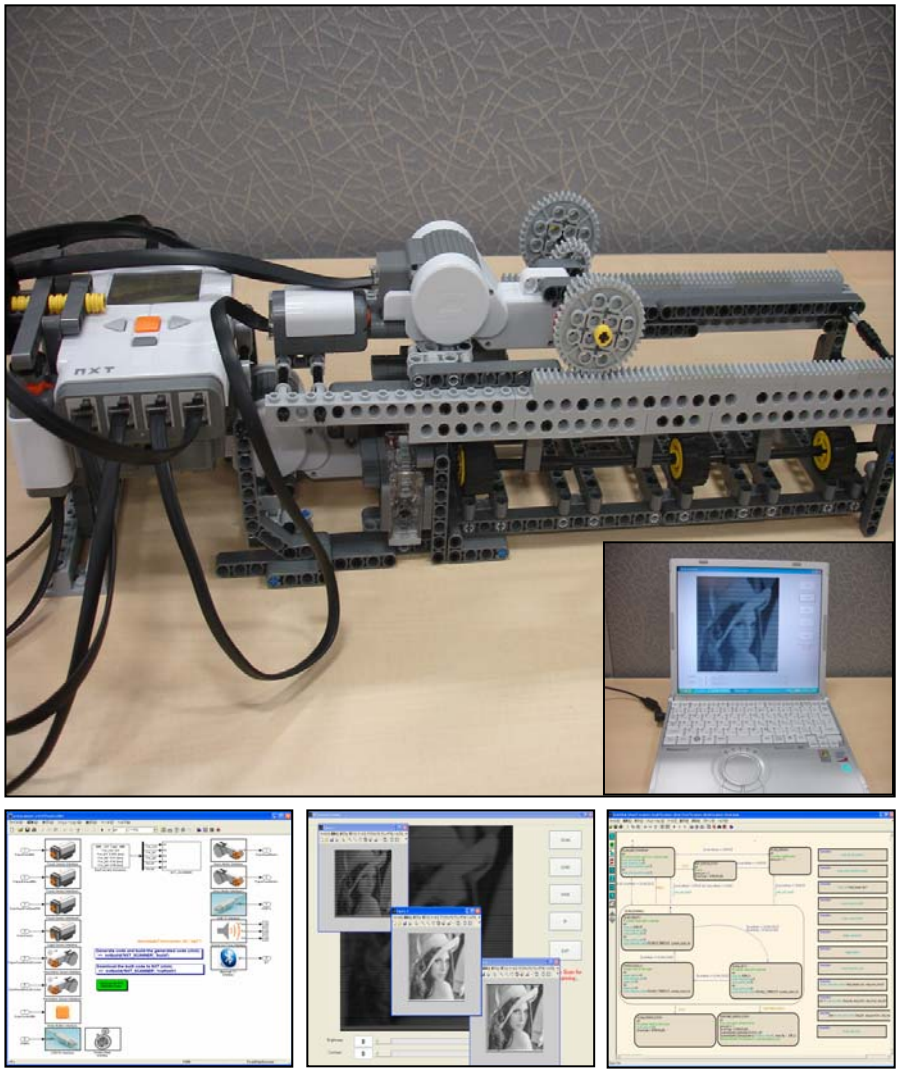
Model-Based Design & an NXT Scanner

- Image Scanner built with LEGO Mindstorms NXT -



■ Author (First Edition)

Yoshiaki Banno : banno@cybernet.co.jp

Tomoki Fukuda : t_fukuda@cybernet.co.jp

Application Engineer

Advanced Support Group 1 Engineering Department

Applied Systems First Division

CYBERNET SYSTEMS CO., LTD.

■ Revision History

Revision	Date	Description	Author/Editor
1.0	May 2009	First Edition	Yoshiaki Banno banno@cybernet.co.jp
		First Edition (chapter 6.2, 6.4, 11.1, 11.2 and 11.3)	Tomoki Fukuda t_fukuda@cybernet.co.jp

The contents and URL described in this document can be changed with no previous notice.

■ Disclaimer

Introduction

This document describes the required products for an NXT Scanner and how to build it. An NXT Scanner is a sheet feed image scanner and is built with Lego Mindstorms NXT. An NXT Viewer is an image viewer and image processing in MATLAB. This document presents Model-Based Design of an NXT Scanner by using MATLAB/Simulink. An NXT Scanner which illustrates MBD using UML and Simulink/Stateflow and using R2008b new features and using many MATLAB/Simulink optional products. The main contents are the following:

- Summary of the NXT Scanner and the NXT Viewer
- The NXT Scanner system
- The NXT Scanner Design
- The NXT Scanner Modeling
- Simulation and Results
- Image Viewer and Image Processing with the NXT Viewer

Preparation

To build an NXT Scanner, please read "NXT Scanner Building Instructions (NXT Scanner Building Instructions.pdf)". You need to download Embedded Coder Robot NXT from the following URL because it is used as Model-Based Design Environment in this document.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13399

Please read "Embedded Coder Robot NXT Instruction Manual (Embedded Coder Robot NXT Instruction En.pdf)" and test sample models / programs preliminarily. The software versions used in this document are as follows.

Software	Version
Embedded Coder Robot NXT	3.16+
nxtOSEK/JSP	2.06
Cygwin	1.5.24
GNU ARM	4.0.2

Required Products

Product	Version	Release
MATLAB®	7.7	R2008b
Simulink®	7.2	R2008b
Stateflow®	7.2	R2008b
Real-Time Workshop®	7.2	R2008b
Real-Time Workshop® Embedded Coder™	5.2	R2008b
Stateflow® Coder™	7.2	R2008b
Image Processing Toolbox™	6.2	R2008b
PolySpace® Server™ for C/C++	6.0	R2008b
PolySpace® Client™ for C/C++	6.0	R2008b
PolySpace® Model Link™ SL	5.2	R2008b
Simulink® Verification and Validation™	2.4	R2008b
Simulink® Design Verifier™	1.3	R2008b

File Lists

Category	File Name	Description
Common	setup_nxtscanner.m	Path for MATLAB
	nxtscanner_ctrl.mdl	The NXT Scanner controller model
	nxtscanner_lib.mdl	The NXT Scanner library model
	mode_control_verification_lib.mdl	[sample] Verification for mode_control.mdl
Data object	nxtscanner.xls	Simulink Data Object management
	SDOxlsIF folder	Excel Interface API for Simulink Data Object Tool
Mode control	mode_control_prm.m	Parameters
	mode_control.mdl	Functional unit model
Paper feed control	paper_feed_mechanism.mdl	Functional unit model
Scan control	scanner_head_prm.m	Parameters
	scanner_head.mdl	Functional unit model
USB control	data_communication.mdl	Functional unit model
	usb_test_signal.mdl	Validation model
	usb_test_data.mat	Validation data
	usb_test_number.mat	Validation data
	usb_test_state.mat	Validation data
Enumerated data types	ControlModeEnum.m	Definition
	ErrorCodeEnum.m	Definition
	FeedModeEnum.m	Definition
	ScanModeEnum.m	Definition
	ScannerBufferInfoEnum.m	Definition
NXT Viewer	NXTScannerViewer.fig	The NXT Viewer GUI design
	NXTScannerViewer.m	The NXT Viewer M-script
PolySpace	nxtscanner_ctrl_polyspace.cfg	Configure file
	polyspace_main.c	Verification file of dummy main loop
	polyspace.h	Verification file for avoidable compile error
	polyspace_additional_file_list.txt	Configure file

Index

1 Model-Based Design

This chapter describes the outline Model-Based Design briefly.

1.1 What is Model-Based Design?

Model-Based Design is a software development technique that uses simulation models. Generally, it is abbreviated as MBD. For control systems, a designer models a plant and a controller or a part of them, and tests the controller algorithm based on a PC simulation or real-time simulation. The real-time simulation enables us to verify and validate the algorithm in real-time, by using code generated from the model. It is Rapid Prototyping (RP) that a controller is replaced by a real-time simulator, and Hardware In the Loop Simulation (HILS) is a plant version of Rapid Prototyping.

Furthermore, auto code generation products like RTW-EC enables us to generate C/C++ code for embedded controllers (microprocessor, DSP, etc.) from the controller model. Figure 1-1 shows the concept of MBD for control systems based on MATLAB product family.

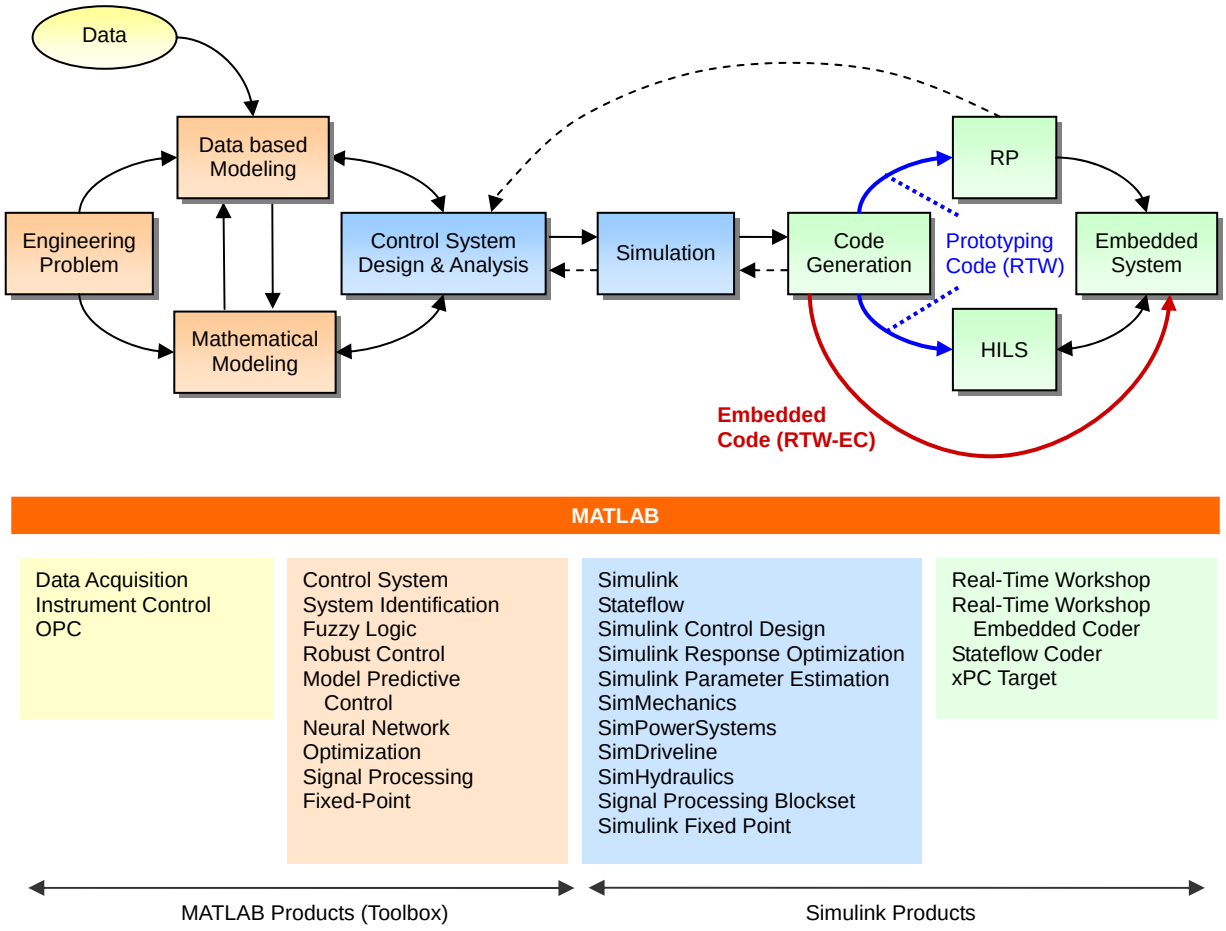


Figure 1-1 MBD for control systems based on MATLAB product family

1.2 V-process

The V-process showed in Figure 1-2 describes the MBD development process for control systems. The V-process consists of the Design, Coding, and Test stage. Each test stages correspond to the appropriate Design stages. A developer makes plant/controller models in the left side of the V-process for early improvement of controller algorithm, and reuses the models in the right side of it for improvement of code verification and validation.

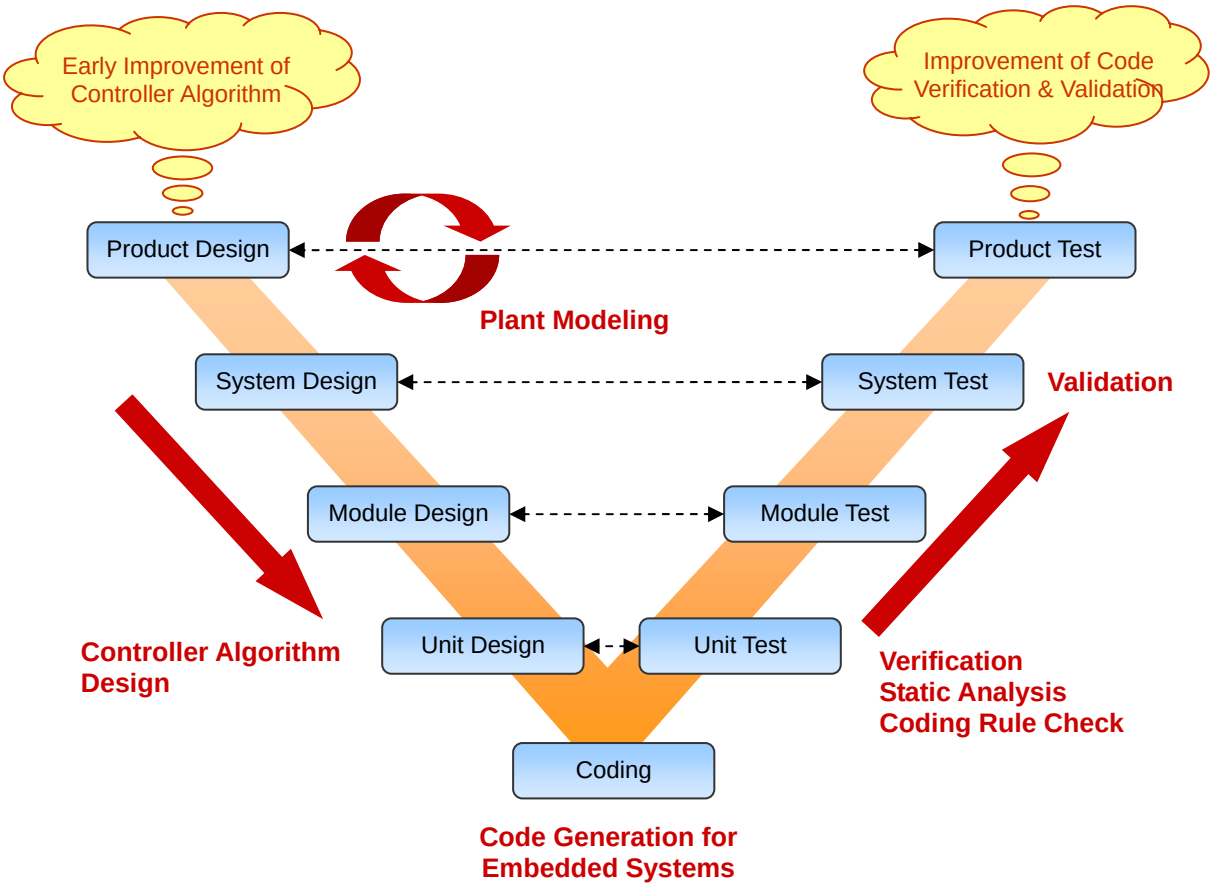


Figure 1-2 V-process for control systems

1.3 Merits of MBD

MBD has the following merits.

- Detection pf specification errors in early stage of development
- Hardware prototype reduction and fail-safe verification by real-time simulation
- Efficient test by model verification
- Effective communication by model usage
- Coding time and error reduction by auto code generation

2 Product design

This chapter describes the outline product design of Model-Based Design.

2.1 What is an NXT Scanner and an NXT Viewer?

The NXT Scanner is a sheet feed image scanner and is built with Lego Mindstorms NXT. The NXT Viewer is an image viewer and an image processor in MATLAB. The NXT Scanner and The NXT Viewer are expressed by UML use case diagram as in Figure 2-1 and Figure 1-2. Each case is shown by use case description as shown in Table 2-1 to Table 2-4.

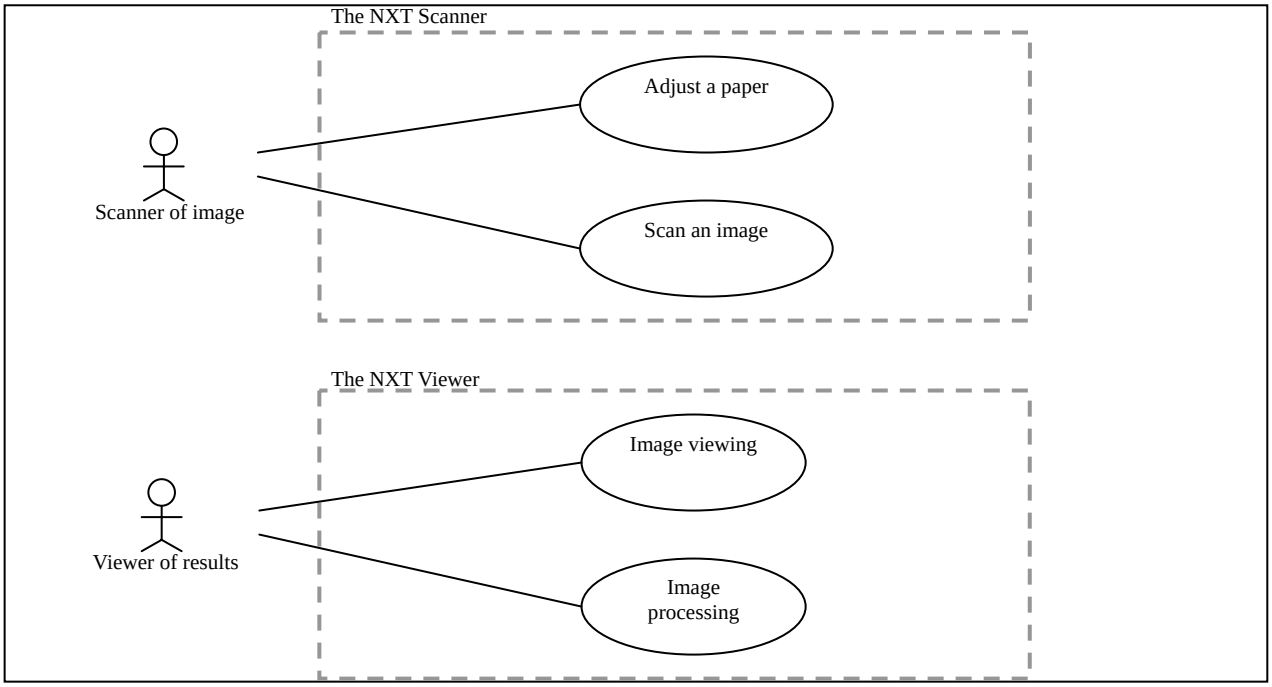


Figure 2-1 Essential UML use case diagram

Table 2-1 Use case description (adjust a paper position)

Use case name	The NXT Scanner Adjust a paper position
Normal flows	1. Actor requests “paper feed” or “paper exhaust” to the NXT Scanner. 2. The NXT Scanner starts adjusting the position of paper. 3. Actor requests stopping to adjust to the NXT Scanner. 4. The NXT Scanner stops adjusting.
Alternate flows	If the NXT Scanner does not succeed, it should stop the whole process.

Table 2-2 Use case description (Scan an image)

Use case name	The NXT Scanner Scan an image
Normal flows	1. Actor requests “start scan” to the NXT Scanner 2. The NXT Scanner starts scan. 3. The NXT Scanner gets data from the light sensor and moves scanner head. 4. The NXT Scanner sends scanning data to the NXT Viewer via the USB port. 5. The NXT Scanner feeds a paper after the end of scan of line. 6. The NXT Scanner keeps on scanning until finished all scan or actor requests to stop scan.
Alternate flows	If the NXT Scanner does not success, it should stop the full process.

Table 2-3 Use case description (View an image)

Use case name	The NXT Viewer View an image
Normal flows	<< **in real time** >> 1. Actor requests “start scan” to the NXT Scanner. 2. The NXT Viewer receives an image data from the Scanner via USB. 3. The NXT Viewer updates the display every line of scanning image. 4. Actor can save scan image. **<< load >>** 1. Actor can load scan image.
Alternate flows	Nothing

Table 2-4 Use case description (Image processing)

Use case name	The NXT Viewer Image processing
Normal flows	1. Actor requests “image processing” for desired image (real time or load). 2. The NXT Viewer processes image processing. 3. The NXT Viewer displays the result of image processing.
Alternate flows	Nothing

3 Mechanisms for the NXT Scanner

This chapter describes the hardware structure that consist sensors and actuators.

3.1 Hardware structure

Figure 3-1 shows structure of the NXT Scanner.

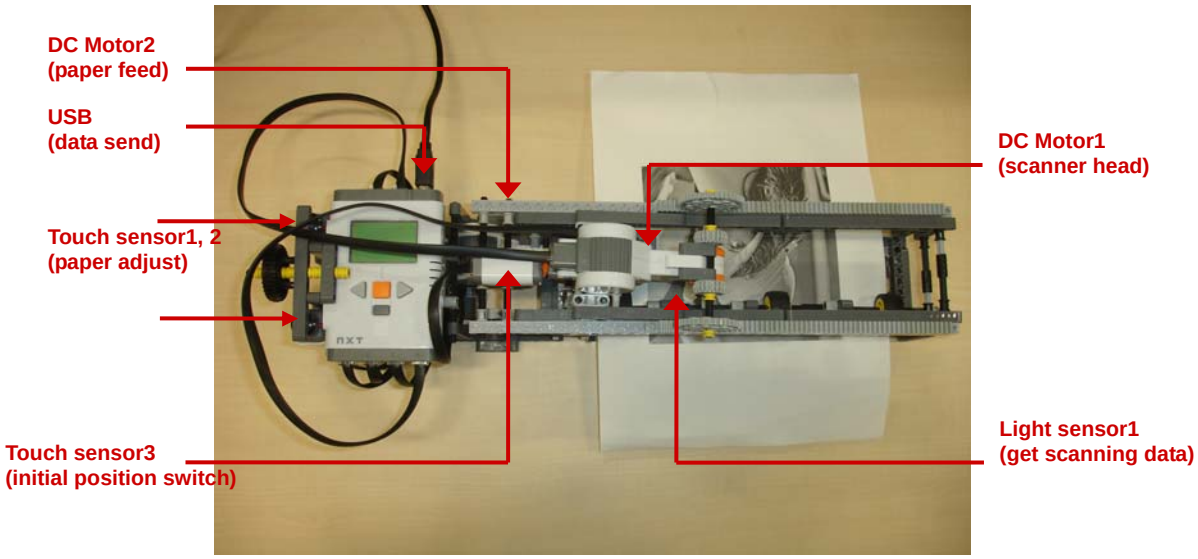


Figure 3-1 The NXT Scanner

The NXT Scanner has following functional categories:

- A user interface unit
- A scanner head unit
- A paper feed unit

A user Interface unit

Figure 3-2 is the user interface unit of the NXT Scanner.

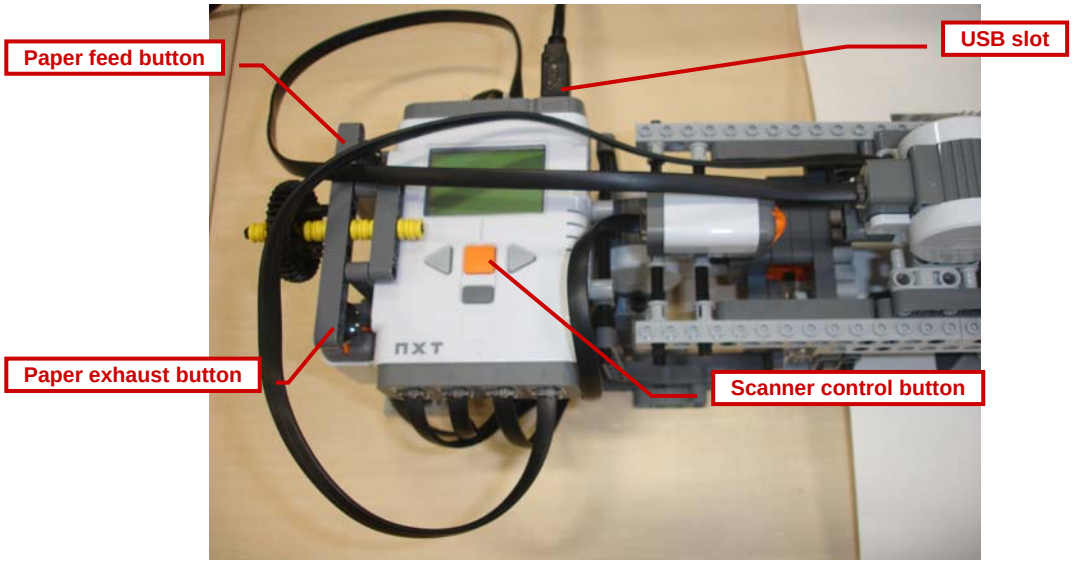


Figure 3-2 The NXT Scanner user interface

A scanner head unit

The scanner head unit moves on the NXT Scanner in order to acquire scan data. Figure 3-3 is an overview of the scanner head unit. The scanner head unit is converted by rotation of scanner head control motor into horizontal position control by the rack and pinion gear system. The scanner head unit has a scan sensor which can receive image data. The purpose of using the scanner head initial position switch is detect the initial position and for protection against overrun.

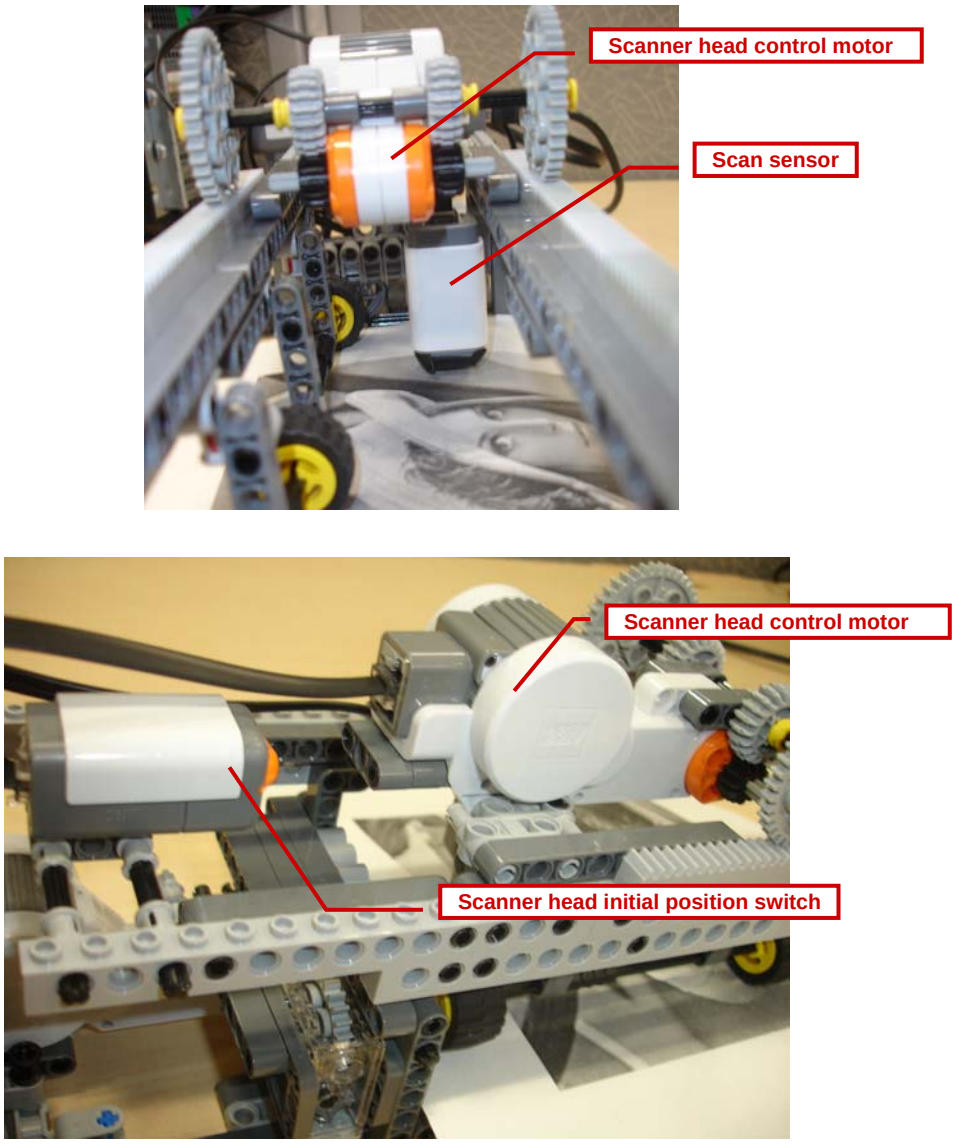


Figure 3-3 Scanner head unit

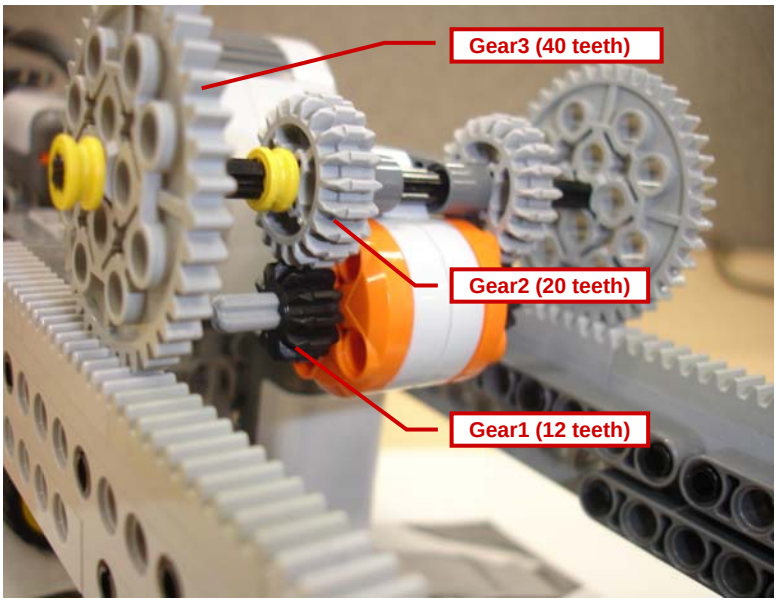


Figure 3-4 Scanner head gear system

The gear reduction ratio g is obtained by the following equation:

$$g = \frac{gear2teeth}{gear1teeth} = \frac{20}{12} = 1.667 \tag{2.1}$$

Gear3 is 2.0292 [cm] in radius, and a circumference of 12.75 [cm]. So the moving distance of the scanner head unit per degree of motor rotation d is given by:

$$d = \frac{12.75[cm]}{360[\text{deg}]*1.667} = 0.0215[cm] \tag{2.2}$$

A paper feed unit

The paper feed unit has a rubber roller. It is rolled for adjusting the paper position by the paper feed control motor. Figure 3-5 and Figure 3-5 are overviews of the paper feed unit. The paper feed unit uses a worm gear which can change the axis of rotation based on the power of the paper feed control motor. So the paper feed unit can control the position of a paper.

