

## Technical Note

### Sigma FSP – Best Programming Practices

Applicable Product: Sigma FSP



Yaskawa Electric America  
2121 Norman Drive South  
Waukegan, IL 60085  
1-800-927-5292

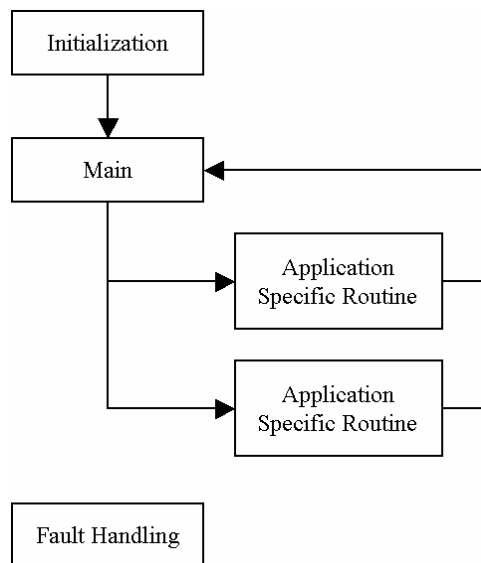
Subject: Technical Note	Product: Sigma FSP	Doc#: TN.MCD.06.060
Title: Sigma FSP – Best Programming Practices		

## Overview

The purpose of this document is to provide the user with a general framework to help ensure success in applying the Sigma FSP. This is a living document that will be edited using the information gained through a variety of successful Sigma FSP applications.

## Program Layout

As with any text-based programming, it is recommended to use a program structure that adheres to the following diagram. This diagram maximizes the robustness of a user program, maximizes code portability, and allows for easy troubleshooting. Additional functionality can also be added or removed to the user program by creating a unique application specific routine and defining an entrance condition in the main loop.



- **Initialization** – any tasks performed at start-up, such as setting application variables, status checks
- **Main** – a continual loop calling application specific routines based upon entrance conditions
- **Application Specific Routine** – code sections performing unique motion tasks, such as homing, indexing, camming, etc.
- **Fault Handling** – code initiated upon faults such as over-travels and E-stops, this section can also contain any interrupt routines that may be included in the user program

Subject: Technical Note	Product: Sigma FSP	Doc#: TN.MCD.06.060
Title: Sigma FSP – Best Programming Practices		

## Label Usage

Labels should be organized in such a way that the user program can be easily understood. FlexWorks does not provide program formatting, so the labels should be assigned according to function within the user program. The table below is organized according to the program structure defined on the previous page. Following this recommendation will make the program easier to follow and allow for code to be easily copied between user programs.

Label Number	Comment	Description
99	Auto Run Label	Beginning of program, parameter Pn2CC
1-9	Initialization	Initialize variables, flags
10-19	Main Program	Call user subroutines
20-29	Application Specific Subroutine A	
30-39	Application Specific Subroutine B	
40-49	Application Specific Subroutine C	
50-59	Application Specific Subroutine D	
60-69	Application Specific Subroutine E	
70-79	Fault Manager and Interrupts	Labels for safety routines and interrupts
80-89	Homing Routine	Home sequence
90-98	Servo Enable Routine	Servo enable sequence

