ADVANCED

Manual for Laboratory
PLC, HMI and SCADA
Version 1.0

Lecturer:  Dr. John McGrory
School of Control Systems and Electrical Engineering,
Dublin Institute of Technology, Room 10, Kevin Street, Dublin 8.
Phone:  +353-(0)1-402-2848
E-Mail:  john.mcgrory@dit.ie
Web Site:  http://eleceng.dit.ie/jmcgrory/
Table of Contents

TABLE OF CONTENTS ................................................................................................................................. 2

CHAPTER 1, STAGES IN DEVELOPING A PLC SYSTEM ................................................................. 4

CHAPTER 2, HOW THE PLC IS WIRED UP TO THE OUTSIDE WORLD ........................................ 5

INSTRUCTION Item 1, Counters INFORMATION Item 2, Data Registers Module 1, CJ Jump INSTRUCTION ................................................................. 5
INSTRUCTION Item 1, Counters INFORMATION Item 2, Data Registers Module 1, CJ Jump INSTRUCTION ................................................................. 5
INSTRUCTION Item 2, Data Registers Module 1, CJ Jump INSTRUCTION ........................................ 7
Module 1, CJ Jump INSTRUCTION ........................................................................................................... 8
Module 1, CJ Jump INSTRUCTION ........................................................................................................... 9
Module 1, CJ Jump INSTRUCTION ........................................................................................................... 10
Module 2, CALL INSTRUCTION .............................................................................................................. 11
Module 2, CALL INSTRUCTION .............................................................................................................. 12
Module 3, FEND INSTRUCTION ............................................................................................................. 13
Module 4, FOR, NEXT Instruction Module 5, Move and Compare Instruction Module 6, ADD INSTRUCTION ............................................................................................................................................... 14
Module 5, Move and Compare Instruction Module 6, ADD Instruction .............................................. 15
Module 5, Move and Compare Instruction Module 6, ADD Instruction .............................................. 16
Module 6, ADD INSTRUCTION ................................................................................................................ 17
Module 6, ADD INSTRUCTION ................................................................................................................ 18
Module 6, ADD INSTRUCTION ................................................................................................................ 19
Module 7, SUB Instruction Module 8, MUL Instruction Module 9, DIV Instruction ................................ 19
Module 7, SUB Instruction Module 8, MUL Instruction Module 9, DIV Instruction ................................ 20
Module 8, MUL Instruction Module 9, DIV Instruction ........................................................................ 21
Module 9, DIV Instruction ....................................................................................................................... 22
Module 10, INC & DEC Instruction Module 11, Analogue-Digital Conversion Module 12, PID INSTRUCTION .................................................................................................................................................. 23
Module 11, Analogue-Digital Conversion Module 12, PID INSTRUCTION ........................................... 24
Module 12, PID INSTRUCTION ............................................................................................................... 25
Module 12, PID INSTRUCTION ............................................................................................................... 26
Module 12, PID INSTRUCTION ............................................................................................................... 27
Module 12, PID INSTRUCTION ............................................................................................................... 28
Module 12, PID INSTRUCTION ............................................................................................................... 29
Chapter 7, Questions to give you experience ....................................................................................... 29
Chapter 7, Questions to give you experience ....................................................................................... 30
Exercise 1, Orange concentrate plant ................................................................................................. 30

CHAPTER 9, HMI, MMI AND OMI .................................................................................................... 31

Step 2: Start a new project ............................................................................................................... 35
Step 3: Getting Started .................................................................................................................... 36
Step 4: More involved HMI, Display Analogue Data ...................................................................... 41
Step 5: Bar Charts .......................................................................................................................... 42
Step 6: Force Outputs ..................................................................................................................... 45

CHAPTER 10, OPC SERVER ........................................................................................................... 49

Step 1: Load up the software ........................................................................................................... 50
Step 2: Add a Channel ..................................................................................................................... 51
Step 3: Add a Device ......................................................................................................................... 52
Step 4: Add a Group ......................................................................................................................... 55
Chapter 1,  **Stages in Developing a PLC System**
All projects will follow the same seven steps so it is worth your while noting them in your head.
1. List the Inputs and Outputs and include a descriptive text of what they are.
2. Construct the ladder logic of the system using the developer software.
3. Convert the ladder logic.
4. Switch the PLC into Stop Mode.
5. Upload the program to the PLC.
6. Switch the PLC into Run Mode.
7. Set the development software to monitor and make sure program is working before leaving it.
Chapter 2, How the PLC is wired up to the outside world

The PLC has particular wiring arrangements. The diagram below highlights how the system is wired up. As you can see the inputs are driven from a 24V supply directly into the contact terminals. This could be in some cases 12V, 110V or 220V on other similar product manufactures equipment.

