perform the function of a rotational sensor by receiving the current position of the output axis with the delivery of external force without the application of force to the motor.

2-1-3. Boundary Set Mode

SAM allows the user to freely set the boundary of the motion angle, and this is useful for applications that require physical movements in a certain range.

2-1-4. Overload Limitation

SAM is designed to detect overload to protect its module as well as the user from excessive external force. The user can decide to prioritize either the performance or safety of an application by freely setting the range of values for overload limitation. If over-current flows for longer than 0.2 seconds above the level set by the user, SAM recognizes it as an overload and is automatically set to the Passive Mode. Its blue LED will be on for 0.3 seconds to inform the user of the change.
If a current over 1.8A flows for longer than 1 seconds, SAM recognizes it as overload and is automatically set to the Passive Mode for 5 seconds.

2-1-5. 360° Rotation

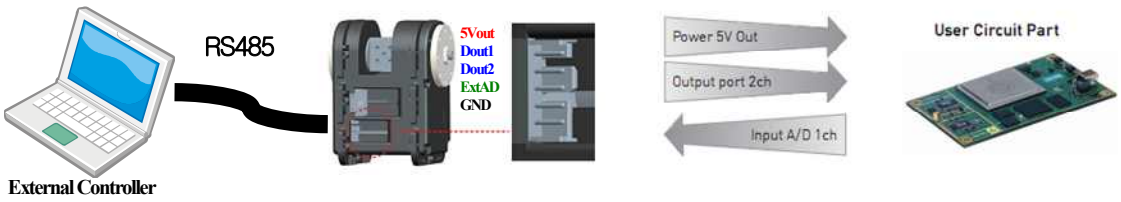
With SAM's 360° Rotation mode, the user can create an application that rotates infinitely at a constant speed set by the user. The user can also select the direction between clockwise or counterclockwise and among 1,000 speed levels per direction.

2-1-6. Brake Mode

SAM's Brake mode allows the stoppage of the module without electricity consumption with the help of the dynamic brake. All SAMs connected to the same control channel of the controller are set to the Brake mode simultaneously.

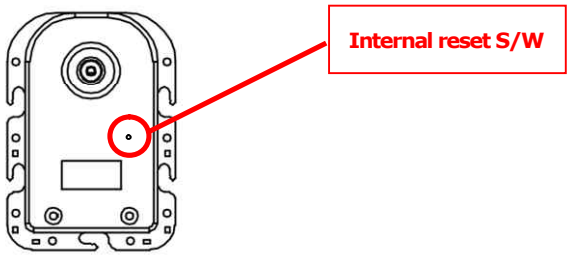
2-1-7. External Input / Output Ports

To support the development of creative applications by users, SAM is equipped with 1 A/D input (0V-5.5V) channel and 2 digital output channels (3.3V). By using the communication channel of the external controller, the user can control the input/ output ports of the SAM. (SAM-xxx models only)



2-1-8. Reset baud rate and ID

While controlling SAM, the need to reset the baud rate and SAM ID can arise due to errors or user's mistakes. The user can simply reset the baud rate and SAM ID by pressing the internal reset switch using a needle or pin until the red and blue LED blinks (approx. 5 seconds). (SAM-xxx models only) (Default ID = 0 / baud rate = 1.5Mbps)



2-1-9. Self Running Motion Mode

The user can implement self-running motion only with an applied power supply without a separate external controller and can program the desired self-running motion onto SAM itself.

2-1-10. SAM Model Table

		SAM-5	SAM-20/28	SAM-xxx
Position Control	Quick mode	◎	◎	◎
	Standard mode	◎	◎	◎
	Standard precise mode			●
Passive mode		◎	◎	◎
Boundary set		◎	◎	◎
Speed / Acceleration set		◎	◎	◎
Overload limitation		◎	◎	◎
360 ° rotation		◎	◎	◎
Brake mode		◎	◎	◎
External I/O Port				●
Baud rate, ID reset		◎	◎	◎
Response level set		◎	◎	◎
Self running motion		◎	◎	◎
LED monitoring				●
Zero position set				●

2-2. SAM Command List

Below is the summary list of protocol commands for SAM control.
Protocol commands can be broadly grouped into Quick commands and Standard commands. The Quick command has the advantage of reduced time for communication compared to the Standard command as it reduces the number of data values while communicating to SAM. But since only SAMs whose ID is 0-30 use the Quick command, it is necessary to use the Standard command if the user wants to control SAM whose ID exceeds 30 or to use all the functions of SAM.
For a detailed explanation for protocol commands, refer to Appendix [A-1 Communication Protocol].

※ There are a few commands using a flash memory to store data, which are marked with the [Flash] sign. As flash memory can be written about 10,000 times, remember that frequent use of commands with [Flash] sign can reduce the lifespan of the product.

2-2-1. Quick Mode Command List

2-2-1-1. SAM Quick Control Commands

(1) Position Control	Move the position of the output axis
(2) Load Status (Torque, Position)	Load the status of torque and position
(3) Passive Mode	Passive mode (sensor mode) command
(4) 360 ° Rotation Mode	360° Rotation Mode
(5) Brake Mode	Brake Mode (Sleep mode) command

2-2-1-2. SAM Quick Control Commands

(1) Set Baud rate [Flash]	Set the communication speed
(2) Set RunTime P, D Gain	Set Temporary P gain, D gain configuration
(3) Set 5-bit ID [Flash]	Set SAM ID configuration
(4) Set OverLoad [Flash]	Set overload limitation
(5) Load OverLoad	Load the limitation of overload
(6) Set Boundary [Flash]	Set boundary of SAM's movements
(7) Load Boundary	Load boundary of SAM's movements

2-2-1-3. SAM Quick Extension Commands

(1) Write Extension Port	Write external extension port data
(2) Load Extension Port	Load external extension port data
(3) Write Motion Data [Flash]	Write self-running motion data
(4) Load Motion Data	Load the number of self-running data

2-2-2. Standard Mode Command List

2-2-2-1. SAM Standard Configuration Commands

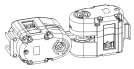
(1) Set ID [Flash]	Sets SAM ID
(2) Set Baud rate [Flash]	Sets Baud rate
(3) Load Current Speed	Loads the current Speed value
(4) Load Current Accel	Loads the current Accel value
(5) Load Current Load	Loads the Load value currently on the output axis
(6) Load Current Position	Loads the position of the current output axis
(7) Set P, D Gain [Flash]	Sets P and D gain values
(8) Load P, D Gain	Loads the set P and D gain values
(9) Set RunTime P, D Gain	Temporarily sets P and D gains
(10) Set OverLoad [Flash]	Sets overload
(11) Load OverLoad	Loads overload
(12) Set Boundary [Flash]	Sets SAM's boundary
(13) Load Boundary	Loads SAM's boundary
(14) Set I Gain [Flash]	Sets I gain value
(15) Load I Gain	Loads I gain value
(16) Set RunTime I Gain	Temporarily sets I gain value
(17) Set Drive Mode [Flash]	Sets the response level and reverse rotation on/off
(18) Load Drive Mode	Loads the set DriveMode
(19) Set Average Torque [Flash]	Sets the average torque to be used for position control
(20) Load Average Torque	Loads the average torque to be used for position control
(21) Load Input Voltage	Loads the SAM power supply voltage

2-2-2-2. SAM Standard Extension Commands

(1) Write Extension Port	Writes LED and external extension port data
(2) Load Extension Port	Loads external extension port AD data
(3) Write Motion Data [Flash]	Writes autonomous motion data
(4) Load Motion Data	Loads the number of autonomous motion data values
(5) PING	Checks the operation of the communication line
(6) Load Firmware Version	Loads the current SAM firmware version
(7) Load Overheat	Loads the overheat status

2-2-2-3. SAM Standard Control Commands

(1) Special Control	Sets Normal/ Passive/ Rotation/ Brake mode
(2) Position Control	Moves the position of the output axis
(3) Precise Position Control	Moves the precise position of the output axis
(4) Load Precise Position	Loads the precise position of the output axis



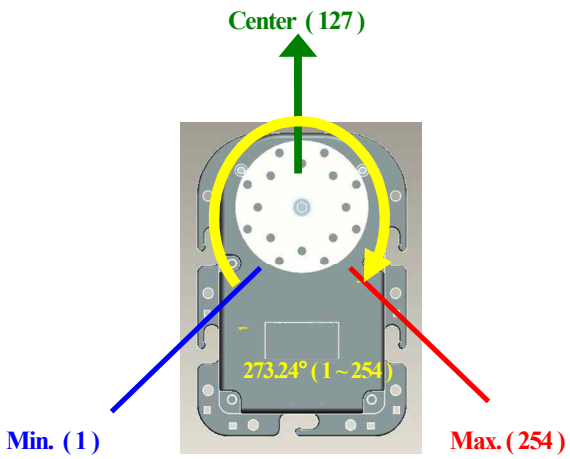
2-3. SAM Command and Response Processing

2-3-1. SAM Quick Control Command

(1) Position Control

The position control command is processed by sending the three units of information – torque, ID, and position value – in the form of 2-byte data. The torque value can be set anywhere between 0 and 4, with 0 the maximum torque and 4 the minimum torque for position control. IDs can be set from 0 to 30 and plays the role of designating the SAM that will perform the command. The position value divides the motion angle of this command, 273.24°, into 254 equal parts (1-254), and converts position value "1" into a 1.08° physical angle to perform position control. To position the output axis in the center, set the position value at "127." The response content (When response configuration is set) from SAM during the execution of the position control command is the average current AD value (8-bit resolution) during the 1ms immediately before the start of the position control and the current position value. The following formula applies to convert the average current AD value into current (A).

- SAM-5: current[A] = current AD value (8-Bit resolution) x 0.01289
- SAM-2x: current [A] = current AD value (8-Bit resolution) x 0.01289
- SAM-1xx: current[A] = current AD value (8-Bit resolution) x 0.016



(2) Load Position

Loads the average current AD value and position value of the motor during the 1ms of the SAM with the ID. The response content is the same as the "Position Control."

(3) Passive Mode

As SAMs are designed to maintain the current position on the position control command, the user cannot change the position of the output axis by force. While the user can change the position in the Brake mode, user must apply significant force to do so as the module is in the dynamic Brake mode. The Passive mode unlocks the torque of the SAM with the ID, allowing the user to easily change the position of the output axis.

(4) 360° Rotation Mode

The 360° mode is used when it is necessary to use SAM as a function of wheel rather than as a joint of a robot with a limited motion angle. The command of the Rotation mode should include the ID, the direction of the rotation (clockwise, counterclockwise), and rotational speed (0-15). The rotational speed can be set from 1 (PWM duty 15%) - 15 (PWM duty 50%) with 0 indicating stop.

(5) Brake Mode

The Brake mode puts SAM into the Brake mode. The SAM with the ID shuts off the power supply applied to the module and shorts both the ends of the module to enable the module to enter the dynamic Brake mode on its own. This mode is also called the sleep mode as the dynamic Brake can stop the movements of the module without power consumption.

2-3-2. SAM Quick Set Commands

(1) Set baud rate [Flash]

This command sets the UART or RS485 baud rate of SAM. The baud rate can be set in 10 units from 0 to 9. The minimum speed is 4800bps and the maximum is 2Mbps. The default value is 1.5Mbps (value : 1)

Value	Baud rate (bps)	1 byte Transmission Time
0	2M	5 us
1	1.5M	6.67 us
2	1M	10 us
3	500K	20 us
4	230.4K	43.4 us
5	115.2K	86.8 us
6	57600	173.6 us
7	38400	260.4 us
8	9600	1.04 ms
9	4800	2.08 ms

(2) Set RunTime P, D Gain

Sets the P and D gains required for PID control needed to control the position of the motor in SAM. Unlike the "Set P.D Gain" function, where the gains are saved in the ROM and are not deleted when the power is turned off and on, this function saves the gains in the RAM among the runtime, meaning faster save and access time, but the deletion of the set values if the power is turned off and on. The user can set Runtime P gain between 1 and 254, and D gain between 0 and 254.