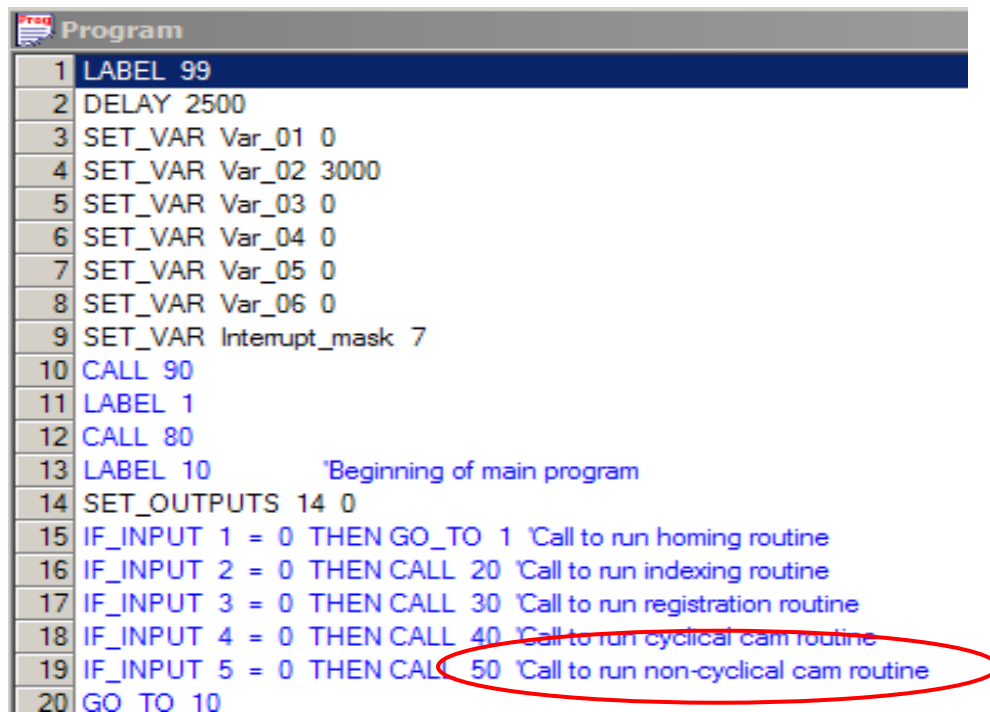
A range has been provided in sections where more than one label may be required in order to perform desired operations such as conditional statements. In addition to the recommendations in the table, below are some additional items to consider:

- Call servo enable and homing routines from the initialization routine on power-up, but also provide an entrance into these routines from the main program loop as needed
- Safety routines should return to the main program loop (which will then call servo enable and homing as necessary)
- Verify that each label has an associated END statement to prevent stack creep
- Consider commenting code using Microsoft Notepad:
  - Locate the .XDR file and open the file in Microsoft Notepad
  - Make comments after a line of code by adding an apostrophe followed by the comment (see below for an example)

```

[Program]
1=LABEL, 99
2=DELAY, 2500
3=SET_VAR, Var_01, 0
4=SET_VAR, Var_02, 3000
5=SET_VAR, Var_03, 0
6=SET_VAR, Var_04, 0
7=SET_VAR, Var_05, 0
8=SET_VAR, Var_06, 0
9=SET_VAR, Interrupt_mask, 7
10=CALL, 90
11=LABEL, 1
12=CALL, 80
13=LABEL, 10      'Beginning of main program
14=SET_OUTPUTS, 14, 0
15=IF_INPUT, 1, =, 0, THEN GO_TO, 1  'Call to run homing routine
16=IF_INPUT, 2, =, 0, THEN CALL, 20  'Call to run indexing routine
17=IF_INPUT, 3, =, 0, THEN CALL, 30  'Call to run registration routine
18=IF_INPUT, 4, =, 0, THEN CALL, 40  'Call to run cyclical cam routine
19=IF_INPUT, 5, =, 0, THEN CALL, 50  'Call to run non-cyclical cam routine
20=GO_TO, 10
    
```

After adding labels to the program in Microsoft Notepad, open the file in FlexWorks



```

Program
1 LABEL 99
2 DELAY 2500
3 SET_VAR Var_01 0
4 SET_VAR Var_02 3000
5 SET_VAR Var_03 0
6 SET_VAR Var_04 0
7 SET_VAR Var_05 0
8 SET_VAR Var_06 0
9 SET_VAR Interrupt_mask 7
10 CALL 90
11 LABEL 1
12 CALL 80
13 LABEL 10      'Beginning of main program
14 SET_OUTPUTS 14 0
15 IF_INPUT 1 = 0 THEN GO_TO 1 'Call to run homing routine
16 IF_INPUT 2 = 0 THEN CALL 20 'Call to run indexing routine
17 IF_INPUT 3 = 0 THEN CALL 30 'Call to run registration routine
18 IF_INPUT 4 = 0 THEN CALL 40 'Call to run cyclical cam routine
19 IF_INPUT 5 = 0 THEN CALL 50 'Call to run non-cyclical cam routine
20 GO TO 10
    
```

Subject: Technical Note	Product: Sigma FSP	Doc#: TN.MCD.06.060
Title: Sigma FSP – Best Programming Practices		

## Variable Usage

The Sigma FSP is limited in the quantity of user variables, so variable efficiency is critical to application success in applications requiring the maximum (or more) available variables in the Sigma FSP. The following table is a suggestion for variable usage.