It is important to isolate the inputs using a fuse or better to avoid the PLC getting damaged. However the Outputs are only ‘Volt Free Contacts’ so a relay is needed to complete the control action and in this case up to 220V can be wired in but usually for safety 24V should be used.
Information Item 1, Counters

FX Series Programmable Controllers

4.10 Counters

Device Mnemonic: C

Purpose: Event driven delays

Alias: Counter(s) C

Available forms: A driven coil sets internal PLC contacts (NO and NC contacts available). Various counter resolutions are possible including:
- General/latched 16bit up counters - see page 4-20
- General/latched 32bit bi-directional counters - see page 4-21
(The availability and use of all these counters is PLC specific - please check availability before use)

Devices numbered in: Decimal, i.e C0 to C9, C10 to C19

Further uses: None

Example device usage:

Available devices:

<table>
<thead>
<tr>
<th>Counter Resolution</th>
<th>FX0(S)</th>
<th>FX0N</th>
<th>FX</th>
<th>FX2(C)</th>
<th>FX2N(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General 16bit up counter</td>
<td>14 (C0 - 13)</td>
<td>16 (C0 - 15)</td>
<td>100 (C0 - 99)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latched 16bit up counter</td>
<td>2 (C14 - 15)</td>
<td>16 (C16 - 31)</td>
<td>100 (C100 - 199)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General 32bit bi-directional counter</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td>20 (C200 - 219)</td>
<td></td>
</tr>
<tr>
<td>Latched 32bit bi-directional counter</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td>15 (C220 - 234)</td>
<td></td>
</tr>
</tbody>
</table>

High speed counters:

- For high speed counters please see page 4-22.

Setting ranges for counters:

- 16bit counters: -32,768 to +32,767
- 32bit counters: -2,147,483,648 to +2,147,483,647
4.12 Data Registers

Device Mnemonic: D
Purpose: A storage device capable of storing numeric data or 16/32 bit patterns
Alias: Data (register/ device/ word)
D (register)
D
Word
Available forms: General use registers - see page 4-34
Battery backed/latched registers - see page 4-35
Special diagnostic registers - see page 4-35
File registers - see page 4-36
RAM file registers - see page 4-36
Externally adjusted registers - see page 4-37
Devices numbered in: Decimal, i.e. D0 to D9, D10 to D19
Further uses: Can be used in the indirect setting of counters and timers
Example device usage: None

Available devices:

<table>
<thead>
<tr>
<th></th>
<th>FX0</th>
<th>FX(N)</th>
<th>FX (CPU Ver. 2.3)</th>
<th>FX(2C)</th>
<th>FX2N(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General use registers</td>
<td>30</td>
<td>128</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(D0 - 29)</td>
<td>(D0 - 127)</td>
<td>(D0 - 199)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latched registers</td>
<td>2</td>
<td>128</td>
<td>312</td>
<td>800</td>
<td>7800</td>
</tr>
<tr>
<td>(D30 - 31)</td>
<td>(D128 - 256)</td>
<td>(D200 - 512)</td>
<td>(D200 - 999)</td>
<td>(D200 - 7999)</td>
<td></td>
</tr>
<tr>
<td>Diagnostic registers</td>
<td>27</td>
<td>39</td>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(D8000 - 8069)</td>
<td>(D8000 - 8255)</td>
<td>(D8000 - 8255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File registers R</td>
<td>N/A</td>
<td>1500</td>
<td>2000</td>
<td>7000</td>
<td></td>
</tr>
<tr>
<td>(D1000 - 2499)</td>
<td>(D1000 - 2999)</td>
<td>(D1000 - 2999)</td>
<td></td>
<td>(D1000 - 7999)</td>
<td></td>
</tr>
<tr>
<td>RAM file registers M</td>
<td>N/A</td>
<td>2000</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>(D8000 - 8069)</td>
<td>(D8000 - 8255)</td>
<td></td>
<td></td>
<td>(D8000 - 7999)</td>
<td></td>
</tr>
<tr>
<td>Adjustable registers F</td>
<td>1</td>
<td>2</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>(D8013)</td>
<td>(D8030 - 8031)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R - These devices are allocated by the user at the expense of available program steps.
On FX2N(C) these devices are a subset of the latched registers.

F - These devices are also included under the count for diagnostic registers.

M - These devices are activated when special auxiliary relay M8074 is turned ON.
No program steps are occupied by RAM file registers.
4.12.1 General Use Registers

Data registers, as the name suggests, store data. The stored data can be interpreted as a numerical value or as a series of bits, being either ON or OFF. A single data register contains 16 bits or one word. However, two consecutive data registers can be used to form a 32 bit device more commonly known as a double word. If the contents of the data register are being considered numerically then the Most Significant Bit (MSB) is used to indicate if the data has a positive or negative bias. As bit devices can only be ON or OFF, 1 or 0 the MSB convention used is, 0 is equal to a positive number and 1 is equal to a negative number.