(3) Set 5-bit ID [Flash]

Sets the ID between 0 and 30. This function allows quicker ID setting as this uses smaller data than the Set ID function in the Standard mode, where the available range is between 0 and 254,

(4) Set OverLoad [Flash]

Sets the limit of overload for module currents. The user can choose whether to prioritize performance or safety as user can set the range of overload limit values. If a module current larger than the set overload value flows for longer than 0.2 seconds, the module enters the Passive mode for at least 0.3 seconds to protect the SAM from mechanical damage. The overload value in the Quick Configuration command is the AD value of the module current (8-bit resolution). To convert the value into the actual current value [A], the following formulas apply.

- SAM-5: Currents [A] to limit = OverLoad [Module current AD value (8-bit resolution)] x 0.01289
- SAM-2x: Currents [A] to limit = OverLoad [Module current AD value (8-bit resolution)] x 0.01289
- SAM-xxx: Currents [A] to limit = OverLoad [Module current AD value (8-bit resolution)] x 0.016

5) Load OverLoad

Loads the overload value of the set 8-bit resolution.

(6) Sets Boundary [Flash]

Sets the Boundary of SAMs with an ID between 0-30. The user can prevent SAM from moving beyond the mechanical motion angle of the application by dividing the control angle in the Quick Control command, 273.24°, into a total of 254 sections between 1 and 254 and setting the floor and ceiling values. When setting the Boundary, the ceiling value must be higher than the floor value. For example, to set the movement range only within 90° left and right of the center of the output axis, the following example applies.

1. -90° / 1.08° = -83 / +90° / 1.08° = +83 / Central value of the output axis = 127
2. Floor (127 – 83) ≤ Control range of the output axis ≤ Ceiling (127 + 83)
3. Floor value = 44, Ceiling value = 210

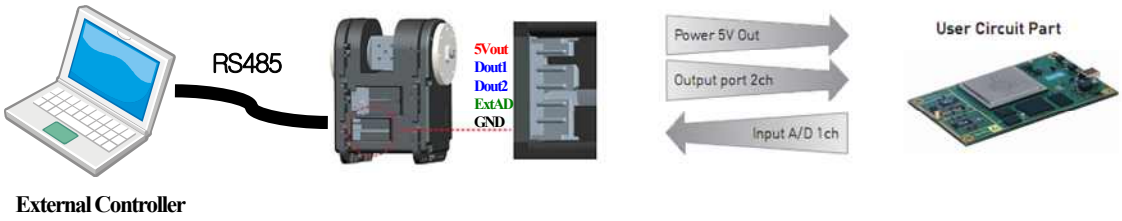
(7) Load Movement Range

Loads the floor and ceiling values set in the SAM with the ID. The range is between 1 and 254.

2-3-3. SAM Quick Set Extension Commands

(1) Write Extension Port

The user can write digital output data in the external extension D-out port of the SAM with an ID between 0 and 30 using this function. Only the SAM-1xx series support this function. The extension D-out port can be used for LEDs, Buzzers, or other external signals as the user needs. The power output level is 5V (Max 150mA), and the data output level is 3.3V. The Write Extension Port function can be controlled by an external controller and accessed by self-running motion.



(2) Load Extension Port (SAM-xxx model only)

Loads the AD value of the signal connected to the external AD pin by using the external AD function in the extension port of SAM with an ID between 0 and 30. Voltages between 0 and 5.5V can be input as an external signal, which then be divided by 3/5 and read in the form of data between 0 and 4095. The following formula applies to convert the signal input in the external AD pin into voltage [V]

Voltage of the external signal [V] = Loaded extension port value [External AD value (12-bit resolution)] x 0.001343

Load Extension Port function can be controlled by an external controller and accessed by self-running motion as well.

(3) Write Motion DATA [Flash]

Implementation of self-running motion with the applied power supply is possible without a separate external controller by utilizing the self-running motion function using SAM's Write Motion Data function. The logic of programming self-running motion in SAM Programmer is as follows. The logic is disabled when the number of motions is 0 and enabled when the number is 1 or more. For relevance with the protocol command packets, refer to the Appendix [A-1-3 Extension commands (24) Write Motion Data].

Protocol Command Packet	Motion Command(1 Byte)		Motion Data(1 Byte)
Main Instruction	Instruction No.	Sub Instruction	Data
None	0	0	X
Position Control	1	Speed(0~4)	8 bit position value

Special Motion Mode	2	1(Passive), 2(Power Down), 3(Wheel CCW), 4(Wheel CW)	Speed value in Wheel Mode (0~15)
Delay (Max. 4095 ms)	3	Upper 4bit Delay value	Lower 8bit Delay value (ms)
DIO	4	X	2 bit extension Output control value
Position Condition Standby	5	1("=") 2(">") 3("<") 4(">=") 5("<=")	8 bit position value
AD Condition Standby	6	1("=") 2(">") 3("<") 4(">=") 5("<=")	8 bit extension AD value
Repeat Counts	7	X	0 : infinite, 1~254 : counts
End	8	X	X

(4) Load DATA Data

Load the numbers of self-running motion data in SAM.

2-3-4. SAM Standard Commands

(1) Set ID [Flash]

The user can set a total of 255 SAM IDs (0-254). This function necessitates more data than that needed for "5-bit ID configuration" in the Quick command (0-30) and be used to set the SAM ID above 31.

(2) Set Baud rate [Flash]

Same as the Set Baud rate command in the Quick Configuration command.

(3) Load Current Speed

Loads the changed amount in position in the resolution higher than 12 bit over the past 10ms (8ms for SAM-170Ex) as the current speed value. The following formulas describe how to convert the current speed values into actual physical angular velocity.

- SAM-160EO200 current angular velocity [deg/s] = (0.01241° x changed amount in position / 7) [deg] / 0.008 [s]
  = Current Speed (changed amount in position) x 0.2216

- SAM-180EO200 current angular velocity [deg/s] = (0.01138° x changed amount in position / 8) [deg] / 0.008 [s]
  = Current Speed (changed amount in position) x 0.1778

- SAM-210EO200 current angular velocity [deg/s] = (0.01187° x changed amount in position / 10) [deg] / 0.008 [s]
  = Current Speed (changed amount in position) x 0.1483

- Angular velocity for models other than SAM-xxx [deg/s] = (0.083° x changed amount in position) [deg] / 0.01 [s]
  = Current Speed (changed amount in position) x 8.3

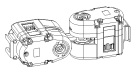
(4) Load Current Accel

Loads the changed amount in speed (pace of the change in amount in position) in the resolution higher than 12 bit over the past 1ms as the current speed value. The following formulas describe how to convert the current Accel values into actual physical angular acceleration.

- SAM-160EO200 current angular acceleration [deg/s^2] = Changed amount in angular velocity [deg/s] / 0.001 [s]
  = Current Accel (changed amount in speed) x0.2216 / 0.001
  = Current Accel (changed amount in speed) x 221.6

- SAM-180EO200 current angular acceleration [deg/s^2] = Changed amount in angular velocity [deg/s] / 0.001 [s]
  = Current Accel (changed amount in speed) x0.1778 / 0.001
  = Current Accel (changed amount in speed) x 177.8

- SAM-160EO200 current angular acceleration [deg/s^2] = Changed amount in angular velocity [deg/s] / 0.001 [s]
  = Current Accel (changed amount in speed) x0.1483/ 0.001
  = Current Accel (changed amount in speed) x 148.3



- Current angular acceleration for models other than SAM-xxx [deg/s^2]
- = Changed amount in angular velocity [deg/s] / 0.001 [s]
  = Current Accel (changed amount in speed) x 8.3 / 0.001
  = Current Accel (Changed amount in speed) x 8300

(5) Load Current Load

Loads the load value currently on the output axis. The load value is the average current AD value of the motor over the past 1ms (12-bit resolution). The following formula describes how to convert the average current AD value into current (A).

- SAM-5: Current [A] = present load value [motor current AD value (12-bit resolution)] x 0.00081
- SAM-2x: present load value [motor current AD value (12-bit resolution)] x 0.00081
- SAM-xxx: Current [A] = present load value [motor current AD value (12-bit resolution)] / 1000

(6) Load Current Position

- SAM -160EO0200 model

The current position value indicates the physical angle of 0.01241° for the position value "1" when the motion angle in the Standard command, 357.99° is divided by 28846 (82-28927). When the output axis is positioned in the center, "14504" will be loaded as current position value.

- SAM -180EO200 model

The current position value indicates the physical angle of 0.01138° for the position value "1" when the motion angle in the Standard command, 357.99° is divided by 31442 (89-31530). When the output axis is positioned in the center, "15809" will be loaded as current position value.

- SAM -210EO200 model

The current position value indicates the physical angle of 0.01187° for the position value "1" when the motion angle in the Standard command, 357.99° is divided by 30139 (86-30224). When the output axis is positioned in the center, "15154" will be loaded as current position value.

- Other SAM series

For SAM series other than 170IE, the current position value indicates the physical angle of 0.083° for the position value "1" when the motion angle in the Standard command, 340° is divided by 4096 (0-4095). When the output axis is positioned in the center, "2047" will be loaded as current position value.

(7) Set P, D Gain [Flash]

Sets the P and D gains required for PID control needed to control the position of the motor in SAM. Unlike the "Set Runtime P.D Gain" function, where the gains are saved in the RAM and are deleted when the power is turned off and on, this function saves the gains in the flash area where the gains are maintained after the power is turned off and on.

(8) Load P, D Gain

Loads the currently set P and D gains, i.e. the P and D gains saved in the RAM field currently used for PID control. If the user applies power to SAM and the "Set P, D Gain" or "Set Runtime P, D Gain," this will load the latest updated P and D gains.

(9) Set RunTime P, D Gain

Sets the P and D gains required for PID control needed to control the position of the motor in SAM. Unlike the "Set P.D Gain" function, where the gains are saved in the ROM and are not deleted when the power is turned off and on, this function saves the gains in the RAM among the runtime values, indicating faster memory save and access time and the deletion of set gains when the power is turned off and on.

(10) Set Overload [Flash]

Sets the limit for overload of motor currents. The user can choose whether to prioritize performance or safety as he or she can set the range of overload limit values. If a motor current larger than the set overload value flows for longer than 0.2 seconds, the motor enters the Manual mode for at least 0.3 seconds to protect SAM from mechanical damage. The overload value in the Standard Configuration command is the AD value of the motor current (12-bit resolution). To convert the value into the actual current value [A], the following formulas apply.

- SAM-5: Currents [A] to limit = OverLoad [Motor current AD value (12-bit resolution)] x 0.00081
- SAM-2x: Currents [A] to limit = OverLoad [Motor current AD value (12-bit resolution)] x 0.00081
- SAM-xxx: Currents [A] to limit = OverLoad [Motor current AD value (12-bit resolution)] x 0.001

(11) Load Overload

Loads the set overload value of the 12-bit resolution.