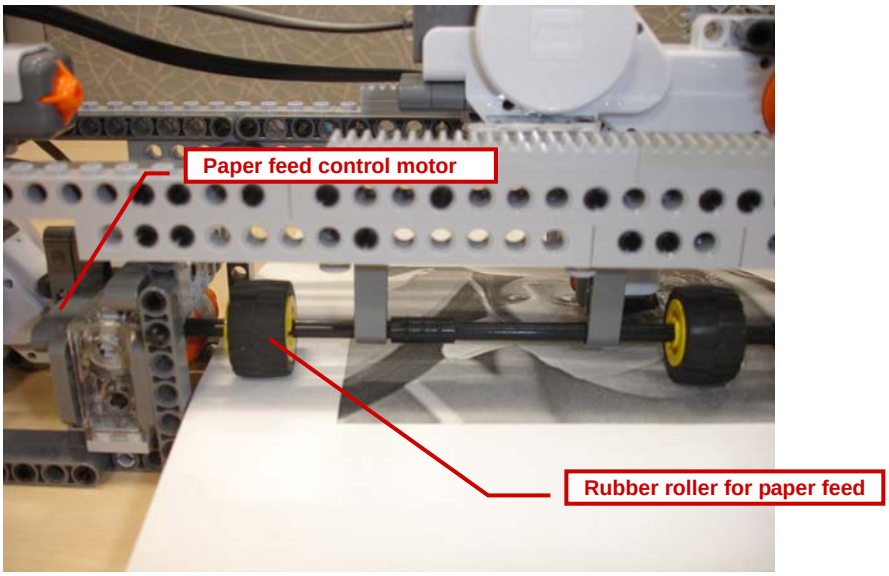


Figure 3-5 Paper feed unit

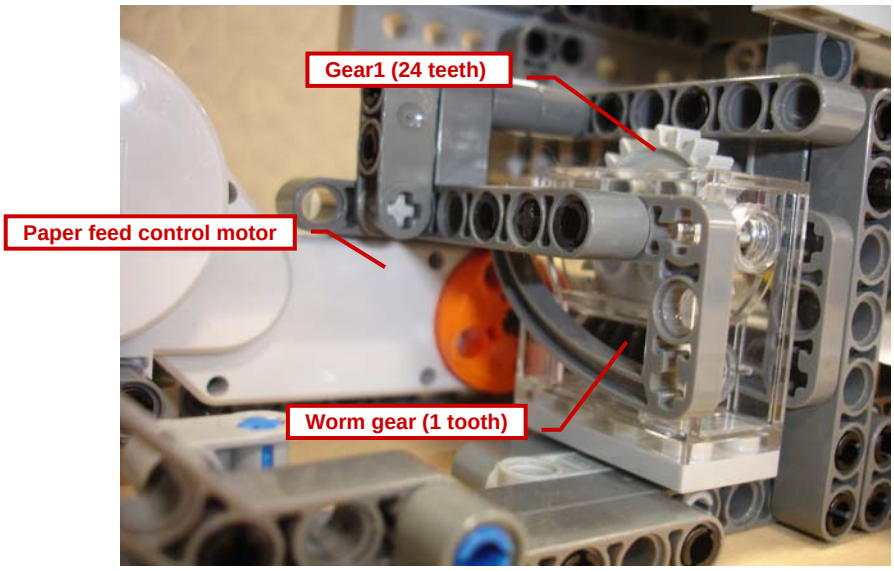


Figure 3-6 Paper feed gear system

The gear reduction ratio g is found by using the following equation:

$$g = \frac{gear1teeth}{1} = \frac{24}{1} = 24 \qquad (2.3)$$

The rubber roller for paper feed is 1.24141 [cm] in radius, and a circumference of 7.8 [cm]. So the moving distance of the paper feed per degree of motor rotation d is given by:

$$d = \frac{7.8[cm]}{360[\text{deg}] * 24} = 9.0278 * 10^{-4}[cm] \tag{2.4}$$

3.2 Backlash

There is a backlash, sometimes called lash or play, between the gears. The backlash has a negative impact on the tracking accuracy because it results in some lost motion when movement is reversed and contact is reestablished. It is necessary to compensate for backlash when engaging the gears. Figure 3-7 shows a basic concept of the backlash. Here, the engaged state means, gears are engaged and the disengaged state means, they are not engaged.

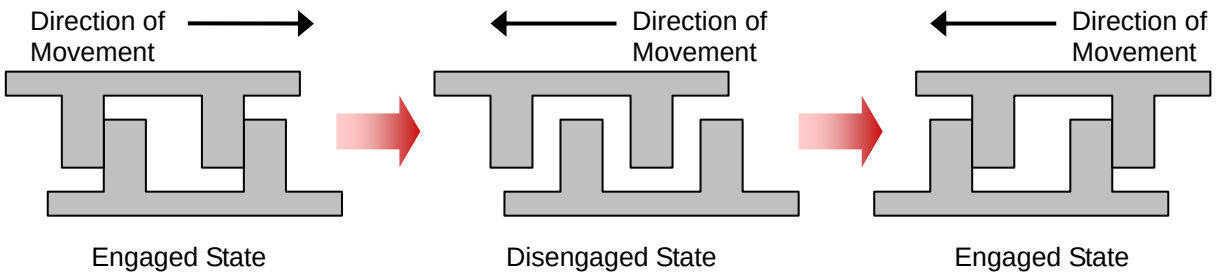


Figure 3-7 Backlash (The upper side is drive gear and under side is driven gear)

3.3 Sensors and Actuators

Table 3-1 and Table 3-2 show sensors and actuators properties:

Table 3-1 Sensor properties

Sensor	Output	Unit	Data Type	Maximum Sample [1/sec]
Rotary encoder	Angle	deg	int32	1000
Touch sensor	Touch ON/OFF		int8	1000
Light sensor	Reflected infrared		uint16	1000

Table 3-2 Actuator properties

Actuator	Input	Unit	Data Type	Maximum Sample [1/sec]
DC motor	PWM	%	int8	500

The reference [2] illustrates many properties of the DC motor. In general, sensors and actuators are individually different.

4 System design

This chapter describes the system design.

4.1 Overview of total system

Figure 4-1 is an architectural concept for the NXT Scanner. The NXT Scanner has four control algorithms, “MODE CONTROL SYSTEM”, “PAPER FEED CONTROL SYSTEM”, “SCAN CONTROL SYSTEM” and “USB COMMUNICATION CONTROL SYSTEM”.

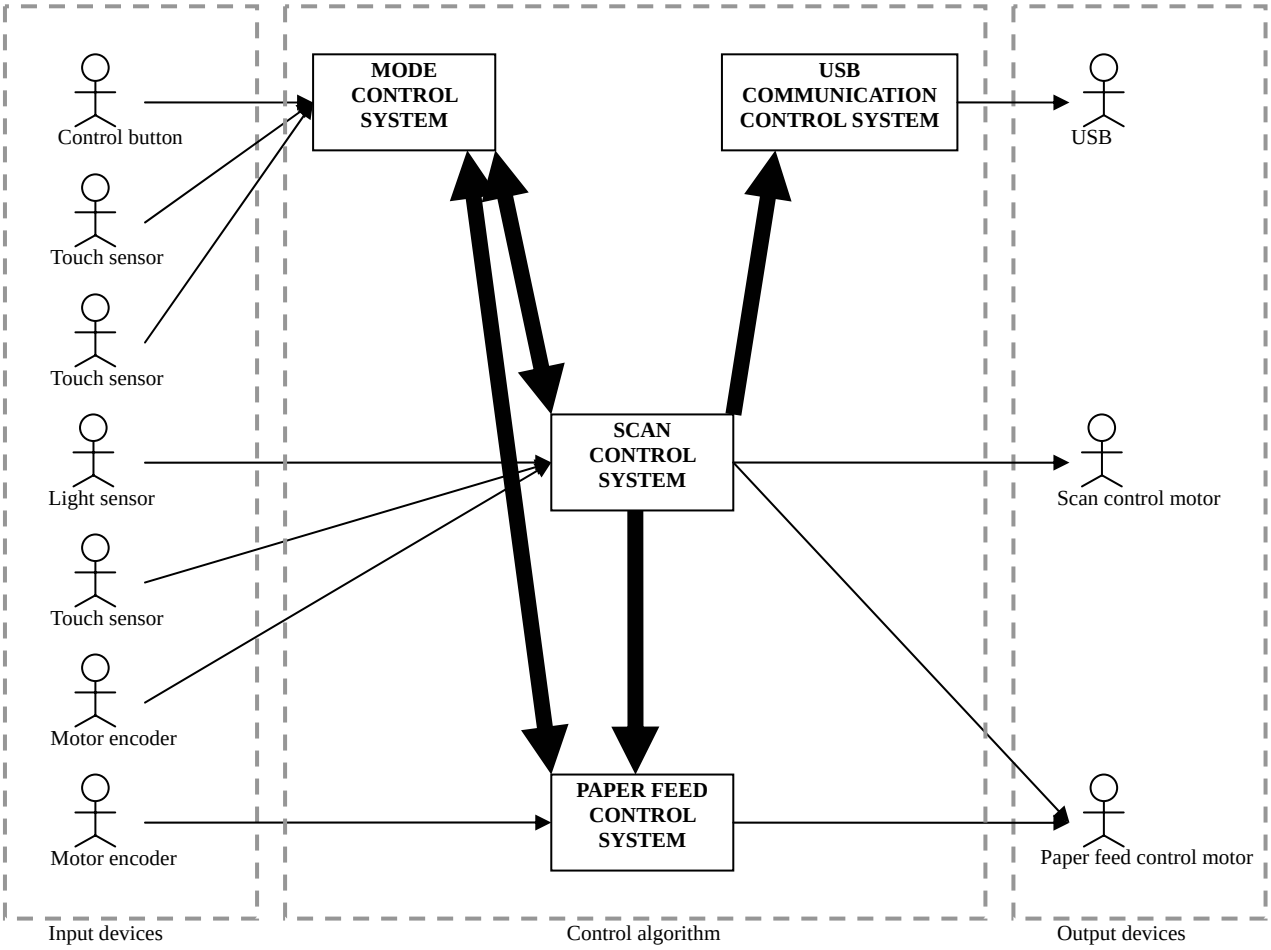


Figure 4-1 An architectural concept

4.2 MODE CONTROL SYSTEM

An internal control mode of the NXT Scanner is changed by active invents (user operation etc.) in the MODE CONTROL SYSTEM. The control mode has six states, “initialization mode”, “idle mode”, “paper feed mode”, “paper exhaust mode”, “scan mode” and “error mode”. Figure 4-2 is state transition diagram for the MODE CONTROL SYSTEM.

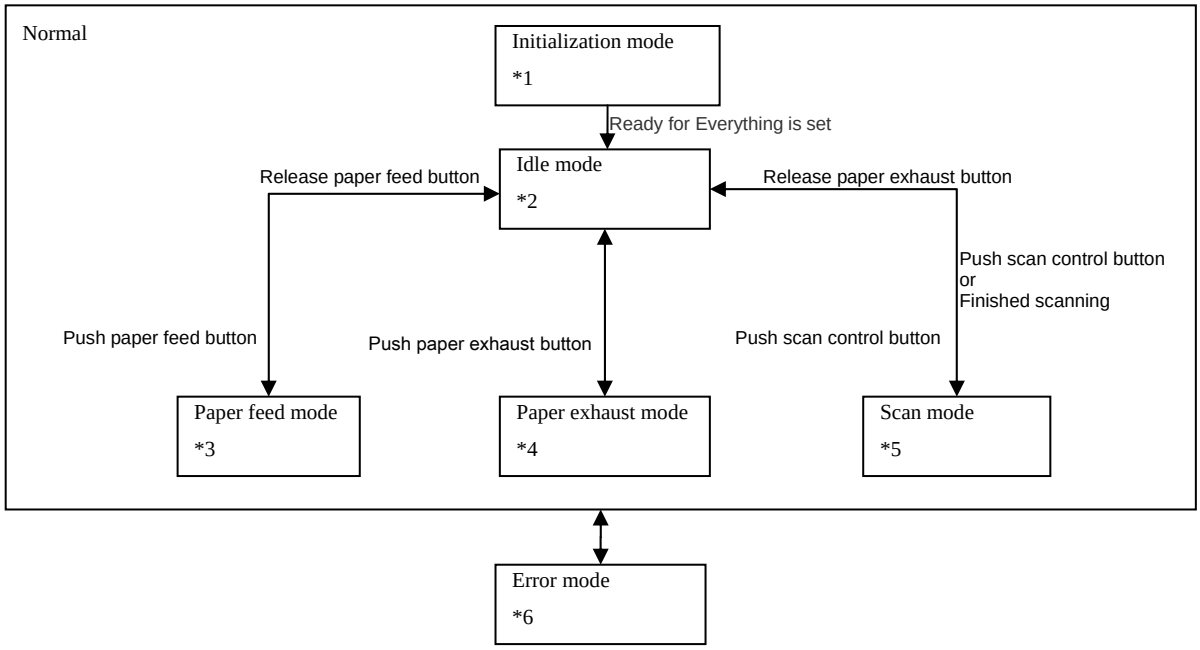


Figure 4-2 State transition diagram for the MODE CONTROL SYSTEM

Table 4-1 The MODE CONTROL SYSTEM additional information

system name	MODE CONTROL SYSTEM
the point	Control the control mode
explain state	1. Initialization mode: during initialization the SCAN CONTROL SYSTEM or the PAPER FEED CONTROL SYSTEM 2. Idle mode: ready and waiting event 3. Paper feed mode: doing paper feed 4. Paper exhaust mode: doing paper exhaust 5. Scan mode: doing scan 6. Error mode: an error has just happened in the SCAN CONTROL SYSTEM or the PAPER FEED CONTROL SYSTEM
note	・ Sound a buzzer during error mode because it differentiates the system in which the error happened. ・ The MODE CONTROL SYSTEM should keep checking the SCAN CONTROL SYSTEM because the control mode should change when scanning is finished.

The SCAN CONTROL SYSTEM and the PAPER FEED CONTROL SYSTEM are controlled by the control mode. Figure 4-3 and Figure 4-4 below are examples of normal and abnormal cases. The control mode is an event request to the SCAN CONTROL SYSTEM and the PAPER FEED CONTROL SYSTEM.

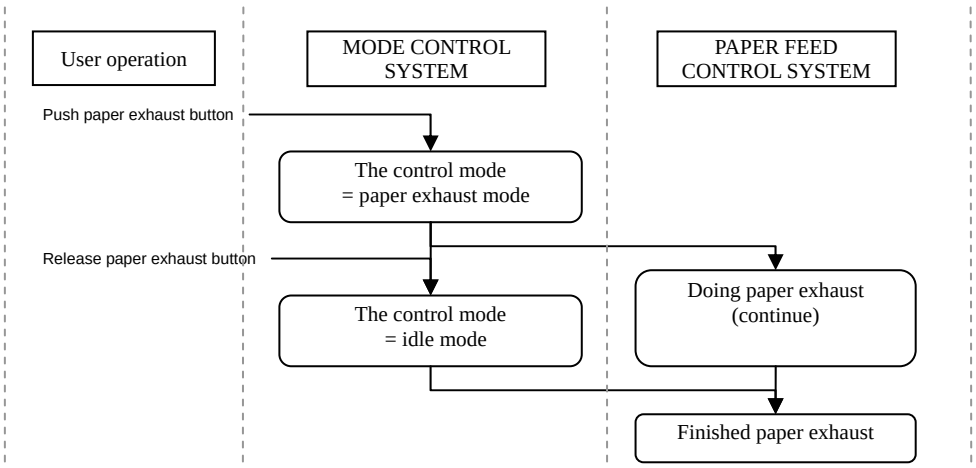


Figure 4-3 Handling the control mode PAPER FEED CONTROL SYSTEM flow in normal case

The system in which an abnormal operation occurred should inform the MODE CONTROL SYSTEM by an error message. The CONTROL MODE SYSTEM receives it and sends it to the whole system by changing the control mode to error mode.

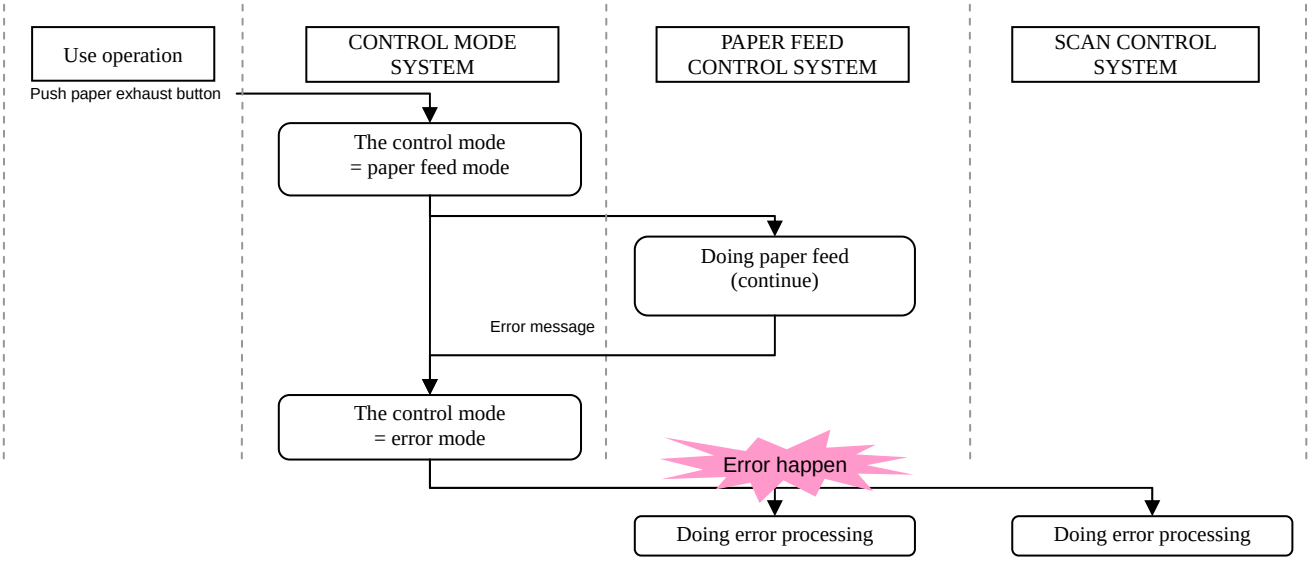


Figure 4-4 Handling the control mode PAPER FEED CONTROL SYSTEM flow in abnormal case

4.3 PAPER FEED CONTROL SYSTEM

The role of the PAPER FEED CONTROL SYSTEM is to adjust the paper position on the NXT Scanner. There are these cases for positioning a paper i.e., before scanning and during scanning. Before scanning case uses the control mode but during scanning case uses the control mode followed by an instruction for the SCAN CONTROL SYSTEM.

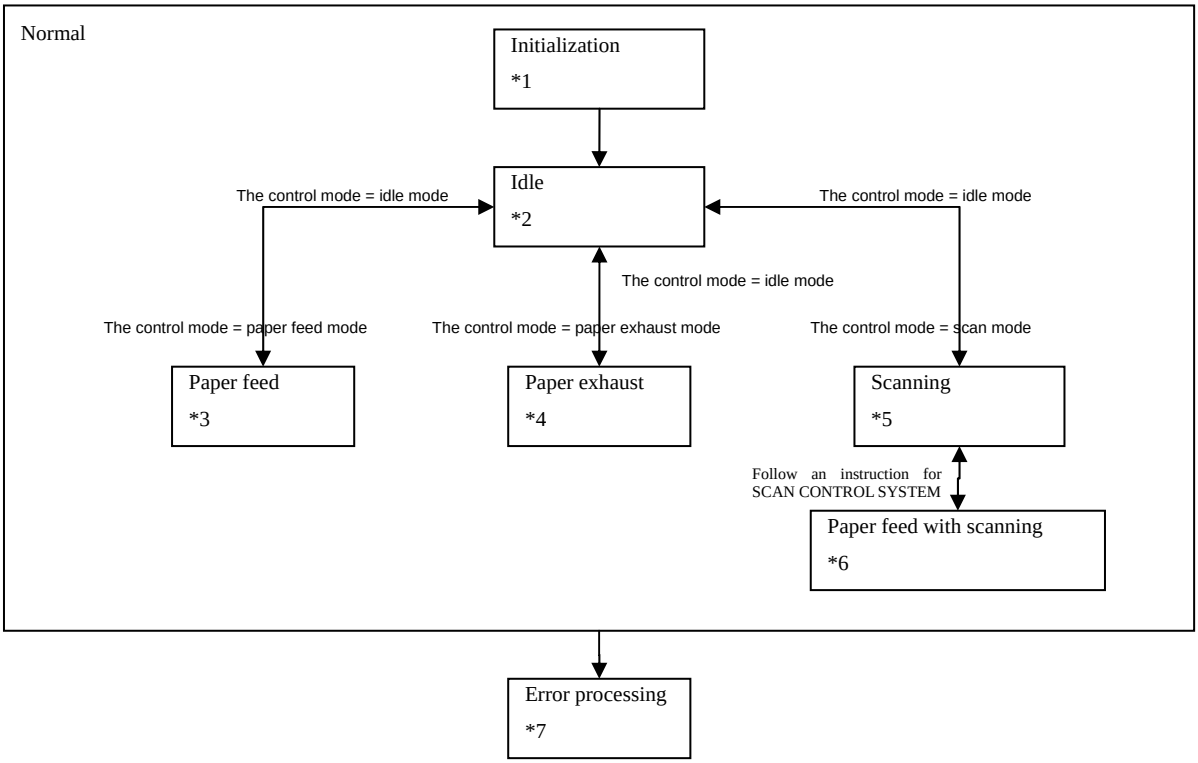


Figure 4-5 State transition diagram for the PAPER FEED CONTROL SYSTEM

Table 4-2 PAPER FEED CONTROL SYSTEM additional information

system name	PAPER FEED CONTROL SYSTEM
the point	Control paper position on the NXT Scanner
explain state	1. Initialization: doing paper feed for initial processing 2. Idle: ready and waiting event 3. Paper feed: doing paper feed 4. Paper exhaust: doing paper exhaust 5. Scanning: scanning and do not paper feed 6. Scanning with paper feed: scanning and doing paper feed 7. Error processing: an error has just happened in the SCAN CONTROL SYSTEM or the PAPER FEED CONTROL SYSTEM
note	・ If an error has happened in the PAPER FEED CONTROL SYSTEM then it should inform the MODE CONTROL SYSTEM by an error message. ・ Stopped paper feed in error processing.

4.4 SCAN CONTROL SYSTEM

The SCAN CONTROL SYSTEM's working to move the scanner head unit and scanning. Figure 4-6 shows the scanning points. It scans the range of about 10cm^2 on the paper for 255*127 points. To speed up the scan, the number of the scan points in the lengthwise direction is half that of the cross direction. Instead, the NXT Viewer adjusts the missing data of the lengthwise direction by image processing. The result is 255*254 data points.

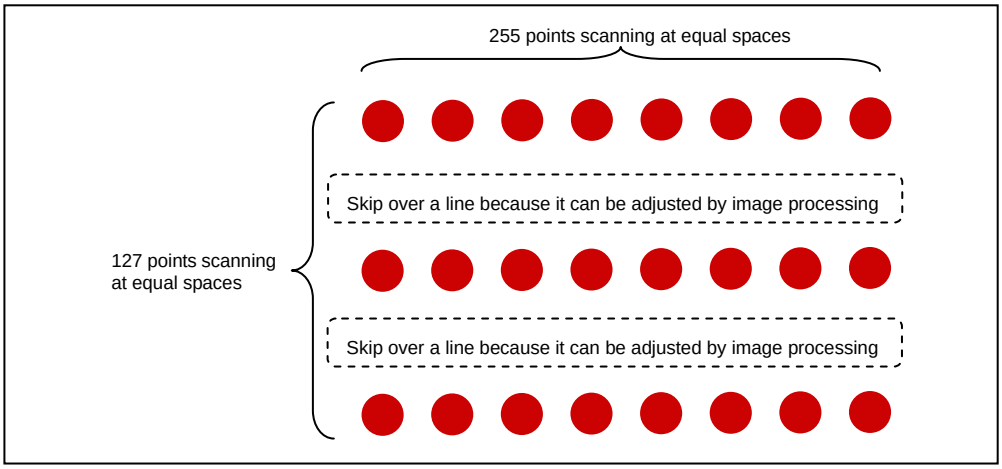


Figure 4-6 Scanning points image (interlace scan)

The SCAN CONTROL SYSTEM has a sequentially algorithm. When scanning, it should take turns at moving the scanner head, getting an image data and requests for paper feed.

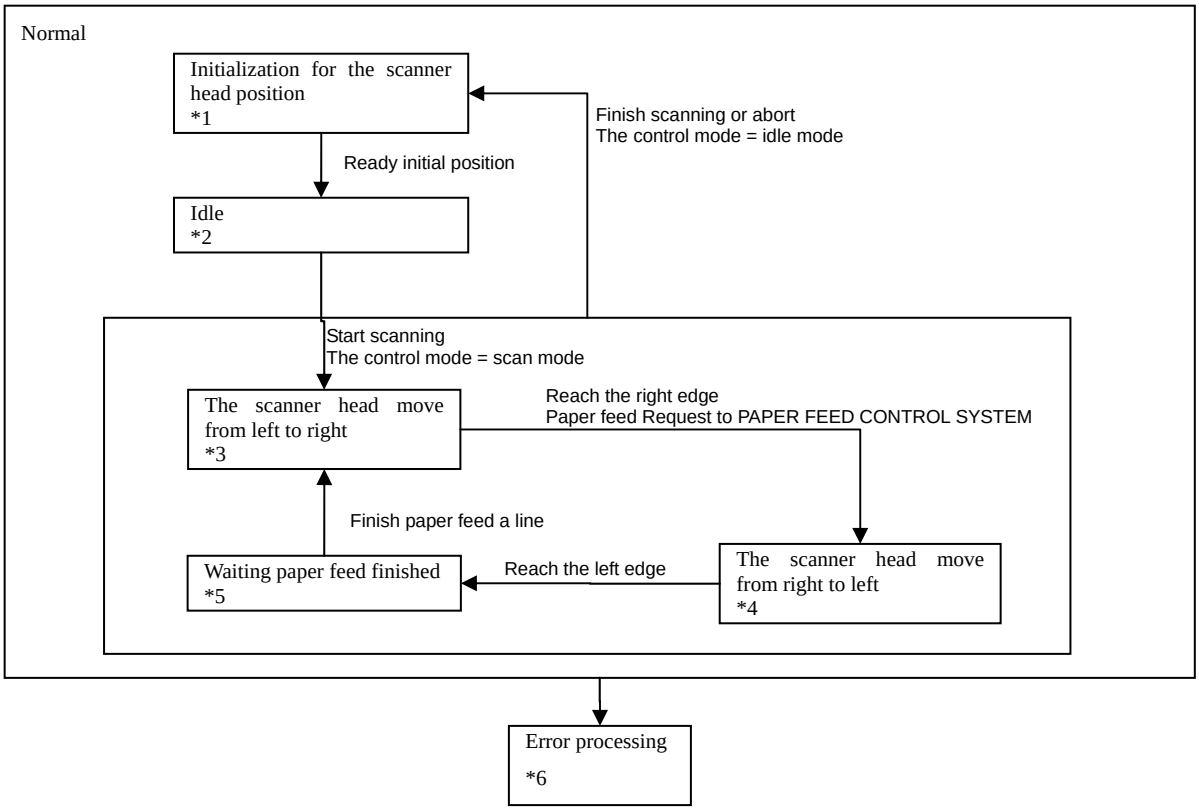


Figure 4-7 State transition diagram for the SCAN CONTROL SYSTEM

Table 4-3 SCAN CONTROL SYSTEM additional information

system name	SCAN CONTROL SYSTEM
the point	Control of scanning of the NXT Scanner
explain state	1. Initialization for the scanner head position: the scanner head is moving to start position (continue scanner head move from right to left for detect switch of the scanner head detection switch) 2. Idle: the scanner head has been ready and waiting for an event 3. The scanner head moves from left to right: receiving from the light sensor (paper feed request will send to the right edge) 4. The scanner head moves from right to left: only moving and do not scanning (paper feed processing is background process) 5. Waiting paper feed processing: waiting message from the PAPER FEED CONTROL SYSTEM 6. Error processing: an error has just happened in the SCAN CONTROL SYSTEM or the PAPER FEED CONTROL SYSTEM
note	・ If the SCAN CONTROL SYSTEM has failed then it should inform the MODE CONTROL SYSTEM by an error message. ・ Getting from the light sensor only when the scanner head moves left to right. ・ After finishing the scan, the scanner head unit should go back to the initialize position. ・ An error occurs, Scanner head stops.

4.5 USB COMMUNICATION CONTROL SYSTEM

The USB COMMUNICATION CONTROL SYSTEM sends scanning data to the NXT Viewer via the USB port. USB communication uses original protocol between the NXT Viewer and the NXT Viewer. The USB COMMUNICATION CONTROL SYSTEM has two buffers for scan data as shown in Figure 4-8, and they are checked every time. When full, it will send the buffer data. It is an asynchronous process between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM. You should require scrupulous attention to the task cycle of the USB COMMUNICATION CONTROL SYSTEM. Generally if the task cycle design is not good, buffer overflow will happen, overwriting, and badly impacts system performance.

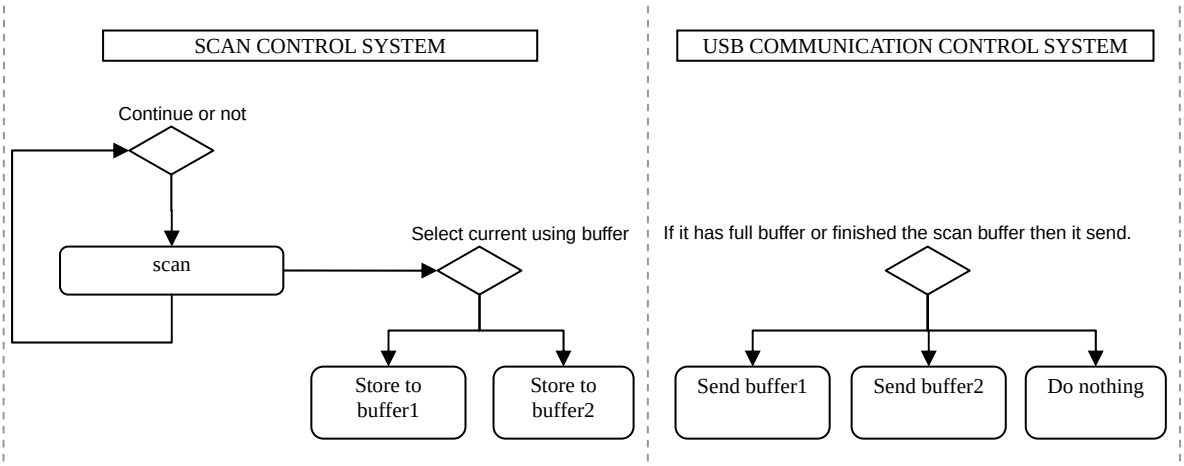


Figure 4-8 The flow chart for USB COMMUNICATION CONTROL SYSTEM

5 Module design for the NXT Scanner

This chapter describes the outline module design for the NXT Scanner.

5.1 Using library model for functional unit models

The NXT Scanner has functional unit models which are designed by system design. Each functional unit model has test environmental. It will be registered pass a validation model with the library model. Each functional unit model is described at next section.

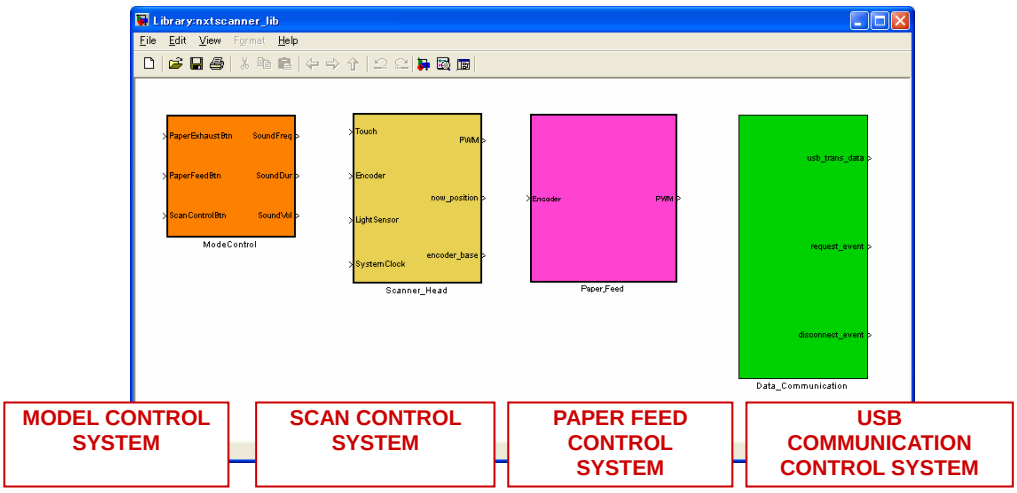


Figure 5-1 Library model

5.2 Shared data (Global variable data) for the total system

Table 5-1 shows shared data for the total system. It means global variable data for the total system.