The diagram above shows both single and double register configurations. In the diagram identified as 1, it should be noted that the 'lower' register D0 no longer has a 'Most Significant Bit'. This is because it is now being considered as part of a 32 bit double word. The MSB will always be found in the higher 16 bits, i.e., in this case D1. When specifying a 32 bit data register within a program instruction, the lower device is always used e.g., if the above example was to be written as a 32 bit instructional operand it would be identified as D0. The second register, D1, would automatically be associated.

Once the data is written to a general data register, it remains unchanged until it is overwritten. When the PLC is turned from RUN to STOP all of the general data registers have their current contents overwritten with a 0 (zero).

Data retention:
- Data can be retained in the general use registers when the PLC is switched from RUN to STOP if special auxiliary relay M8033 is ON.

Data register updates:
- Writing a new data value to a data register will result in the data register being updated with the new data value at the end of the current program scan.
4.12.2 Battery Backed/ Latched Registers

Once data is written to a battery backed register, it remains unchanged until it is overwritten. When the PLC’s status is changed from RUN to STOP, the data in these registers is retained. The range of devices that is battery backed can be changed by adjusting the parameters of the PLC. For details of how to do this please refer to the appropriate programming tools manual.

**Using the FX2-40AW/AP:**

- When using an FX with either the FX2-40AW or the FX2-40AP a proportion of the latched data registers are automatically assigned for communications use by the FX2-40AW/AP module.

  Communication between Master and Slave 100 points M800 to M899
  10 points D490 to D499

  Communication between Slave and Master 100 points M900 to M999
  10 points D500 to D509

4.12.3 Special Diagnostic Registers

Special registers are used to control or monitor various modes or devices inside the PLC. Data written in these registers are set to the default values when the power supply to the PLC is turned ON.

- Note: When the power is turned ON, all registers are first cleared to 0 (zero) and then the default values are automatically written to the appropriate registers by the system software. For example, the watchdog timer data is written to D8000 by the system software. To change the setting, the user must write the required value over what is currently stored in D8000.

Data stored in the special diagnostic registers will remain unchanged when the PLC is switched from STOP mode into RUN.

**Use of diagnostic registers:**

- On no account should unidentified devices be used. If a device is used, it should only be for the purpose identified in this manual. Please see chapter 6 for tables containing data and descriptions of the available devices for each PLC.
Module 1, CJ Jump Instruction

FX Series Programmable Controllers

Applied Instructions 5

5.1.1 CJ (FNC 00)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>CJ FNC 00</td>
<td>Jumps to the identified pointer position</td>
<td>Valid pointers from the range 0 to 63</td>
<td>CJ, CJP 3 steps</td>
</tr>
<tr>
<td>(Conditional Jump)</td>
<td></td>
<td></td>
<td>Jump pointer PPP: 1 step</td>
</tr>
</tbody>
</table>

Operation:
When the CJ instruction is active it forces the program to jump to an identified program marker. While the jump takes place the intervening program steps are skipped. This means they are not processed in any way. The resulting effect is to speed up the programs operational scan time.

Points to note:

a) Many CJ statements can reference a single pointer.
b) Each pointer must have a unique number. Using pointer P63 is equivalent to jumping to the END instruction.
c) Any program area which is skipped, will not update output statuses even if the input devices change. For example, the program opposite shows a situation which loads X1 to drive Y1. Assuming X1 is ON and the CJ instruction is activated the load X1, out Y1 is skipped. Now even if X1 is turned OFF Y1 will remain ON while the CJ instruction forces the program to skip to the pointer P0. The reverse situation will also apply, i.e. if X1 is OFF to begin with and the CJ instruction is driven, Y1 will not be turned ON if X1 is turned ON. Once the CJ instruction is deactivated X1 will drive Y1 in the normal manner. This situation applies to all types of outputs, e.g. SET, RST, OUT, Y, M and S devices.
d) The CJ instruction can jump to any point within the main program body or after an FEND instruction.
e) A CJ instruction can be used to Jump forwards through a program, i.e. towards the END instruction OR it can jump backwards towards step 0. If a backwards jump is used care must be taken not to overrun the watchdog timer setting otherwise the PLC will enter an error situation. For more information on the watchdog timer please see page 5-12.

i) Unconditional jumps can be entered by using special auxiliary coils such as M8000. In this situation while the PLC is in RUN the program will ALWAYS execute the CJ instruction in an unconditional manner.

**IMPORTANT:**

- Timers and counters will freeze their current values if they are skipped by a CJ instruction. For example if Y1 in the previous program (see point c) was replaced by T0 K100 and the CJ instruction was driven, the contents of T0 would not change/increase until the CJ instruction is no longer driven, i.e. the current timer value would freeze.

High speed counters are the only exception to this situation as they are processed independently of the main program.