(12) Sets Boundary [Flash]

- SAM – xxx Model

Sets the movement range of SAMs with an ID between 0-254. The user can prevent SAM from moving beyond the mechanical motion angle of the application by dividing the control angle in the Standard Control Command, 358° , into a total of 254 sections between 1 and 254 (1.41° per section) and setting the floor and ceiling values. When setting the movement range, the ceiling value must be higher than the floor value. For example, to set the movement range only within 90° left and right of the center of the output axis, the following example applies.

1. -90° / 1.41° = -64 / +90° / 1.41° = +64 / Central value of the output axis = 127
2. Floor (127 – 64) ≤ Control range of the output axis ≤ Ceiling (127 + 64)
3. Floor value = 63, Ceiling value = 191

- Other SAM series

Sets the movement range of SAMs other than the 170Ex. The user can prevent SAM from moving beyond the mechanical motion angle of the application by dividing the control angle in the Standard Control Command, 334.66° , into a total of 254 sections between 1 and 254 (1.32° per section) and setting the floor and ceiling values. When setting the movement range, the ceiling value must be higher than the floor value. For example, to set the movement range only within 90° left and right of the center of the output axis, the following example applies.

1. -90° / 1.32° = -68 / +90° / 1.32° = +68 / Central value of the output axis = 127
2. Floor (127 – 68) ≤ Control range of the output axis ≤ Ceiling (127 + 68)
3. Floor value = 59, Ceiling value = 195

(13) Loads Boundary

Loads the set Boundary values. The range can be set between 1-254 for both the floor and ceiling alike.

(14) Set I Gain [Flash]

Sets I gains required for the PID control needed to control the position of the motor in SAM. Unlike the "Set Runtime I Gain" function, where gains are saved and accessed quicker but deleted once the power is turned off and on as they are saved in the RAM among runtime, this function saves the gains in the flash area, where the gains are maintained after the power is turned off and on.

(15) Load I Gain

Loads the currently set I gain values from the RAM area used for the PID control. If the user has applied the power supply to SAM and enabled the "Set I Gain" or "Set Runtime I Gain," this command will load the latest updated I gain values.

(16) Set RunTime I Gain

Sets I gains required for the PID control needed to control the position of the motor in SAM. Unlike the "Set I Gain" function, where gains are saved and accessed quicker and not deleted after the power is turned off and on as they are saved in the flash area, this function saves the gains in the RAM among the runtime values, where the gains are saved and accessed quicker but deleted once the power is turned off and on.

(17) Set Drive Mode [Flash]

Sets SAM's response level and reverse rotation functions.

- Set Response Level

  The default value for SAM's response level is "0," at which SAM responds only to load commands to minimize the communication load. At level "1," SAM responds to all commands, and at "2," SAM performs but do not responds to all commands.

- Reverse mode (SAM-340 model)

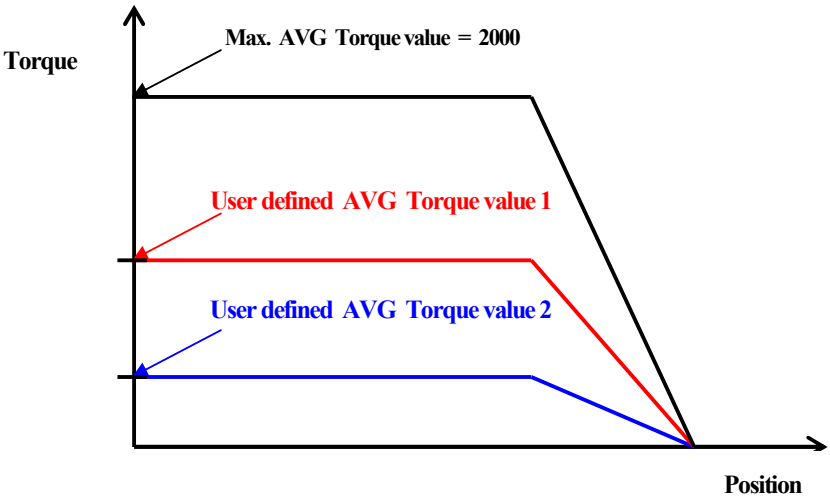
  If the spin direction of the output axis is different, the user can reverse the direction by using the Reverse mode.

(18) Load Drive Mode

Loads the Drive mode set in the SAM with the ID.

(19) Set Average Torque [Flash]

Sets the average torque value between 0 and 2,000 to be used to control the position in the Standard Control command. If the average torque value is set to "0," the torque does not move. If the value is set to "2, 000," the torque performs position control at its full capacity. This is used for a more precise control as the resolution is 400 times that of the 5-level torque control in the Quick Position Control command.



(20) Loads Average Torque

Loads the average torque value to be used to control the position in the Standard command.

(21) Load Input Voltage

Used to load the power supply voltage applied to SAM. The power supply voltage for each SAM series product must be operated in accordance with the maximum input allowable voltage. The following formulas apply to convert the input voltage into actually applied voltage [V].

- SAM-5: Applied power voltage [V] = {Input Voltage AD value (12-bit resolution) x 0.00483 } + 0.65
- SAM-2x: Applied power voltage [V] = {Input Voltage AD value (12-bit resolution) x 0.00886 } + 0.65
- SAM-1xx: Applied power voltage [V] = {Input Voltage AD value (12-bit resolution) x 0.00886 } + 0.65

2-3-5. SAM Standard Extension Commands

(1) Write Extension Port (SAM-xxx model only)

(2) Load Extension Port AD (SAM-xxx model only)

(3) Write Motion Data

(4) Load Motion Data

The same command as the SAM Quick Extension command, except that this controls a total of 255 SAMs with IDs between 0 and 254.

(5) PING

This is a command to check the status of communication with a connected SAM. The user can check whether the communication is functioning well or which SAM is connected. The user is advised to connect only one SAM to check the status of communication as, if several SAMs are connected, using this command can cause collisions on the communication line when the SAMs respond.

(6) Load Firmware Version

Load the model of the SAM with the ID between 0 and 254 as well as the current firmware version. The MSByte refers to the SAM model, whereas the LSByte means the currently applied firmware version.

Response MSByte	Response LSByte	SAM Model / FW Version
0x05	Xy	SAM-5 / Version Xy
0x20	Xy	SAM-20 / Version Xy
0x28	Xy	SAM-28 / Version Xy
0x16	Xy	SAM-160EO200 / Version Xy
0x18	Xy	SAM-180EO200 / Version Xy
0x21	Xy	SAM-210EO200 / Version Xy

(7) Load Overheat status

If the internal temperature of SAM is 65 ℃ or higher, SAM position control function is stopped.
In this case, user can check the status of SAM temperature. By using Temperature Error protocol, "RED LED" is blinking as user can see visually.

2-3-6. SAM Standard Control Commands

(1) Special Control

Sets the SAM with the ID between 0 and 254 to Normal (position control)/ Manual/ Break/ Rotation mode. If the SAM is set to the Rotation mode, the rotational speed value must be also given. The rotational speed is a value increasing by PWM duty 0.1% for the value "1" and can be set between 0 and 2,000. The direction of the rotational speed is decided depending on the scope.

0-999 → CW/ 1000-2000 → CCW/ Stops rotation at 0 or 1,000

(For CCW rotation, (rotational speed value – 1,000) is computed as the actual rotational speed)

(2) Position Control

The Standard position control command is processed by sending two units of information –ID and position value – in the form of 3-byte data. This does not individually set the torque value in contrast with the Quick position control command. When controlling the position, this will refer to the "AVG Torque "and "CTRL Torque" values set in the Standard Configuration command. IDs can be controlled between 0 and 254, and the position value has a resolution higher than 12-bit depending on the model, allowing a more precise control that the Quick command.

- SAM-160EO0200 model

  The position value in SAM Standard mode performs position control by dividing the control angle of 358° by 28846 (82-28927) and converting it into a physical angle of 0.01241° for the position value "1." To set the position of the output axis in the center, enter"14504" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control starts in the range of 82 to 28927.

- SAM-180EO200 model

  The position value in SAM Standard mode performs position control by dividing the control angle of 358° by 31442 (89-31530) and converting it into a physical angle of 0.01138° for the position value "1." To set the position of the output axis in the center, enter "15809" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control starts in the range of 89 to 31530.

- SAM-210EO200 model

  The position value in SAM Standard mode performs position control by dividing the control angle of 358° by 30139 (86-30224) and converting it into a physical angle of 0.01187° for the position value "1." To set the position of the output axis in the center, enter "15154" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control starts in the range of 86 to 30224.

- Other SAM series

The position value in SAM Standard mode performs position control by dividing the control angle of 334.66° by 4033 and converting it into a physical angle of 0.0083° for the position value "1." To set the position of the output axis in the center, enter "2047" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control starts and is of 12-bit resolution in the range of 0 to 4095.

The position control characteristics for Quick/Standard commands by SAM model is as follows.

		Quick Position Control	Standard Position Control	Precise Position Control
Command Data Transmission Rating		4 Byte (response 2 byte)	7 Byte (response 2 byte)	8 Byte (response 3 byte)
Valid Data Content		ID(0-30) / Torq / position value (8 bit)	ID (0-254) / position value (12 bit or over)	ID (0-254) /position value (18 bit or over)
Response Content		current (8 bit) / position (8 bit)	position (12 bit or over)	position (18 bit or over)
Control Angle (Resolution)	SAM-5/2x	273.24° (1 - 254)	334.66° (32 - 4064)	–
	SAM–160EO200		357.99° (82 - 28927)	357.99° (566-202494)
	SAM–180EO200		357.99° (89- 31530)	357.99° (704-252248)
	SAM–210EO200		357.99° (86- 30224)	357.99° (843-302251)
Physical Position value (per 1 control unit)	SAM-5/2x	1.08°	0.083°	–
	SAM–160EO200		0.01241°	0.001773°
	SAM–180EO200		0.01138°	0.001423°
	SAM–210EO200		0.01187°	0.001187°

(3) Precise Position Control

When precise position control command is necessary, it is possible to control the position by the unit of 0.001773°, 0.001423° or 0.001187° for SAM-xxxEO200 series. This command consists of 8-byte command and 3-byte protocol.

- **SAM-160EO200 Model**

The position value in SAM Standard mode performs position control by dividing the control angle of 359.99° by 203059 (0-203059) and converting it into a physical angle of 0.001778° for the position value "1." To set the position of the output axis in the center, enter"101529" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control.

The actual control range is between 566 and 202494, 1°- 359° due to boundary limits.

- **SAM-180EO200 Model**

The position value in SAM Standard mode performs position control by dividing the control angle of 359.99° by 252951 (0-252951) and converting it into a physical angle of 0.001423° for the position value "1." To set the position of the output axis in the center, enter"126475" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control. The actual control range is between 704 and 252248, 1°- 359° due to boundary limits.

- **SAM-210EO200 Model**

  The position value in SAM Standard mode performs position control by dividing the control angle of 359.99° by 303093 (0-303093) and converting it into a physical angle of 0.001187° for the position value "1." To set the position of the output axis in the center, enter"151546" for the position value. The response content from the SAM when executing the position control command is the position value right before the position control. The actual control range is between 843 and 302251, 1°- 359° due to boundary limits.

(5) Load Precise Position Control

- SAM-160EO200 model

  The current position value indicates the physical angle of 0.001772° for the position value "1" when the motion angle in the Standard Precise Position Control command, 359.99° is divided by 203059 (0-203059). When the output axis is positioned in the center, "101529" will be loaded as current position value.

- SAM-180EO200 model

  The current position value indicates the physical angle of 0.001423° for the position value "1" when the motion angle in the Standard Precise Position Control command, 359.99° is divided by 252951 (0-252951). When the output axis is positioned in the center, "126475" will be loaded as current position value.