Table 5-1 Shared data for total system

Data name	Detail
The control mode: (ControlMode)	・ Control mode for total system and data type is enumerated type ERROR (error states) INITIALIZE (each unit is doing initializing) IDLE (do nothing) PAPERFEED (doing paper feed) PAPEREXHAUST (doing paper exhaust) SCAN (scanning) ・ Initial value is INITIALIZE. ・ Data scope is total system ・ Writable is only the MODE CONTROL SYSTEM

The error code: (ErrorCode)	・ Error status for total system and data type is enumerated type NO_ERROR (expected states) ERROR_001 (the scanner head system has error) ERROR_002 (paper feed has error) ・ Initial value is NO_ERROR. ・ Data scope is total system.
The scan mode: (ScanMode)	・ Scan mode in the SCAN CONTROL SYSTEM and data type is enumerated type SCAN_INIT (initialization) SCAN_IDLE (ready and waiting event) SCAN_SCAN (scanning) SCAN_FEED (paper feed with scanning) ・ Initial value is SCAN_INIT. ・ Data scope is total system. ・ Writable is only the SCAN CONTROL SYSTEM
The paper feed mode: (FeedMode)	・ Paper feed mode for the SCAN CONTROL SYSTEM and the PAPER FEED CONTROL SYSTEM and data type is enumerated type FEED_INIT (initialization) FEED_IDLE (ready and waiting event) FEED_PAPERFEED (doing paper feed or exhaust) ・ Initial value is FEED_INIT. ・ Data scope is total system. ・ Writable are only the SCAN CONTROL SYSTEM and the PAPER FEED CONTROL SYSTEM.
The scan buffer information: (ScannerBuffer_Info)	・ Basically buffer information for between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM and data type is enumerated type TRANS_FINISH (finished data transfer) STORING (storing buffer) FULL_UP (full buffer) ・ Initial value is TRANS_FINISH. ・ Data scope is total system. ・ Writable are only the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM.
The scan buffer status: (ScannerBuffer_Status)	・ Buffer status for between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM ・ After using the scan buffer, it has to initialize to ZERO. ・ Initial value is ZERO. ・ ELP (end-of-line packet information) :bit4 0: other than end-of-line packet (it means beginning or mid-packet)

	1: end-of-line packet (include final packet for scan end) ・ START (START information) :bit3 0: normal scanning 1 : START ・ STOP (STOP information) :bit2 0: normal scanning 1: STOP ・ DIR (direction for scanning) :bit1 0: to the right 1: to the left ・ VALID (identification information of valid data) :bit0 0: invalid data 1: valid data ・ Data scope is total system. ・ Writable are only the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM.
The scan buffer number: (ScannerBuffer_Number)	・ Number (0 to 30) of send data information for between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM. ・ After using the scan buffer, it has to initialize to ZERO. ・ Initial value is ZERO. ・ Data scope is total system. ・ Writable are only the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM.
The scan buffer data: (ScannerBuffer_Data)	・ Data of scan data for between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM. ・ After using the scan buffer, it has to initialize to ZERO. ・ Initial value is ZERO. ・ Data scope is total system. ・ Writable are only the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM.
The scan packet number: (ScannerBuffer_Pnum)	・ The total number of current scanning packets for between the SCAN CONTROL SYSTEM and the USB COMMUNICATION CONTROL SYSTEM. ・ After using the scan buffer, it has to initialize to ZERO. ・ Initial value is ZERO. ・ Data scope is total system. ・ Writable is only the SCAN CONTROL SYSTEM.

6 Unit design

This chapter describes the unit design and introduces new features of R2008b.

6.1 Using enumerated type (New feature of R2008b)

Each functional unit model uses enumerated type which is a new feature of R2008b. The NXT Scanner is:

- Enumerated type used in Data Store Memory block for using global variable data
- Using Simulink Data Object for setting custom storage class

How to use enumerated type

Enumerated type setting:

1. Define for enumerated type in M-script
2. Data Store Memory block property sets enumerated type

Concrete examples of setting

The control mode in the MODE CONTROL SYSTEM is defined by the global variable the ControlMode. It registers enumerated type named ControlModeEnum. The M-script has to register enumerated type as follows:

ControlModeEnum.m

```
classdef(Enumeration) ControlModeEnum < Simulink.IntEnumType
    enumeration
        ERROR(-1)       % error status
        INITIALIZE(0)   % each unit are doing initialization
        IDLE(1)         % doing nothing (no operation)
        PAPERFEED(2)    % doing paper feed
        PAPEREXHAUST(3) % doing paper exhaust
        SCAN(4)         % scanning
    end
end
```

Specifically, Figure 6-1 shows how to set for enumerated type for Data Store Memory block. Data type sets ControlModeEnum from M-script (ControlModeEnum.m). Initial value also sets INITIALIZE from M-script.

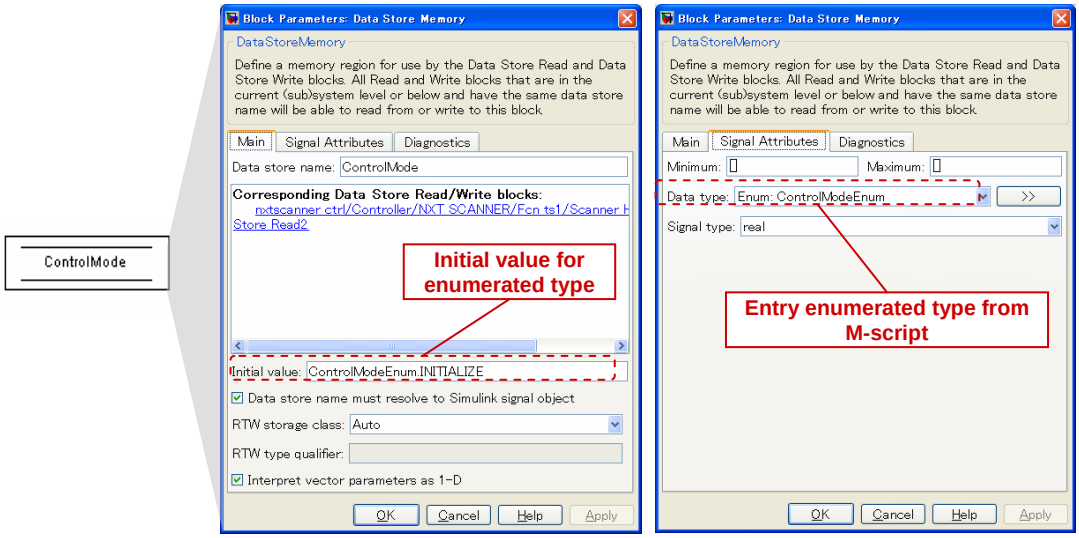


Figure 6-1 Enumerated type parameters

6.2 Utilize Simulink function (New feature of R2008b)

The PAPER FEED CONTROL MODEL uses Simulink function in Stateflow which is a new feature of R2008b. Simulink function can model Simulink subsystem of a function in Stateflow. Double-clicks the Simulink Function block to view or edit the Stateflow chart.

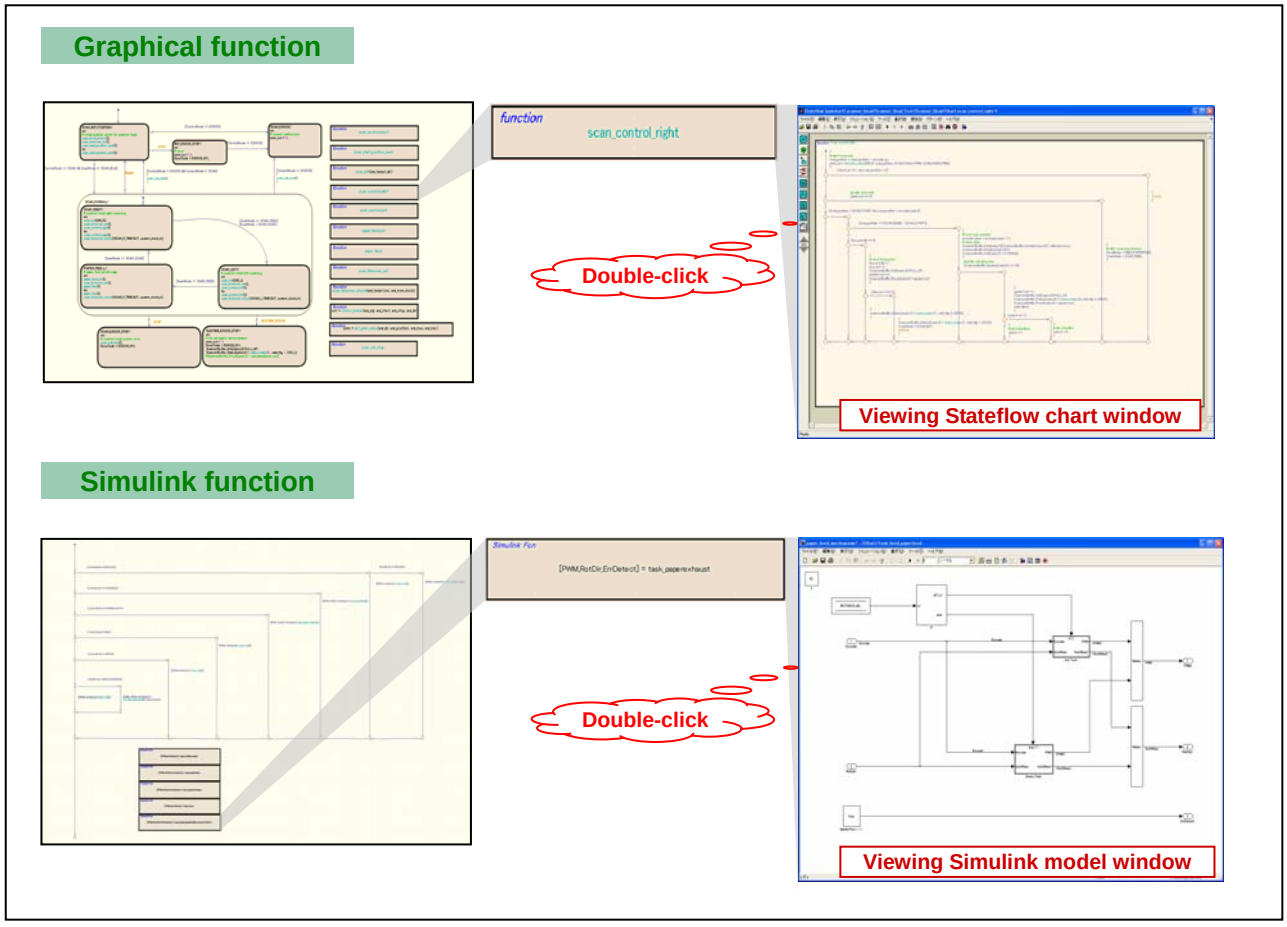


Figure 6-2 Difference between graphical and Simulink function

6.3 MODE CONTROL MODEL

The mode_control.mdl is a model for the MODE CONTROL SYSTEM. Figure 6-3 shows top layer of the CONTROL MODEL.

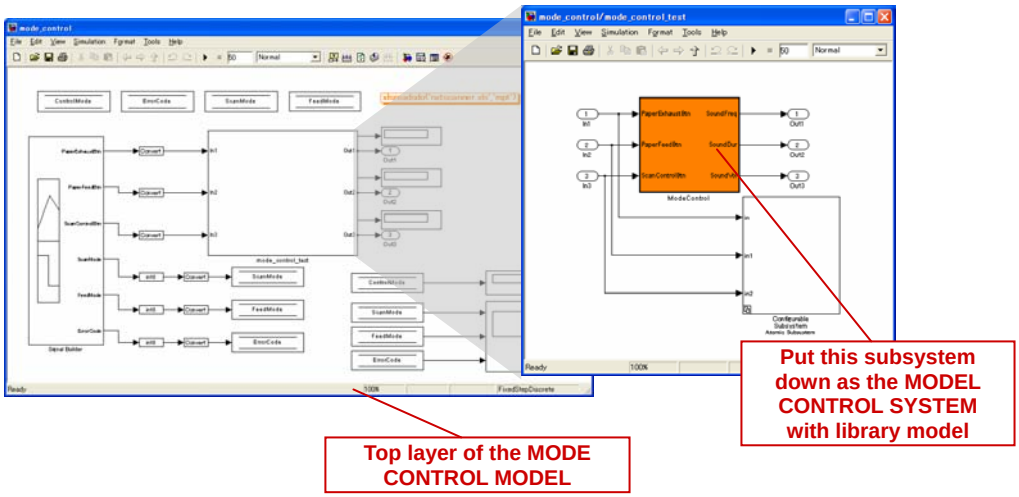


Figure 6-3 Relating the MODEL CONTROL MODEL to library model

This model provides input signals for simulation, and output signals for the purpose of monitoring in the top layer, just like in case of the other function models. By running the model, we can validate this model in unit function.

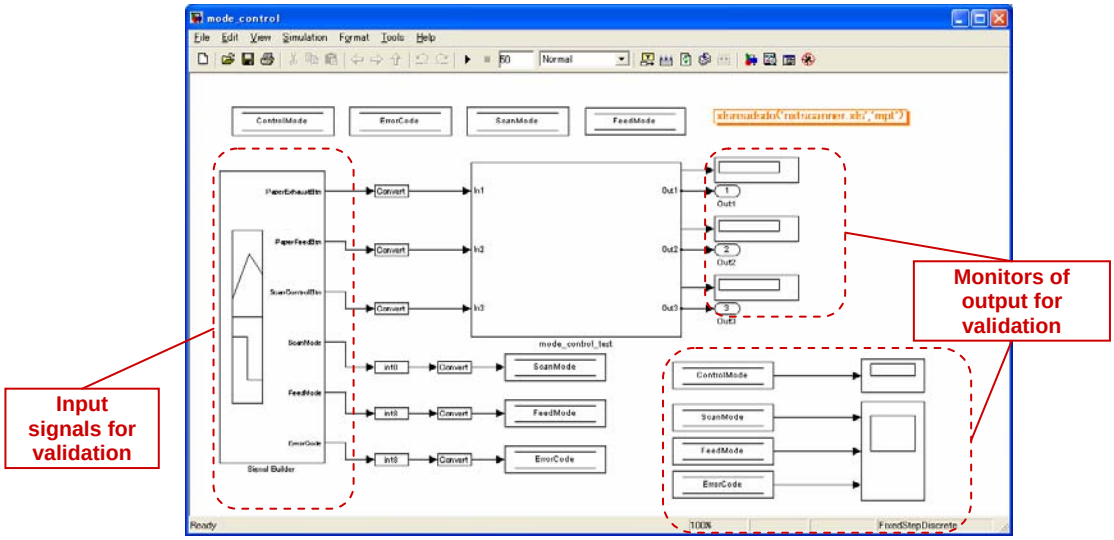


Figure 6-4 Top layer of the MODE CONTROL MODEL

Shared data

Global variable data to share between other tasks by uses of Data Store Memory blocks. Also the ErrorCode, the ScanMode and the FeedMode use the enumerated type and Data Store Memory block. The setup step is the same as that of the ControlMode.

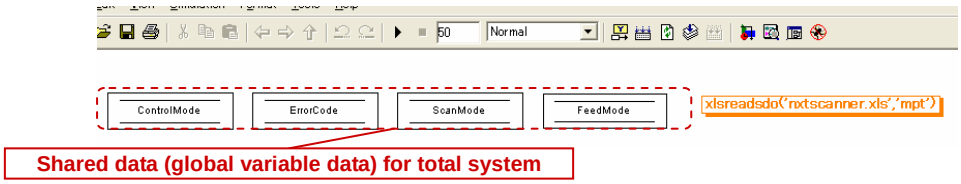


Figure 6-5 Shared data of the MODE CONTROL MODEL

Algorithm implementation

Figure 6-6 shows the main algorithm model. Input signals (the ScanControl) to Stateflow are shown by Figure 6-7. The ScanControl has two trigger patterns which are all finished and user operation. All finished trigger happens when the ScanMode changes state of to send a latest scan data. As shown in Figure 6-8, the Stateflow is the same as the one shown in system design of state transition diagram.

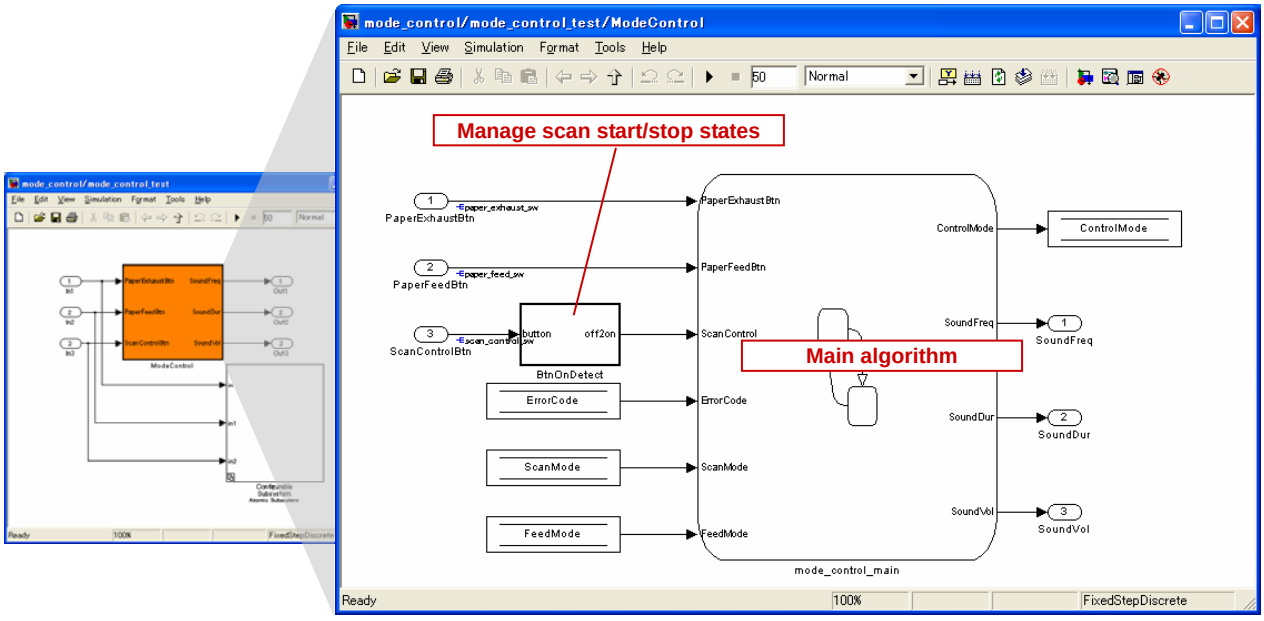


Figure 6-6 The MODE CONTROL MODEL and library model

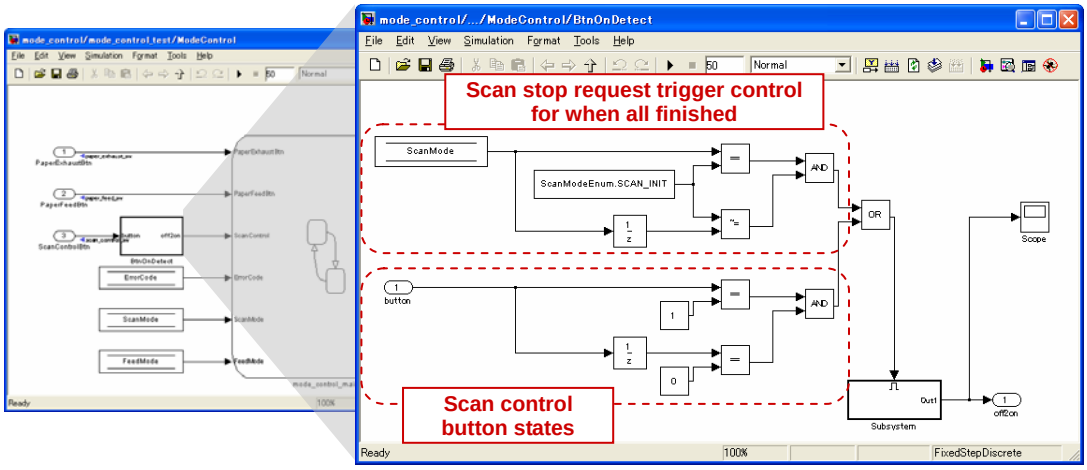


Figure 6-7 Manage scan start/stop states

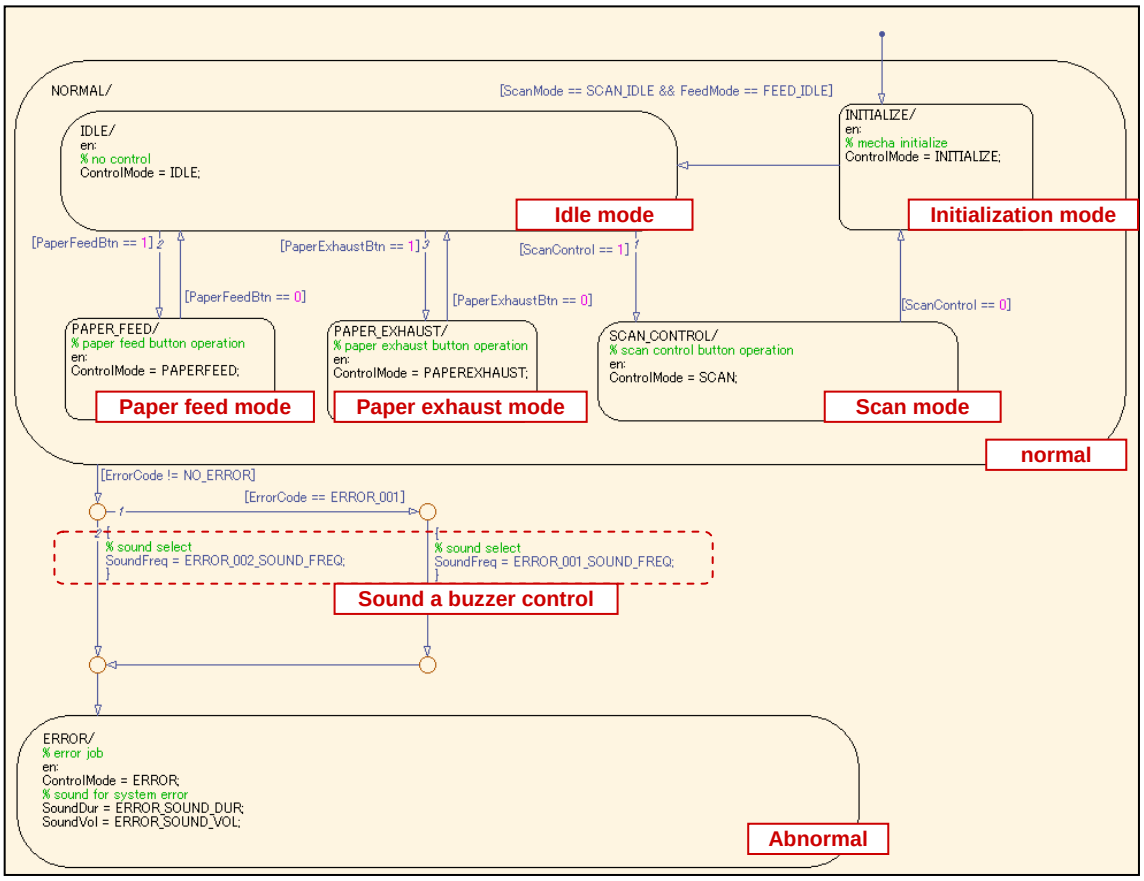


Figure 6-8 Main algorithm of the MODE CONTROL MODEL in stateflow

6.4 PAPER FEED CONTROL MODEL

The paper_feed_mechanism.mdl is a model for the PAPER FEED CONTROL SYSTEM. Figure 6-9 shows the top layer of the PAPER FEED CONTROL MODEL.

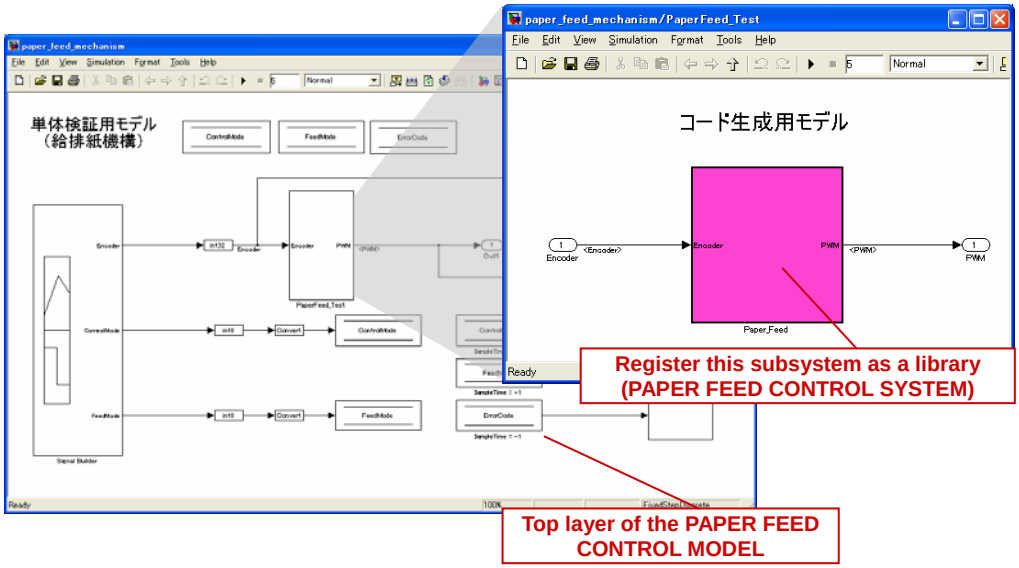


Figure 6-9 Relation the PAPER FEED CONTROL MODEL to library model

This model provides input signals for simulation, and output signals for the purpose of monitoring in the top layer, just like in case of the other function models. By running the model, we can validate this model in unit function.

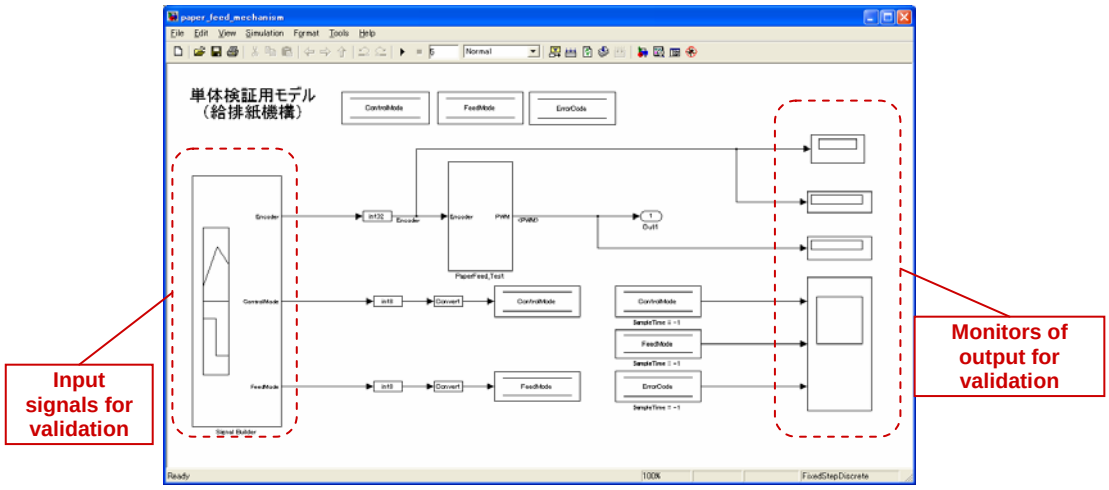


Figure 6-10 Top layer of the PAPER FEED CONTROL SYSTEM

Algorithm implementation

The PAPER FEED CONTROL SYSTEM consists of two parts, the control flow part (written by Stateflow) and the error detection part (written by Simulink model). See Figure 6-11.

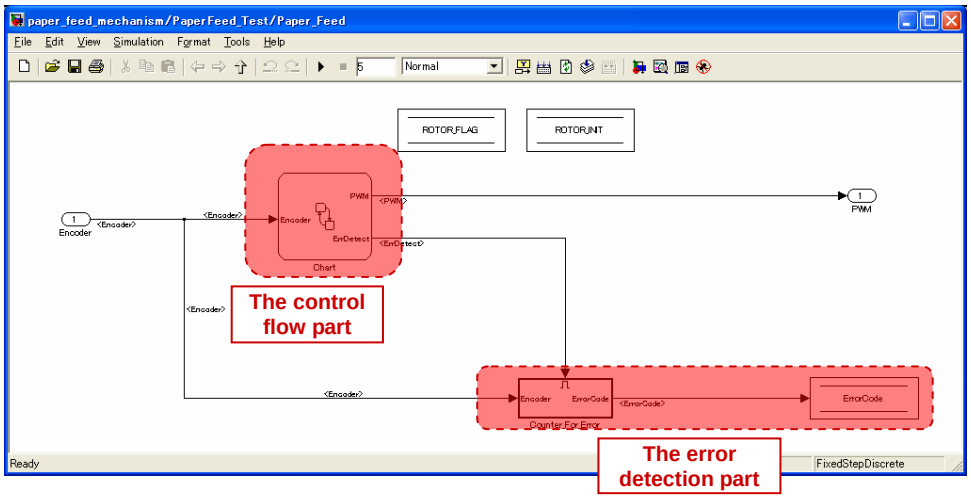


Figure 6-11 Top layer of the PAPER FEED CONTROL

The control flow part

Figure 6-12 is the control flow part written in the flow chart style. The branching algorithm processes the data (the ControlMode, the PaperFeedMode) shared with the other control system models. The Simulink Functions called from this chart are placed at the bottom of this chat. See Table 6-1 for details of each Simulink Function.

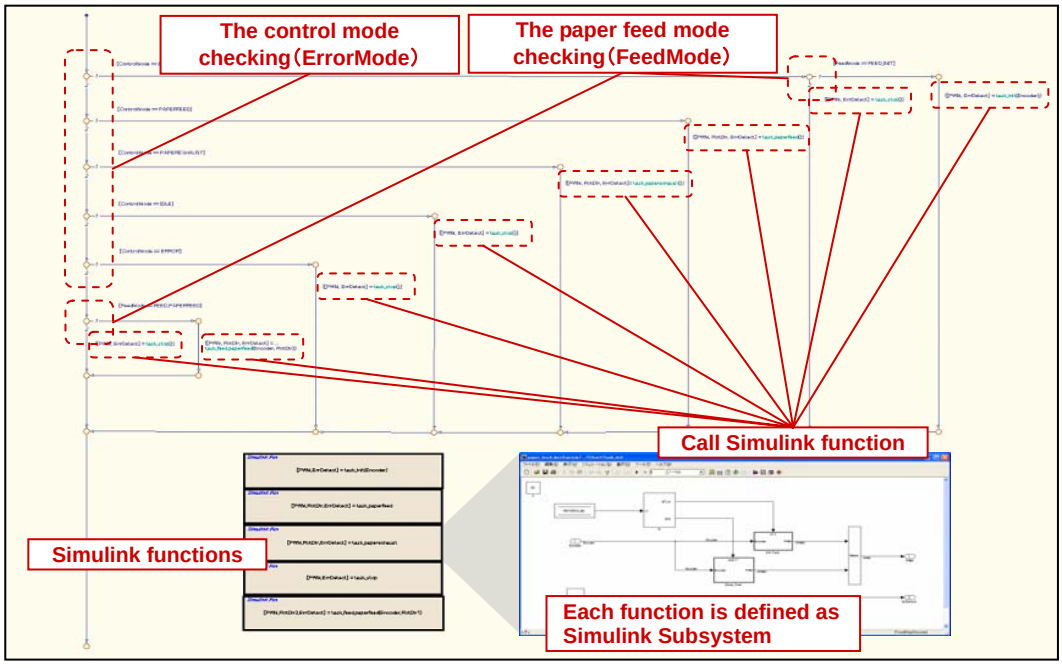


Figure 6-12 The control flow part of the PAPER FEED CONTROL

Table 6-1 Simulink function list in the PAPER FEED CONTROL MODEL

Simulink function	Arguments	Return value	Note
task_init	encoder value (DC motor2 for paper feed)	current PWM value enable/disable flag (error judge)	initialize processing (After power-on, automatically move for initialization)
task_paperfeed	-	current PWM value paper move direction enable/disable flag (Error judge)	paper feed processing (paper feed button operation)
task_paperexhaust	-	current PWM value paper move direction enable/disable flag (Error judge)	paper exhaust processing (paper exhaust button operation)
task_stop		current PWM value enable/disable flag (Error judge)	stop of paper feed and exhaust processing
task_feed_paperfeed	encoder value (DC motor2 for paper feed)	current PWM value paper move direction enable/disable flag (Error judge)	paper feed with scanning processing

The algorithm of the control flow part

Figure 6-13 is the flow chat of the branching algorithm of the control flow part.