Using applied instructions:

- Applied instructions are also skipped if they are programmed between the CJ instruction and the destination pointer. However, The PLSY (FNC 57) and PWM (FNC 58) instructions will operate continuously if they were active before the CJ instruction was driven, otherwise they will be processed, i.e. skipped, as standard applied instructions.

Details of using CJ with other program flow instructions

- Further details can be found on pages 7-12 and 7-13 about the combined use of different program flow techniques (such as master control, MC etc).
Module 2, CALL Instruction

5.1.2 CALL (FNC 01)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL FNC 01 (Call subroutine)</td>
<td>Executes the subroutine program starting at the identified pointer position</td>
<td>Valid pointers from the range 0 to 62 Nest levels: 5 including the initial CALL</td>
<td>CALL, CALLP: 3 step Subroutine pointer PPP: 1 steps</td>
</tr>
</tbody>
</table>

**Operation:**

When the CALL instruction is active it forces the program to run the subroutine associated with the called pointer (area identified as subroutine P10). A CALL instruction must be used in conjunction with FEND (FNC 06) and SRET (FNC 02) instructions. The program jumps to the subroutine pointer (located after an FEND instruction) and processes the contents until an SRET instruction is encountered. This forces the program flow back to the line of ladder logic immediately following the original CALL instruction.

**Points to note:**

a) Many CALL statements can reference a single subroutine.

b) Each subroutine must have a unique pointer number. Subroutine pointers can be selected from the range P0 to P62. Subroutine pointers and the pointers used for CJ (FNC 00) instructions are NOT allowed to coincide.

c) Subroutines are not normally processed as they occur after an FEND instruction. When they are called, care should be taken not to overrun the watchdog timer setting. For more information on watchdog timers please see page 5-12.

d) Subroutines can be nested for 5 levels including the initial CALL instruction. As an example the program shown opposite shows a 2 level nest. When X1 is activated the program calls subroutine P11. Within this subroutine is a CALL to a second subroutine P12. When both subroutines P11 and P12 are active simultaneously, they are said to be nested. Once subroutine P12 reaches its SRET instruction it returns the program control to the program step immediately following its original CALL (see ①). P11 then completes its operation, and once its SRET instruction is processed the program returns once again to the step following the CALL P11 statement (see ②).
Module 3, FEND Instruction

5.1.5 FEND (FNC 06)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEND</td>
<td>Used to indicate the end of the main program block</td>
<td>N/A</td>
<td>FEND: 1 step</td>
</tr>
<tr>
<td>FNC 06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(First end)</td>
<td>Note: Can be used with CJ (FNC 00), CALL (FNC 01) and interrupt routines</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:
An FEND instruction indicates the first end of a main program and the start of the program area to be used for subroutines. Under normal operating circumstances the FEND instruction performs a similar action to the END instruction, i.e. output processing, input processing and watchdog timer refresh are all carried out on execution.

Points to note:

a) The FEND instruction is commonly used with CJ-P-FEND, CALL-P-SRET and I-RET program constructions (P refers to program pointer, I refers to interrupt pointer). Both CALL pointers/subroutines and interrupt pointers (!) subroutines are ALWAYS programmed after an FEND instruction, i.e. these program features NEVER appear in the body of a main program.

b) Multiple occurrences of FEND instructions can be used to separate different subroutines (see diagram above).

c) The program flow constructions are NOT allowed to be split by an FEND instruction.

d) FEND can never be used after an END instruction.
Module 4, FOR, NEXT Instruction

FX Series Programmable Controllers

Applied Instructions 5

5.1.7 FOR, NEXT (FNC 08, 09)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands $</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR FNC 08 (Start of a FOR-NEXT loop)</td>
<td>Identifies the start position and the number of repeats for the loop</td>
<td>K, H, KnX, KnY, KnM, KnS, T, C, D, V, Z</td>
<td>FOR: 3 step</td>
</tr>
<tr>
<td>NEXT FNC 09 (End of a FOR-NEXT loop)</td>
<td>Identifies the end position for the loop</td>
<td>N/A</td>
<td>NEXT: 1 step</td>
</tr>
</tbody>
</table>

Note: The FOR-NEXT loop can be nested for 5 levels, i.e. 5 FOR-NEXT loops are programmed within each other.

Operation:
The FOR and NEXT instructions allow the specification of an area of program, i.e. the program enclosed by the instructions, which is to be repeated $S$ number of times.

Points to note:

a) The FOR instruction operates in a 16 bit mode hence, the value of the operand $S$ may be within the range of 1 to 32,767. If a number between the range -32,768 and 0 (zero) is specified it is automatically replaced by the value 1, i.e. the FOR-NEXT loop would execute once.

b) The NEXT instruction has NO operand.

c) The FOR-NEXT instructions must be programmed as a pair e.g. for every FOR instruction there MUST be an associated NEXT instruction. The same applies to the NEXT instructions, there MUST be an associated FOR instruction. The FOR-NEXT instructions must also be programmed in the correct order. This means that programming a loop as a NEXT-FOR (the paired NEXT instruction proceeds the associated FOR instruction) is NOT allowed.