- SAM-210EO200 model

  The current position value indicates the physical angle of 0.001187° for the position value "1" when the motion angle in the Standard Precise Position Control command, 359.99° is divided by 303093 (0-303093). When the output axis is positioned in the center, "151546" will be loaded as current position value.

A. Appendix

A-1 Communication Protocol

A-1-1 SAM Quick Control Commands

(1) Position Control

Makes a SAM send information to the external controller on the current load and the current position of the output terminal and move to a designated position.

▶ Command Packet

1byte	1byte	1byte	1byte
Header	Data1	Target position	Checksum

Header = 0xFF

Data1

Torque			ID				
7	6	5	4	3	2	1	0

※ Torq : 0(Max.)~4(Min.), ID : 0~30

Target position = 0~254

Checksum = (Data1 X OR Target position) AND 0x7F

▶ Response Packet

1byte	1byte
Load	Position

※ Load: 0~254, Position : 0~254

Example) ID : 0, Torque : 0, Target Position : 127 ((0x7F)

- Command Packet

1byte	1byte	1byte	1byte
0xFF	0x00	0x7F	0x7F

- Response Packet

1byte	1byte
Load	Position

(2) Read Status (Torque, Position)

Read current SAM Load and position.

▶ Command Packet

1byte	1byte	1byte	1byte
Header	Data1	Data2	Checksum

Header = 0xFF

Data1

Mode(=5)			ID				
7	6	5	4	3	2	1	0

※ Mode: 5, ID : 0~30

Data2 = Default value

Checksum = (Data1 XOR Data2) AND 0x7F

▶ Response Packet

1byte	1byte
Load	Position

※ Load : 0~254, Position : 0~254

Example) Read SAM ID 0 status.

- Command Packet

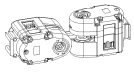
1byte	1byte	1byte	1byte
0xFF	0xA0	0x00	0x20

- Response Packet

1byte	1byte
Load	Position

(3) Passive Mode

Notice the current SAM position and is released the torque. It is moved by external force.



▶ Command Packet

1byte	1byte	1byte	1byte
Header	Data1	Data2	Checksum

Header = 0xFF

Data1

Mode(=6)			ID				
7	6	5	4	3	2	1	0

※ Mode : 6, ID : 0~30

Data2

Mode(=1)			Default value				
7	6	5	4	3	2	1	0

※ Mode : 1

Checksum = (Data1 XOR Data2) AND 0x7F

▶ Response Packet

1byte	1byte
ID	ID

※ ID : 0~30

Example) Enter into Passive mode of ID 0

- Command Packet

1byte	1byte	1byte	1byte
0xFF	0xC0	0x10	0x50

- Response Packet

1byte	1byte
0x00	0x00

(4) Wheel Mode (360° rotation)

Enter into Wheel mode. Speed is divided into 16 levels.

▶ Command Packet

1byte	1byte	1byte	1byte
Header	Data1	Data2	Checksum

Header = 0xFF

Data1

Mode(=6)			ID				
7	6	5	4	3	2	1	0

※ Mode: 6, ID : 0~30

Data2

Wheel Direction				Speed			
7	6	5	4	3	2	1	0

※ Wheel Direction : 3 (CCW), 4(CW)

※ Speed : 0 (Stop), 1(Min.)~15(Max.)

Checksum = (Data1 XOR Data2) AND 0x7F

▶ Response Packet

1byte	1byte
Rotation No.	Position

※ Rotation No. : 0~255

※ Position = 0~254

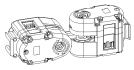
Example) ID 0, Wheel mode (CCW, Speed : 15)

- Command Packet

1byte	1byte	1byte	1byte
0xFF	0xC0	0x3F	0x7F

- Response Packet

1byte	1byte
Rotation No.	Position



(5) Brake Mode

Enter into Brake mode. It is not moved well because power is blocked-status.

▶ Command Packet

1byte	1byte	1byte	1byte
Header	Data1	Data2	Checksum

Header = 0xFF

Data1

Mode(=6)			ID				
7	6	5	4	3	2	1	0

※ Mode : 6

Data2

Mode(=2)				임의 값			
7	6	5	4	3	2	1	0

※ Mode : 2

Checksum = (Data1 XOR Data2) AND 0x7F

▶ Response Packet

1byte	1byte
ID	Position

※ ID : 0~30, Position : 0~254

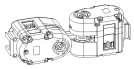
Example) ID 0, Enter into Brake mode.

- Command Packet

1byte	1byte	1byte	1byte
0xFF	0xDF	0x20	0x7F

- Response Packet

1byte	1byte
0x00	Position



A-1-2 SAM Quick Control

(1) Set Baud Rate[Flash]

It sets baud rate of SAM. It supports4800 bps, 9600 bps, 38400 bps, 57600 bps, 115200 bps, 230400 bps, 500000 bps, 1Mbps, 1.5Mbps , 2Mbps.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x08

Data3 = 0(2Mbps) / 1(1.5Mbps) / 2(1Mbps) / 3(0.5Mbps) / 4(230400bps) / 5(115200bps) / 6(57600bps) / 7(38400bps) / 8(9600bps) / 9(4800bps)

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
new Baud rate	new Baud rate

※ new Baud rate : 0~9

Example) ID 0, set baud rate 115200bps

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x08	0x05	0x05	0x6D

- Response Packet

1byte	1byte
0x05	0x05

(2) Runtime P, D Gain Set

It sets SAM의 runtime P and D gain.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF (Packet starts.)

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x0B

Data3 = 1~254 (P-gain)

Data4 = 0~254 (D-gain)

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
P gain	D gain

※ P gain : 1~254, D gain : 0~254

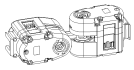
Example) ID 0, set P gain 100, D gain 100 in runtime.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x0B	0x64	0x64	0x69

- Response Packet

1byte	1byte
0x64	0x64



(3) 5 Bit ID [Flash]

Set 5 Bit ID

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x0C

Data3 = new ID(0~254)

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
new ID	new ID

※ new ID : 0~254

Example) Change ID from 0 to 30.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x0C	0x1E	0x1E	0x6C

- Response Packet

1byte	1byte
0x1E	0x1E

(4) Set OverLoad [Flash]

Set SAM overload. (If overload is happens, it enters into "passive mode")

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x0F

Data3 = OverLoad (0 ~ 254)

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
new overload	new overload

※ new overload : 0~254

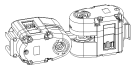
Example) Set ID 0, Overload 104.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x0F	0x68	0x68	0x6F

- Response Packet

1byte	1byte
0x68	0x68



(5) Read OverLoad

Read SAM overload.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x10

Data3 = Default overload value

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
Overload	Overload

※ Overload : 0~254

Example) Read ID 0 Overload.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x10	0x00	0x00	0x70

- Response Packet

1byte	1byte
Overload	Overload

(6) Set Boundary Range [Flash]

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x11

Data3 = new Upper Boundary(2~254)

Data4 = new Lower Boundary(1~253)

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
New Upper Boundary	New Lower Boundary

※ new Upper Boundary : 0~254, new Lower Boundary : 0~254

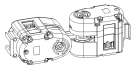
Example) ID 0, Upper Boundary = 100, Lower Boundary = 50

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x11	0x64	0x32	0x20

- Response Packet

1byte	1byte
0x64	0x32



(7) Read Boundary Range

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x12

Data3 = default value

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
Upper Bound	Lower Bound

※ new Upper Boundary : 0~254, new Lower Boundary : 0~254

Example) Read ID 0 Boundary Range.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x12	0x00	0x00	0x72

- Response Packet

1byte	1byte
Upper Boundary	Lower Boundary

A-1-3 SAM Quick Extension Command

(1) Write External Port

It is to set up SAM D/O(Digital Output) value.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x64

Data3

default value				Ch		LED	
7	6	5	4	3	2	1	0

※

bit 0 : D/O1 output value

bit 1 : D/O2 output value

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
D/O Value	D/O Value

※ D/O Value = Data3

Example) ID 0, Set up SAM D/O(Digital Output) CH0 = 1, CH1 = 1.

• Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x64	0x03	0x03	0x04

• Response Packet

1byte	1byte
0x03	0x03

(2) Read Extension Port AD

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x65

Data3 = default value

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1byte	1byte
Ext AD (H3/H5)	Ext AD(L7)

※ External AD Value structure

Ext AD (H5) + Ext AD (L7): 0 ~ 4095

Example) Read ID 0 external AD (If AD value is 1000,)

• Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x65	0x00	0x00	0x04

• Response Packet

1byte	1byte
0x07	0x68

(3) Write Motion DATA [Flash]

It is for self-running motion programming. Please refer to the "3-11 self running motion" section.

▶ Command Packet

1byte	1byte	1byte	1byte		1byte	1byte	1byte
Header	Data1	Data2	Motion Count	···	Motion command X	Motion Data X	Checksum

Header = 0xFF

Data1

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x 96

Motion Count = 0 ~ 8

Motion command X = 0 ~ 254

Motion data X = 0 ~ 254

Checksum = (byte2 XOR byte3 XOR ··· byten) AND 0x7f

▶ Response Packet

1byte	1byte
Motion Count	Motion Count

Motion Count : 0 ~ 8

Example) ID 0, Self-running motion programming, repeat from position 82 to 172.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x96	0x04	0x10	0x52	0x31	0x0A	0x10	0xAC	0x31	0x0a	0x0c

- Response Packet

1byte	1byte
0x04	0x04

(4) Read Motion DATA

Read self running motion program line nos.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Checksum

Header = 0xFF(Start Packet)

Data1 =

Mode(=7)			ID				
7	6	5	4	3	2	1	0

※ Mode : 7, ID : 0~30

Data2 = 0x 97

Data3 = default value

Data4 = Data3

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4) AND 0x7F

▶ Response Packet

1 byte	1 byte
Motion Count	Motion Count

※ Motion Count : 0 ~ 8

Example) Read ID 0 self running motion program nos.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0x97	0x00	0x00	0x77

- Response Packet

1byte	1byte
Motion Count	Motion Count

A-1-4 SAM Standard Command

(1) Set ID [Flash]

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 160(0xA0)

Data3 = ID

Data4 = new ID(0~254)

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new ID	new ID

※ new ID : 0~254

Example) Chang ID from 0 to 30.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xA0	0x00	0x1E	0x1E	0x40

- Response Packet

1byte	1byte
0x1E	0x1E

(2) Set Baud rate[Flash]

Supported baud rates are 4800 bps, 9600 bps, 38400 bps, 57600 bps, 115200 bps, 230400 bps, 460800 bps, 921600 bps.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 161(0xA1)

Data3 = ID

Data4 = 0(2Mbps) / 1(1.5Mbps) / 2(1Mbps) / 3(0.5Mbps) / 4(230400bps) / 5(115200bps) / 6(57600bps) / 7(38400bps) / 8(9600bps) / 9(4800bps)

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new Baud rate	new Baud rate

※ new Baud rate : 0~9

Example) Set ID 0 baud rate to 115200bps.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xA1	0x00	0x07	0x07	0x41

- Response Packet

1byte	1byte
0x07	0x07

(3) Read Current Speed

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 170(0xAA)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Current Speed (H5)	Current Speed (L7)

※ Current Speed (H5) + Current Speed (L7) : 0~1023 / 4095

Example) Read current Speed of ID 0, speed 100 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAA	0x00	0x00	0x00	0x4A

- Response Packet

1byte	1byte
0x00	0x64

(4) Read current Accel

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 171(0xAB)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
current Accel (H5)	current Accel (L7)

※ current Accel (H5) + current Accel (L7) : -1023 ~1023

If negative(-) sign, it means it is decelerating.