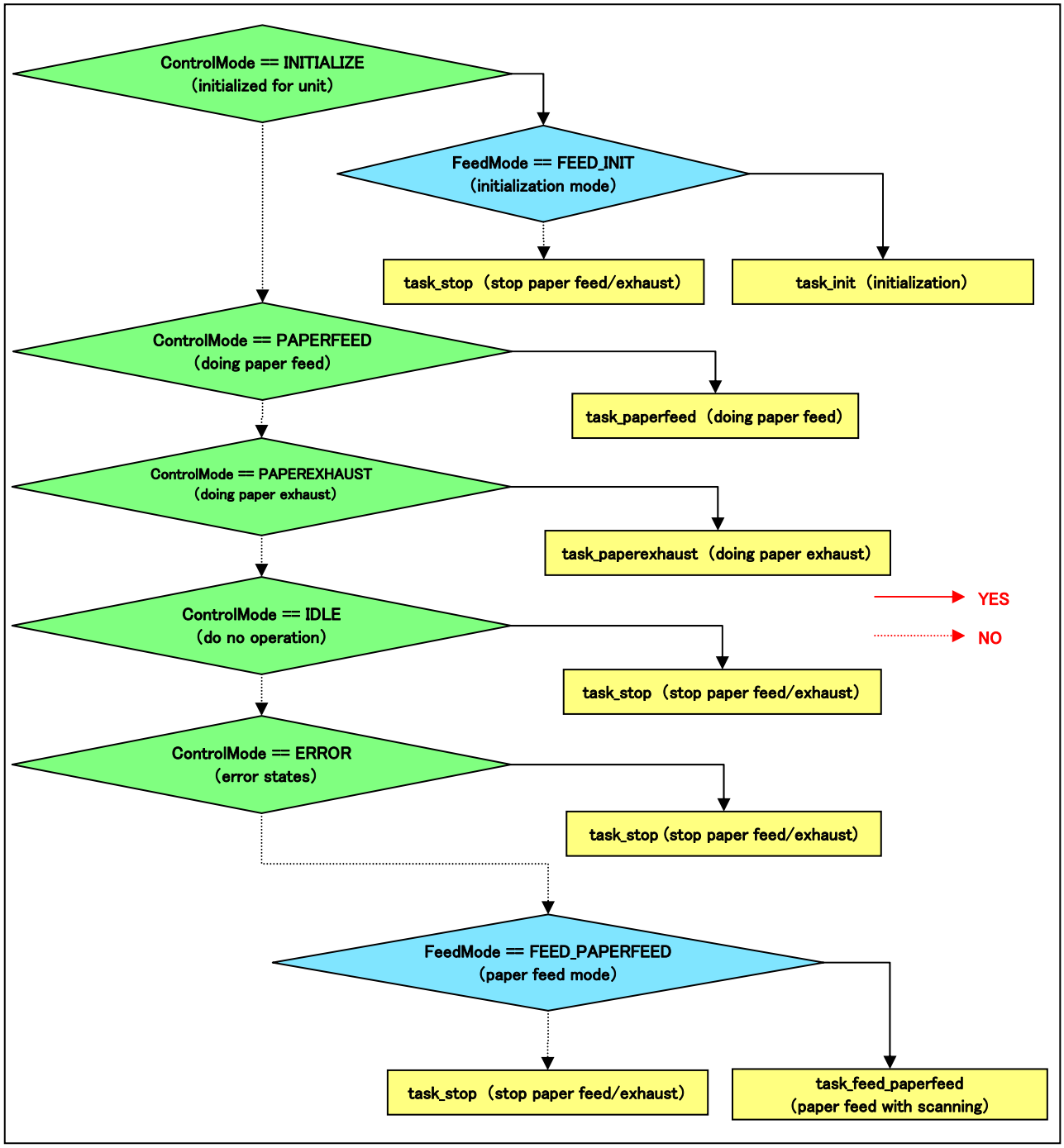


Figure 6-13 The algorithm of the control flow part

Initialization (task_init)

Right after the NXT Scanner is powered, constant paper feed is forced automatically. This process is necessary to avoid the backlash of DC motor for paper feed.

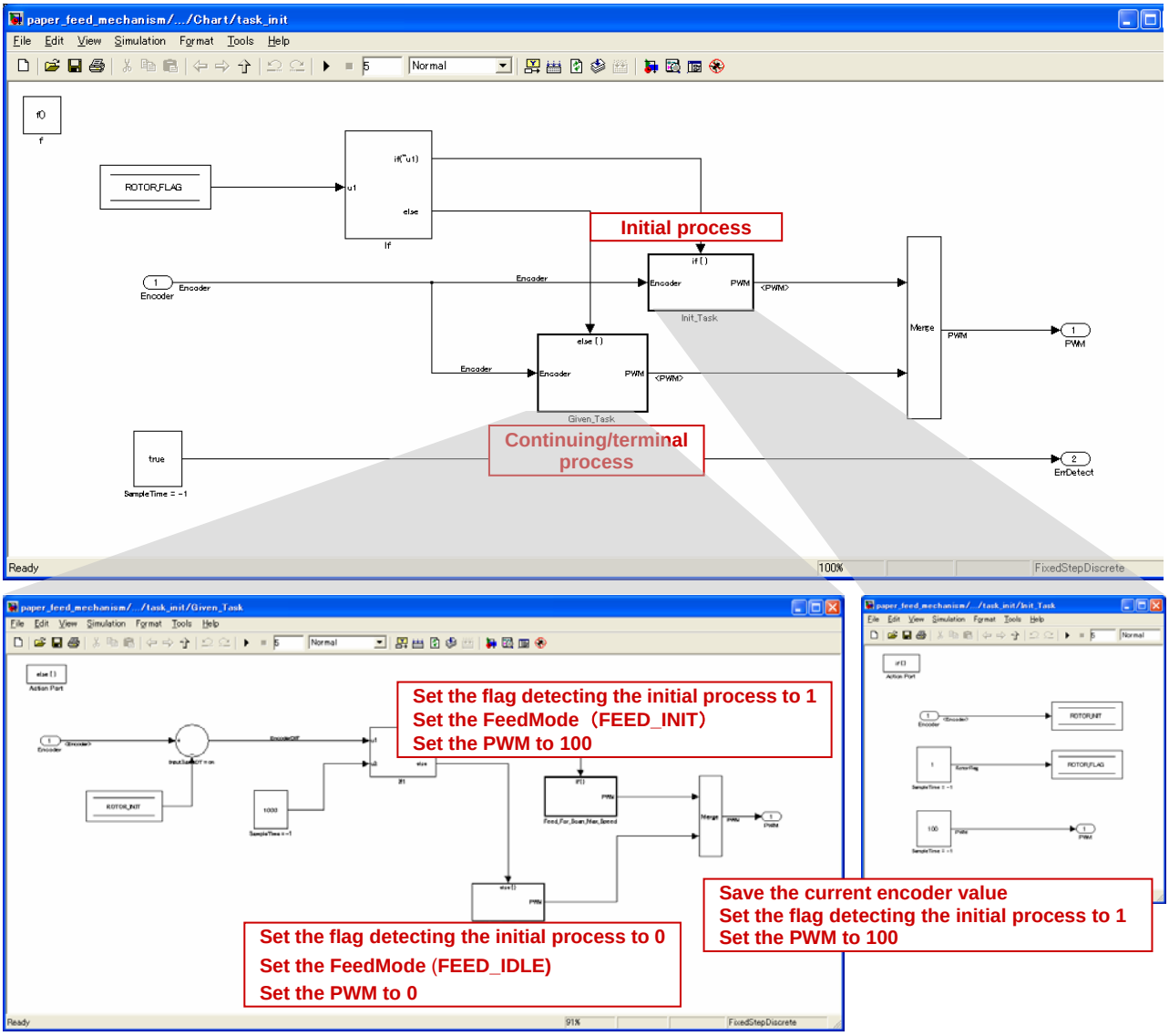


Figure 6-14　Initialization process (task_init Simulink function)

The initialization process has two subsystems which are the first process and the continue/finished process. The value of Data Store Memory is used for selecting subsystem.

Initial process has functions described below.

- Save the current encoder value
- Set the flag detecting the initial process to 1: This will make the transition to the continuing process.
- Set the PWM value to MAX (100).

The continuing process follows after the initial process, and paper feed operation will start.

- Set the flag detecting the initial process to 1
- Set the PaperFeedMode to FEED_INIT
- Set the PWM value to MAX (100)

If the encoder value increases for 1000 degrees of rotation from the value saved at the initial process, Paper feed terminates by the following terminal process.

- Set the flag detecting the initial process to 0
- Set the PaperFeedMode to FEED_IDLE
- Set the PWM value to MIN (0)

After the initialization, The PaperFeedMode changes FEED_INIT to FEED_IDLE. This change implies the end of initialization, and it is informed to the MODE CONTROL MODEL to switch the ControlMode.

Paper feed action (paper feed button) (task_paperfeed)

This process is called when the paper feed button is pushed down by users. This process has functions described below.

- Set the PWM value to MAX (100)
- Set the backlash direction flag to 0: This means that the direction of the current backlash is same as that of paper feed.
- Set the flag to 1 for enabling error detection: This information will be sent to the error detection part as a signal.

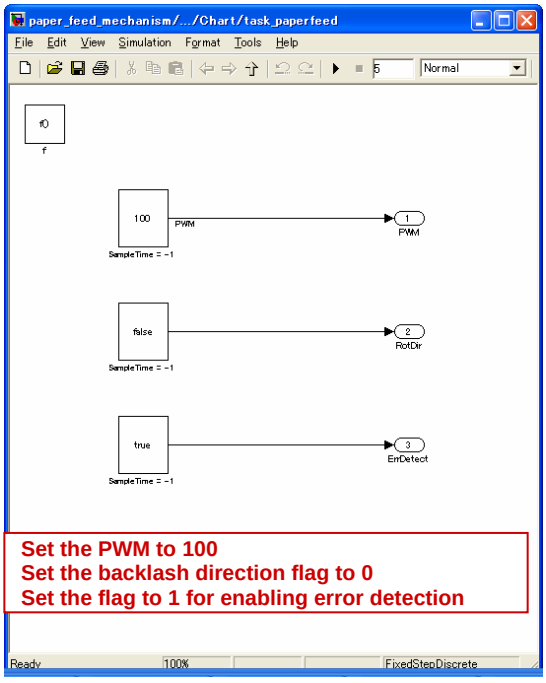


Figure 6-15 Paper feed action (paper feed button) (task_paperfeed Simulink function)

Paper exhaust action (paper exhaust button) (task_paperexhaust)

This process is called when the paper exhaust button is pushed down by the user. This process has functions described below.

- Set the PWM value to MAX (-100)
- Set the backlash direction flag to 1: This means that the direction of the current backlash is same as that of paper exhaust.
- Set the flag to 1 for enabling error detection: This information will be sent to the error detection part as a signal.

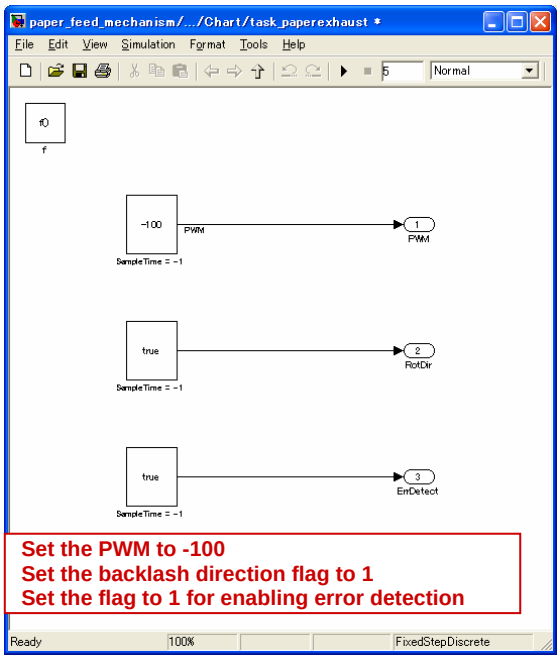


Figure 6-16 Paper exhaust action (paper exhaust button) (task_paperexhaust Simulink function)

Stop of paper feed/exhaust action (task_stop)

This process is called to stop the DC motor for paper feed, while the paper feed/paper exhaust are not in action, or the scanner head is moving. This process has functions described below.

- Set the PWM value to MIN (0)
- Set the flag to 0 for disabling error detection: This information will be sent to the error detection part as a signal.

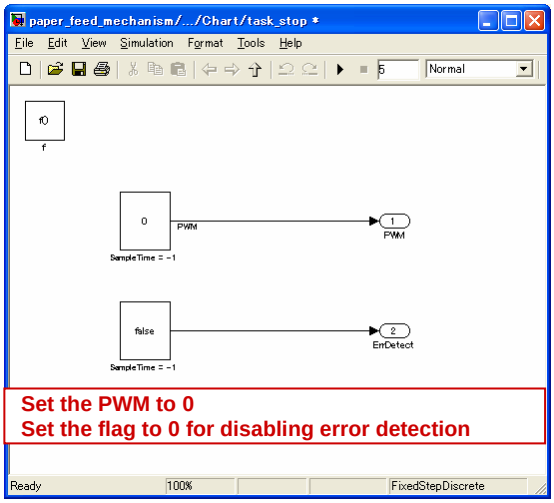


Figure 6-17 Stop of paper feed/exhaust action (task_stop Simulink function)

Paper feed action under scanning (task_feed_paperfeed)

This function defines the process of the paper feed action required during the process of scanning. This process is quite similar to the initialization (task_init), thus a part of this model is diverted from the initialization (task_init). This section shows you the specification of this function which differs from the one of initialization (task_init). One of the differences is that the backlash taken care of in this function. The direction of the backlash is passed to this function as an input argument. The incremental value for the encoder is determined depending on this information. For example, if the direction flag is 0, it is decided that the paper feed was done just before, and gives 102 degrees to feed one row of the image data. If the flag is 1, it is decided that the paper exhaust was done just before, and gives 102+50 = 152 degrees to feed. In the latter case, the 50 degrees is the offset to cancel out the backlash.

LookUp Table is used to define the PWM output response to the encoder input. This is because we would like to avoid the coasting of the rotation of the motor while adjusting the PWM output. We used the data logging function of The NXT GamePad to tune the table data. The NXT GamePad is a PC utility which provides a function of Bluetooth communication with The NXT Scanner.

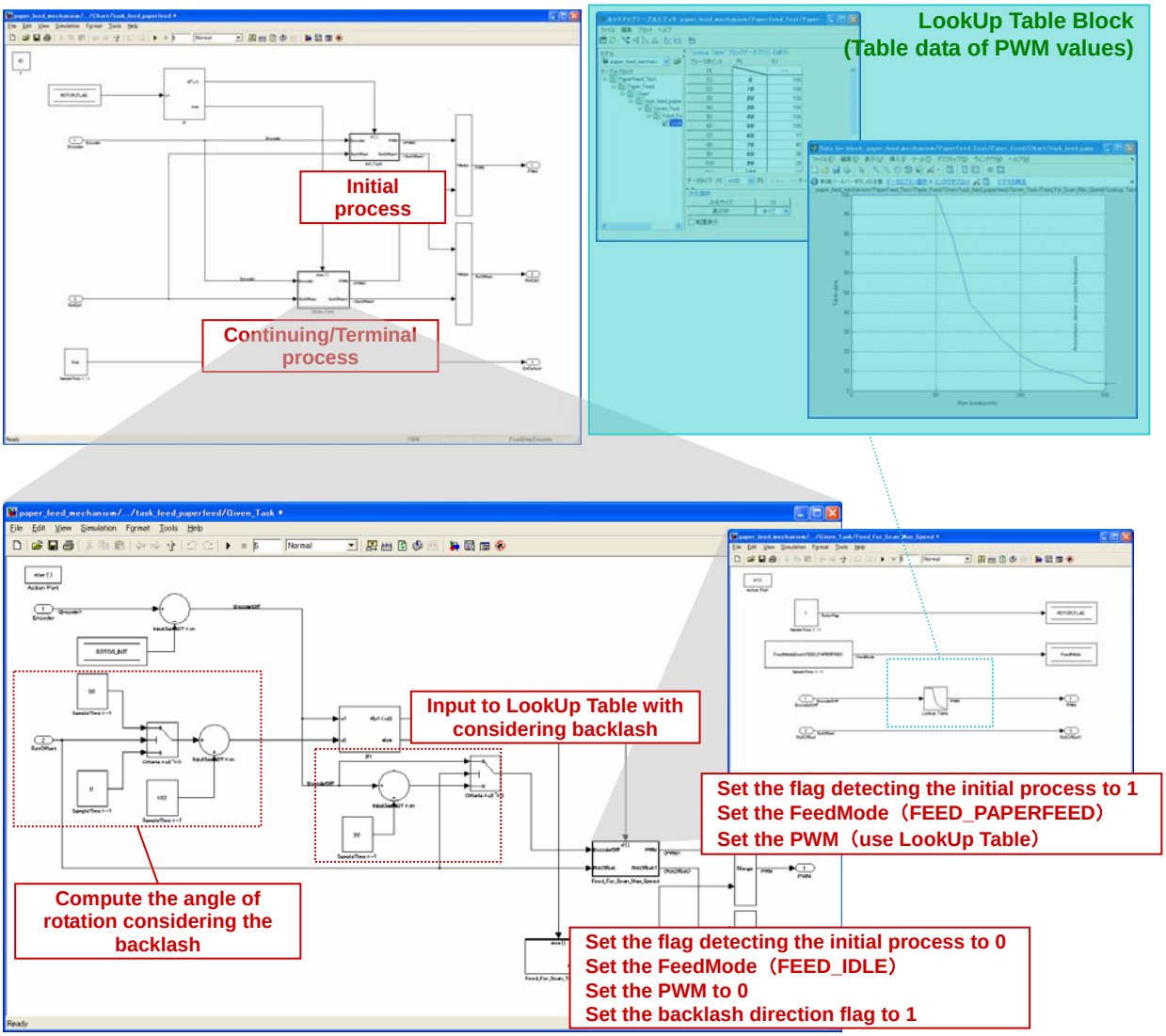


Figure 6-18　Paper feed action under scanning (task_feed_paperfeed Simulink function)

There are 127 rows to be scanned, so the paper feed is done 127 times during the process of scanning. Interlaced scanning is employed. It gives a square image. Figure 6-19 is a part of the image, and you can see the image is lack of data in columns. The NXT Viewer can interpolate this incomplete image by using the MATLAB image processing functions.

Figure 6-19　Result of scanning (sample)

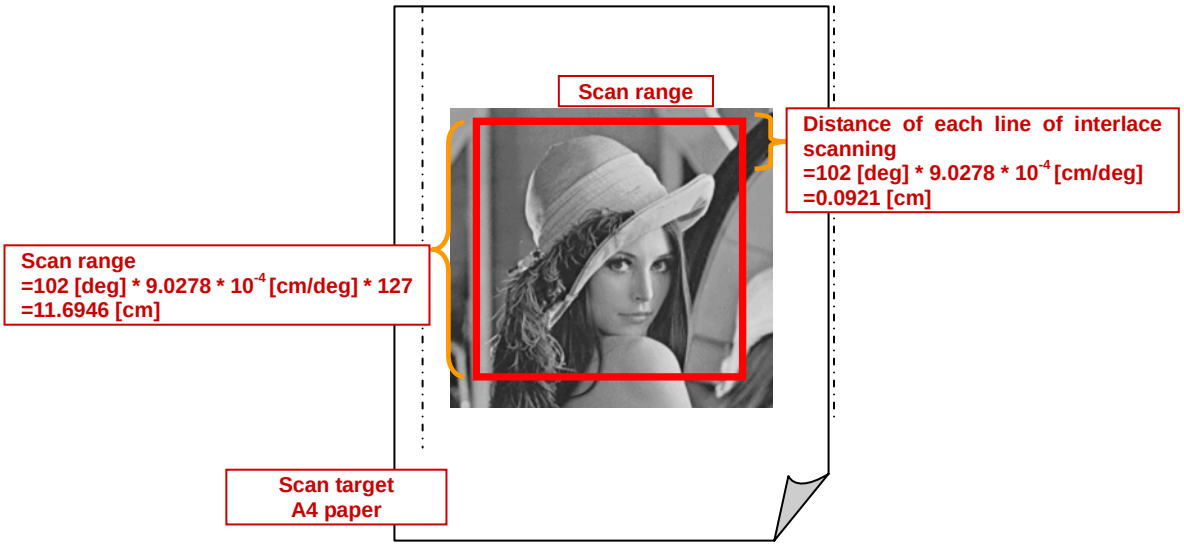


Figure 6-20 Scan range (paper feed direction)

Figure 6-20 shows the data computed from the data in Section 3 (Mechanisms for the NXT Scanner). The moving distance of the paper feed per degree of motor rotation is 9.0278×10^{-4} [cm/deg]. So the resolution of the paper feed is about 27dpi [dots per inch].

The error detection part

Figure 6-21 is the error detection part. The subsystem of the error detection part is enabled by the signal sent from the control flow part. If this subsystem is enabled, every 10 steps the current encoder value is saved, and compared with the previous value. If these two encoder values are equivalent, the error code is updated from NO_ERROR to ERROR_002, and the error condition is passed to the MODE CONTROL MODEL.

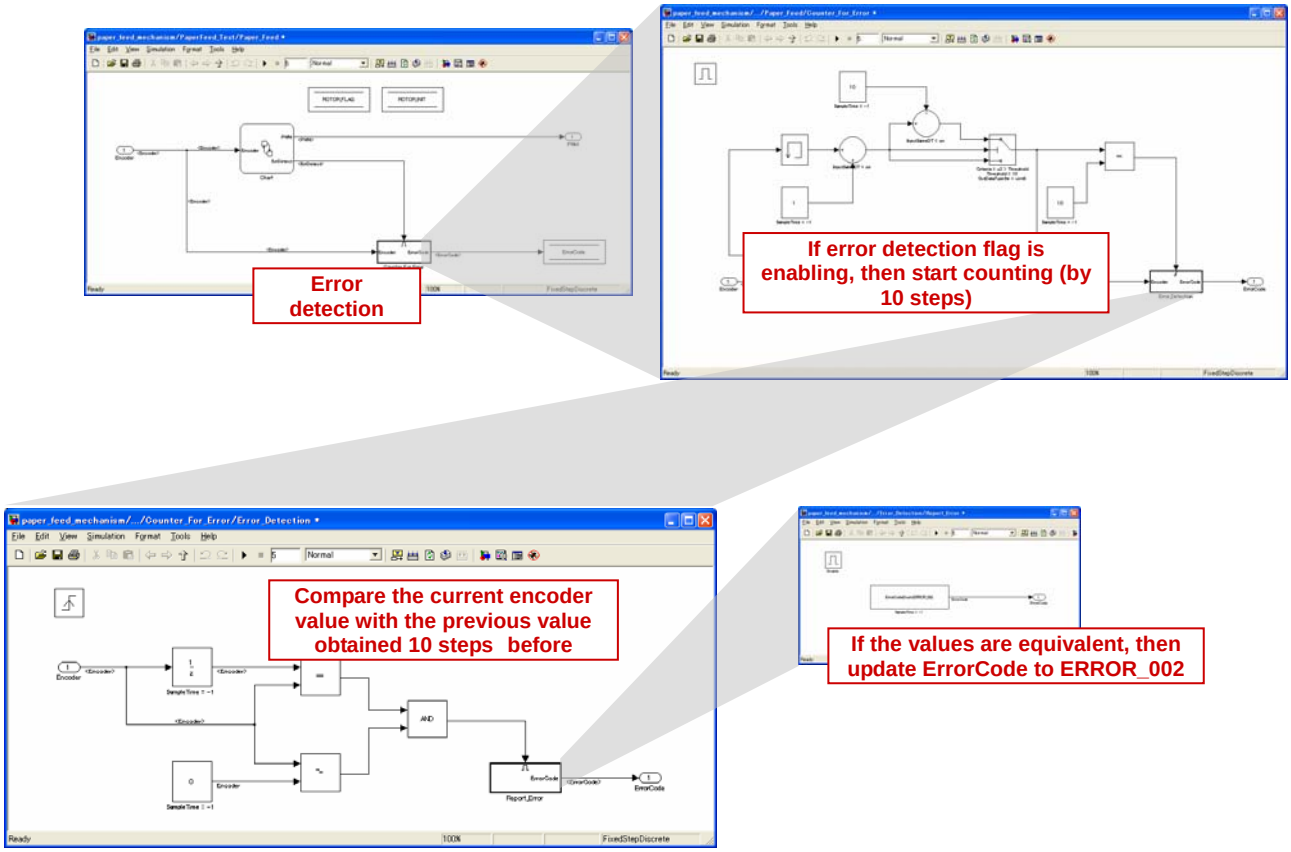


Figure 6-21 The error detection part (subsystem)

6.5 SCAN CONTROL MODEL

The scanner_head.mdl is model for the SCAN CONTROL SYSTEM. Figure 6-22 shows top layer of the SCAN CONTROL MODEL.

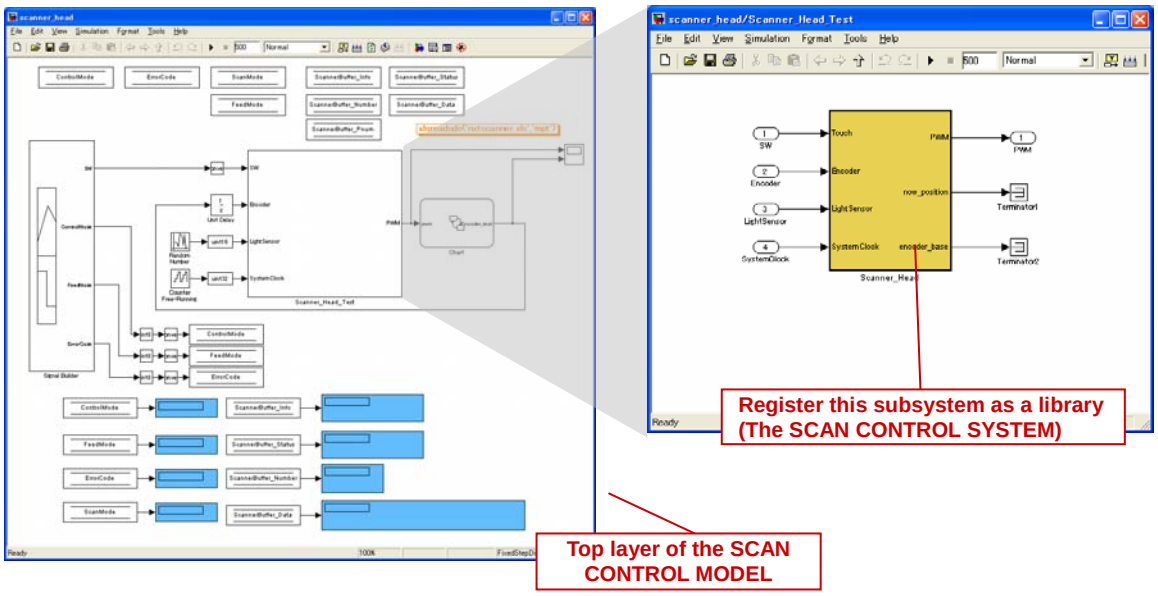


Figure 6-22 Relation the SCAN CONTROL MODEL to library model

This model provides input signals for simulation, and output signals for the purpose of monitoring in the top layer, just like in case of the other function models. By running the model, we can validate this model in unit function.

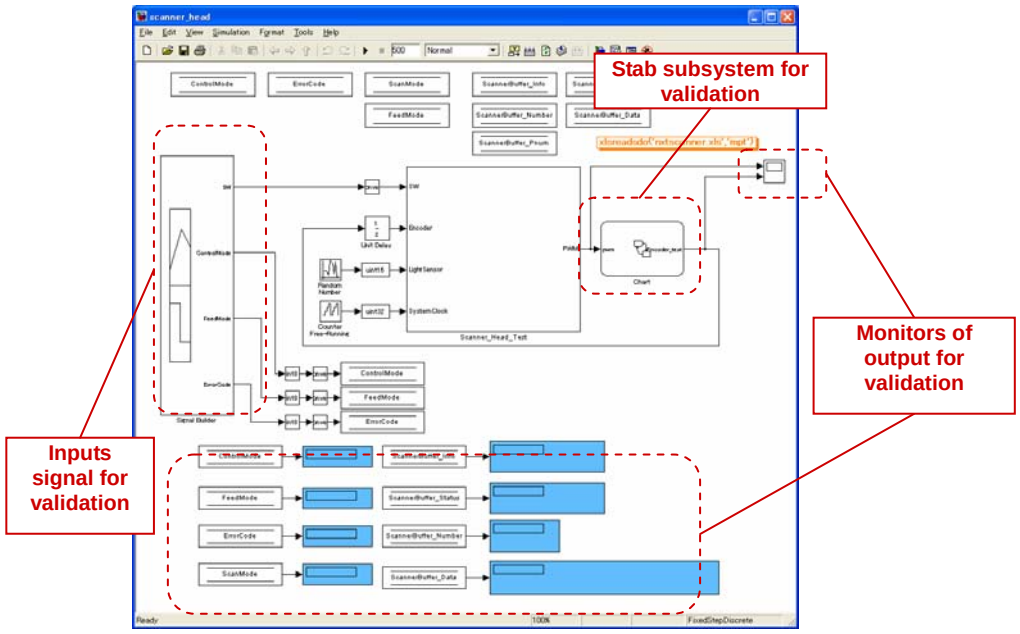


Figure 6-23 Top layer of the SCAN CONTROL MODEL

Algorithm implementation

The SCAN CONTROL MODEL only uses Stateflow. The Left part of Figure 6-24 is the state chart, and the right part are the graphical functions designed by a flow chart. Table 6-2 is a list of graphical functions.

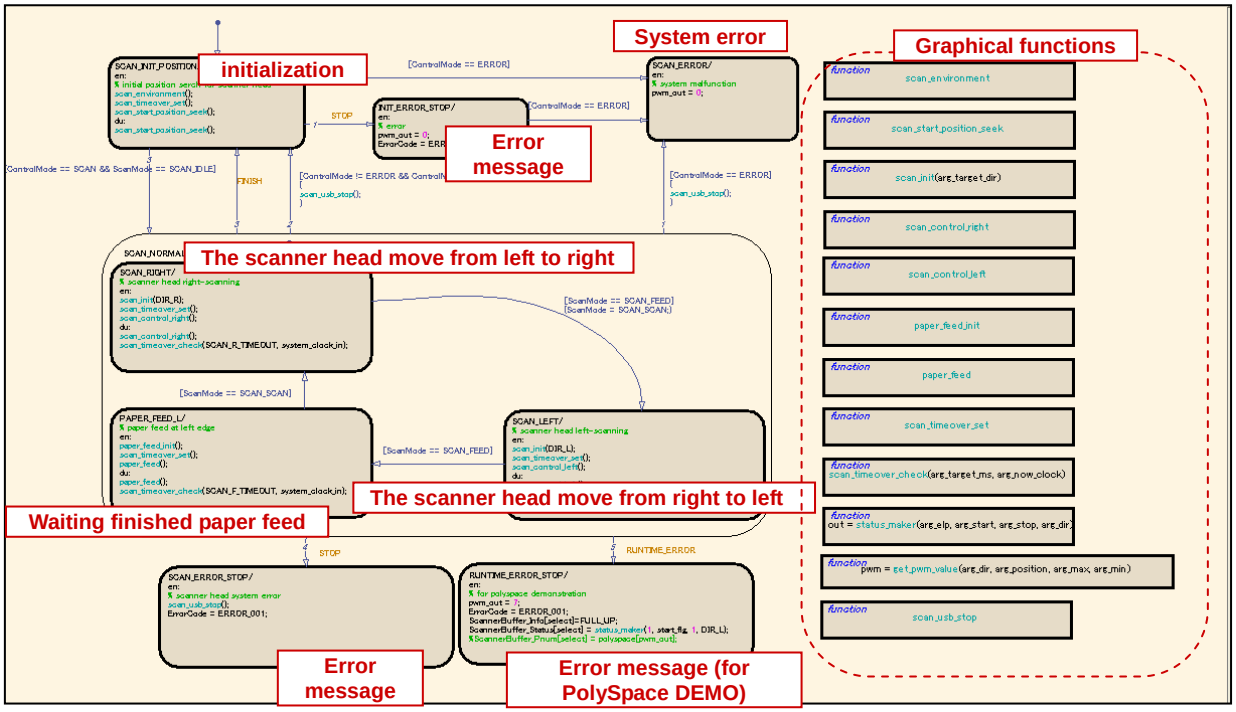


Figure 6-24 Main algorithm of the SCAN CONTROL MODEL

Table 6-2 Graphical functions list in the SCAN CONTROL MODEL

graphical function	argument	return	note
scan_environment	-	-	initialization processing for moving to start position
scan_start_position_seek	-	-	moving to start position algorithm function
scan_init	target direction	-	initialization processing for scan start
scan_control_right	-	-	Move from left to right algorithm function
scan_control_left	-	-	Move from right to left algorithm function
paper_feed_init	-	-	initialization processing for waiting finished paper feed
paper_feed	-	-	Waiting finished paper feed function
scan_timeover_set	-	-	initialization processing for timer start
scan_timeover_check	limited time current time	-	Checking limited time function
status_maker	ELP information START information STOP information DIR information	send status	Making status packet for USB send function
get_pwm_value	target direction current position target max speed PWM target min speed PWM	current PWM value	Get PWM value function
scan_usb_stop	-	-	Making final USB data transfer for forced termination (All scan finished)

Storing scan data

Storing scan data is processing of scan_control_right graphical function during moving the scanner head unit from left to right. Figure 6-25 shows Stateflow algorithm of the scan_control_right graphical function.