Inserting an FEND instruction between the FOR-NEXT instructions, i.e. FOR-FEND-NEXT, is NOT allowed. This would have the same effect as programming a FOR without a NEXT instruction, followed by the FEND instruction and a loop with a NEXT and no associated FOR instruction.

d) A FOR-NEXT loop operates for its set number of times before the main program is allowed to finish the current program scan.

e) When using FOR-NEXT loops care should be taken not the exceed the PLC’s watchdog timer setting. The use of the WDT instruction and/or increasing the watchdog timer value is recommended.
Nested FOR-NEXT loops:

FOR-NEXT instructions can be nested for 5 levels. This means that 5 FOR-NEXT loops can be sequentially programmed within each other.

In the example a 3 level nest has been programmed. As each new FOR-NEXT nest level is encountered the number of times that loop is repeated is increased by the multiplication of all of the surrounding/previous loops.

For example, loop C operates 4 times. But within this loop there is a nested loop, B. For every completed cycle of loop C, loop B will be completely executed, i.e. it will loop D0Z times. This again applies between loops B and A.

The total number of times that loop A will operate for ONE scan of the program will equal:

1) The number of loop A operations multiplied by
2) The number of loop B operations multiplied by
3) The number of loop C operations

If values were associated to loops A, B and C, e.g. 7, 6 and 4 respectively, the following number of operations would take place in ONE program scan:

Number of loop C operations = 4 times
Number of loop B operations = 24 times (C × B, 4 × 6)
Number of loop A operations = 168 times (C × B × A, 4 × 6 × 7)

Note:

The use of the CJ programming feature, causing the jump to P22 allows the ‘selection’ of which loop will be processed and when, i.e. if X10 was switched ON, loop A would no longer operate.
Module 5, Move and Compare Instruction

5.2.1 CMP (FNC 10)

The data of S1 is compared to the data of S2. The result is indicated by 3 bit devices specified from the header address entered as D. The bit devices indicate:
- S2 is less than S1 - bit device D is ON
- S2 is equal to S1 - bit device D+1 is ON
- S2 is greater than S1 - bit device D+2 is ON

Note: The destination (D) device statuses will be kept even if the CMP instruction is deactivated. Full algebraic comparisons are used, i.e. -10 is smaller than +2 etc.

5.2.2 ZCP (FNC 11)

The operation is the same as the CMP instruction except a single data value (S3) is compared against a data range (S1-S2).
- S3 is less than S1 and S2 - bit device D is ON
- S3 is equal to or between S1 and S2 - bit device D+1 is ON
- S3 is greater than both S1 and S2 - bit device D+2 is ON
5.2.3 MOV (FNC 12)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV FNC 12 (Move)</td>
<td>Moves data from one storage area to a new storage area</td>
<td>K, H, KnX, KnY, KnM, KnS, T, C, D, V, Z</td>
<td>MOV, MOVPE: 5 steps, DMOV, DMVPE: 9 steps</td>
</tr>
</tbody>
</table>

**Operation:**

The contents of the source device (S) is copied to the destination (D) device when the control input is active. If the MOV instruction is not driven, no operation takes place.

**Note:** This instruction has a special programming technique which allows it to mimic the operation of newer applied instructions when used with older programming tools. See page 1-5 for more details.

5.2.4 SMOV (FNC 13)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMOV FNC 13 (Shift move)</td>
<td>Takes elements of an existing 4 digit decimal number and inserts them into a new 4 digit number</td>
<td>m1, m2, n, K, H, KnX, KnY, KnM, KnS, T, C, D, V, Z</td>
<td>SMOV, SMOVPE: 11 steps</td>
</tr>
</tbody>
</table>

**Operation 1:**

This instruction copies a specified number of digits from a 4 digit decimal source (S) and places them at a specified location within a destination (D) number (also a 4 digit decimal). The existing data in the destination is overwritten. The decimal manipulation mode is available to all FX and FX2C units.

**Key:**
m1 - The source position of the 1st digit to be moved
m2 - The number of source digits to be moved
n - The destination position for the first digit

**Note:** The selected destination must NOT be smaller than the quantity of source data.

Digit positions are referenced by number: 1= units, 2= tens, 3= hundreds, 4= thousands.
### 5.2.6 BMOV (FNC 15)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNC 15</td>
<td>(Block move)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Operation:
A quantity of consecutively occurring data elements can be copied to a new destination. The source data is identified as a device head address (S) and a quantity of consecutive data elements (n). This is moved to the destination device (D) for the same number of elements (n).