Example) Read current Accel of ID 0, Decel 100 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAB	0x00	0x00	0x00	0x4B

- Response Packet

1byte	1byte
0x00	0x64

(5) Read current Load

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 172(0xAC)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
current Load (H5)	current Load (L7)

※ current Load (H5) + current Load (L7) : 0 ~1023 / 4095

Example) Read current Load of ID 0, Load 100 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAC	0x00	0x00	0x00	0x4C

- Response Packet

1byte	1byte
0x00	0x64

(6) Read current Position

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 173(0xAD)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
current Pos. (H5)	current Pos. (L7)

※ current Position (H5) + current Position (L7) : 0 ~1023 / 4095

Example) Read current Position of ID 0, Position 100 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAD	0x00	0x00	0x00	0x4D

- Response Packet

1byte	1byte
0x00	0x64

(7) Set P, D gain [Flash]

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 174(0xAE)

Data3 = ID

Data4 = P Gain(0~254)

Data5 = D Gain(0~254)

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new P gain	new D gain

※ new gain : 0~254

Example) Set ID 0. P gain 30, D gain 30.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAE	0x00	0x1E	0x1E	0x4E

- Response Packet

1byte	1byte
0x1E	0x1E

(8) Read P, D gain

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 175(0xAF)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
P gain	D gain

※ gain : 0~254

Example) Read ID 0 P gain and D gain value of P, D gain 30 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xAF	0x00	0x00	0x00	0x4F

- Response Packet

1byte	1byte
0x1E	0x1E

(9) Set Runtime P, D gain

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 176(0xB0)

Data3 = ID

Data4 = new Runtime P Gain(0~254)

Data5 = new Runtime D Gain(0~254)

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new RP gain	new RD gain

※ new gain : 0~254

Example) Set ID 0 Runtime P gain 30, Runtime D gain 30

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB0	0x00	0x1E	0x1E	0x50

- Response Packet

1byte	1byte
0x1E	0x1E

(10) Set OverLoad [Flash]

(※ Note : If load is over, it goes "Passive" mode automatically.)

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 177(0xB1)

Data3 = ID

Data4 = new OverLoad(Upper(H5) 5 bit)

X			New OverLoad (H5)				
7	6	5	4	3	2	1	0

Data5 = new Load Limit (Lower(L7) 7 bit)

X	New OverLoad (L7)						
7	6	5	4	3	2	1	0

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
New OverLoad (H5)	New OverLoad (L7)

※ New OverLoad (H5) + New OverLoad (L7) : 0~1023 / 4095

Example) Set ID 0 OverLoad 1000

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB0	0x00	0x03	0xE8	0xBB

- Response Packet

1byte	1byte
0x03	0xE8

(11) Read OverLoad

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 178(0xB2)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
OverLoad (H5)	OverLoad (L7)

※ OverLoad (H5) + OverLoad (L7) : 0~1023 / 4095

Example) Read Overload value of ID 0 Overload 1000 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB1	0x00	0x00	0x00	0x52

- Response Packet

1byte	1byte
0x03	0xE8

(12) Set Boundary Range [Flash]

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 179(0xB3)

Data3 = ID

Data4 = new Lower Boundary(1~253)

Data5 = new Upper Boundary(2~254)

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new L Bound	new U Bound

※ new Bound : 1~254

Example) Set ID 0 Boundary Range "30, 200".

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB2	0x00	0x1E	0xC8	0x83

- Response Packet

1byte	1byte
0x1E	0xC8

(13) Read Boundary Range

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 180(0xB4)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
L Bound	U Bound

※ Bound : 1~254

Example) Read Boundary Range of ID 0, Boundary Range 30, 200 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB3	0x00	0x00	0x00	0x54

- Response Packet

1byte	1byte
0x1E	0xC8

(14) Set I gain [Flash]

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 181(0xB5)

Data3 = ID

Data4 = I Gain(0~254)

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new I gain	new I gain

※ new gain : 0~254

Example) Set ID 0, I gain 30

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB4	0x00	0x1E	0x1E	0x55

- Response Packet

1byte	1byte
0x1E	0x1E

(15) Read I gain

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 182(0xB6)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
I gain	I gain

※ gain : 0~254

Example) Read I gain of ID 0, I gain 30 setting.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB5	0x00	0x00	0x00	0x56

- Response Packet

1byte	1byte
0x1E	0x1E

(16) Set Runtime I gain

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 183(0xB7)

Data3 = ID

Data4 = new Runtime I Gain(0~10)

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new RI gain	new RI gain

※ new gain : 0~10

Example) Set ID 0, Runtime I gain 10.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB7	0x00	0x0A	0x0A	0x57

- Response Packet

1byte	1byte
0x0A	0x0A

(17) Set Drive mode [Flash]

Upper 4 bit is Response level(0: Response for Read Command only, 1 : Response for all Command,
2 : No response for all Command)

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 184(0xB8)

Data3 = ID

Data4 = new Drive mode

Set Response level				Set Reverse mode			
7	6	5	4	3	2	1	0

※ Bit 4,5 is 0 or 1 in accordance with Response level.

bit0 - 0(Inactivate) or 1 (Activate) depends on Reverse mode

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
new Drive mdoe	new Drive mdoe

Example) Set - Response for all Commands and activate Reverse mode of ID 0.

• Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB8	0x00	0x11	0x11	0x58

• Response Packet

1byte	1byte
0x10	0x10

(18) Read Drive mode

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 185(0xB9)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Drive mode	Drive mode

Example) Read ID 0 Driver Mode.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xB9	0x00	0x00	0x00	0x59

- Response Packet

1byte	1byte
0x10	0x10

(19) Set AVG Torq [Flash]

It is to set the average torque value from 0 to 2,000..

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 186(0xBA)

Data3 = ID

Data4 = new AVG Torq(Upper(H5) 5 bit)

X			New AVG Torq (H5)				
7	6	5	4	3	2	1	0

Data5 = new AVG Torq (Lower(L7) 7 bit)

X	New AVG Torq(L7)						
7	6	5	4	3	2	1	0

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
New AVG Torq (H5)	New AVG Torq(L7)

※ New AVG Torq (H5) + New AVG Torq (L7) : 0~2000

Example) Set ID 0 AVG Torq 1000

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBA	0x00	0x03	0xE8	0x31

- Response Packet

1byte	1byte
0x03	0xE8

(20) Read AVG Torq

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 187(0xBB)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
AVG Torq (H5)	AVG Torq (L7)

※ AVG Torq (H5) + AVG Torq (L7) : 0~2000

Example) Read ID 0 AVG Torq that is set to 1000.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBB	0x00	0x00	0x00	0x5B

- Response Packet

1byte	1byte
0x03	0xE8

(21) Read Input Voltage

Read AD value of Input Voltage.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 188(0xBC)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Input Volt (H5)	Input Volt (L7)

※ Input Volt (H5) + Input Volt (L7) : 0~4095

Example) Read ID 0 SAM Input Voltage value that is 1000.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBC	0x00	0x00	0x00	0x5B

- Response Packet

1byte	1byte
0x03	0xE8

A-1-5 SAM Standard Extension Command

(1) Write External Port

It is to set LED on/off or write the data to external port.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 186(0xBA)

Data3 = ID

Data4 =

				Ch		LED	
7	6	5	4	3	2	1	0

※ bit 0 : 1(Red LED on) / 0(Red LED off) / bit 1 : 1(Blue LED on) / 0(Blue LED off)

bit 2 : D/O1 output / bit 3 : D/O2 output

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Port output	Port output

Example) Set ID 0 LED on and output 1 in D/O1.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBA	0x00	0x0E	0x0E	0x5A

- Response Packet

1byte	1byte
0x0E	0x0E

(2) Read External Port AD

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 187(0xBB)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Ext AD (H3/H5)	Ext AD(L7)

※ Ext AD (H5) + Ext AD (L7) : 0 ~ 4095

Example) Read ID 0 External Port (outside AD value 1,000)

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBB	0x00	0x00	0x00	0x5B

- Response Packet

1byte	1byte
0x07	0x68

(3) Write Motion Data [Flash]

It is for self-running motion programming. Refer to the programming logic [3-11] section.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte		1byte	1byte	1byte
Header	Data1	Data2	ID	Motion Count	···	Motion command X	Motion Data X	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 0x188(0xBC)

Data3 = ID

Motion Count = 0 ~ 8

Motion command X = 0 ~ 254

Motion data X = 0 ~ 254

Checksum = (byte2 XOR byte3 XOR ··· byte) AND 0x7f

▶ Response Packet

1byte	1byte
Motion Count	Motion Count

Motion Count : 0 ~ 8

Example) Set self running motion programming function, move from position 82 to 172 repeatedly.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBC	0x00	0x04	0x10	0x52	0x31	0x0A	0x10	0xAC	0x31	0x0a	0x26

- Response Packet

1byte	1byte
0x04	0x04

(4) Read Motion Data

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF(Set Packet)

Data1 = 0xE0

Data2 = 189(0xBD)

Data3 = ID

Data4 = don't care

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1 byte	1 byte
Motion Count	Motion Count

※ Motion Count : 0 ~ 8

Example) Read ID 0, Self running motion program line nos.

• Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBD	0x00	0x00	0x00	0x5D

• Response Packet

1byte	1byte
Motion Count	Motion Count

(5) Ping

It is to check the communications between SAM.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF(Packet start)

Data1 = 0xE0

Data2 = 190(0xBE)

Data3 = ID

Data4 = don't care

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1 byte	1 byte
ID	ID

Example) Check ID 1 communication with others.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xBE	0x01	0x00	0x00	0x5F

- Response Packet

1byte	1byte
0x01	0x01

(6) Read Firmware Version

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 195 (0xC3)

Data3 = ID

Data4 = don't care

Data5 = don't care

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
Model Number	FW Version

※ Model Number

0x05 ➔ SAM-5 / 0x20 ➔ SAM-20 / 0x28 ➔ SAM-28

0x14 ➔ SAM-140 / 0x16 ➔ SAM-160

0x18 ➔ SAM-180E0200 / 0x21 ➔ SAM-210E0200

Example) Read ID 0 SAM model and Firmware version.

• Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xC3	0x00	0x00	0x00	0x23

• Response Packet

1byte	1byte
0x18	0x03

(7) Read Temp Error

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0XE0

Data2 = 197 (0xC5)

Data3 = ID

Data4 = Don't care

Data5 = Data4

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
0: Temp OK 1:Temp Error	0: Temp OK 1: Temp Error

Example) Read ID 0 Temperature status

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xC5	0x00	0x00	0x00	0x24

- Response Packet

1byte	1byte
0x00	0x00

- Temperature status is normal

- Response Packet

1byte	1byte
0x01	0x01

- Temperature status is error – position control function will not work until normal.

A-1-6 SAM Standard Command

(1) Special Control

General Mode (0), Passive Mode (1), Brake Mode (2), Wheel Mode (3).

For Wheel Mode, CW direction range is from 0 to 999 and CCW direction range is from 1000 to 2000.

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 199(0xC7)

Data3 = ID

Data4 =

Mode				Rotation Speed Upper 4 bit			
7	6	5	4	3	2	1	0

※Mode : 0(General) / 1(Passive) / 2(Brake) / 3(Wheel)

Data5 =

X	Rotation Speed Lower 7 bit						
7	6	5	4	3	2	1	0

※ Rotation Speed : 0~999 (CW) / 1000~2000(CCW)

Each direction has 1000 steps.