Variable	Comment	Function
Var_01	Status Word	User defined control bits for program flow
Var_02	Home Offset	Index distance after C-pulse – set as zero point
Var_03	Target Position	Holds target position for motion commands
Var_04		
Var_05		
Var_06		
Var_07		
Var_08		
Var_09		
Var_10	Accumulator	Working register for arithmetic operations

These suggestions provide the user with a flexible pattern that covers a broad range of typical application needs. The status word is a flexible variable that holds user-defined status bits that can be controlled through bit manipulation. Creating a HOMED bit is an example of a user-defined status bit that could be included in this status word and be enabled or disabled in the fault and application specific subroutines. The home offset is a variable found in many applications, and the target position allows for all target positions to be passed through this variable so that programs can be written efficiently by copying common commands. Lastly, the accumulator can be used as a working register for many arithmetic operations in order to minimize extraneous variable usage.

This provides the user with additional variables that can be used to hold critical program data. It is recommended that the user write down the function of each user variable in the program since the variables cannot be commented in the FlexWorks programming environment.

Subject: Technical Note	Product: Sigma FSP	Doc#: TN.MCD.06.060
Title: Sigma FSP – Best Programming Practices		

## Fault Manager

The Sigma FSP comes equipped with a special fault subroutine called the Fault Manager. This subroutine is executed when a fault or alarm occurs and has a higher priority than any user interrupt. Using the Fault Manager provides a robust solution to handling faults on an application. A list of faults and alarms is available in Chapter 7 of the FlexWorks User's Manual (YEA-SIA-FSP-4).

The following code example provides the recommended structure of the Fault Manager subroutine. This structure allows for fault handling subroutines to be easily added or removed, and allows for standard fault handling routines to easily be copied among applications.

```

FAULT_MANAGER
IF Fault_code = [CODE#1] THEN GO_TO [LABEL#1]
IF Fault_code = [CODE#2] THEN GO_TO [LABEL#2]
IF Fault_code = [CODE#3] THEN GO_TO [LABEL#3]
END

LABEL [LABEL#1]
...program code for handling CODE#1...
FAULT_MESSAGE_CLEAR
FAULT_MANAGER_RETURN -1

LABEL [LABEL#2]
...program code for handling CODE#2...
FAULT_MESSAGE_CLEAR
FAULT_MANAGER_RETURN -1

LABEL [LABEL#3]
...program code for handling CODE#3...
FAULT_MESSAGE_CLEAR
FAULT_MANAGER_RETURN -1

```

Using this structure, the fault manager will only execute fault handling subroutines for the selected fault codes. The Fault Manager subroutine will be run for all fault codes, but codes with no subroutine are passed over. After executing the fault handling subroutines, the program will return to the Fault Manager label, and if no other faults are active, the program will return to the program line executed when the fault occurred.

Subject: Technical Note	Product: Sigma FSP	Doc#: TN.MCD.06.060
Title: Sigma FSP – Best Programming Practices		

The table below provides a list of fault codes that are commonly used in the Fault Manager.

<b>Fault Code</b>	<b>Description</b>
151	Positive Over-travel
152	Negative Over-travel
50	Emergency Stop
74	Overload: High Load
75	Overload: Low Load

### **Additional Recommendations**

The following is a list of other recommendations that have been identified during product use to help prevent against errors.

- Add a two-second delay in the program immediately after the auto-start label to allow the amplifier to fully power-up prior to program execution
- Use a conditional IF statement to check latched position status with a timeout check instead of a WAIT for latched position status to prevent the user program from permanently holding if no registration occurs