#### Points to note:

a) If the quantity of source devices (n) exceeds the actual number of available source devices, then only those devices which fall in the available range will be used.

b) If the number of source devices exceeds the available space at the destination location, then only the available destination devices will be written to.

c) The BMOV instruction has a built-in automatic feature to prevent overwriting errors from occurring when the source (S - n) and destination (D - n) data ranges coincide. This is clearly identified in the following diagram:

(Note: The numbered arrows indicate the order in which the BMOV is processed)

<table>
<thead>
<tr>
<th>X0</th>
<th>BMOV</th>
<th>D5</th>
<th>D7</th>
<th>K3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>BMOV</td>
<td>D20</td>
<td>D18</td>
<td>K4</td>
</tr>
</tbody>
</table>

#### Note:
- Using file registers as the destination devices [D] may only be performed on FX Main Processing Units (MPUs) with a CPU version 3.07 or greater or on any FX2C or FX2NC MPU.
Module 6, ADD Instruction

5.3.1 ADD (FNC 20)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>FNC 20 (Addition)</td>
<td>K, H, KnX, KnY, KnM, KnS,</td>
<td>ADD, ADDDP:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T, C, D, V, Z</td>
<td>7 steps</td>
</tr>
<tr>
<td></td>
<td>The value of the two source devices is added</td>
<td>KnY, KnM, KnS,</td>
<td>DADD,</td>
</tr>
<tr>
<td></td>
<td>and the result stored in the destination</td>
<td>T, C, D, V, Z</td>
<td>DADDDP:</td>
</tr>
<tr>
<td></td>
<td>device</td>
<td></td>
<td>13 steps</td>
</tr>
</tbody>
</table>

When using M8023 to add floating point data, only double word (32 bit) data registers (D) or constants (K/H) may be used. See page 4-46 for more details regarding floating point format.

Operation 1: (Applicable to all units)
The data contained within the source devices (S1, S2) is combined and the total is stored at the specified destination device (D).

Points to note:

a) All calculations are algebraically processed, i.e. 5 + (-8) = -3.

b) The same device may be used as a source (S1 or S2) and as the destination (D). If this is the case then the ADD instruction would actually operate continuously. This means on every scan the instruction would add the result of the last scan to the second source device. To prevent this from happening the pulse modifier should be used or an interlock should be programmed.

c) If the result of a calculation is "0" then a special auxiliary flag, M8020 is set ON.

d) If the result of an operation exceeds 32,767 (16 bit limit) or 2,147,483,647 (32 bit limit) the carry flag, M8022 is set ON. If the result of an operation exceeds -32,768 or -2,147,483,648 the borrow flag, M8021 is set ON. When a result exceeds either of the number limits, the appropriate flag is set ON (M8021 or M8022) and a portion of the carry/borrow is stored in the destination device. The mathematical sign of this stored data is reflective of the number limit which has been exceeded, i.e. when -32,768 is exceeded negative numbers are stored in the destination device but if 32,767 was exceeded positive numbers would be stored at D.

e) If the destination location is smaller than the obtained result, then only the portion of the result which directly maps to the destination area will be written, i.e if 25 (decimal) was the result, and it was to be stored at K1Y4 then only Y4 and Y7 would be active. In binary terms this is equivalent to a decimal value of 0 a long way short of the real result of 25I.

Continued over the page...
### Module 7, SUB Instruction

**FX Series Programmable Controllers**

#### 5.3.2 SUB (FNC 21)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB FNC 21 (Subtract)</td>
<td>One source device is subtracted from the other - the result is stored in the destination device</td>
<td>K, H, KnX, KnY, KnM, KnS, T, C, I, D, V, Z</td>
<td>SUB, SUBP: 7 steps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KnY, KnM, KnS, T, I, C, D, V, Z</td>
<td>DSUB, DSUBP: 13 steps</td>
</tr>
</tbody>
</table>

#### Operation 1:
(Applicable to all units)
The data contained within the source device, S2 is subtracted from the contents of source device S1. The result or remainder of this calculation is stored in the destination device D. Note: the Points to note* under the ADD instruction (previous page) can also be similarly applied to the subtract instruction.

#### Operation 2:
(Applicable units: FX2C) This function is equivalent to FNC 121 ESUB. The information regarding 'Operation 2' of the ADD instruction apply similarly to this second operation of the SUB instruction (with the exception of a subtraction being performed instead of an addition). Again, only constants and double data words can be manipulated with only DSUB, DSUBP instruction formats being valid.

#### FX2N Support for floating point operations

*Note: The use of M8023 is not supported in FX2N units. The appropriate dedicated floating point instruction should be used instead. E.g. Instead of DADD with M8023 ON, use FNC 120, DEADD.