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

1byte	1byte
ID	Mode

Example) Set ID 0, Wheel Mode, and Direction is CCW, Speed is 500.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xC7	0x00	0x3B	0x5C	0x40

- Response Packet

1byte	1byte
0x00	0x03

(2) Position Control

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 200(0xC8)

Data3 = ID

Data4 = target position(Upper(H3 / H5) 3 bit or 5bit)

			Target (H3 / H5)				
7	6	5	4	3	2	1	0

Data5 = target position(Lower(L7) 7 bit)

X	Target (L7)						
7	6	5	4	3	2	1	0

Checksum = (Data1 XOR Data2 XOR Data3 XOR Data4 XOR Data5) AND 0x7F

▶ Response Packet

Position(H3/H5)	Position(L7)
0~1023 / 4095	

Example) Move ID 0 Position 700.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xC8	0x00	0x05	0x3C	0x11

- Response Packet

1byte	1byte
Current Position(H3/H5)	Current Position(L7)

(3) Precision Position Control

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Data6	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 202(0xCA)

Data3 = ID

Data4 = target position(Upper(H5) 5bit)

			Target (H5)				
7	6	5	4	3	2	1	0

Data5 = target position(Middle (L7) 7 bit)

X	Target (M7)						
7	6	5	4	3	2	1	0

Data6 = target position(Lower(L7) 7 bit)

X	Target (L7)						
7	6	5	4	3	2	1	0

Checksum=(Data1 xor Data2 xor Data3 xor Data4 xor Data5 xor Data6) AND 0x7F

▶ Response Packet

1byte	1byte	1byte
Current Position (H5)	Current Position(M7)	Current Position(L7)

※ current Position : current Position(H5) + current Position (M7) + current Position (L7)

0 ~ 183317 / 469293

Example) Move ID 1 to Position 15276.

- Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xCA	0x01	0x00	0x77	0x2D	0x71

- Response Packet

1byte	1byte	1byte
Current Position(H5)	Current Position(M7)	Current Position(L7)

(4) Read Precision Position Control

▶ Command Packet

1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
Header	Data1	Data2	Data3	Data4	Data5	Data6	Checksum

Header = 0xFF

Data1 = 0xE0

Data2 = 203 (0xCB)

Data3 = ID

Data4 = don't care

Data5 = don't care

Data6 = don't care

Checksum=(Data1 xor Data2 xor Data3 xor Data4 xor Data5 xor Data6) AND 0x7F

▶ Response Packet

1byte	1byte	1byte
Current position(H5)	Current Position (M7)	Current position(L7)

※ current position : current position(H5) + current position(M7) + current position(L7)

0 ~ 183317 / 469293

Example) Read current position 15276 of ID 1.

- Command Packet

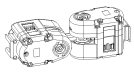
1byte	1byte	1byte	1byte	1byte	1byte	1byte	1byte
0xFF	0xE0	0xCB	0x01	0x00	0x00	0x00	0x2A1

- Response Packet

1byte	1byte	1byte
Current Position (H5)	Current Position (M7)	Current position(L7)

A-2 Code Example

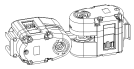
```
/*----------------data definition ----------------------------------------------------------------------*/
typedef signed long  s32;      // 4 byte
typedef signed short s16;      // 2 byte
typedef signed char  s8;       // 1 byte
typedef unsigned long  u32;    // 4 byte
typedef unsigned short u16;    // 2 byte
typedef unsigned char  u8;     // 1 byte

/*----------------Serial Communication function ----------------------------------------------------------*/
/* ※ User should define the functions in accordance with Micro-Controller Unit or PC Application support.  */
/*-------------------------------------------------------------------------------------------------------*/
void SendByte(u8 data);        // UART Transmit Function
u8 GetByte(u16 timeout);       // UART Receive Function

/*----------------Packet formation function ------------------------------------------*/
#define HEADER 0xff

void Quick_Ctrl_CMD (u8 Data1, u8 Data2);
void Quick_Set_Expand_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4);
void Standard_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4);
void Standard_8Byte_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4, u8 Data5);

/*----------------Quick  control function ----------------------------------------------------------------*/
u16 Quick_PosControl_CMD (u8 SamId, u8 Torq, u8 TargetPos);
void Quick_StatusRead_CMD (u8 SamId );
u16 Quick_PassiveMode_CMD (u8 SamId );
u16 Quick_WheelMode_CMD (u8 SamId, u8 Dir, u8 WheelSpeed);
u16 Quick_BrakeMode_CMD (void);

/*----------------Quick  setting function ----------------------------------------------------------------*/
u16 Quick_BaudSet_CMD(u8 SamId, u8 Baud);
u16 Quick_RuntimePDgainSet_CMD(u8 SamId, u8 RPgain, u8 RDgain);
u16 Quick_IdSet_CMD(u8 SamId, u8 NewID);
u16 Quick_OverLoadSet_CMD(u8 SamId, u8 OverLoad);
u16 Quick_OverLoadRead_CMD(u8 SamId);
u16 Quick_BoundarySet_CMD(u8 SamId, u8 LowerBound, u8 UpperLowerBound);
u16 Quick_BoundaryRead_CMD(u8 SamId);
```

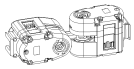
```
/*----------Quick extension function----------*/
u16 Quick_ExpandPortWrite_CMD(u8 SamId, u8 ChannelData, U8 LEDData);
u16 Quick_ExpandPortADRead_CMD(u8 SamId);
u16 Quick_MotionDataRead_CMD(u8 SamId);

/*----------Standard control function----------*/
u16 Standard_IdSet_CMD(u8 SamId);
u16 Standard_IdSet_CMD(u8 SamId);
u16 Standard_CTRLSpeedSet_CMD(u8 SamId, u16 CTRLSpeed);
u16 Standard_CTRLSpeedRead_CMD(u8 SamId);
u16 Standard_CTRLTorqSet_CMD(u8 SamId, u16 CTRLTorq);
u16 Standard_CTRLTorqRead_CMD(u8 SamId);
u16 Standard_CTRLAccelSet_CMD(u8 SamId, u16 CTRLAccel);
u16 Standard_CTRLAccelRead_CMD(u8 SamId);
u16 Standard_CTRLDecelSet_CMD(u8 SamId, u16 CTRLDecel);
u16 Standard_CTRLDecelRead_CMD(u8 SamId);
s16 Standard_CurrentSpeedRead_CMD(u8 SamId);
s16 Standard_CurrentAccelRead_CMD(u8 SamId);
u16 Standard_CurrentLoadRead_CMD(u8 SamId);
u16 Standard_CurrentPosRead_CMD(u8 SamId);
u16 Standard_PDgainSet_CMD(u8 SamId, u8 Pgain, u8 Dgain);
u16 Standard_PDgainRead_CMD(u8 SamId);
u16 Standard_RuntimePDgainSet_CMD(u8 SamId, u8 RPgain, u8 RDgain);
u16 Standard_OverLoadSet_CMD(u8 SamId, u16 OverLoad);
u16 Standard_OverLoadRead_CMD(u8 SamId);
u16 Standard_BoundarySet_CMD(u8 SamId, u8 LowerBoundary, u8 UpperBoundary);
u16 Standard_BoundaryRead_CMD(u8 SamId);
u16 Standard_IgainSet_CMD(u8 SamId, u8 Igain);
u16 Standard_IgainRead_CMD(u8 SamId);
u16 Standard_RuntimeIgainSet_CMD(u8 SamId, u8 RIgain);
u16 Standard_DrivemodeSet_CMD(u8 SamId, u8 ResponseLevel, u8 Reverse);
u16 Standard_DrivemodeRead_CMD(u8 SamId);
u16 Standard_AVGTorqSet_CMD(u8 SamId, u16 AVGTorq);
u16 Standard_AVGTorqRead_CMD(u8 SamId);
u16 Standard_InputVoltageRead_CMD(u8 SamId);
```

```
/*-----------------Standard extension function -----------------------------------------------------------------*/
u16 Standard_ExpandPortWrite_CMD(u8 SamId, u8 ChannelData, u8 LEDData);
u16 Standard_ExpandPortADRead_CMD(u8 SamId);
u16 Standard_MotionDataRead_CMD(u8 SamId);
u16 Standard_PING_CMD(u8 SamId);
u16 Standard_FirmwareVersionRead_CMD(u8 SamId);
u16 Standard_OffsetControl_CMD(u8 SamId, u8 OffsetControl);
u16 Standard_SpecialControl_CMD(u8 SamId, u8 ControlMode, u16 WheelSpeed);
u16 Standard_PosControl_CMD(u8 SamId, u16 Position);
u32 Standard_PrecisionPosControl_CMD(u8 SamId, u32 PrecisionPosition);
u32 Standard_PrecisionPosRead_CMD(u8 SamId);


/*-----------------Define Packet formation function -----------------------------------------------------------*/
/*****************************************************************************/
/* Send Quick control Command Packet(4 bytes) to SAM                         */
/* Input : Data1, Data2                                                      */
/* Output : None                                                             */
/*****************************************************************************/
void Quick_Ctrl_CMD (u8 Data1, u8 Data2)
{
    u8 CheckSum;
    CheckSum = (Data1^Data2) & 0x7f;
    SendByte(HEADER);
    SendByte(Data1);    SendByte(Data2);
    SendByte(CheckSum);
}

/*****************************************************************************/
/* Send Quick setting Command Packet(6 bytes) to SAM                 */
/* Input : Data1, Data2, Data3, Data4                                        */
/* Output : None                                                             */
/*****************************************************************************/
void Quick_Set_Expand_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4)
{
    u8 CheckSum;
    CheckSum = (Data1^Data2^Data3^Data4) & 0x7f;
    SendByte(HEADER);
    SendByte(Data1);    SendByte(Data2);    SendByte(Data3);    SendByte(Data4);
    SendByte(CheckSum);
}

/*****************************************************************************/
```

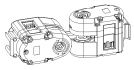
```
/* Send Standard Command Packet(7 bytes) to SAM                              */
/* Input : Data1, Data2, Data3, Data4                                        */
/* Output : None                                                             */
/*****************************************************************************/
void Standard_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4)
{
    u8 CheckSum;
    CheckSum = (0xe0 ^Data1^Data2^Data3^Data4) & 0x7f;
    SendByte(HEADER);    SendByte(0xe0);
    SendByte(Data1);    SendByte(Data2);    SendByte(Data3);    SendByte(Data4);
    SendByte(CheckSum);
}

/*****************************************************************************/
/* Send Standard 8byte Command Packet(8 bytes) to SAM                        */
/* Input : Data1, Data2, Data3, Data4 , Data5                                */
/* Output : None                                                             */
/*****************************************************************************/
void Standard_8Byte_CMD (u8 Data1, u8 Data2, u8 Data3, u8 Data4, u8 Data5)
{
    u8 CheckSum;
    CheckSum = (0xe0 ^Data1^Data2^Data3^Data4^Data5) & 0x7f;
    SendByte(HEADER);    SendByte(0xe0);
    SendByte(Data1);    SendByte(Data2);    SendByte(Data3);    SendByte(Data4);    SendByte(Data5);
    SendByte(CheckSum);
}


/*----------------Define Quick control function ----------------------------------------------------------------*/
/*****************************************************************************/
/* Send Quick position control Command Packet(4 bytes) to SAM                */
/* Input : SamId, Torq, TargetPos                                            */
/* Output : Response 2Byte (Upper Byte : Current / Lower Byte : Position )   */
/*****************************************************************************/
u16 Quick_PosControl_CMD (u8 SamId, u8 Torq, u8 TargetPos)
{
    u16 ResponseData = 0;
    If( (SamId <= 30) && (Torq <= 4) && (TargetPos <= 254) )
    {
        Quick_Ctrl_CMD ( (Torq<<5) | SamId, TargetPos );
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
```