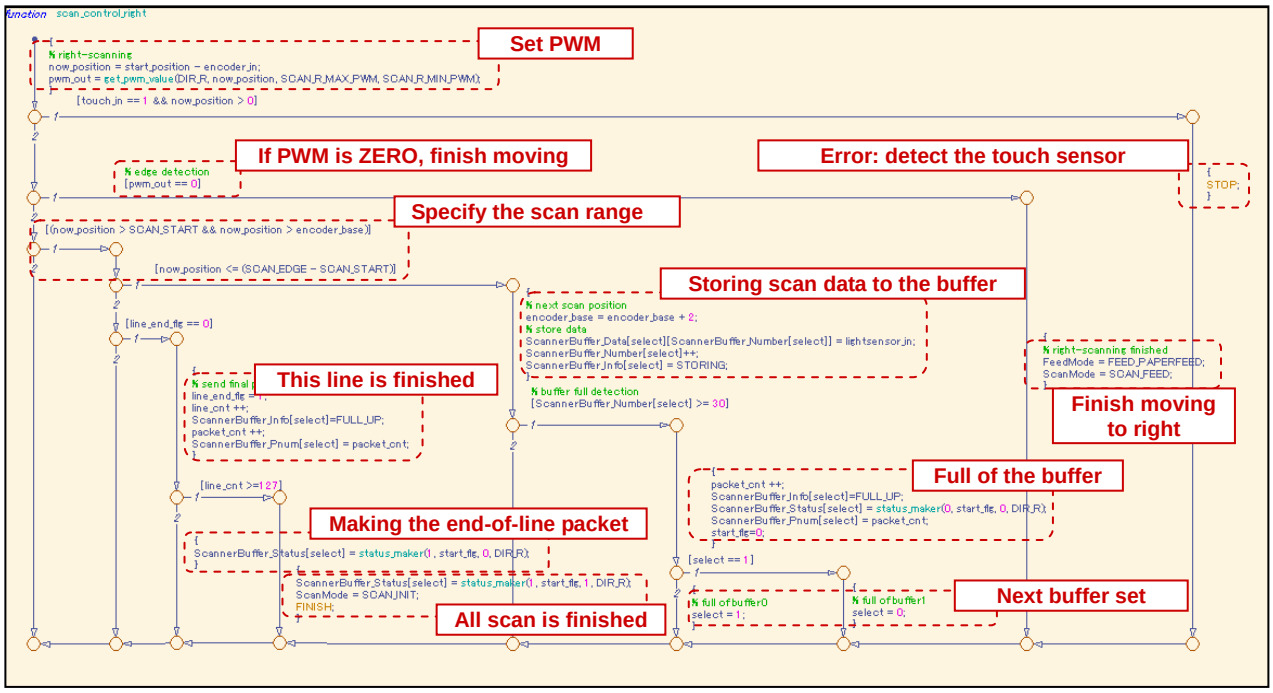


Figure 6-25 Algorithm of the scanner head moving from left to right

Figure 6-26 shows the scan range in the red boxed text on an A4 paper. SCAN_EDGE is defined the range of moving the scanner head unit and the unit of SCAN_EDGE is the rotary encoder value (degree). Scanable range is SCAN_EDGE minus both edges of SCAN_START. Figure 6-26 shows each value which are using below value in Section 3 (Mechanisms for the NXT Scanner). The moving distance of the scanner head unit per degree of motor rotation is 0.0215 [cm/deg]. So the resolution of the scanner head is about 118dpi [dots per inch].

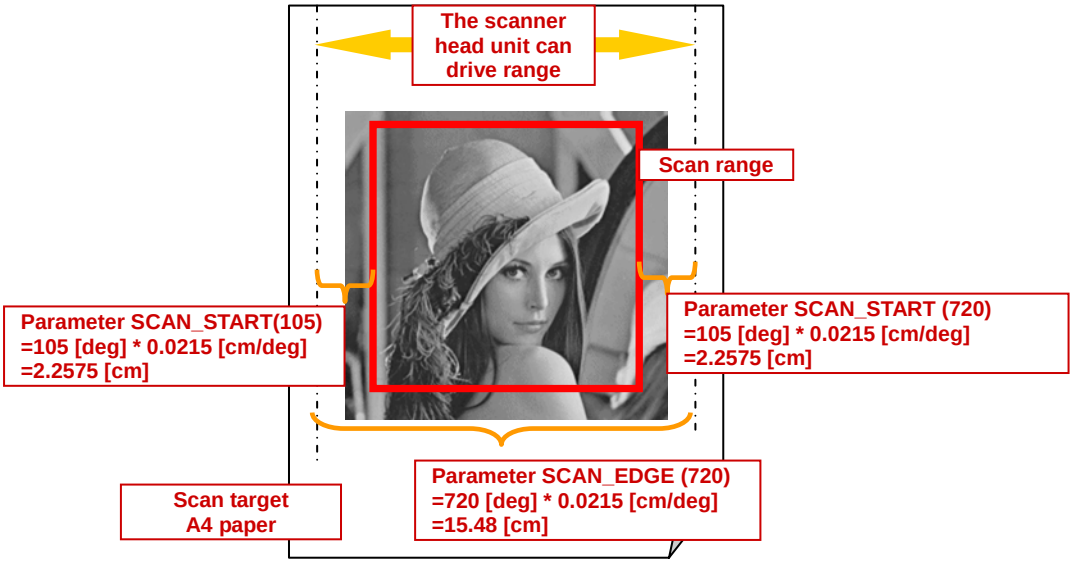


Figure 6-26 Scan range (the scanner head moving direction)

Regular intervals scan position

Scan is done 255 time every line. At this time the most important thing is scan timing. If scan timing will be not equidistant, the resulting image will not look good. To use timing that depends almost entirely on the task cycle is not a good idea. This is because the scanner head motor movement is non-constant. Therefore, the NXT Scanner uses feedback from the rotary encoder value of the scanner head motor. So the control should be designd to scan regularly in the target position. Figure 6-27 shows it code in red boxed text. As you see, the scan target positions are at 0.0439 [cm] intervals.

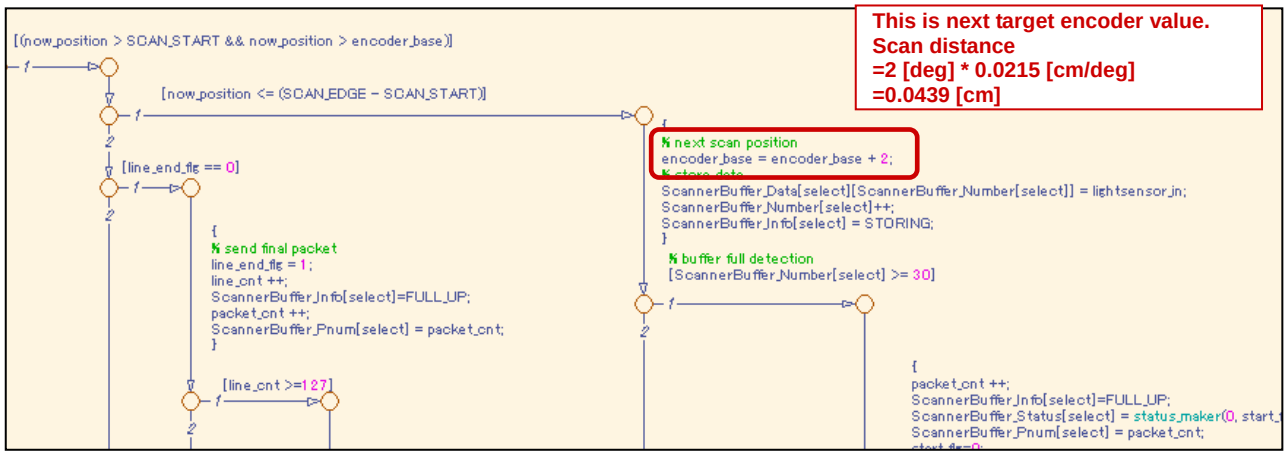


Figure 6-27 Scan distance control (scan_control_right graphical function)

Control of scanner head stop position

If the result of scanning the image is not good looking, one of the reasons is the stop position of the scanner head unit. It means the scanner head unit can not stop at the predetermined position. The main cause will be the inertial torque effect. The PWM control algorithm should stop exactly the same position (SCAN_EDGE) every time.

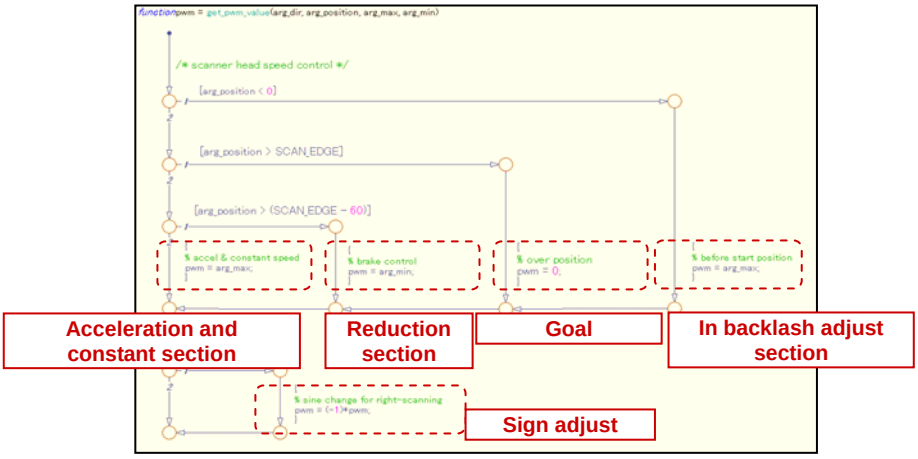


Figure 6-28 Scanner head speed and stop control (get_pwm_value graphical function)

The target minimum speed PWM is fixed by metering experiments, because the scanner head unit has the frictional force and the load torque from the cable. The target minimum speed PWM means it is the minimum power required to move the scanner head unit. The starting point of the reduction section is also fixed by metering experiments. Therefore, if you should change the any specification (hardware structure, PWM, etc.), you need to exercise caution.

Avoid backlash

If the result of scanning the image is not good looking, one of the reasons and most the potentially influential factor is a backlash. Target positioning and current position discord from the real position because the gear has backlash. Figure 6-29 shows backlash cancel for each case.

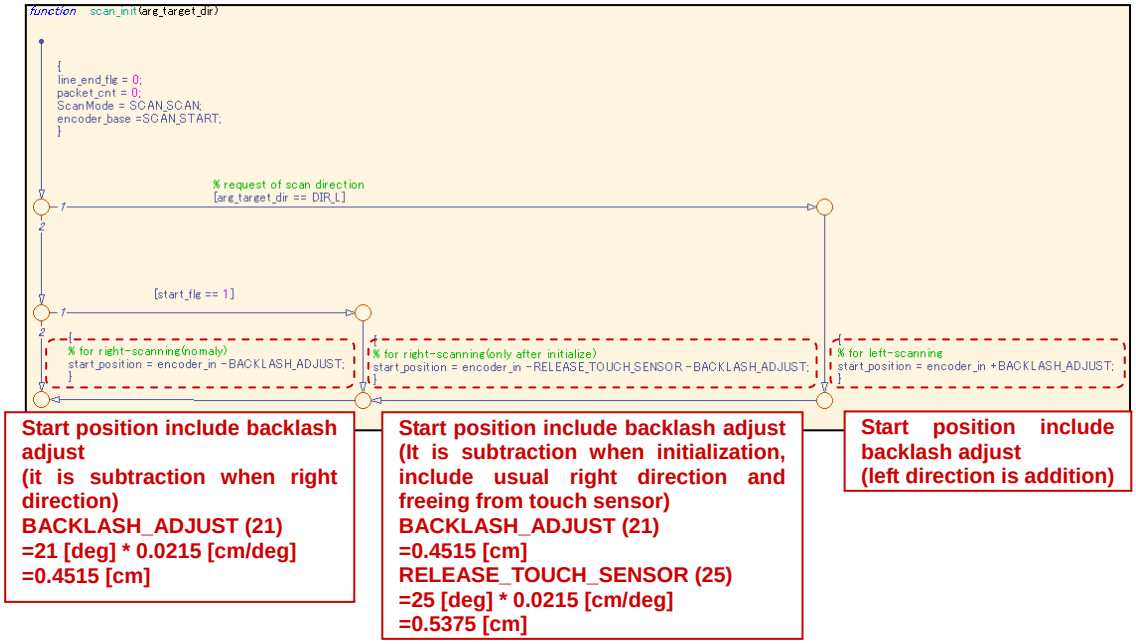


Figure 6-29 Backlash adjust control (scan_init graphical function)

Reduced scanning time

Moving to left of the scanner head unit and paper feed are parallel process. This can reduce scanning time.

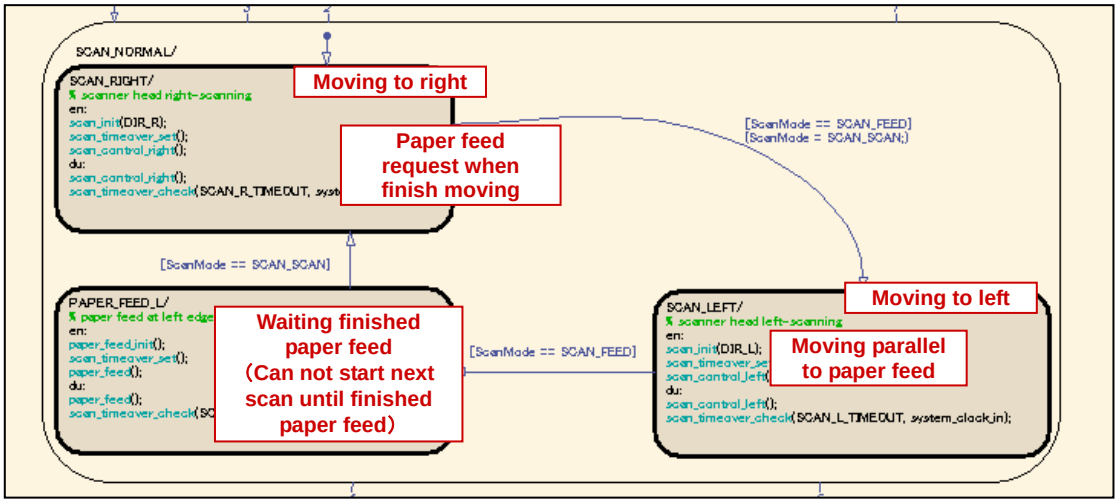


Figure 6-30 Scanner head move from right to left and paper feed control

6.6 USB COMMUNICATION CONTROL MODEL

The data_communication.mdl is model for the USB COMMUNICATION CONTROL SYSTEM. Figure 6-31 shows top layer of the USB COMMUNICATION CONTROL MODEL.

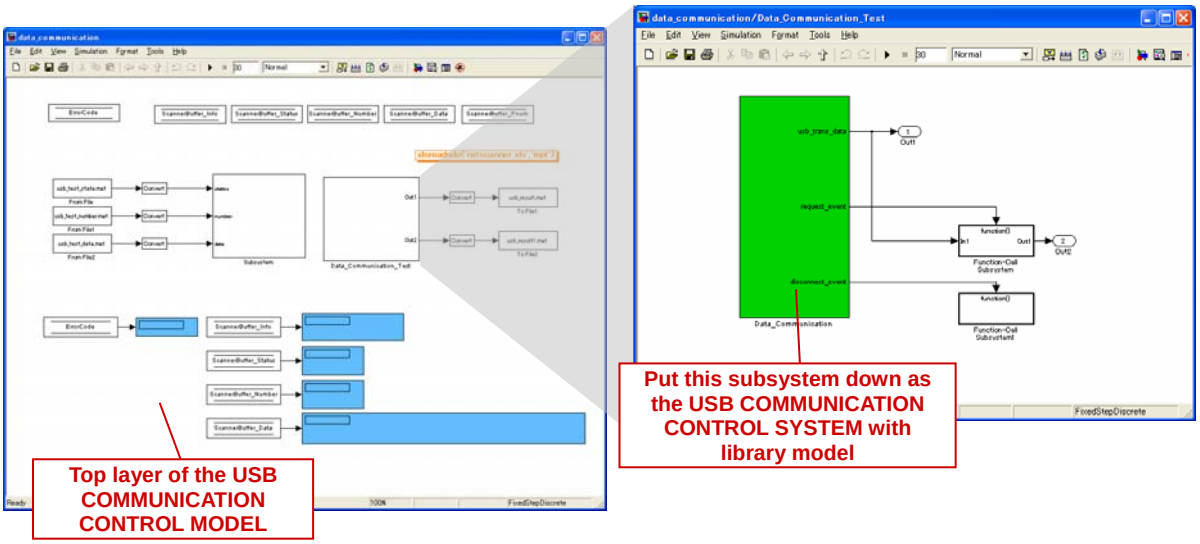


Figure 6-31 Relation the USB COMMUNICATION CONTROL MODEL to library

This model provides input signals for simulation, and output signals for the purpose of monitoring in the top layer, just like in case of the other function models. By running the model, we can validate this model in unit function.

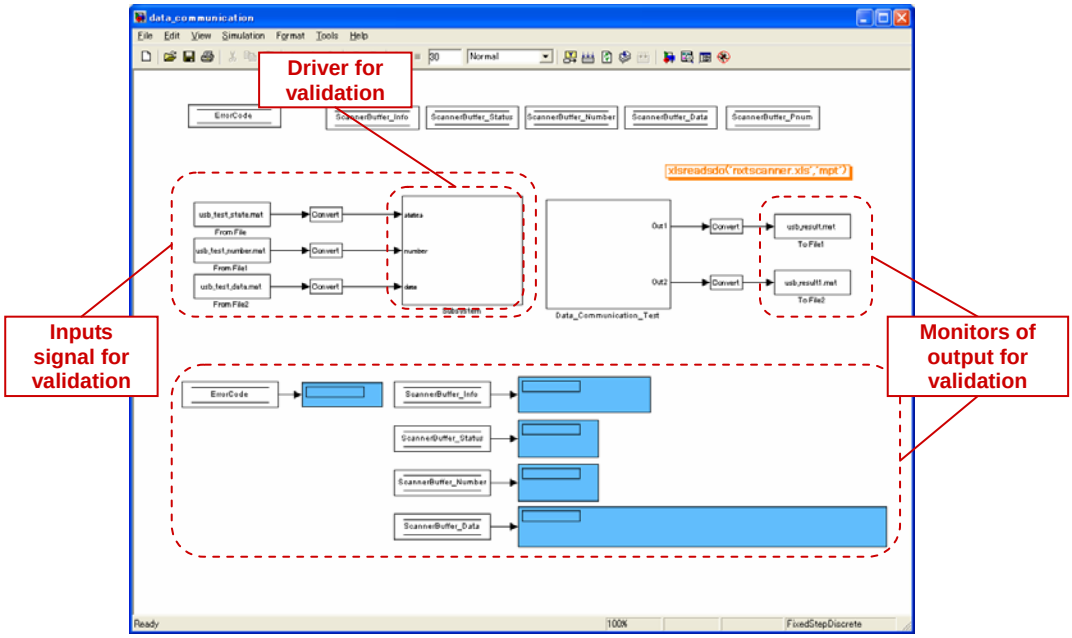


Figure 6-32 Top layer of the USB COMMUNICATION CONTROL MODEL

Algorithm implementation

Figure 6-33 shows the USB COMMUNICATION CONTROL MODEL. Figure 6-34 shows the USB communication algorithm which is written using Stateflow. After the Stateflow process, USB send data (64 Bytes) is merged status (2Bytes), number (2Bytes) and data (60Bytes).

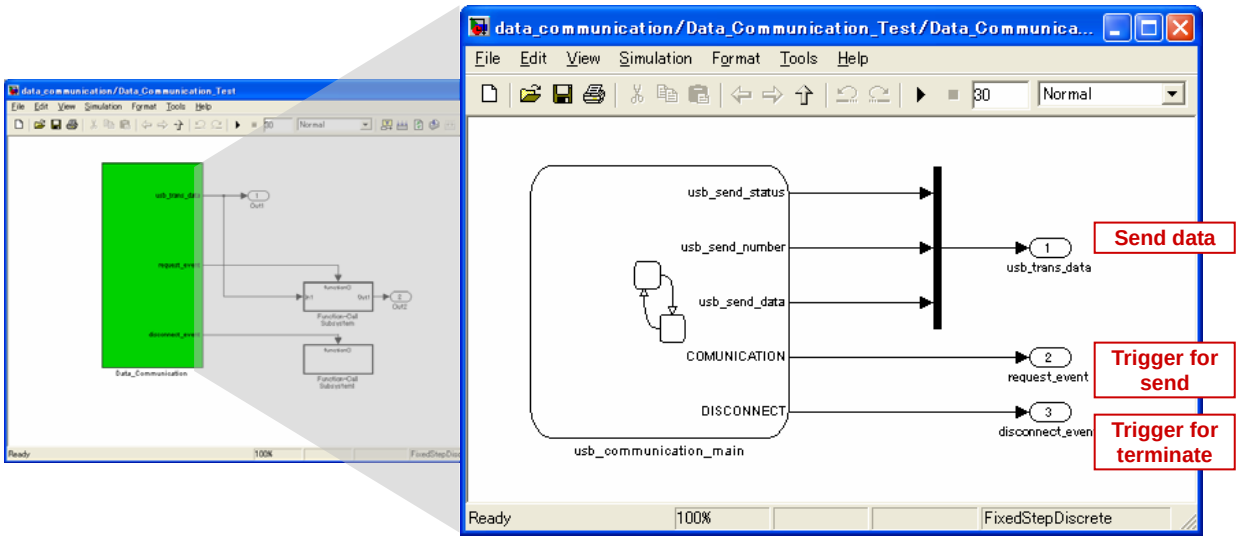


Figure 6-33 The USB COMMUNICATION CONTROL MODEL outputs

When finished scanning is finished, the connection should be terminated. Thus the USB COMMUNICATION CONTROL MODE should be monitoring the scan buffer. After latest scan buffer send, the USB COMMUNICATION CONTROL MODEL waits 1 sample time and then terminates the connection. Using state (CLOSE) generates 1 sample time wait.

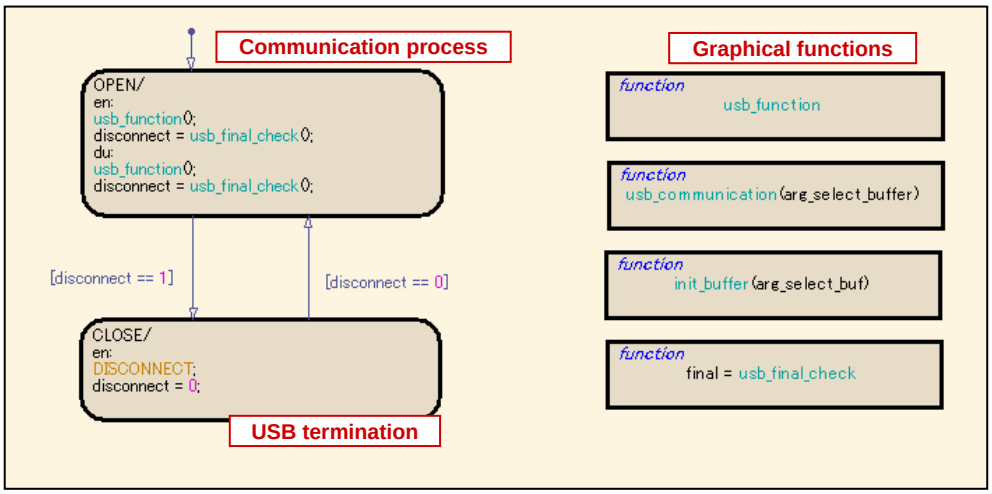


Figure 6-34 Main algorithm of the USB COMMUNICATION CONTROL MODEL

Requisites for send buffer are full of buffer or latest data is stored when finished scanning. The buffer is a double buffer structure. When designing the task cycle in the system design, you should take care of two points. First, it is necessary to design the task cycle such that both buffers not fill up. Second, it is necessary to alternately to use the double buffer. This is because when scanning is finished, sometimes each buffer becomes full. So ScannerBuffer_Pnum is used.

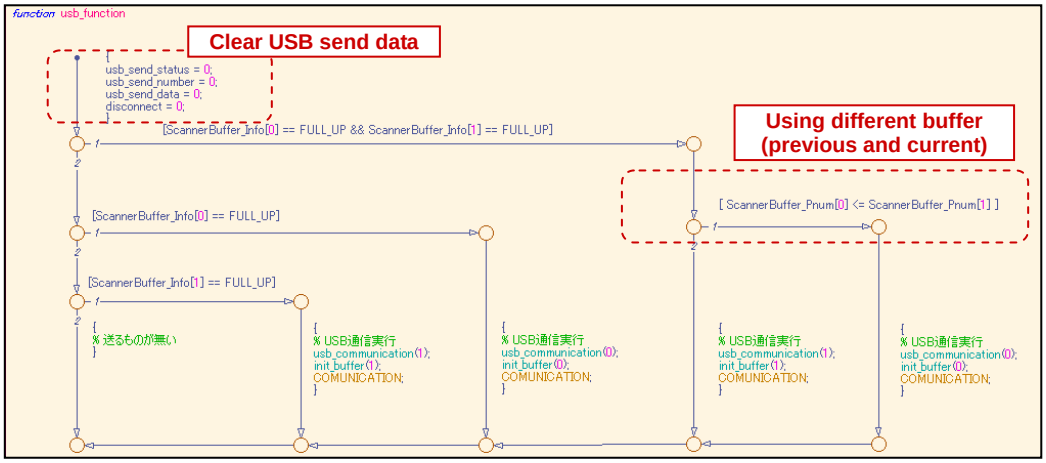


Figure 6-35 Communication process (usb_function graphical function)

7 Simulation for each models

This chapter describes the simulation process and the results of the NXT Scanner. Before simulation, please execute the setup_nxtscanner.m. This file sets a path for the NXT Scanner environment folder.

7.1 Test signals for Simulation

Every function unit model has 100% coverage test signal as shown Figure 7-1.You can use it for simulation and unit test for each function unit model.

Figure 7-1 Change test signal

Note

Simulink® Design Verifier™ does not support Enumerated Type and Stateflow Simulink Function of new feature of R2008b. Therefore, the NXT Scanner is marked 100% coverage test signal by hand. So verification is not automatic, you should judge it by yourself.

7.2 Introduce of verification tools and function

Simulink Design Verifier and Simulink Verification and Validation are verification tools for Simulink and Stateflow models.

Simulink Verification and Validation

Simulink Verification and Validation has the following features.

- It is possible to specify the coverage target by the subsystem.
- It is possible to make a distinction between the two cases (transit path or not) in the model by color.
- It is possible to accumulate the coverage by two or more test signal.

Simulink Verification and Validation exposes design flaws, inadequate requirements, incomplete tests, and unnecessary design constructs early in the development process. You can trace requirement documents to your design models, component tests, and generated code. You can also verify your designs and tests through model coverage and modeling standards checking.

Simulink Design Verifier

Simulink Design Verifier performance functions below for your Simulink and Stateflow models.

- Automatically generate the test harness model.
- Property proving.

Simulink Design Verifier uses formal methods for Verification and Validation. Simulink Design Verifier can generate 100% coverage test harness model by coverage measurement feature of Simulink Verification and Validation. By using test harness model it I possible to validate equivalence checking model and generated code (convert to S-Function).

For property proving, you can directly capture design requirements and performance objectives as properties in their Simulink or Stateflow models. Simulink Design Verifier mathematically proves whether those properties are satisfied and, if not, provides counter-examples that would violate the properties. As a result, you can find design flaws, unsatisfied requirements, and unreachable states or logic that would be difficult to uncover using simulation alone.

Figure 7-2 shows a model that includes verification specification subsystem for the MODE CONTROL MODEL. In this case, this model has a switch in the verification specific subsystem. Because Simulink Design Verifier doesn't support the new feature of R2008b but the model simulation should be possible. The switch can select blank subsystem or verification specification subsystem.

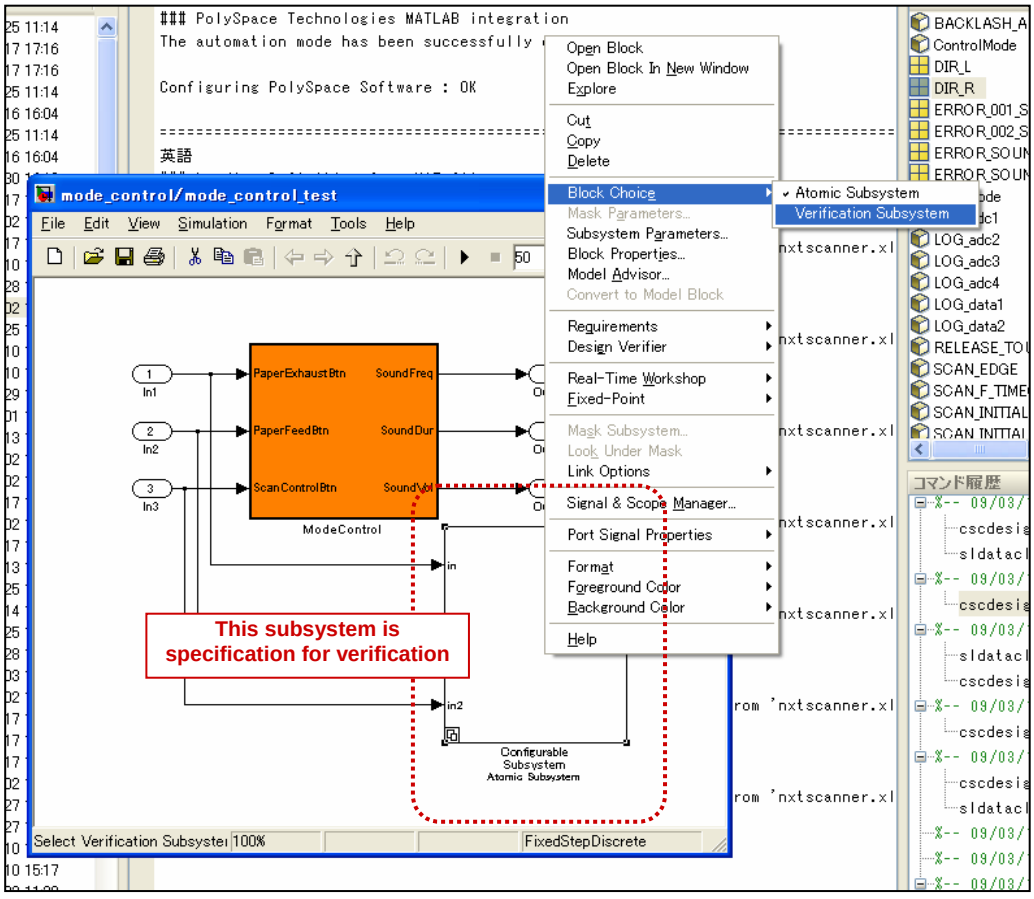


Figure 7-2 Change specification for verification subsystem

Figure 7-3 is one example of a model for the verification specification subsystem. It uses the truth table in Stateflow. Simulink Design Verifier proved the model by this specification of truth table. Simulink Design Verifier documents the valid blocks in a detailed report and generates counterexamples for invalid blocks. Counterexamples include input data and parameter values that demonstrate a specific violation. They are incorporated into the same style harness models that are produced during test generation.