*See section 5.11*
Module 8, MUL Instruction

5.3.3 MUL (FNC 22)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL</td>
<td>Multiplying the two source devices together the result is stored in the destination device</td>
<td>S1, S2, D</td>
<td></td>
</tr>
<tr>
<td>FNC 22, FNC 22 (Multiplier)</td>
<td></td>
<td>K, H, KnX, KnY, KnM, KnS, T, C, D, V, Z</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>See page 4-46 for more details regarding floating point format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When using M8023 to subtract</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>floating point data, only double word (32 bit) data registers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(D) or constants (K/H) may be used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MUL, MULP: 7 steps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMUL, DMULP: 13 steps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PULSE-P</th>
<th>16 BIT OPERATION</th>
<th>32 BIT OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation 1: (Applicable to all units)

The contents of the two source devices (S1, S2) are multiplied together and the result is stored at the destination device (D). Note the normal rules of algebra apply.

Points to note:

a) When operating the MUL instruction in 16 bit mode, two 16 bit data sources are multiplied together. They produce a 32 bit result. The device identified as the destination address is the lower of the two devices used to store the 32 bit result. Using the above example with some test data:

5 (D0) x 7 (D2) = 35 - The value 35 is stored in (D4, D5) as a single 32 bit word.

b) When operating the MUL instruction in 32 bit mode, two 32 bit data sources are multiplied together. They produce a 64 bit result. The device identified as the destination address is the lower of the four devices used to store the 64 bit result.

c) If the location of the destination device is smaller than the obtained result, then only the portion of the result which directly maps to the destination area will be written, i.e. if a result of 72 (decimal) is to be stored at K1Y4 then only Y4 would be active. In binary terms this is equivalent to a decimal value of 8, a long way short of the real result of 72!

Viewing 64 bit numbers

- It is currently impossible to monitor the contents of a 64 bit result. However, the result can be monitored in two smaller, 32 bit, blocks, i.e. a 64 bit result is made up of the following parts: (upper 32 bits) x 2^32 + (lower 32 bits).

Operation 2: (Applicable units: FX(2C))

This function is equivalent to FNC 122 EMUL. When ‘floating point mode flag’ M8023 is active, i.e. ON, DMUL and DMULP instructions can be used to perform floating point multiplications. When M8023 is reset, i.e. OFF floating point manipulation will not occur. Constants (K/H) and floating point numbers (stored in double data registers D) can be used in any configuration. The constants (K/H) will automatically be converted to the ‘floating point format’ for the operation. Answers for an operation are stored (completely) in one pair of double (32 bits) data registers and not 2 pairs (64 bits) as used in ‘Operation 1’. The normal rules of algebra apply to floating point multiplication.
Module 9, DIV Instruction

DIV (FNC 23)

5.3.4 DIV (FNC 23)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td>Divides one source value by another the result is stored in the destination device</td>
<td>K, H, KnX, KnY, KnM, KnS, T, C, D, V, Z</td>
<td>DIV/DIVF: 7 steps</td>
</tr>
<tr>
<td>FNC 23 (Division)</td>
<td></td>
<td>KnY, KnM, KnS, T, C, D, (V)</td>
<td>DDIV, DDIVF: 13 steps</td>
</tr>
</tbody>
</table>

See page 4-46 for more details regarding floating point format. When using M8023 to subtract floating point data, only double word (32 bit) data registers (D) or constants (K/H) may be used to perform

PULSE-P 16 BIT OPERATION 32 BIT OPERATION

X0 [S1] [S2] [D1] [D2] [D3] [D4]

Operation: (Applicable to all units)
The primary source (S1) is divided by the secondary source (S2). The result is stored in the destination (D). Note the normal rules of algebra apply.

Points to note:

a) When operating the DIV instruction in 16 bit mode, two 16 bit data sources are divided into each other. They produce two 16 bit results. The device identified as the destination address is the lower of the two devices used to store the two results. This storage device will actually contain a record of the number of whole times S2 will divide into S1 (the quotient).

The second, following destination register contains the remainder left after the last whole division (the remainder). Using the previous example with some test data:

\[
51 (D0) \div 10 (D2) = 5(D4) \equiv (D5)
\]

This result is interpreted as 5 whole divisions with 1 left over \((5 \times 10 + 1 = 51)\).

b) When operating the DIV instruction in 32 bit mode, two 32 bit data sources are divided into each other. They produce two 32 bit results. The device identified as the destination address is the lower of the two devices used to store the quotient and the following two devices are used to store the remainder, i.e. if D30 was selected as the destination of 32 bit division operation then D30, D31 would store the quotient and D32, D33 would store the remainder. If the location of the destination device is smaller than the obtained result, then only the portion of the result which directly maps to the destination area will be written. If bit devices are used as the destination area, no remainder value is calculated.

c) If the value of the source device S2 is 0 (zero) then an operation error is executed and the operation of the DIV instruction is cancelled.