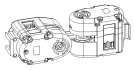
```
    return ResponseData;
}

/*****************************************************************************/
/* Send Quick read status Command Packet(4bytes) to SAM                      */
/* Input : SamId, *Torq, *CurrentPos                                         */
/* Output : Response 2Byte (Upper Byte : Torq / Lower Byte : Position )      */
/*****************************************************************************/
void Quick_StatusRead_CMD (u8 SamId)
{
    u16 ResponseData = 0;
    if( SamId <=30 )
    {
        Quick_Ctrl_CMD ( 0xa0 | SamId, 0 );
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send for entering into Quick Passive mode Command Packet(4bytes)          */
/* Input : SamId                                                             */
/* Output : Response 2Byte (Upper Byte : ID / Lower Byte : ID )              */
/*****************************************************************************/
u16 Quick_PassiveMode_CMD (u8 SamId )
{
    u16 ResponseData = 0;
    if( SamId <=30 )
    {
        Quick_Ctrl_CMD ( 0xc0 | SamId, 0x10 );
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send for entering into Quick Wheel mode Command Packet(4bytes)            */
/* Input : SamId, Dir, WheelSpeed                                            */
/* Output : Response 2Byte (Upper Byte : ID / Lower Byte : ID )              */
/*****************************************************************************/
u16 Quick_WheelMode_CMD (u8 SamId, u8 Dir, u8 WheelSpeed)
```

```
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (Dir==3 || Dir==4) && (WheelSpeed <=15) )
    {
        Quick_Ctrl_CMD ( 0xc0 | SamId, (Dir<<4)| WheelSpeed);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
/*****************************************************************************/
/* Send for entering into Quick Brake mode Command Packet(4bytes)            */
/* Input : None                                                              */
/* Output : Response 2Byte (Upper Byte : ID / Lower Byte : Position )         */
/*****************************************************************************/
u16 Quick_BrakeMode_CMD (void)
{
    u16 ResponseData = 0;
    Quick_Ctrl_CMD (0xc0 | SamId, 0x20);
    ResponseData = GetByte(TIMEOUT) << 8;
    ResponseData |= GetByte(TIMEOUT);

    return ResponseData;
}

/*----------------Define Quick setting function ----------------------------------------------------------*/
/*****************************************************************************/
/* Send for setting of Quick baud rate Command Packet(6bytes)            */
/* Input : SamId, Baud                                                       */
/* Output : Response 2Byte (Upper Byte : Baud / Lower Byte : Baud )           */
/*****************************************************************************/
u16 Quick_BaudSet_CMD(u8 SamId, u8 Baud)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (Baud <= 9) )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 8, Baud, Baud);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

```
/*****************************************************************************/
/* Send Quick RunTime P,D Gain setting Command Packet(6bytes)                */
/* Input : SamId, RPgain, RDgain                                             */
/* Output : Response 2Byte (Upper Byte : RPgain / Lower Byte : RDgain )      */
/*****************************************************************************/
u16 Quick_RuntimePDgainSet_CMD(u8 SamId, u8 RPgain, u8 RDgain)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (1 <= RPgain) && (RPgain <= 254) && (0 <= RDgain) && (RDgain <= 254) )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 11, RPgain, RDgain);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send Quick 5bit ID  setting Command Packet(6bytes)                        */
/* Input : SamId, NewID                                                      */
/* Output : Response 2Byte (Upper Byte : NewID / Lower Byte : NewID )        */
/*****************************************************************************/
u16 Quick_IdSet_CMD(u8 SamId, u8 NewID)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (1 <= NewID) && (NewID <= 30) )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 12, NewID, NewID);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send Quick OverLoad setting Command Packet(6bytes)                        */
/* Input : SamId, OverLoad                                                   */
/* Output : Response 2Byte (Upper Byte : OverLoad / Lower Byte : OverLoad)   */
/*****************************************************************************/
u16 Quick_OverLoadSet_CMD(u8 SamId, u8 OverLoad)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (1 <= OverLoad) && (OverLoad <= 30) )
```

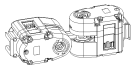



```
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 15, OverLoad, OverLoad);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send Quick read OverLoad Command Packet(6bytes)                           */
/* Input : SamId, OverLoad                                                   */
/* Output : Response 2Byte (Upper Byte : OverLoad / Lower Byte : OverLoad)   */
/*****************************************************************************/
u16 Quick_OverLoadRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( SamId <= 30 )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 16, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send Quick boundary range setting Command Packet(6bytes)                  */
/* Input : SamId, LowerBound, UpperBound                                     */
/* Output : Response 2Byte (Upper Byte : LowerBound / Lower Byte :UpperBound)*/
/*****************************************************************************/
u16 Quick_BoundarySet_CMD(u8 SamId, u8 LowerBound, u8 UpperLowerBound)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (0 <= LowerBound) && (LowerBound <=253) && (1<= UpperBound) && (UpperBound <= 254) )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 17, LowerBound, UpperBound);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
```

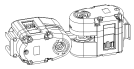
```
/* Send Quick read boundary Command Packet(6bytes)                              */
/* Input : SamId                                                                */
/* Output : Response 2Byte (Upper Byte : LowerBound / Lower Byte :UpperBound)   */
/******************************************************************************/
u16 Quick_BoundaryRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( SamId <= 30 )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 18, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}


/*----------------------Define Quick extension function----------------------*/
/******************************************************************************/
/* Send Quick write extension port Command Packet(6bytes)                     */
/* Input : SamId, ChannelData, LEDData                                        */
/* Output : Response 2Byte (Upper Byte : WriteData / Lower Byte : WriteData)  */
/******************************************************************************/
u16 Quick_ExpandPortWrite_CMD(u8 SamId, u8 ChannelData, U8 LEDData)
{
    u16 ResponseData = 0;
    if( (SamId <= 30) && (ChannelData <= 3) && (LEDData <= 3))
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 100, (ChannelData<<2) | LEDData, (ChannelData<<2) | LEDData);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/******************************************************************************/
/* Send Quick read extension port AD value Command Packet(6bytes)             */
/* Input : SamId                                                              */
/* Output : Expand AD Value                                                   */
/******************************************************************************/
u16 Quick_ExpandPortADRead_CMD(u8 SamId)
{
    u16 ExpandAD_Value = 0;
```

```
    if( SamId <= 30 )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 101, 0, 0);
        ExpandAD_Value = GetByte(TIMEOUT) << 7;
        ExpandAD_Value |= GetByte(TIMEOUT);
    }
    return ExpandAD_Value;
}

/*****************************************************************************/
/* Send Quick read motion data Command Packet(6bytes)                        */
/* Input : SamId                                                             */
/* Output : Response 2Byte (Upper Byte : Motion No./ Lower Byte : Motion No.) */
/*****************************************************************************/
u16 Quick_MotionDataRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( SamId <= 30 )
    {
        Quick_Set_Expand_CMD (0xe0 | SamId, 151, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}


/*-------------------Define Standard control function -------------------------------------------------------*/
/*******************************************************************************************/
/* Send Standard ID setting Command Packet(7 bytes)                                        */
/* Input : SamId, NewID                                                                    */
/* Output : Response 2Byte (Upper Byte : NewID / Lower Byte : NewID)                       */
/*******************************************************************************************/
u16 Standard_IdSet_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (NewID <= 254) )
    {
        Standard_CMD (160, SamId, NewID, NewID);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
```

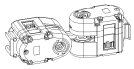
```
}

/*********************************************************************************************/
/* Send Standard baud rate setting Command Packet(7bytes)                                    */
/* Input : SamId, Baud                                                                       */
/* Output : Response 2Byte (Upper Byte : Baud / Lower Byte : Baud)                           */
/*********************************************************************************************/
u16 Standard_IdSet_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (Baud <= 9) )
    {
        Standard_CMD (161, SamId, Baud, Baud);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard CTRL Speed setting Command Packet(7bytes)                                   */
/* Input : SamId, CTRLSpeed                                                                  */
/* Output : CTRLSpeed                                                                        */
/*********************************************************************************************/
u16 Standard_CTRLSpeedSet_CMD(u8 SamId, u16 CTRLSpeed)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (CTRLSpeed <= 4095) )
    {
        Standard_CMD (162, SamId, CTRLSpeed>>7, CTRLSpeed & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read CTRL Speed Command Packet(7bytes)                                      */
/* Input : SamId                                                                             */
/* Output : CTRLSpeed                                                                        */
/*********************************************************************************************/
u16 Standard_CTRLSpeedRead_CMD(u8 SamId)
{
```

```
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (163, SamId, 0,0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
/*******************************************************************************************/
/* Send Standard CTRL Torq setting Command Packet(7bytes)                                  */
/* Input : SamId, CTRLTorq                                                                 */
/* Output : CTRLTorq                                                                       */
/*******************************************************************************************/
u16 Standard_CTRLTorqSet_CMD(u8 SamId, u16 CTRLTorq)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (CTRLTorq <= 4095) )
    {
        Standard_CMD (164, SamId, CTRLTorq>>7, CTRLTorq & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*******************************************************************************************/
/* Send Standard read CTRL Torq Command Packet(7bytes)                                     */
/* Input : SamId                                                                           */
/* Output : CTRLTorq                                                                       */
/*******************************************************************************************/
u16 Standard_CTRLTorqRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (165, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

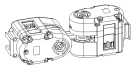
```
/*********************************************************************************************/
/* Send Standard CTRL Accel setting Command Packet(7bytes)                                   */
/* Input : SamId, CTRLAccel                                                                  */
/* Output : CTRLAccel                                                                        */
/*********************************************************************************************/
u16 Standard_CTRLAccelSet_CMD(u8 SamId, u16 CTRLAccel)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (CTRLAccel <= 4095) )
    {
        Standard_CMD (166, SamId, CTRLAccel>>7, CTRLAccel & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read CTRL Accel Command Packet(7bytes)                                      */
/* Input : SamId                                                                             */
/* Output : CTRLAccel                                                                        */
/*********************************************************************************************/
u16 Standard_CTRLAccelRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (167, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard CTRL Decel setting Command Packet(7bytes)                                   */
/* Input : SamId, CTRLDecel                                                                  */
/* Output : CTRLDecel                                                                        */
/*********************************************************************************************/
u16 Standard_CTRLDecelSet_CMD(u8 SamId, u16 CTRLDecel)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (CTRLDecel <= 4095) )
```

```
    {
        Standard_CMD (168, SamId, CTRLDecel>>7, CTRLDecel & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*******************************************************************************************/
/* Send Standard read CTRL Decel Command Packet(7bytes)                                    */
/* Input : SamId                                                                           */
/* Output : CTRLDecel                                                                      */
/*******************************************************************************************/
u16 Standard_CTRLDecelRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (169, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*******************************************************************************************/
/* Send Standard read current Speed Command Packet(7bytes)                                 */
/* Input : SamId                                                                           */
/* Output : CurrentSpeed                                                                   */
/*******************************************************************************************/
s16 Standard_CurrentSpeedRead_CMD(u8 SamId)
{
    s16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (170, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);

        If(ResponseData & 0x4000) ResponseData = - ResponseData;
    }
    return ResponseData;
}
```

```
/*********************************************************************************************/
/* Send Standard read current Accel Command Packet(7bytes)                                   */
/* Input : SamId                                                                             */
/* Output : CurrentAccel                                                                     */
/*********************************************************************************************/
s16 Standard_CurrentAccelRead_CMD(u8 SamId)
{
    s16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (171, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);

        If(ResponseData & 0x4000) ResponseData = - ResponseData;
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read current Load Command Packet(7bytes)                                    */
/* Input : SamId                                                                             */
/* Output : CurrentLoad                                                                      */
/*********************************************************************************************/
u16 Standard_CurrentLoadRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (172, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read current position Command Packet(7bytes)                                */
/* Input : SamId                                                                             */
/* Output : CurrentPos                                                                       */
/*********************************************************************************************/
u16 Standard_CurrentPosRead_CMD(u8 SamId)
```

```
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (173, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard P,D gain setting Command Packet(7bytes)                                     */
/* Input : SamId, Pgain, Dgain                                                               */
/* Output : Response 2Byte (Upper Byte : Pgain / Lower Byte : Dgain )                        */
/*********************************************************************************************/
u16 Standard_PDgainSet_CMD(u8 SamId, u8 Pgain, u8 Dgain)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (1 <= Pgain) && (Pgain <= 254) && (Dgain <= 254))
    {
        Standard_CMD (174, SamId, Pgain, Dgain);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read P,D gain Command Packet(7bytes)                                        */
/* Input : SamId                                                                             */
/* Output : Response 2Byte (Upper Byte : Pgain / Lower Byte : Dgain )                        */
/*********************************************************************************************/
u16 Standard_PDgainRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254))
    {
        Standard_CMD (175, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
```