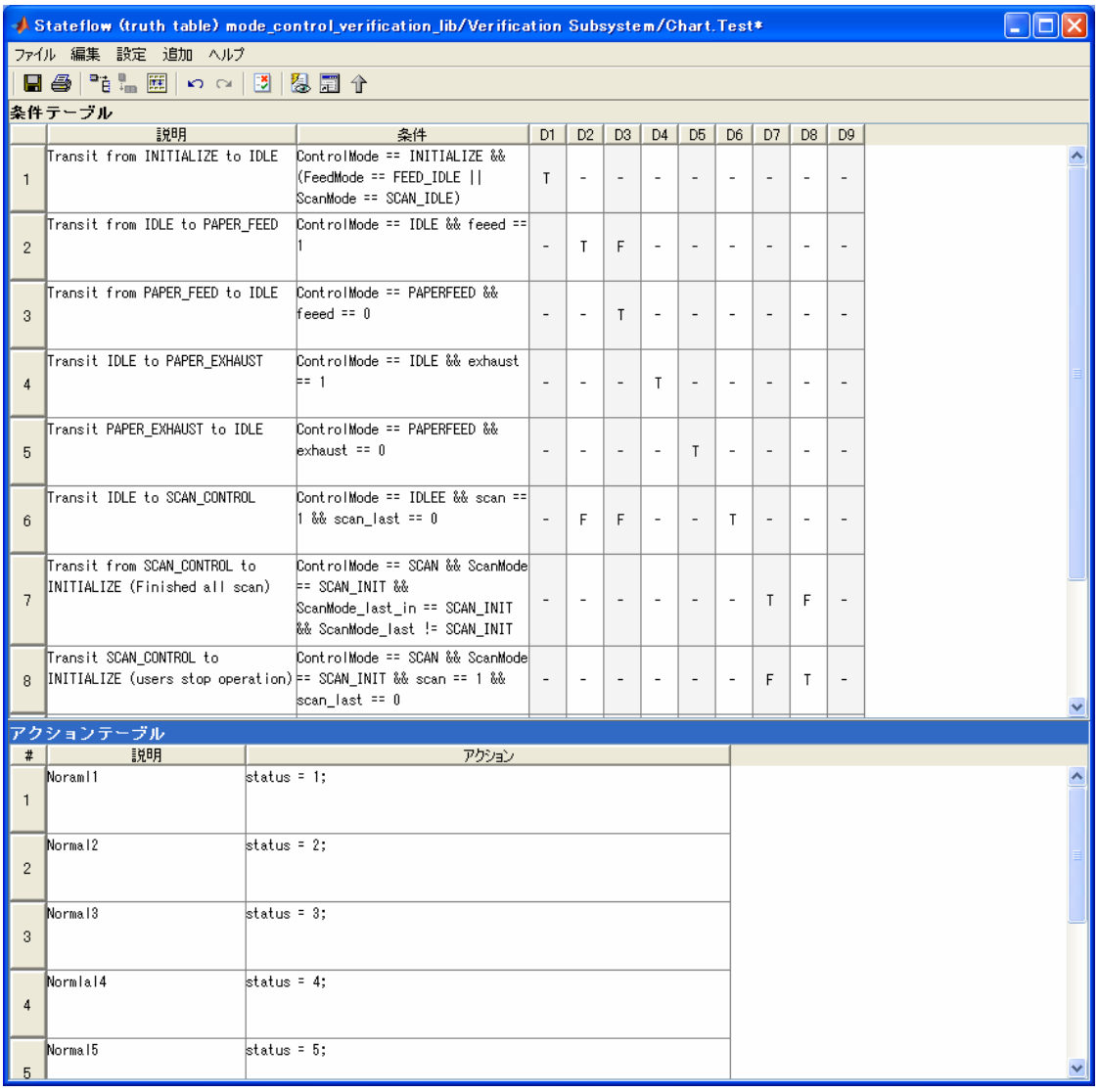


Figure 7-3 Example model of specification for verification

8 The NXT Scanner controller model (integrated each models)

This chapter describes the control program, task configuration, and model contents of the nxtscanner_ctrl.mdl.

8.1 Control program summary

The NXT Scanner controller has five tasks described in Table 8-1.

Table 8-1 Task organization

Task	Period	Works
task_init	initialization only	initial value setting
task_ts1	2 [ms] cycle	SCAN CONTROL SYSTEM
task_ts2	10 [ms] cycle	PAPER FEED CONTROL SYSTEM
task_ts3	20 [ms] cycle	MODE CONTROL SYSTEM
task_ts4	60 [ms] cycle	USB COMMUNICATION CONTROL SYSTEM

8.2 The NXT Scanner model summary

The nxtscanner_ctrl.mdl is based on Embedded Coder Robot NXT framework.

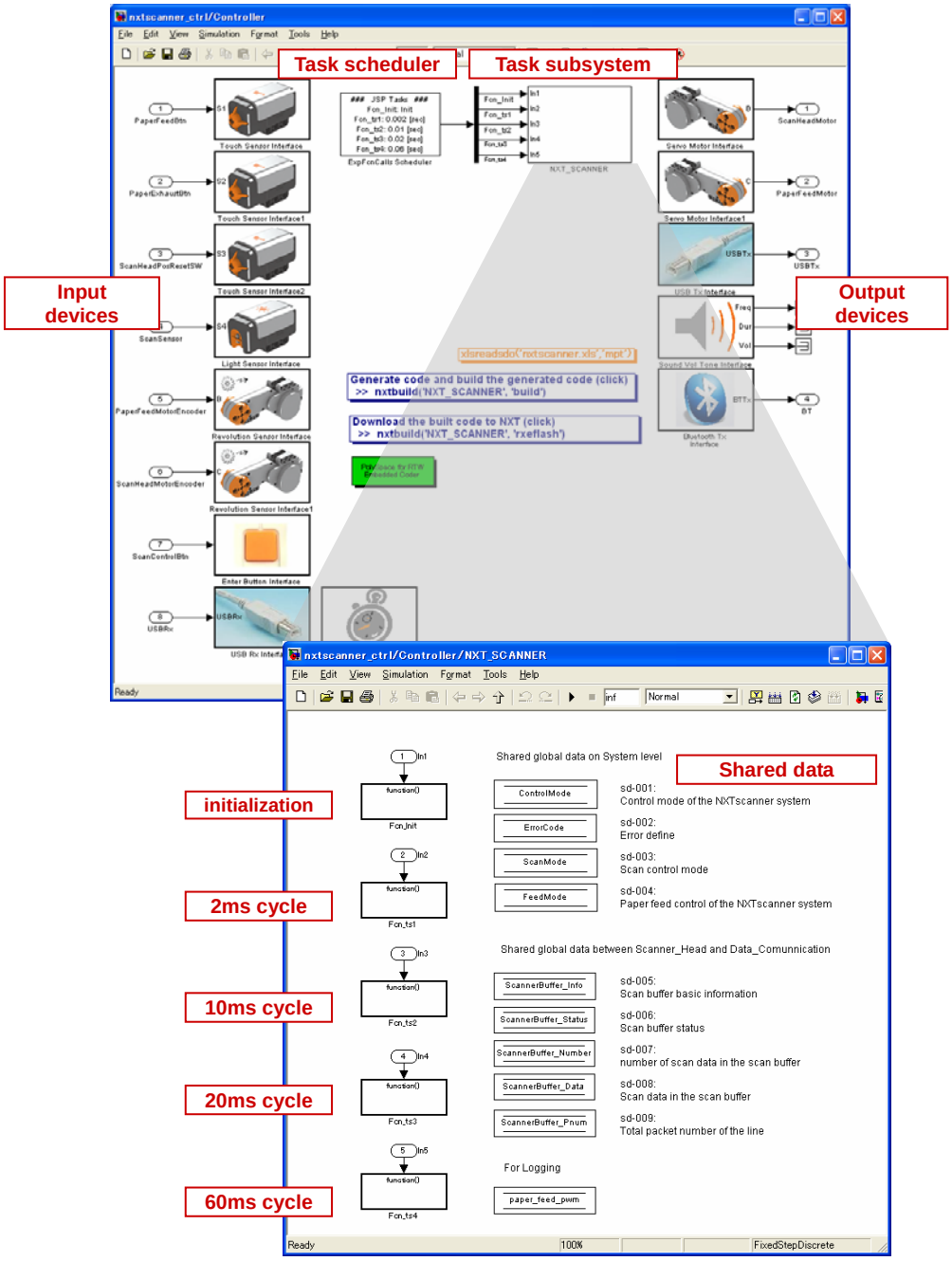


Figure 8-1 nxtscanner_ctrl.mdl

Device interface

We can make device interfaces by using the sensor and actuator blocks provided in Embedded Coder Robot NXT library.

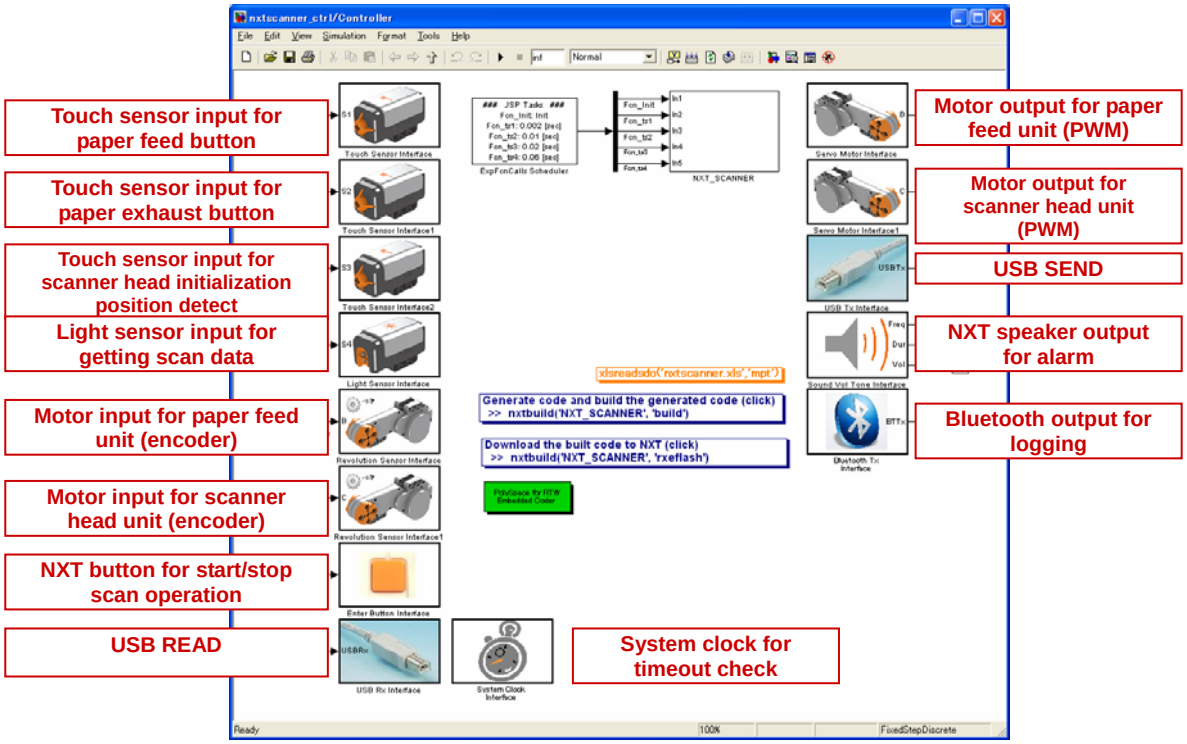


Figure 8-2 Device interface

Scheduler and tasks

The ExpFcnCalls Scheduler block has task configuration such as task name, task period, platform, and stack size. You can make task subsystems by connecting function-call signals from the scheduler to function-call subsystems. So you can select the platform which is OSEK or JSP.

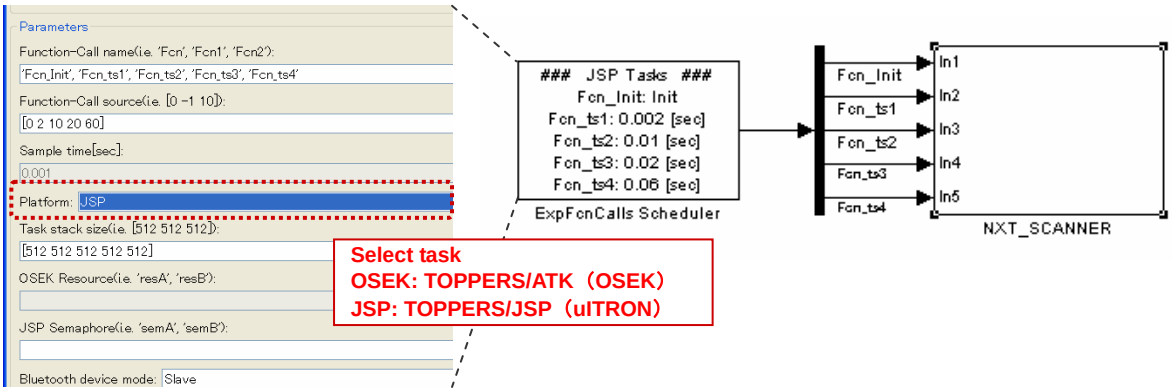


Figure 8-3 Scheduler and tasks

Priority

You have to set the priority of the device blocks and the ExpFcnCalls Scheduler block at root level by sorting them in order (1: device inputs, 2: tasks , 3: device outputs). The low number indicates high priority and negative numbers are allowed.

To display priority, right click the block and choose [**Block Properties**].

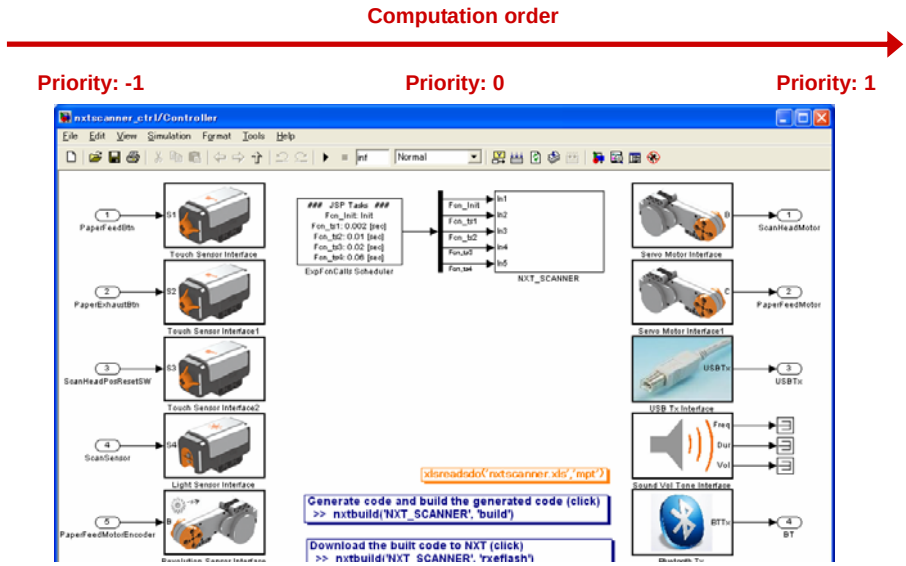


Figure 8-4 Priority setting

Shared data

Shared data is using Data Store Memory blocks as shared data between tasks.

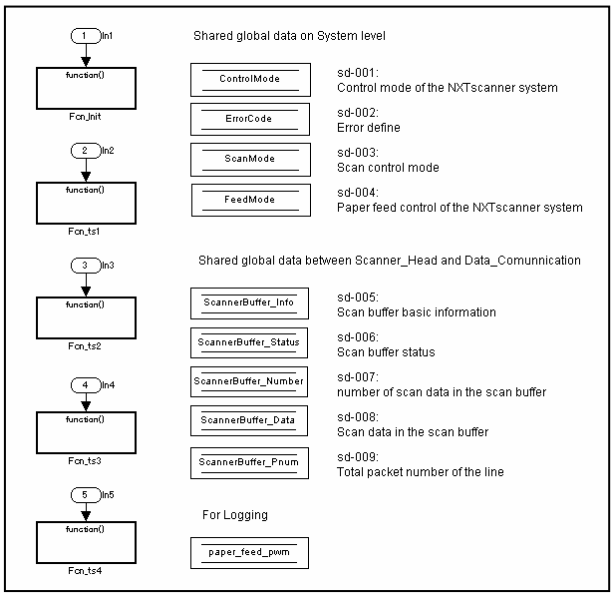


Figure 8-5 Shared data

8.3 Initialization task: task_init

This task sets the initial values. Motors initialize to ZERO at this task.

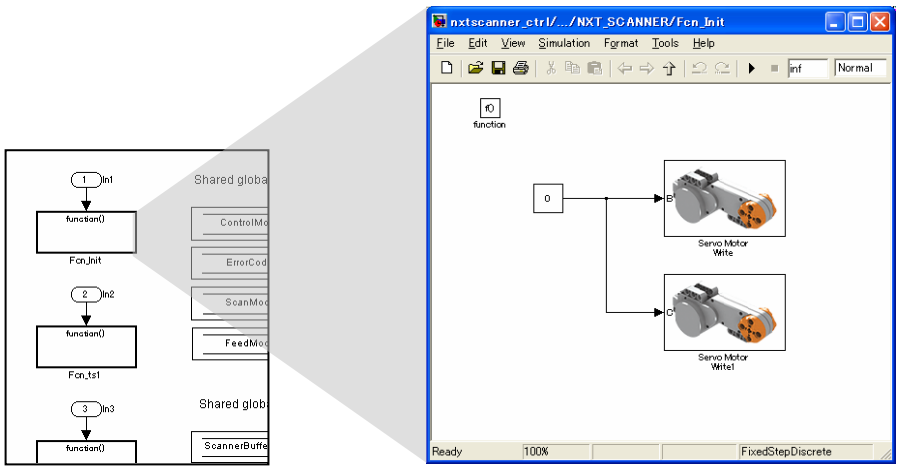


Figure 8-6 task_init

8.4 2ms task: task_ts1

This task includes the SCAN CONTROL MODEL and data logging via Bluetooth.

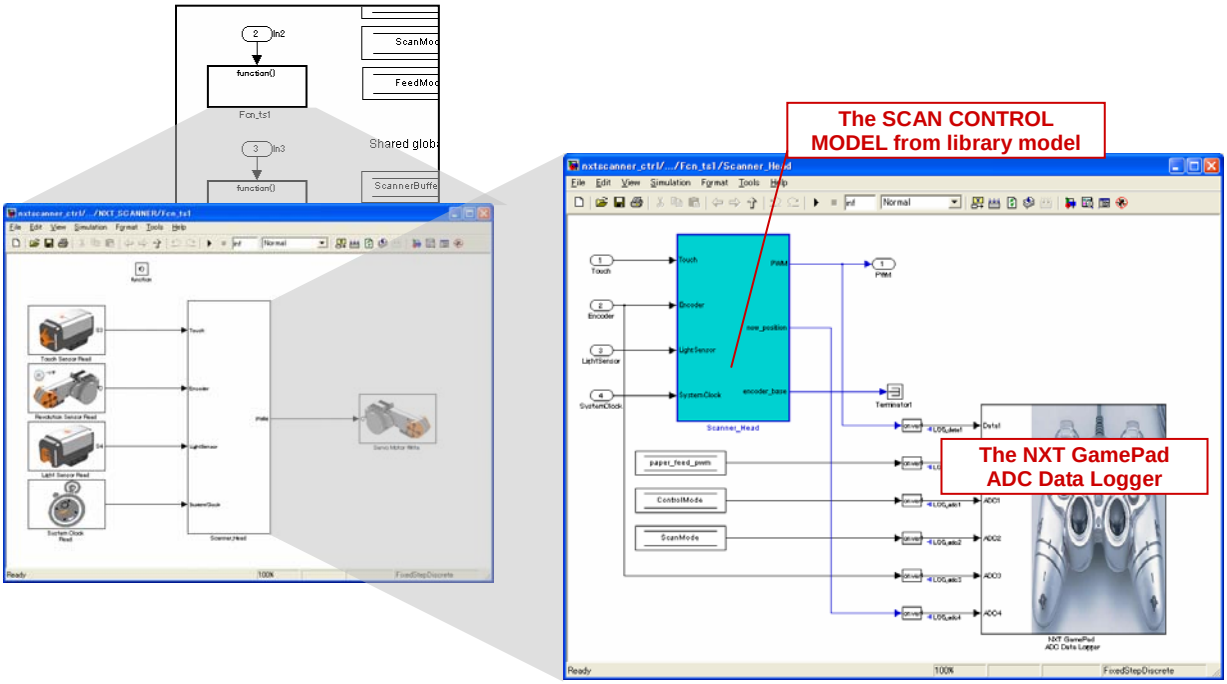


Figure 8-7 task_ts1

8.5 10ms task: task_ts2

This task includes the PAPER FEED CONTROL MODEL.

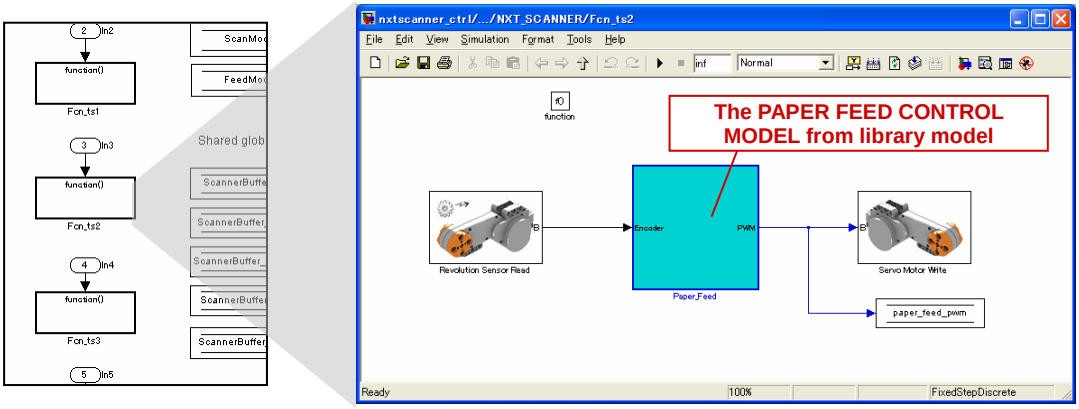


Figure 8-8 task_ts2

8.6 20ms task: task_ts3

This task includes the MODE CONTROL MODEL.

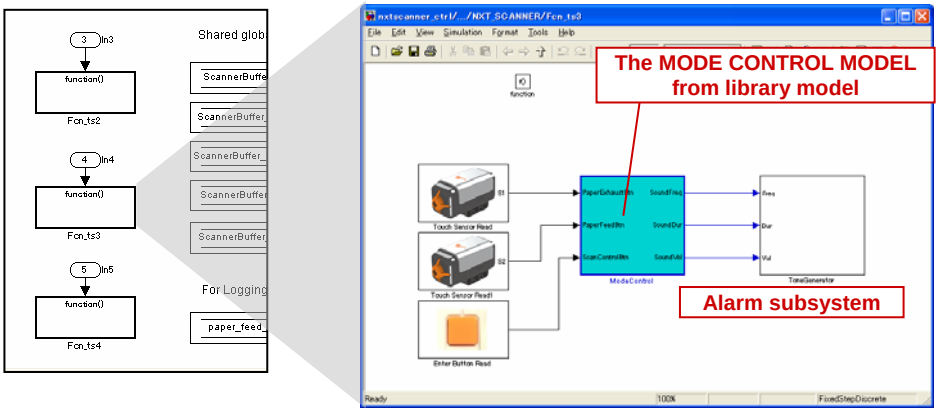


Figure 8-9 task_ts3

8.7 60ms task: task_ts4

This task includes the USB COMMUNICATION CONTROL MODEL.

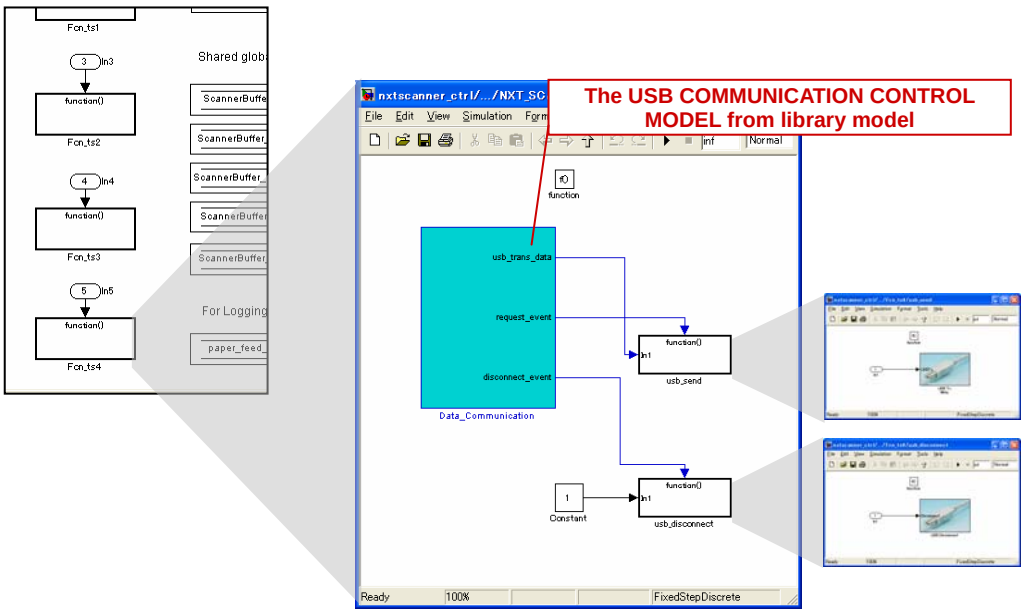


Figure 8-10 task_ts4

8.8 Tuning parameters

All parameters used in the nxtscanner_ctrl.mdl are defined by the nxtscanner.xls. Table 8-2 shows the tuning parameters. You might have to tune these parameters because the parts, blocks, sensors, and actuators are individually different.

Table 8-2 Tuning parameters

Parameter	Description
BACKLASH_ADJUST	backlash adjust
RELEASE_TOUCH_SENSOR	freeing from the touch sensor adjust
SCAN_EDGE	the scanner head unit drives range
SCAN_START	scan start position
SCAN_INITIALIZE_PWM	PWM for moving the scanner head initialization
SCAN_INITIALIZE_TIMEOUT	time out for the scanner head moving initialization [ms]
SCAN_R_TIMEOUT	time out for the scanner head moving to right [ms]
SCAN_L_TIMEOUT	time out for the scanner head moving to left [ms]
SCAN_F_TIMEOUT	time out for waiting the paper feed with scanning [ms]
SCAN_R_MAX_PWM	MAX PWM for the scanner head moving to right
SCAN_R_MIN_PWM	MIN PWM for the scanner head moving to right
SCAN_L_MAX_PWM	MAX PWM for the scanner head moving to left
SCAN_L_MIN_PWM	MIN PWM for the scanner head moving to left

9 Code generation and implementation

This chapter describes how to generate code from the nxtscanner_ctrl.mdl and download it to the NXT Intelligent Brick. The experimental results are also shown.

9.1 Target hardware and software

Table 9-1 shows the target hardware specification of LEGO Mindstorms NXT and the software used in Embedded Coder Robot NXT.

Table 9-1 LEGO Mindstorms NXT & Embedded Coder Robot NXT specification

hardware	processor	ATMEL 32-bit ARM 7 (AT91SAM7S256) 48MHz
	flash memory	256 Kbytes (10000 times writing guarantee)
	RAM	64 Kbytes
interface	actuator	3 DC motor
	sensor	ultrasonic, touch sensor, light sensor, sound sensor
	display	100 * 64 pixel LCD
	communication	Bluetooth / USB
software	RTOS	nxtOSEK / nxtJSP
	compiler	GCC
	library	GCC library

9.2 How to generate code and download

You can generate code from the model, build it, and download the program into NXT by clicking the annotations in nxtscanner_ctrl.mdl shown in Figure 9-1. The procedure as follows:

1. Setting Simulink data object by clicking [**xlsreadsdo (`nxtscanner.xls`,'mpt`)**]. Simulink data object can add the generated code to the user's specific information (name, allocation, instruction modifier, etc.). For details, please refer to the reference [3].
2. Generate code and build the generated code by clicking [**Generate code and build the generated code**].
3. Connect NXT and PC via USB. Download the program into NXT by clicking [**Download the built code to NXT**].

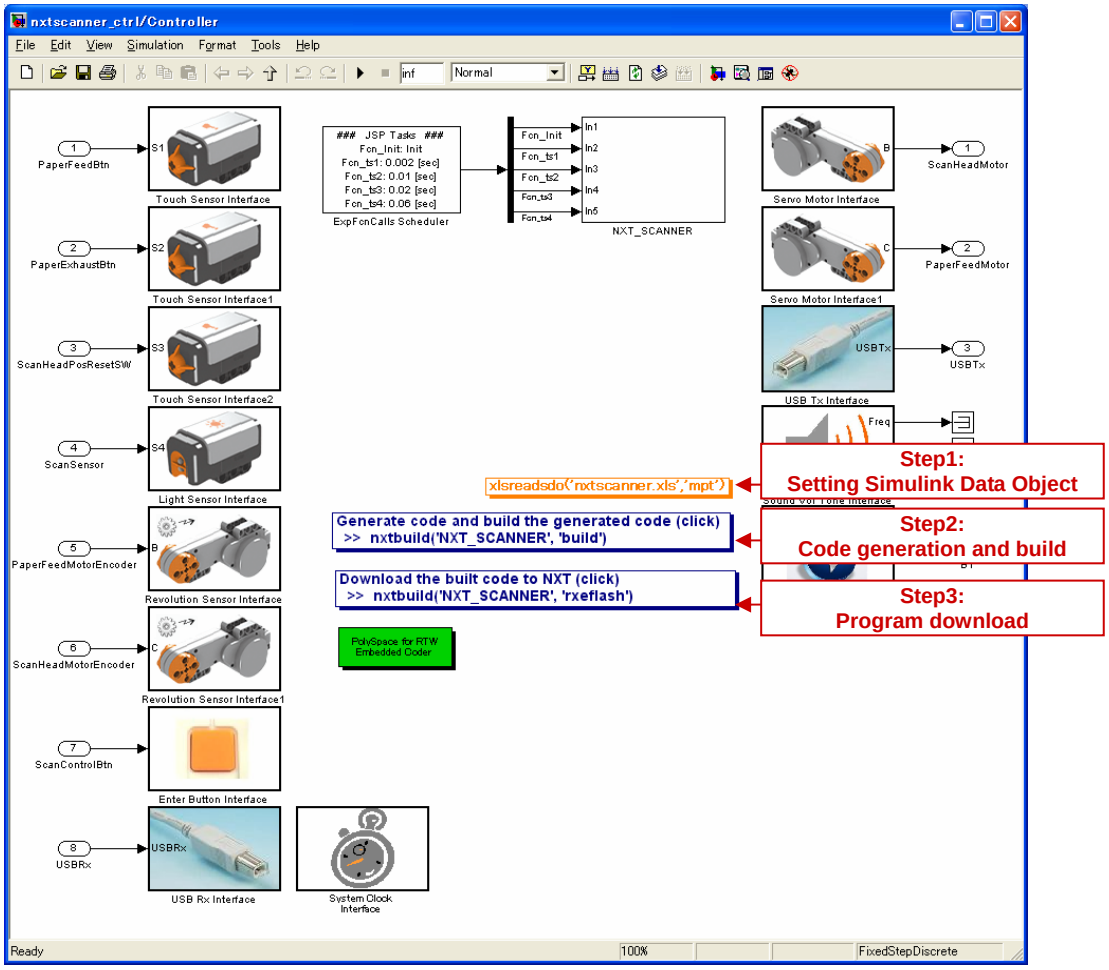


Figure 9-1 Annotations for code generation & build / download

A part of the generated code is described in Appendix.

10 Verification of generated code

Generated code from the NXT Scanner is validated by PolySpace®. This chapter describes the configuration for validation and it doesn't describe about installation.

10.1 What is PolySpace?

PolySpace is abstract interpretation static verification tool that is necessary for high reliability software development. PolySpace has two features; one is that PolySpace can find runtime errors, and the second is that PolySpace can certify that errors don't exist. PolySpace uses color-coding to indicate the status of each element in the code, as follows:

- Green: Reliable (It shows safe instructions: these are code sections which can never lead to a runtime error.)
- Red: Faulty (It shows runtime errors will occur every time that piece of code is executed)
- Gray: Dead (It shows code which is unreachable (dead code))
- Orange: Unproven (It is a warning)

You can use PolySpace to verify handwritten code, generated code, or a combination of the two, before compilation and test.

10.2 PolySpace configuration

PolySpace Analyzer is opened pushing the block button of Figure 10-1 or from the select menu in the model window.

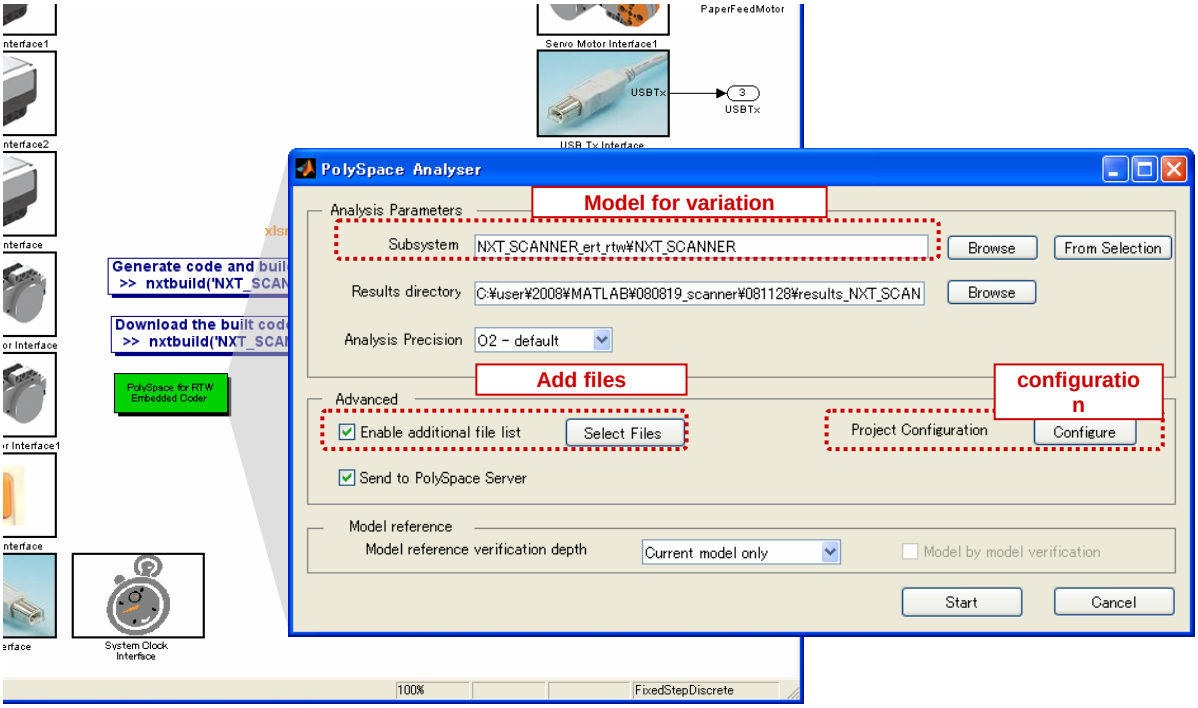


Figure 10-1 PolySpace setting menu

Caution

PolySpace configuration something use direct path, please be careful.