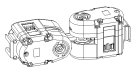



```
}

/*******************************************************************************************/
/* Send Standard Runtime P,D gain setting Command Packet(7bytes)                           */
/* Input : SamId, RPgain, RDgain                                                           */
/* Output : Response 2Byte (Upper Byte : RPgain / Lower Byte : RDgain )                    */
/*******************************************************************************************/
u16 Standard_RuntimePDgainSet_CMD(u8 SamId, u8 RPgain, u8 RDgain)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (1 <= RPgain) && (RPgain <= 254) && (RDgain <= 254))
    {
        Standard_CMD (176, SamId, RPgain, RDgain);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*******************************************************************************************/
/* Send Standard OverLoad setting Command Packet(7bytes)                                   */
/* Input : SamId, OverLoad                                                                 */
/* Output : OverLoad                                                                       */
/*******************************************************************************************/
u16 Standard_OverLoadSet_CMD(u8 SamId, u16 OverLoad)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (OverLoad <= 4095) )
    {
        Standard_CMD (177, SamId, OverLoad>>7, OverLoad & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*******************************************************************************************/
/* Send Standard read OverLoad Command Packet(7bytes)                                      */
/* Input : SamId                                                                           */
/* Output : OverLoad                                                                       */
/*******************************************************************************************/
u16 Standard_OverLoadRead_CMD(u8 SamId)
{
```

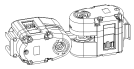
```
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (178, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard Boundary setting Command Packet(7bytes)                                 */
/* Input : SamId, LowerBoundary, UpperBoundary                                           */
/* Output : Response 2Byte (Upper Byte : LowerBoundary / Lower Byte : UpperBoundary)     */
/*****************************************************************************************/
u16 Standard_BoundarySet_CMD(u8 SamId, u8 LowerBoundary, u8 UpperBoundary)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (LowerBoundary <= 253) && (1 <= UpperBoundary) && (UpperBoundary <= 254) )
    {
        Standard_CMD (179, SamId, LowerBoundary, UpperBoundary);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard read Boundary Command Packet(7bytes)                                    */
/* Input : SamId                                                                         */
/* Output : Response 2Byte (Upper Byte : LowerBoundary / Lower Byte : UpperBoundary)     */
/*****************************************************************************************/
u16 Standard_BoundaryRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (180, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

```
/*****************************************************************************************/
/* Send Standard I gain setting Command Packet(7bytes)                                   */
/* Input : SamId, Igain                                                                  */
/* Output : Response 2Byte (Upper Byte : Igain / Lower Byte :Igain)                      */
/*****************************************************************************************/
u16 Standard_IgainSet_CMD(u8 SamId, u8 Igain)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (Igain <= 254) )
    {
        Standard_CMD (181, SamId, Igain, Igain);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard read I gain Command Packet(7bytes)                                      */
/* Input : SamId                                                                         */
/* Output : Response 2Byte (Upper Byte : Igain / Lower Byte :Igain)                      */
/*****************************************************************************************/
u16 Standard_IgainRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (182, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard Runtime I gain setting Command Packet(7bytes)                           */
/* Input : SamId, RIgain                                                                 */
/* Output : Response 2Byte (Upper Byte : RIgain / Lower Byte : RIgain)                   */
/*****************************************************************************************/
u16 Standard_RuntimeIgainSet_CMD(u8 SamId, u8 RIgain)
{
    u16 ResponseData = 0;
```

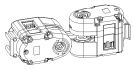
```
    if( (SamId <= 254) && (RIgain <= 254) )
    {
        Standard_CMD (183, SamId, RIgain, RIgain);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard Drive Mode setting Command Packet(7bytes)                               */
/* Input : SamId, ResponseLevel, Reverse                                                 */
/* Output : Response 2Byte (Upper Byte : DriveMode / Lower Byte : DriveMode)             */
/*****************************************************************************************/
u16 Standard_DrivemodeSet_CMD(u8 SamId, u8 ResponseLevel, u8 Reverse)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (ResponseLevel <= 2) && (Reverse <= 1) )
    {
        Standard_CMD (184, SamId, (ResponseLevel<<4) | Reverse, (ResponseLevel<<4) | Reverse);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard read Drive Mode Command Packet(7bytes)                                  */
/* Input : SamId                                                                         */
/* Output : Response 2Byte (Upper Byte : DriveMode / Lower Byte : DriveMode)             */
/*****************************************************************************************/
u16 Standard_DrivemodeRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (185, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

```
/*****************************************************************************************/
/* Send Standard AVGTorq setting Command Packet(7bytes)                                  */
/* Input : SamId, AVGTorq                                                                */
/* Output : AVGTorq                                                                      */
/*****************************************************************************************/
u16 Standard_AVGTorqSet_CMD(u8 SamId, u16 AVGTorq)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (AVGTorq <= 4095) )
    {
        Standard_CMD (186, SamId, AVGTorq>>7, AVGTorq & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard read AVGTorq Command Packet(7bytes)                                     */
/* Input : SamId                                                                         */
/* Output : AVGTorq                                                                      */
/*****************************************************************************************/
u16 Standard_AVGTorqRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (187, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************************/
/* Send Standard read Input Voltage Command Packet(7bytes)                               */
/* Input : SamId                                                                         */
/* Output : InputVoltage                                                                 */
/*****************************************************************************************/
u16 Standard_InputVoltageRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
```

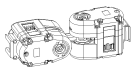



```
    {
        Standard_CMD (188, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*---------- Define Standard extension function ----------*/
/*****************************************************************************/
/* Send Standard write extension port Command Packet(7bytes)                 */
/* Input : SamId, ChannelData, LEDData                                       */
/* Output : Response 2Byte (Upper Byte : WriteData/ Lower Byte :WriteData)   */
/*****************************************************************************/
u16 Standard_ExpandPortWrite_CMD(u8 SamId, u8 ChannelData, u8 LEDData)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (Channel <= 3) && (LEDData <= 3) )
    {
        Standard_CMD (190, SamId, (ChannelData<<2) | LEDData, (ChannelData<<2) | LEDData);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*****************************************************************************/
/* Send Standard read extension port Command Packet(7bytes)                  */
/* Input : SamId                                                             */
/* Output : ExpandAD                                                         */
/*****************************************************************************/
u16 Standard_ExpandPortADRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (191, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

```
/*********************************************************************************************/
/* Send Standard read MotionData Command Packet(7bytes)                                      */
/* Input : SamId                                                                             */
/* Output : Output : Response 2Byte (Upper Byte : Motion 수/ Lower Byte : Motion 수)          */
/*********************************************************************************************/
u16 Standard_MotionDataRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (193, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard PING Command Packet(7bytes)                                                 */
/* Input : SamId                                                                             */
/* Output : Output : Response 2Byte (Upper Byte : ID / Lower Byte : ID)                       */
/*********************************************************************************************/
u16 Standard_PING_CMD(u8 SamId)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) )
    {
        Standard_CMD (194, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard read FirmwareVersion  Command Packet(7bytes)                                */
/* Input : SamId                                                                             */
/* Output : Output : Response 2Byte (Upper Byte : ModelNumber / Lower Byte : FWversion)      */
/*********************************************************************************************/
u16 Standard_FirmwareVersionRead_CMD(u8 SamId)
{
    u16 ResponseData = 0;
```

```
    if( (SamId <= 254) )
    {
        Standard_CMD (195, SamId, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard zero position setting Command Packet(7bytes)                                */
/* Input : SamId, OffsetControl                                                              */
/* Output : Output : Response 2Byte (Upper Byte : OffsetControl / Lower Byte :OffsetControl) */
/*********************************************************************************************/
u16 Standard_OffsetControl_CMD(u8 SamId, u8 OffsetControl)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (OffsetControl <= 1))
    {
        Standard_CMD (196, SamId, OffsetControl, OffsetControl);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard special mode setting Command Packet(7bytes)                                 */
/* Input : SamId, ControlMode, WheelSpeed                                                    */
/* Output : Output : Response 2Byte (Upper Byte : ID / Lower Byte : ControlMode)             */
/*********************************************************************************************/
u16 Standard_SpecialControl_CMD(u8 SamId, u8 ControlMode, u16 WheelSpeed)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (ControlMode <= 3) && (WheelSpeed <= 2000) )
    {
        If( ControlMode == 3 ) Standard_CMD (199, SamId, ((WheelSpeed>>7) & 0x0f) | 0x30, WheelSpeed);
        else Standard_CMD (199, SamId, ControlMode<<4, 0);
        ResponseData = GetByte(TIMEOUT) << 8;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```

```
/*********************************************************************************************/
/* Send  Standard position control Command Packet(7bytes)                                    */
/* Input : SamId, Position                                                                   */
/* Output : Position                                                                         */
/*********************************************************************************************/
u16 Standard_PosControl_CMD(u8 SamId, u16 Position)
{
    u16 ResponseData = 0;
    if( (SamId <= 254) && (Position <= 4095) )
    {
        Standard_CMD (200, SamId, Position>>7, Position & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send Standard precision position control Command Packet(8bytes)                           */
/* Input : SamId, Position                                                                   */
/* Output : Position                                                                         */
/*********************************************************************************************/
u32 Standard_PrecisionPosControl_CMD(u8 SamId, u32 PrecisionPosition)
{
    u32 ResponseData = 0;
    if( (SamId <= 254) && (PrecisionPosition <= 4096766) )
    {
        Standard_8Byte_CMD (202, SamId, PrecisionPosition>>14, PrecisionPosition>>7, PrecisionPosition & 0x7f);
        ResponseData = GetByte(TIMEOUT) << 14;
        ResponseData |= GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}

/*********************************************************************************************/
/* Send  Standard read precision control Command Packet(8bytes)                              */
/* Input : SamId                                                                             */
/* Output : Position                                                                         */
/*********************************************************************************************/
u32 Standard_PrecisionPosRead_CMD(u8 SamId)
{
```

```
    u32 ResponseData = 0;
    if( (SamId <= 254))
    {
        Standard_8Byte_CMD (203, SamId, 0, 0, 0);
        ResponseData = GetByte(TIMEOUT) << 14;
        ResponseData |= GetByte(TIMEOUT) << 7;
        ResponseData |= GetByte(TIMEOUT);
    }
    return ResponseData;
}
```