Model for validation

This automatically to sets the model path for PolySpace. It means that the NXT_SCANNER subsystem is validated by Simulation and generated code in the NXT_SCANNER_ert_rtw folder is validated by PolySpace.

Configuration

It opens the configuration setting window. A sample window is shown in Figure 10-2. Changing from default value for validation is shown by red boxed texts.

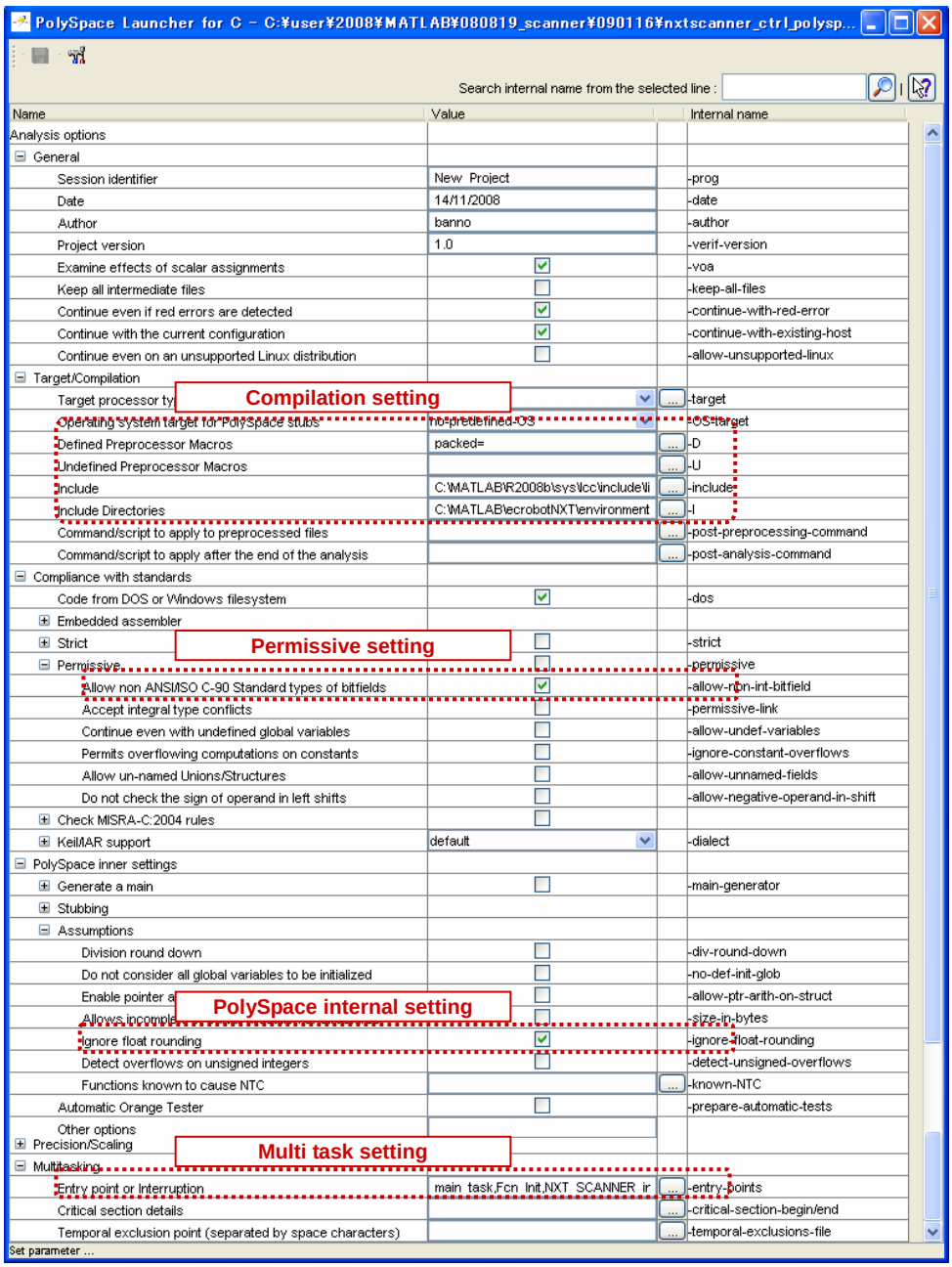


Figure 10-2 PolySpace Configuration diapley

Generated code in the NXT_SCANNE_ert_rtw folder does not have multitasking information. Thus PolySpace uses the original main function, polyspace_main.c, to validate model. The polyspace_main.c defines the entry points of the Fcn_Init function, The NXT_SCANNER_initialize function and the main_task function.

Add files

The polyspace_main.c is configuration of multitasking definition. The NXT Scanner has task subsystems which are driven by a trigger from scheduler. The main_task function in the polyspace_main.c defines each task process without initialize task. The task cycle is arbitrary set for verification of PolySpace.

polyspace_main.c

```
#include "NXT_SCANNER.h"

extern int anyvalue(void);

int main(void){
    Fcn_Init();
    NXT_SCANNER_initialize();
    while(1){
        main_task();
    }
    return(0);
}
void main_task(void){
    while(anyvalue()){
        if(anyvalue()){
            Fcn_ts1();
        }
        if(anyvalue()){
            Fcn_ts2();
        }
        if(anyvalue()){
            Fcn_ts3();
        }
        if(anyvalue()){
            Fcn_ts4();
        }
    }
}
```

10.3 PolySpace can find runtime errors

The NXT Scanner has intentional error code modeling for PolySpace demo. The code is in a state of RUNTIME_ERROR_STOP which is driven by “RUNTIME_ERROR” event in Stateflow of the SCAN CONTROL MODEL. When PolySpace finds a runtime error, the result is a warning by red color as in Figure 10-3.

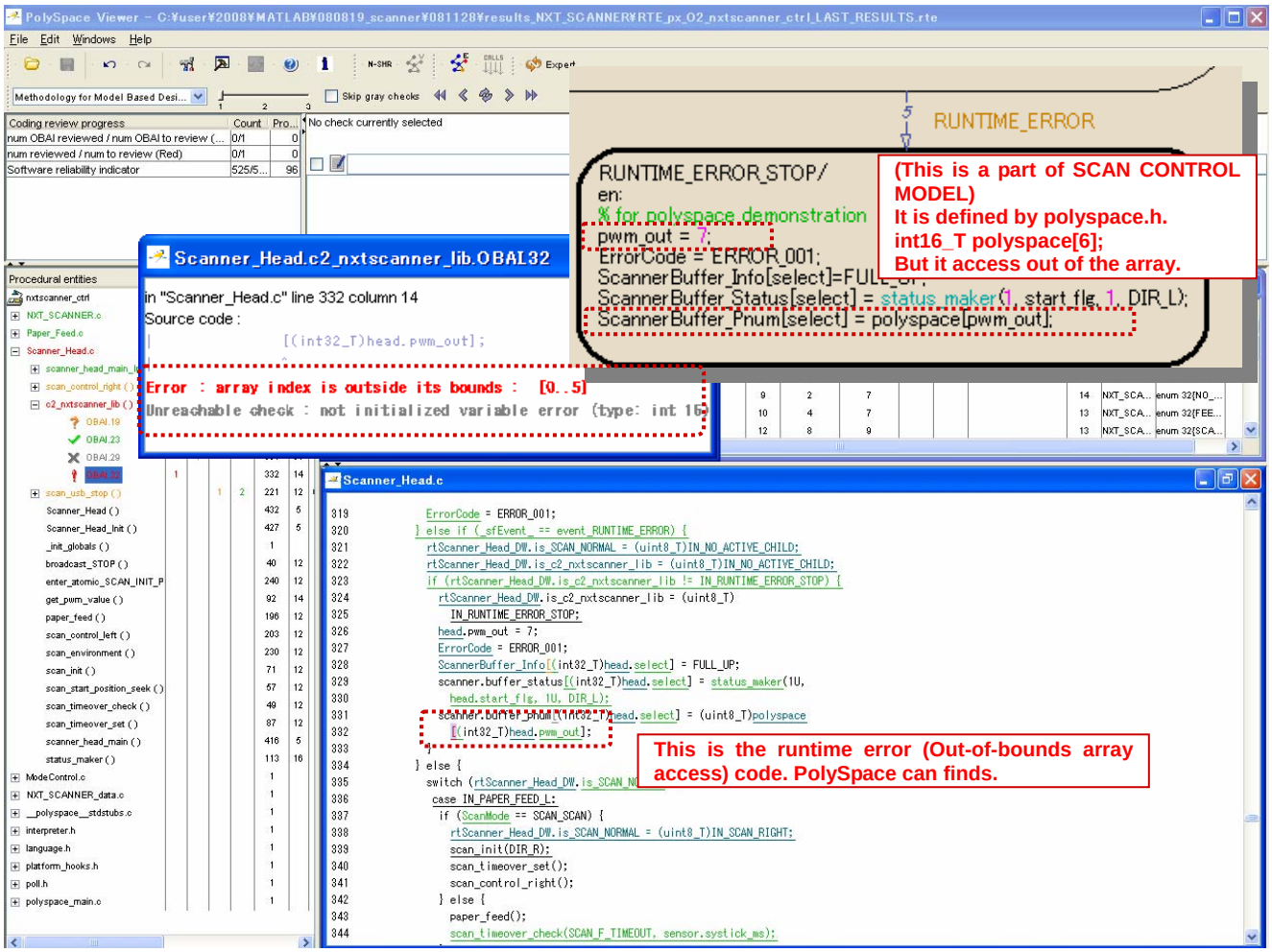


Figure 10-3 Example of PolySpace finds runtime error

Note

Runtime error is occurs when there is no syntax error in the program code but comes during execution of the program. PolySpace can find the following errors without test cases, without code instrumentation and without execution.

- Overflows, underflows, division by zero, and other arithmetic errors
- Out-of-bounds array access and illegally dereferences pointers
- Read access to non-initialized data
- Dead code
- Access to null this pointer (C++)
- Dynamic errors related to object programming, inheritance, and exception handling (C++)
- Non-initialized class members (C++)
- Other errors, including dangerous type conversions

10.4 Results of PolySpace verification

After commenting out of the runtime error code, PolySpace tests again. Figure 10-4 is the result window of PolySpace. There is no red and gray code, but orange has 18 points. Orange means it is not red, gray, or green. Orange is checked by review.

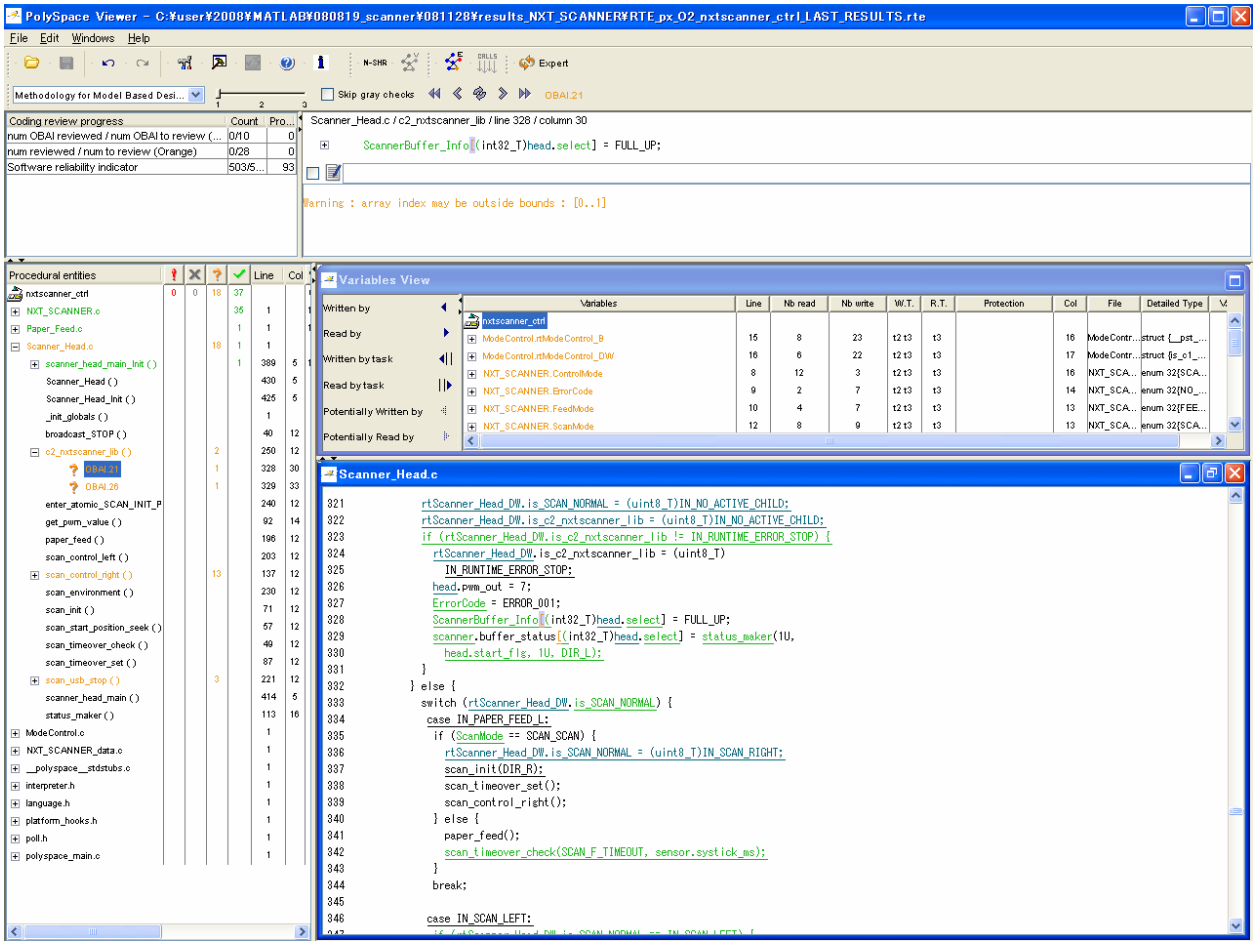


Figure 10-4 Results PolySpace verification

11 What is an NXT Viewer?

The NXT Viewer is a proprietary the NXT Scanner's GUI program. It has two features as below.

- Image display: Receive scan data from the NXT Scanner via USB, and display them in real time for each line. Merge received data to show image data.
- Image processing: Image processing after the data has been received

The NXT Viewer was developed using a feature called GUIDE, which is a function to create GUI for MATLAB. GUIDE is based on M-language. Particularly, the image processing function of the NXT Viewer uses the Image Processing Toolbox, which is an optional product of MATLAB.

11.1 How to use the NXT Viewer

This chapter describes how to use the NXT Viewer.

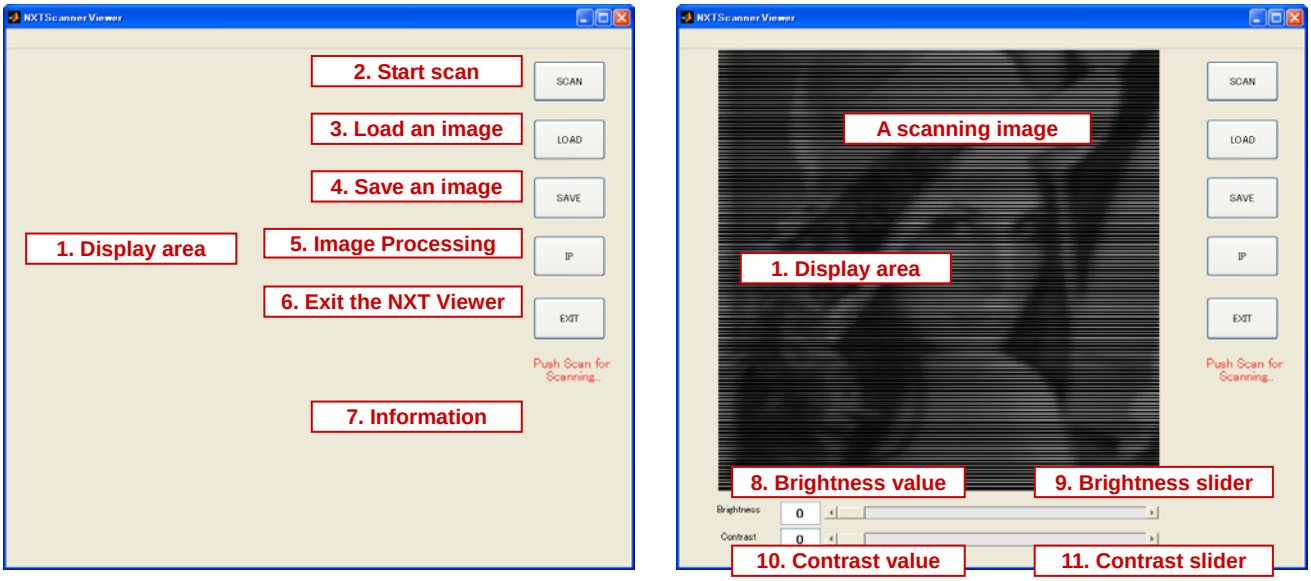


Figure 11-1 Design of the NXT Viewer display

Table 11-1 is list of details for each component of the NXT Viewer introduced in Figure 11-1.

Table 11-1 The NXT Viewer specification

No.	compone nt name	component type	details
1	–	axis (viewer area)	Scanning: real time viewer of scanning Other: viewer of LOAD image from mat file (For details, please refer to section 11.3) The image size is M rows and 255 lines (M =< 255).
2	SCAN	push button	Display a confirmation dialogue. Clicking OK button starts waiting data and scanning. The scan stops if the scan data reached 255 lines, or if the user is forced to stop scan.
3	LOAD	push button	Display a dialogue for selecting MAT file. After the MAT file is selected, the scan data is loaded to the viewer area.
4	SAVE	push button	Display a dialogue for saving MAT file. The scan data is saved to the MAT file after deciding which file to save.
5	IP	push button	It is the Image processing for viewer area (scan image or load image). After adjusting, the result is shown in another viewer display. (For details, please refer to Section 11.4)
6	EXIT	push button	Display a confirmation dialogue. Clicking OK button closes the NXT Viewer.
7	–	static text	Display current status and information.
8	–	edit text	Display the brightness level.
9	–	slider	Adjust the brightness level (0 to100)
10	–	edit text	Display the contrast level.
11	–	slider	Setting the contrast level (0 to100)

11.2 Command for USB communication (nxtusb)

The NXT Viewer uses a command (nxtusb) for USB communication to receive scan data from the NXT Scanner. The nxtusb communicates with the USB driver (fantom) of Mindstorms NXT via its own private functions. The private function is created as a MEX-file. (It is a feature to build an executable function called by MATLAB).

NXTScannerViewer.m

```
(elision)
% make a nxtusb object
nuObj = nxtusb;
  :
% USB data receive by nxtusb
[len, buf] = read(nuObj, 'uint16', 32);
  :
% USB terminate
delete(nuObj);
clear nuObj
(elision)
```

11.3 Overview of the image display

The NXT Viewer displays gray scale image. Each raw data consists of 10 bits data received from the NXT Scanner. The NXT Viewer converts the raw data (data type is uint16) to uint8 (0-255). This conversion method is detected by experiment and data logging.

NXTScannerViewer.m

```
(elision)
% simple brightness adjust
data1 = uint8(747 - data1);
scanData(row, 1:length(data1)) = data1;
imshow(scanData), drawnow
(elision)
```

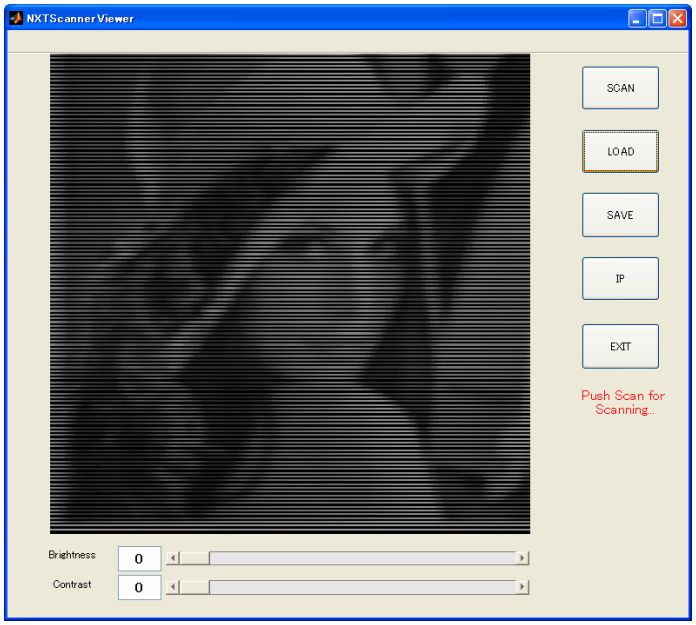


Figure 11-2 The NXT Viewer display

11.4 About image processing

The NXT Viewer has the following three stages for image processing.

1. Color and contrast adjustment (Figure1)
2. Interpolation (Figure2)
3. Deconvolution (Figure3)

Color and contrast adjustment

This stage is an optimization of the gray scale. The red bold texts in the following M-Script indicate the commands of the Image Processing Toolbox.

- Imadjust：Adjust image intensity values.
- Imshow：Handle Graphics figure.

NXTScannerViewer.m

```
(elision)
% color and contrast adjustment
brLvl = get(handles.slider1,'Value');
cnLvl = get(handles.slider2, 'Value');
scanData = scanData + brLvl;
scanData = scanData * ((100 + cnLvl) / 100);
[xs,ys] = size(scanData);
mx = max(scanData(1:2:end));
mn = min(scanData(1:2:end));
scanData = imadjust(scanData,[double(mn)/255 double(mx)/255],[0 1],1);
(elision)
figure, imshow(scanData)
```

Figure 11-3　Color and contrast adjustment by the NXT Viewer (Figure1)

Interpolation

This stage is the process to interpolate data between each line. This process is needed because the NXT Scanner sends interlace scan data. It skips one line after scan each line is scanning by jump a line. So an image has a deficit.

- imresize：Resize image by Lanczos3 interpolated method.
- edgetaper：Taper edges for area of edge of an image
- Imshow：Display an image on Handle Graphics figure.

NXTScannerViewer.m

```
(elision)
% interpolation
I1 = scanData(1:2:end,:);
I1 = imresize(I1,[xs,ys],'lanczos3');
scanData2 = I1;
k = ones(3,6);
k = k/(size(k,1)*size(k,2));
scanData2 = edgetaper(scanData2,k);
(elision)
figure, imshow(scanData2)
```

Figure 11-4　Image interpolation by the NXT Viewer (Figure2)

Deconvolution

This is the deconvolution process for blurred image.

- edge：Find edges in intensity image.
- strel：Create morphological structuring element.
- imdilate：Dilate image.
- deconvblind：Using blind deconvolution for to remove blur

NXTScannerViewer.m

```
(elision)
% deconvolution
a = edge(scanData2,'sobel',0.10);
se = strel('rect',[2 5]);
b = imdilate(a,se);
b = ~b;
b([1:3 end-[0:2]],:) = 0;
b(:,[1:3 end-[0:2]]) = 0;
[J,PSF] = deconvblind(scanData2,k,22,uint8(0),double(b),uint8(255));
(elision)
figure, imshow(J)
```

Figure 11-5　Image deconvolution by the NXT Viewer (Figure3)

12 Experimental results

You can watch a movie of the NXT Scanner control experiment and the NXT Viewer image at the following URL.

http://www.youtube.com/watch?hl=en&v=hadVwPJw3_0

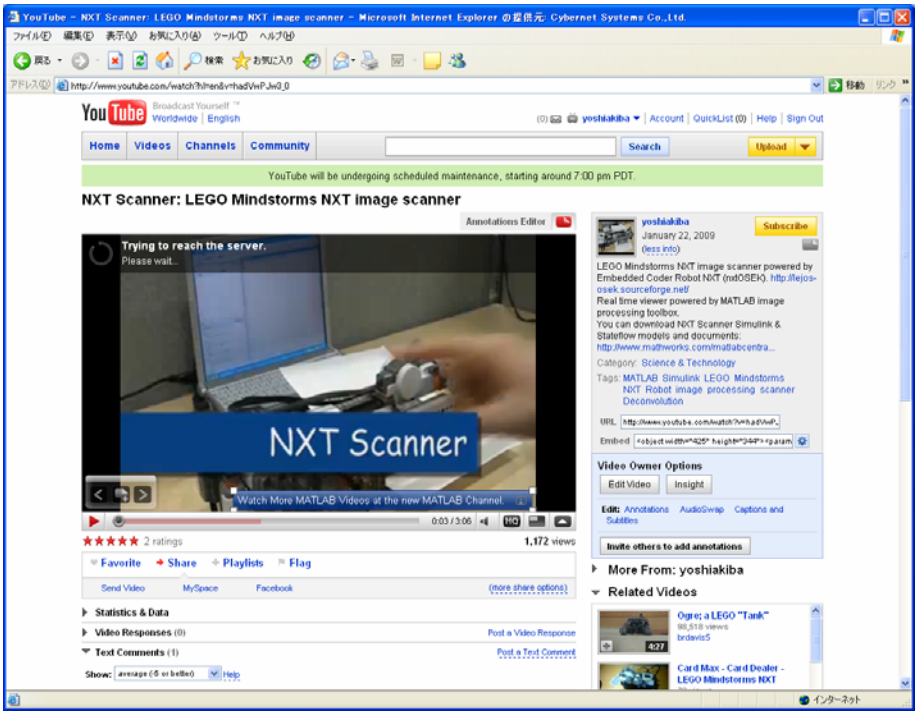


Figure 12-1 The Movie of the NXT Scanner and the NXT Viewer

13 Challenges for readers

We provide the following problems as challenges for readers. Please try them if you have interest.

- The NXT Scanner: faster reduction of scanning time
- The NXT Scanner: improve how to build (for example, the scanner head change to use from a gear to a tire)
- The NXT Viewer: improve the image processing algorithm

Appendix Generated code

This appendix describes the main code generated from nxtscanner_ctrl.mdl. It is the default code generated by RTW-EC. RTW-EC allows us to assign user variable attributes such as variable name, storage class, modifiers etc. by using Simulink Data Object. The comments are omitted for compactness.

NXT_SCANNER.c

```
#include "NXT_SCANNER.h"
#include "NXT_SCANNER_private.h"

#define IN_CLOSE                       (1)
#define IN_OPEN                        (2)

uint8_T _sfEvent_;
ControlModeEnum ControlMode;
ErrorCodeEnum ErrorCode;
FeedModeEnum FeedMode;
ScannerBufferInfoEnum ScannerBuffer_Info[2];
ScanModeEnum ScanMode;
head_type head;
logger_type logger;
scanner_type scanner;
sensor_type sensor;
usb_type usb;
BlockIO rtB;
D_Work rtDWork;
PrevZCSigStates rtPrevZCSigState;
void Fcn_Init(void)
{
  ecrobot_set_motor_mode_speed(NXT_PORT_B, 1, 0);
  ecrobot_set_motor_mode_speed(NXT_PORT_C, 1, 0);
}

void Fcn_ts1_Init(void)
{
  Scanner_Head_Init();
}

void Fcn_ts1(void)
{
  sensor.scanner_head_reset_sw = ecrobot_get_touch_sensor(NXT_PORT_S3);
  sensor.scanner_head_rev = ecrobot_get_motor_rev(NXT_PORT_C);
  sensor.light_data = ecrobot_get_light_sensor(NXT_PORT_S4);
  sensor.systick_ms = ecrobot_get_systick_ms();
  Scanner_Head();
  ecrobot_set_motor_mode_speed(NXT_PORT_C, 1, head.pwm_out);
  logger.LOG_data1 = head.pwm_out;
  logger.LOG_data2 = rtDWork.paper_feed_pwm;
  logger.LOG_adc1 = (int16_T)ControlMode;
  logger.LOG_adc2 = (int16_T)ScanMode;
  logger.LOG_adc3 = (int16_T)sensor.scanner_head_rev;
  logger.LOG_adc4 = (int16_T)head.now_position;
  ecrobot_bt_adc_data_logger(logger.LOG_data1, logger.LOG_data2, logger.LOG_adc1,
    logger.LOG_adc2, logger.LOG_adc3, logger.LOG_adc4);
}

void Fcn_ts2_Init(void)
{
  Paper_Feed_Init();
}

void Fcn_ts2_Disable(void)
{
  Paper_Feed_Disable();
}
```

```
void Fcn_ts2_Start(void)
{
  Paper_Feed_Start();
}

void Fcn_ts2(void)
{
  rtB.RevolutionSensor_B = ecrobot_get_motor_rev(NXT_PORT_B);
  Paper_Feed();
  rtDWork.paper_feed_pwm = rtB.PWM;
  ecrobot_set_motor_mode_speed(NXT_PORT_B, 1, rtB.PWM);
}

void Fcn_ts3_Init(void)
{
  ModeControl_Init();
}

void Fcn_ts3(void)
{
  {
    boolean_T rtb_RelationalOperator_k;
    sensor.paper_exhaust_sw = ecrobot_get_touch_sensor(NXT_PORT_S1);
    sensor.paper_feed_sw = ecrobot_get_touch_sensor(NXT_PORT_S2);
    sensor.scan_control_sw = ecrobot_is_ENTER_button_pressed();
    ModeControl();
    rtb_RelationalOperator_k = (rtModeControl_B.SoundVol > 0U);
    if (rtb_RelationalOperator_k && (rtPrevZCSigState.EnabledSubsystem_Trig_ZCE
        != POS_ZCSIG)) {
      ecrobot_sound_tone(rtModeControl_B.SoundFreq, rtModeControl_B.SoundDur,
                         rtModeControl_B.SoundVol);
    }

    rtPrevZCSigState.EnabledSubsystem_Trig_ZCE = rtb_RelationalOperator_k ?
      POS_ZCSIG : ZERO_ZCSIG;
  }
}

void usb_send(void)
{
  uint8_T rtb_TmpHiddenBufferAtUSBTxWrite[64];

  {
    int32_T i;
    for (i = 0; i < 2; i++) {
      rtb_TmpHiddenBufferAtUSBTxWrite[i] = usb.send_status[i];
    }

    for (i = 0; i < 2; i++) {
      rtb_TmpHiddenBufferAtUSBTxWrite[i + 2] = usb.send_number[i];
    }

    for (i = 0; i < 60; i++) {
      rtb_TmpHiddenBufferAtUSBTxWrite[i + 4] = usb.send_data[i];
    }

    ecrobot_send_usb(rtb_TmpHiddenBufferAtUSBTxWrite, 0, MAX_USB_DATA_LEN);
  }
}
```

```
static void usb_communication(uint8_T sf_arg_select_buffer);
static void init_buffer(uint8_T sf_arg_select_buf);
static void usb_function(void);
static uint8_T usb_final_check(void);
static void usb_communication(uint8_T sf_arg_select_buffer)
{
  uint8_T sf_index;
  usb.send_status[1] = 0U;
  usb.send_status[0] = (uint8_T)scanner.buffer_status[sf_arg_select_buffer];
  usb.send_number[1] = 0U;
  usb.send_number[0] = scanner.buffer_number[sf_arg_select_buffer];
  for (sf_index = 0U; sf_index < 30; sf_index++) {
    usb.send_data[(sf_index << 1) + 1] = (uint8_T)((scanner.buffer_data
      [(sf_index << 1) + sf_arg_select_buffer] & 0xFF00) >> 8);
    usb.send_data[sf_index << 1] = (uint8_T)(scanner.buffer_data[(sf_index << 1)
      + sf_arg_select_buffer] & 0x00FF);
  }
}

static void init_buffer(uint8_T sf_arg_select_buf)
{
  uint8_T sf_index;
  ScannerBuffer_Info[sf_arg_select_buf] = TRANS_FINISH;
  scanner.buffer_status[sf_arg_select_buf] = 0U;
  scanner.buffer_number[sf_arg_select_buf] = 0U;
  scanner.buffer_pnum[sf_arg_select_buf] = 0U;
  for (sf_index = 0U; sf_index < 30; sf_index++) {
    scanner.buffer_data[sf_arg_select_buf + (sf_index << 1)] = 0U;
  }
}

static void usb_function(void)
{
  int32_T i;
  for (i = 0; i < 2; i++) {
    usb.send_status[i] = 0U;
    usb.send_number[i] = 0U;
  }

  for (i = 0; i < 60; i++) {
    usb.send_data[i] = 0U;
  }

  rtDWork.disconnect = 0U;
  if ((ScannerBuffer_Info[0] == FULL_UP) && (ScannerBuffer_Info[1] == FULL_UP))
  {
    if (scanner.buffer_pnum[0] <= scanner.buffer_pnum[1]) {
      usb_communication(0U);
      init_buffer(0U);
      usb_send();
    } else {
      usb_communication(1U);
      init_buffer(1U);
      usb_send();
    }
  } else if (ScannerBuffer_Info[0] == FULL_UP) {
    usb_communication(0U);
    init_buffer(0U);
    usb_send();
  } else {
    if (ScannerBuffer_Info[1] == FULL_UP) {
      usb_communication(1U);
      init_buffer(1U);
      usb_send();
    }
  }
}
```

```
void usb_communication_main_Init(void)
{
  {
    int32_T i;
    rtDWork.is_active_c4_nxtscanner_lib = 0U;
    rtDWork.is_c4_nxtscanner_lib = 0U;
    rtDWork.disconnect = 0U;
    for (i = 0; i < 2; i++) {
      usb.send_status[i] = 0U;
      usb.send_number[i] = 0U;
    }

    for (i = 0; i < 60; i++) {
      usb.send_data[i] = 0U;
    }
  }
}

void usb_communication_main(void)
{
  if (rtDWork.is_active_c4_nxtscanner_lib == 0) {
    rtDWork.is_active_c4_nxtscanner_lib = 1U;
    rtDWork.is_c4_nxtscanner_lib = (uint8_T)IN_OPEN;
    usb_function();
    rtDWork.disconnect = usb_final_check();
  } else {
    switch (rtDWork.is_c4_nxtscanner_lib) {
     case IN_CLOSE:
      if (rtDWork.disconnect == 0) {
        rtDWork.is_c4_nxtscanner_lib = (uint8_T)IN_OPEN;
        usb_function();
        rtDWork.disconnect = usb_final_check();
      }
      break;

     case IN_OPEN:
      if (rtDWork.disconnect == 1) {
        rtDWork.is_c4_nxtscanner_lib = (uint8_T)IN_CLOSE;
        if (((uint8_T)1U) == 1) {
          ecrobot_disconnect_usb();
        }

        rtDWork.disconnect = 0U;
      } else {
        usb_function();
        rtDWork.disconnect = usb_final_check();
      }
      break;

     default:
      rtDWork.is_c4_nxtscanner_lib = (uint8_T)IN_OPEN;
      usb_function();
      rtDWork.disconnect = usb_final_check();
      break;
    }
  }
}

void Fcn_ts4_Init(void)
{
  usb_communication_main_Init();
}

void Fcn_ts4(void)
{
  usb_communication_main();
}
```

```
void NXT_SCANNER_initialize(void)
{
  {
    rtB.ErrorCode_i = NO_ERROR;
  }

  ControlMode = ERROR;
  ErrorCode = NO_ERROR;
  FeedMode = FEED_INIT;

  {
    int_T i;
    for (i = 0; i < 2; i++) {
      ScannerBuffer_Info[i] = TRANS_FINISH;
    }
  }

  ScanMode = SCAN_INIT;
  ModeControl_initialize();
  Fcn_ts2_Start();
  ErrorCode = NO_ERROR;
  rtPrevZCSigState.EnabledSubsystem_Trig_ZCE = POS_ZCSIG;
  rtPrevZCSigState.Error_Detection_Trig_ZCE = POS_ZCSIG;
  _sfEvent_ = CALL_EVENT;
  Fcn_ts1_Init();
  Fcn_ts2_Init();
  Fcn_ts3_Init();
  Fcn_ts4_Init();
}
```

ModeControl.c

```
#include "ModeControl.h"

#include "NXT_SCANNER.h"
#include "NXT_SCANNER_private.h"

#define IN_ERROR                       (1)
#define IN_IDLE                        (1)
#define IN_INITIALIZE                   (2)
#define IN_NORMAL                      (2)
#define IN_NO_ACTIVE_CHILD_e0           (0)
#define IN_PAPER_EXHAUST                (3)
#define IN_PAPER_FEED                  (4)
#define IN_SCAN_CONTROL                 (5)

rtB_ModeControl rtModeControl_B;
rtDW_ModeControl rtModeControl_DW;
void ModeControl_Init(void)
{
  rtModeControl_DW.is_NORMAL = 0U;
  rtModeControl_DW.is_active_c1_nxtscanner_lib = 0U;
  rtModeControl_DW.is_c1_nxtscanner_lib = 0U;
  rtModeControl_B.ControlMode_d = ERROR;
  rtModeControl_B.SoundFreq = 0U;
  rtModeControl_B.SoundDur = 0U;
  rtModeControl_B.SoundVol = 0U;
}

void ModeControl(void)
{
  {
    ScanModeEnum rtb_DataStoreRead2_k;
    rtb_DataStoreRead2_k = ScanMode;
    if (((ScanMode == SCAN_INIT) && (SCAN_INIT !=
         rtModeControl_DW.UnitDelay1_DSTATE)) || ((sensor.scan_control_sw ==
         ((uint8_T)1U)) && (rtModeControl_DW.UnitDelay2_DSTATE == ((uint8_T)0U))))
    {
      if (rtModeControl_DW.UnitDelay2_DSTATE_m >= ((uint8_T)1U)) {
        rtModeControl_B.Switch = ((uint8_T)0U);
      } else {
        rtModeControl_B.Switch = ((uint8_T)1U);
      }

      rtModeControl_DW.UnitDelay2_DSTATE_m = rtModeControl_B.Switch;
    }

    rtModeControl_DW.UnitDelay1_DSTATE = rtb_DataStoreRead2_k;
    rtModeControl_DW.UnitDelay2_DSTATE = sensor.scan_control_sw;
    if (rtModeControl_DW.is_active_c1_nxtscanner_lib == 0) {
      rtModeControl_DW.is_active_c1_nxtscanner_lib = 1U;
      rtModeControl_DW.is_c1_nxtscanner_lib = (uint8_T)IN_NORMAL;
      rtModeControl_DW.is_NORMAL = (uint8_T)IN_INITIALIZE;
      rtModeControl_B.ControlMode_d = INITIALIZE;
    } else {
      switch (rtModeControl_DW.is_c1_nxtscanner_lib) {
       case IN_ERROR:
        break;

       case IN_NORMAL:
        if (ErrorCode != NO_ERROR) {
          if (ErrorCode == ERROR_001) {
            rtModeControl_B.SoundFreq = 880U;
          } else {
            rtModeControl_B.SoundFreq = 220U;
          }
          rtModeControl_DW.is_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD_e0;
          rtModeControl_DW.is_c1_nxtscanner_lib = (uint8_T)IN_ERROR;
          rtModeControl_B.ControlMode_d = ERROR;
          rtModeControl_B.SoundDur = 3000U;
          rtModeControl_B.SoundVol = 20U;
```

```
      } else {
        switch (rtModeControl_DW.is_NORMAL) {
         case IN_IDLE:
          if (rtModeControl_B.Switch == 1) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_SCAN_CONTROL;
            rtModeControl_B.ControlMode_d = SCAN;
          } else if (sensor.paper_feed_sw == 1) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_PAPER_FEED;
            rtModeControl_B.ControlMode_d = PAPERFEED;
          } else {
            if (sensor.paper_exhaust_sw == 1) {
              rtModeControl_DW.is_NORMAL = (uint8_T)IN_PAPER_EXHAUST;
              rtModeControl_B.ControlMode_d = PAPEREXHAUST;
            }
          }
          break;

         case IN_INITIALIZE:
          if ((ScanMode == SCAN_IDLE) && (FeedMode == FEED_IDLE)) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_IDLE;
            rtModeControl_B.ControlMode_d = IDLE;
          }
          break;

         case IN_PAPER_EXHAUST:
          if (sensor.paper_exhaust_sw == 0) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_IDLE;
            rtModeControl_B.ControlMode_d = IDLE;
          }
          break;

         case IN_PAPER_FEED:
          if (sensor.paper_feed_sw == 0) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_IDLE;
            rtModeControl_B.ControlMode_d = IDLE;
          }
          break;

         case IN_SCAN_CONTROL:
          if (rtModeControl_B.Switch == 0) {
            rtModeControl_DW.is_NORMAL = (uint8_T)IN_INITIALIZE;
            rtModeControl_B.ControlMode_d = INITIALIZE;
          }
          break;

         default:
          rtModeControl_DW.is_NORMAL = (uint8_T)IN_INITIALIZE;
          rtModeControl_B.ControlMode_d = INITIALIZE;
          break;
        }
       }
       break;

      default:
       rtModeControl_DW.is_c1_nxtscanner_lib = (uint8_T)IN_NORMAL;
       rtModeControl_DW.is_NORMAL = (uint8_T)IN_INITIALIZE;
       rtModeControl_B.ControlMode_d = INITIALIZE;
       break;
     }
    }

    ControlMode = rtModeControl_B.ControlMode_d;
  }
}

void ModeControl_initialize(void)
{
  {
    rtModeControl_B.ControlMode_d = ERROR;
  }
}
```

Scanner_Head.c

```
#include "Scanner_Head.h"

#include "NXT_SCANNER.h"
#include "NXT_SCANNER_private.h"

#define DIR_L                         (1U)
#define DIR_R                         (0U)
#define IN_INIT_ERROR_STOP             (1)
#define IN_NO_ACTIVE_CHILD             (0)
#define IN_PAPER_FEED_L                (1)
#define IN_RUNTIME_ERROR_STOP          (2)
#define IN_SCAN_ERROR                  (3)
#define IN_SCAN_ERROR_STOP             (4)
#define IN_SCAN_INIT_POSITION          (5)
#define IN_SCAN_LEFT                   (2)
#define IN_SCAN_NORMAL                 (6)
#define IN_SCAN_RIGHT                  (3)
#define event_FINISH                   (1U)
#define event_RUNTIME_ERROR            (2U)
#define event_STOP                     (0U)

rtDW_Scanner_Head rtScanner_Head_DW;
static void broadcast_STOP(void);
static void scan_timeover_check(uint32_T sf_arg_target_ms, uint32_T
  sf_arg_now_clock);
static void scan_start_position_seek(void);
static void scan_init(uint8_T sf_arg_target_dir);
static void scan_timeover_set(void);
static int8_T get_pwm_value(uint8_T sf_arg_dir, int32_T sf_arg_position, uint8_T
  sf_arg_max, uint8_T sf_arg_min);
static uint16_T status_maker(uint8_T sf_arg_elp, uint8_T sf_arg_start, uint8_T
  sf_arg_stop, uint8_T sf_arg_dir);
static void scan_control_right(void);
static void paper_feed(void);
static void scan_control_left(void);
static void scan_usb_stop(void);
static void scan_environment(void);
static void enter_atomic_SCAN_INIT_POSITION(void);
static void c2_nxtscanner_lib(void);
static void broadcast_STOP(void)
{
  uint8_T sf_previousEvent;
  sf_previousEvent = _sfEvent_;
  _sfEvent_ = event_STOP;
  c2_nxtscanner_lib();
  _sfEvent_ = sf_previousEvent;
}

static void scan_timeover_check(uint32_T sf_arg_target_ms, uint32_T
  sf_arg_now_clock)
{
  if (sf_arg_target_ms < sf_arg_now_clock - head.start_time) {
    broadcast_STOP();
  }
}

static void scan_start_position_seek(void)
{
  if (sensor.scanner_head_reset_sw == 1) {
    head.pwm_out = 0;
    ScanMode = SCAN_IDLE;
    head.scan_init_end_flg = 1U;
  } else if (head.scan_init_end_flg == 1) {
    broadcast_STOP();
  } else {
    head.pwm_out = SCAN_INITIALIZE_PWM;
    scan_timeover_check(SCAN_INITIALIZE_TIMEOUT, sensor.systick_ms);
  }
}
```

```
static void scan_init(uint8_T sf_arg_target_dir)
{
  head.line_end_flg = 0U;
  head.packet_cnt = 0U;
  ScanMode = SCAN_SCAN;
  head.encoder_base = (int32_T)SCAN_START;
  if (sf_arg_target_dir == DIR_L) {
    head.start_position = sensor.scanner_head_rev + BACKLASH_ADJUST;
  } else if (head.start_flg == 1) {
    head.start_position = (sensor.scanner_head_rev - RELEASE_TOUCH_SENSOR) -
      BACKLASH_ADJUST;
  } else {
    head.start_position = sensor.scanner_head_rev - BACKLASH_ADJUST;
  }
}

static void scan_timeover_set(void)
{
  head.start_time = sensor.systick_ms;
}

static int8_T get_pwm_value(uint8_T sf_arg_dir, int32_T sf_arg_position, uint8_T
  sf_arg_max, uint8_T sf_arg_min)
{
  int8_T sf_pwm;
  if (sf_arg_position < 0) {
    sf_pwm = (int8_T)sf_arg_max;
  } else if (sf_arg_position > SCAN_EDGE) {
    sf_pwm = 0;
  } else if (sf_arg_position > SCAN_EDGE - 60) {
    sf_pwm = (int8_T)sf_arg_min;
  } else {
    sf_pwm = (int8_T)sf_arg_max;
  }

  if (sf_arg_dir == 0) {
    return (int8_T)(-sf_pwm);
  }

  return sf_pwm;
}

static uint16_T status_maker(uint8_T sf_arg_elp, uint8_T sf_arg_start, uint8_T
  sf_arg_stop, uint8_T sf_arg_dir)
{
  uint16_T sf_out;
  sf_out = 1U;
  if (sf_arg_elp == 1) {
    sf_out = 17U;
  }

  if (sf_arg_start == 1) {
    sf_out += 8;
  }

  if (sf_arg_stop == 1) {
    sf_out += 4;
  }

  if (sf_arg_dir == 1) {
    sf_out += 2;
  }

  return sf_out;
}
```

```
static void scan_control_right(void)
{
  uint8_T sf_previousEvent;
  head.now_position = head.start_position - sensor.scanner_head_rev;
  head.pwm_out = get_pwm_value(DIR_R, head.now_position, SCAN_R_MAX_PWM,
    SCAN_R_MIN_PWM);
  if ((sensor.scanner_head_reset_sw == 1) && (head.now_position > 0)) {
    broadcast_STOP();
  } else if (head.pwm_out == 0) {
    FeedMode = FEED_PAPERFEED;
    ScanMode = SCAN_FEED;
  } else {
    if ((head.now_position > SCAN_START) && (head.now_position >
         head.encoder_base)) {
      if (head.now_position <= SCAN_EDGE - SCAN_START) {
        head.encoder_base = head.encoder_base + 2;
        scanner.buffer_data[head.select + (scanner.buffer_number[(int32_T)
          head.select] << 1)] = sensor.light_data;
        scanner.buffer_number[(int32_T)head.select] = (uint8_T)
          (scanner.buffer_number[(int32_T)head.select] + 1);
        ScannerBuffer_Info[(int32_T)head.select] = STORING;
        if (scanner.buffer_number[(int32_T)head.select] >= 30) {
          head.packet_cnt = (uint8_T)(head.packet_cnt + 1);
          ScannerBuffer_Info[(int32_T)head.select] = FULL_UP;
          scanner.buffer_status[(int32_T)head.select] = status_maker(0U,
            head.start_flg, 0U, DIR_R);
          scanner.buffer_pnum[(int32_T)head.select] = head.packet_cnt;
          head.start_flg = 0U;
          if (head.select == 1) {
            head.select = 0U;
          } else {
            head.select = 1U;
          }
        }
      } else {
        if (head.line_end_flg == 0) {
          head.line_end_flg = 1U;
          head.line_cnt = head.line_cnt + 1U;
          ScannerBuffer_Info[(int32_T)head.select] = FULL_UP;
          head.packet_cnt = (uint8_T)(head.packet_cnt + 1);
          scanner.buffer_pnum[(int32_T)head.select] = head.packet_cnt;
          if (head.line_cnt >= 127U) {
            scanner.buffer_status[(int32_T)head.select] = status_maker(1U,
              head.start_flg, 1U, DIR_R);
            ScanMode = SCAN_INIT;
            sf_previousEvent = _sfEvent_;
            _sfEvent_ = event_FINISH;
            c2_nxtscanner_lib();
            _sfEvent_ = sf_previousEvent;
          } else {
            scanner.buffer_status[(int32_T)head.select] = status_maker(1U,
              head.start_flg, 0U, DIR_R);
          }
        }
      }
    }
  }
}

static void paper_feed(void)
{
  if (FeedMode == FEED_IDLE) {
    ScanMode = SCAN_SCAN;
  }
}
```

```
static void scan_control_left(void)
{
  uint8_T sf_previousEvent;
  head.now_position = sensor.scanner_head_rev - head.start_position;
  head.pwm_out = get_pwm_value(DIR_L, head.now_position, SCAN_L_MAX_PWM,
    SCAN_L_MIN_PWM);
  if ((sensor.scanner_head_reset_sw == 1) && (head.now_position > 0)) {
    sf_previousEvent = _sfEvent_;
    _sfEvent_ = event_RUNTIME_ERROR;
    c2_nxtscanner_lib();
    _sfEvent_ = sf_previousEvent;
  } else {
    if (head.pwm_out == 0) {
      ScanMode = SCAN_FEED;
    }
  }
}

static void scan_usb_stop(void)
{
  head.pwm_out = 0;
  ScannerBuffer_Info[(int32_T)head.select] = FULL_UP;
  scanner.buffer_status[(int32_T)head.select] = status_maker(1U, head.start_flg,
    1U, DIR_R);
  scanner.buffer_pnum[(int32_T)head.select] = (uint8_T)(head.packet_cnt + 1);
}

static void scan_environment(void)
{
  head.start_flg = 1U;
  head.select = 0U;
  head.line_cnt = 0U;
  head.line_end_flg = 0U;
  head.packet_cnt = 0U;
  head.scan_init_end_flg = 0U;
}

static void enter_atomic_SCAN_INIT_POSITION(void)
{
  if (rtScanner_Head_DW.is_c2_nxtscanner_lib != IN_SCAN_INIT_POSITION) {
    rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_INIT_POSITION;
    scan_environment();
    scan_timeover_set();
    scan_start_position_seek();
  }
}

static void c2_nxtscanner_lib(void)
{
  if (rtScanner_Head_DW.is_active_c2_nxtscanner_lib == 0) {
    rtScanner_Head_DW.is_active_c2_nxtscanner_lib = 1U;
    if (rtScanner_Head_DW.is_c2_nxtscanner_lib != IN_SCAN_INIT_POSITION) {
      rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_INIT_POSITION;
      scan_environment();
      scan_timeover_set();
      scan_start_position_seek();
    }
  } else {
    switch (rtScanner_Head_DW.is_c2_nxtscanner_lib) {
     case IN_INIT_ERROR_STOP:
      if (ControlMode == ERROR) {
        rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_ERROR;
        head.pwm_out = 0;
      }
      break;

     case IN_RUNTIME_ERROR_STOP:
      break;

     case IN_SCAN_ERROR:
      break;
```

```
    case IN_SCAN_ERROR_STOP:
     break;

    case IN_SCAN_INIT_POSITION:
     if (_sfEvent_ == event_STOP) {
       rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_INIT_ERROR_STOP;
       head.pwm_out = 0;
       ErrorCode = ERROR_001;
     } else if (ControlMode == ERROR) {
       rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_ERROR;
       head.pwm_out = 0;
     } else if ((ControlMode == SCAN) && (ScanMode == SCAN_IDLE)) {
       rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_NORMAL;
       if (rtScanner_Head_DW.is_SCAN_NORMAL != IN_SCAN_RIGHT) {
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_SCAN_RIGHT;
         scan_init(DIR_R);
         scan_timeover_set();
         scan_control_right();
       }
     } else {
       scan_start_position_seek();
     }
     break;

    case IN_SCAN_NORMAL:
     if (rtScanner_Head_DW.is_c2_nxtscanner_lib == IN_SCAN_NORMAL) {
       if (ControlMode == ERROR) {
         scan_usb_stop();
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
         rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_ERROR;
         head.pwm_out = 0;
       } else if ((ControlMode != ERROR) && (ControlMode != SCAN)) {
         scan_usb_stop();
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
         rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_NO_ACTIVE_CHILD;
         enter_atomic_SCAN_INIT_POSITION();
       } else if (_sfEvent_ == event_FINISH) {
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
         rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_NO_ACTIVE_CHILD;
         enter_atomic_SCAN_INIT_POSITION();
       } else if (_sfEvent_ == event_STOP) {
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
         rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_SCAN_ERROR_STOP;
         scan_usb_stop();
         ErrorCode = ERROR_001;
       } else if (_sfEvent_ == event_RUNTIME_ERROR) {
         rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
         rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_NO_ACTIVE_CHILD;
         if (rtScanner_Head_DW.is_c2_nxtscanner_lib != IN_RUNTIME_ERROR_STOP) {
           rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)
             IN_RUNTIME_ERROR_STOP;
           head.pwm_out = 7;
           ErrorCode = ERROR_001;
           ScannerBuffer_Info[(int32_T)head.select] = FULL_UP;
           scanner.buffer_status[(int32_T)head.select] = status_maker(1U,
             head.start_flg, 1U, DIR_L);
         }
       } else {
         switch (rtScanner_Head_DW.is_SCAN_NORMAL) {
          case IN_PAPER_FEED_L:
           if (ScanMode == SCAN_SCAN) {
             rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_SCAN_RIGHT;
             scan_init(DIR_R);
             scan_timeover_set();
             scan_control_right();
           } else {
             paper_feed();
             scan_timeover_check(SCAN_F_TIMEOUT, sensor.systick_ms);
           }
           break;
```

```
          case IN_SCAN_LEFT:
           if (rtScanner_Head_DW.is_SCAN_NORMAL == IN_SCAN_LEFT) {
             if (ScanMode == SCAN_FEED) {
               rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_PAPER_FEED_L;
               scan_timeover_set();
               paper_feed();
             } else {
               scan_control_left();
               scan_timeover_check(SCAN_L_TIMEOUT, sensor.systick_ms);
             }
           }
           break;

          case IN_SCAN_RIGHT:
           if (rtScanner_Head_DW.is_SCAN_NORMAL == IN_SCAN_RIGHT) {
             if (ScanMode == SCAN_FEED) {
               ScanMode = SCAN_SCAN;
               rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_SCAN_LEFT;
               scan_init(DIR_L);
               scan_timeover_set();
               scan_control_left();
             } else {
               scan_control_right();
               scan_timeover_check(SCAN_R_TIMEOUT, sensor.systick_ms);
             }
           }
           break;

          default:
           rtScanner_Head_DW.is_SCAN_NORMAL = (uint8_T)IN_NO_ACTIVE_CHILD;
           break;
         }
       }
     }
     break;

    default:
     rtScanner_Head_DW.is_c2_nxtscanner_lib = (uint8_T)IN_NO_ACTIVE_CHILD;
     break;
   }
  }
}

void scanner_head_main_Init(void)
{
  {
    int32_T i;
    rtScanner_Head_DW.is_SCAN_NORMAL = 0U;
    rtScanner_Head_DW.is_active_c2_nxtscanner_lib = 0U;
    rtScanner_Head_DW.is_c2_nxtscanner_lib = 0U;
    head.start_position = 0;
    head.select = 0U;
    head.start_flg = 0U;
    head.line_cnt = 0U;
    head.line_end_flg = 0U;
    head.packet_cnt = 0U;
    head.scan_init_end_flg = 0U;
    head.start_time = 0U;
    for (i = 0; i < 6; i++) {
      polyspace[i] = 0;
    }

    head.pwm_out = 0;
    head.now_position = 0;
    head.encoder_base = 0;
  }
}
```

```
void scanner_head_main(void)
{
  {
    uint8_T sf_previousEvent;
    sf_previousEvent = _sfEvent_;
    _sfEvent_ = CALL_EVENT;
    c2_nxtscanner_lib();
    _sfEvent_ = sf_previousEvent;
  }
}

void Scanner_Head_Init(void)
{
  scanner_head_main_Init();
}

void Scanner_Head(void)
{
  scanner_head_main();
}
```

Paper_Feed.c

```
#include "Paper_Feed.h"

#include "NXT_SCANNER.h"
#include "NXT_SCANNER_private.h"

void task_stop_Start(rtB_task_stop *localB)
{
  localB->PWM = 0;
  localB->Constant2 = FALSE;
}

void task_stop(rtB_task_stop *localB)
{
  localB->PWM = 0;
  localB->Constant2 = FALSE;
}

void Paper_Feed_Init(void)
{
  rtDWork.RotDir = false;
}

void Paper_Feed_Disable(void)
{
  rtDWork.Counter_For_Error_MODE = SUBSYS_DISABLED;
}

void Paper_Feed_Start(void)
{
  rtB.PWM_f = 100;
  rtB.Constant1_c = FALSE;
  rtB.Constant2_m = TRUE;
  rtB.PWM_o = -100;
  rtB.Constant1 = TRUE;
  rtB.Constant2_i = TRUE;
  task_stop_Start(&rtB.task_stop_n);
  rtB.PWM_c = 100;
  rtB.PWM_c = 100;
  rtB.Constant2_l = TRUE;
  rtB.PWM_l = 100;
  rtB.PWM_l = 100;
  rtB.PWM_l = 100;
  rtB.Constant2 = TRUE;
  rtDWork.Memory1_PreviousInput = ((uint8_T)0U);
  rtDWork.UnitDelay_DSTATE = 0;
  rtB.ErrorCode_i = NO_ERROR;
}

void Paper_Feed(void)
{
  int32_T rtb_Switch1;

  {
    boolean_T rtb_ErrDetect;
    uint8_T rtb_Sum1_d;
    int32_T rtb_Switch;
    int32_T rtb_EncoderDiff_o;
    if (ControlMode == INITIALIZE) {
      if (FeedMode == FEED_INIT) {
        rtB.SFunction_o6 = rtB.RevolutionSensor_B;
        if (!rtDWork.ROTOR_FLAG) {
          rtDWork.ROTOR_INIT = rtB.SFunction_o6;
          rtDWork.ROTOR_FLAG = TRUE;
          rtB.PWM_l = 100;
        } else {
          if (rtB.SFunction_o6 - rtDWork.ROTOR_INIT < 1000) {
            rtDWork.ROTOR_FLAG = TRUE;
```

```
          FeedMode = FEED_INIT;
          rtB.PWM_l = 100;
        } else {
          rtDWork.ROTOR_FLAG = FALSE;
          FeedMode = FEED_IDLE;
          rtB.PWM_l = 0;
        }
      }

      rtB.Constant2 = TRUE;
      rtB.PWM = rtB.PWM_l;
      rtb_ErrDetect = rtB.Constant2;
    } else {
      task_stop(&rtB.task_stop_n);
      rtB.PWM = rtB.task_stop_n.PWM;
      rtb_ErrDetect = rtB.task_stop_n.Constant2;
    }
  } else if (ControlMode == PAPERFEED) {
    rtB.PWM_f = 100;
    rtB.Constant1_c = FALSE;
    rtB.Constant2_m = TRUE;
    rtB.PWM = rtB.PWM_f;
    rtDWork.RotDir = rtB.Constant1_c;
    rtb_ErrDetect = rtB.Constant2_m;
  } else if (ControlMode == PAPEREXHAUST) {
    rtB.PWM_o = -100;
    rtB.Constant1 = TRUE;
    rtB.Constant2_i = TRUE;
    rtB.PWM = rtB.PWM_o;
    rtDWork.RotDir = rtB.Constant1;
    rtb_ErrDetect = rtB.Constant2_i;
  } else if (ControlMode == IDLE) {
    task_stop(&rtB.task_stop_n);
    rtB.PWM = rtB.task_stop_n.PWM;
    rtb_ErrDetect = rtB.task_stop_n.Constant2;
  } else if (ControlMode == ERROR) {
    task_stop(&rtB.task_stop_n);
    rtB.PWM = rtB.task_stop_n.PWM;
    rtb_ErrDetect = rtB.task_stop_n.Constant2;
  } else if (FeedMode == FEED_PAPERFEED) {
    rtB.SFunction_o4 = rtB.RevolutionSensor_B;
    rtB.SFunction_o5 = rtDWork.RotDir;
    if (!rtDWork.ROTOR_FLAG) {
      rtDWork.ROTOR_INIT = rtB.SFunction_o4;
      rtDWork.ROTOR_FLAG = TRUE;
      rtB.PWM_c = 100;
      rtB.RotOffset = rtB.SFunction_o5;
    } else {
      rtb_EncoderDiff_o = rtB.SFunction_o4 - rtDWork.ROTOR_INIT;
      if (rtB.SFunction_o5) {
        rtb_Switch1 = rtb_EncoderDiff_o;
      } else {
        rtb_Switch1 = rtb_EncoderDiff_o + 50;
      }

      if (rtB.SFunction_o5) {
        rtb_Switch = 50;
      } else {
        rtb_Switch = 0;
      }

      if (rtb_EncoderDiff_o < rtb_Switch + 102) {
        rtDWork.ROTOR_FLAG = TRUE;
        FeedMode = FEED_PAPERFEED;
        {
          uint32_T iLeft;
          if (rtb_Switch1 <= 0 ) {
            iLeft = 0;
          } else if (rtb_Switch1 >= 150 ) {
            iLeft = 15U;
```

```
          } else {
            iLeft = (uint32_T)( rtb_Switch1 ) / 10;

            {
              uint32_T remainder;
              remainder = (uint32_T)( rtb_Switch1 ) % 10;
              if (( 10 - remainder ) <= remainder ) {
                iLeft++;
              }
            }
          }

          rtB.PWM_c = (rtConstP.LookupTable_YData[iLeft]);
        }

        rtB.RotOffset = rtB.SFunction_o5;
      } else {
        rtDWork.ROTOR_FLAG = FALSE;
        FeedMode = FEED_IDLE;
        rtB.PWM_c = 0;
        rtB.RotOffset = FALSE;
      }
    }

    rtB.Constant2_l = TRUE;
    rtB.PWM = rtB.PWM_c;
    rtDWork.RotDir = rtB.RotOffset;
    rtb_ErrDetect = rtB.Constant2_l;
  } else {
    task_stop(&rtB.task_stop_n);
    rtB.PWM = rtB.task_stop_n.PWM;
    rtb_ErrDetect = rtB.task_stop_n.Constant2;
  }

  if (rtb_ErrDetect) {
    if (rtDWork.Counter_For_Error_MODE == SUBSYS_DISABLED) {
      rtDWork.Memory1_PreviousInput = ((uint8_T)0U);
      rtDWork.UnitDelay_DSTATE = 0;
      rtDWork.Counter_For_Error_MODE = SUBSYS_ENABLED;
    }

    rtb_Sum1_d = (uint8_T)(uint32_T)(rtDWork.Memory1_PreviousInput + ((uint8_T)
      1U));
    if (rtb_Sum1_d > ((uint8_T)10U)) {
      rtb_Sum1_d -= ((uint8_T)10U);
    }

    rtb_ErrDetect = (rtb_Sum1_d == ((uint8_T)10U));
    if (rtb_ErrDetect && (rtPrevZCSigState.Error_Detection_Trig_ZCE !=
                          POS_ZCSIG)) {
      if ((rtDWork.UnitDelay_DSTATE == rtB.RevolutionSensor_B) &&
          (rtB.RevolutionSensor_B != 0)) {
        rtB.ErrorCode_i = ERROR_002;
      }

      rtDWork.UnitDelay_DSTATE = rtB.RevolutionSensor_B;
    }

    rtPrevZCSigState.Error_Detection_Trig_ZCE = rtb_ErrDetect ? POS_ZCSIG :
      ZERO_ZCSIG;
    rtDWork.Memory1_PreviousInput = rtb_Sum1_d;
  } else {
    if (rtDWork.Counter_For_Error_MODE == SUBSYS_ENABLED) {
      rtDWork.Counter_For_Error_MODE = SUBSYS_DISABLED;
    }
  }

  ErrorCode = rtB.ErrorCode_i;
 }
}
```

Reference

[1] Embedded Coder Robot NXT
http://www.mathworks.com/matlabcentral/fileexchange/13399

[2] Philo's Home Page　LEGO Mindstorms NXT
http://www.philohome.com/

[3] NXT GamePad
http://lejos-osek.sourceforge.net/utilities.htm

[4] Excel Interface API for Simulink Data Object
http://www.mathworks.com/matlabcentral/fileexchange/20316