Operation 2: (Applicable units FX(2C)) This function is equivalent to FNC 123 EDIV. The information regarding ‘Operation 2’ of the MUL instruction apply similarly to this second operation of the DIV instruction (with the exception of a division being performed instead of a multiplication). Again, only constants and double data words can be manipulated with only DDIV, DDIVF instruction formats being valid. Answers for an operation are stored (completely) in one pair of double (32 bits) data registers, i.e. answers are not split in to quotient and remainder as in ‘Operation 1’. The normal rules of algebra apply to floating point division.
Module 10, INC & DEC Instruction

5.3.5 INC (FNC 24)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC</td>
<td>The designated device is incremented by 1 on every execution of the instruction</td>
<td>D</td>
<td>INC, INCP: 3 steps</td>
</tr>
<tr>
<td>FNC 24</td>
<td>(Increment)</td>
<td>KnY, KnM, KnS, T, C, D, V, Z</td>
<td>DINC, DINCP: 5 steps</td>
</tr>
</tbody>
</table>

**Operation:**
On every execution of the instruction the device specified as the destination D, has its current value incremented (increased) by a value of 1.
In 16 bit operation, when +32,767 is reached, the next increment will write a value of -32,768 to the destination device.
In 32 bit operation, when +2,147,483,647 is reached the next increment will write a value of -2,147,483,646 to the destination device.
In both cases there is no additional flag to identify this change in the counted value.

5.3.6 DEC (FNC 24)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>The designated device is decremented by 1 on every execution of the instruction</td>
<td>D</td>
<td>DEC, DECP: 3 steps</td>
</tr>
<tr>
<td>FNC 25</td>
<td>(Decrement)</td>
<td>KnY, KnM, KnS, T, C, D, V, Z</td>
<td>DDEC, DDECP: 5 steps</td>
</tr>
</tbody>
</table>

**Operation:**
On every execution of the instruction the device specified as the destination D, has its current value decremented (decreased) by a value of 1.
In 16 bit operation, when -32,768 is reached the next increment will write a value of +32,767 to the destination device.
In 32 bit operation, when -2,147,483,646 is reached the next increment will write a value of +2,147,483,647 to the destination device.
In both cases there is no additional flag to identify this change in the counted value.
Module 11, Analogue-Digital Conversion

1. INTRODUCTION

- The FX2N-4AD analog special function block has four input channels. The input channels receive analog signals and convert them into a digital value. This is called an A/D conversion. The FX2N-4AD has maximum resolution of 12 bits.
- The selection of voltage or current-based inputs/output is by user wiring. Analog ranges of -10 to 10V DC (resolution: 5mV), -20 to 20mA (resolution: 20μA) may be selected.
- Data transfer between the FX2N-4AD and the FX2N main unit is by buffer memory exchange. There are 32 buffer memories (each of 16 bits) in the FX2N-4AD.
- The FX2N-4AD occupies 9 points of I/O on the FX2N expansion bus. The 8 points can be allocated from either inputs or outputs. The FX2N-4AD draws 30mA from the 5V rail of the FX2N main unit or powered extension unit.

1.1 EXTERNAL DIMENSIONS

Weight: Approx. 0.5 kg (0.66 lbs)  Dimensions: mm (inches)

2. TERMINAL LAYOUTS

- The analog input is received through a twisted pair shielded cable. This cable should be wired separately from power lines or any other lines which may induce electrical noise.
- If a voltage ripple occurs during input, or there is an electrical induced noise on the external wiring, connect a smoothing capacitor of 0.1 to 0.47μF, 25V.
- If you are using current input, connect the V+ and I- terminals to each other.
- If there is excessive electrical noise, connect the FG frame ground terminal with the grounded terminal on the FX2N-4AD.
- Connect the ground terminal on the FX2N-4AD unit with the grounded terminal on the main unit. Use class 3 grounding on the main unit, if available.
3

INSTALLATION NOTES AND USAGE

3.1 Environment specification

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental specifications (excluding following)</td>
<td>Same as those for the FX2N main unit</td>
</tr>
<tr>
<td>Dielectric withstand voltage</td>
<td>500VAC, 1min (between all terminals and ground)</td>
</tr>
</tbody>
</table>

3.2 Power supply specification

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog circuits</td>
<td>24V DC ±10%, 55mA (external power supply from main unit)</td>
</tr>
<tr>
<td>Digital circuits</td>
<td>5V DC, 30mA (internal power supply from main unit)</td>
</tr>
</tbody>
</table>

3.3 Performance specification

Analog Inputs

<table>
<thead>
<tr>
<th>Item</th>
<th>Voltage input</th>
<th>Current input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog input range</td>
<td>DC -10V to +10V (input resistance: 200kΩ), Warning: this unit may be damaged by input voltage in excess of ±10V</td>
<td>DC -20mA to +20mA (input resistance: 250kΩ), Warning: this unit may be damaged by input currents in excess of ±32mA</td>
</tr>
<tr>
<td>Digital output</td>
<td>12-bit conversion stored in 16-bit 2's complement form. Maximum value: +2047, Minimum value: -2048</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>5mV (10V default range 1/2000)</td>
<td>20mA (20mA default range 1/1000)</td>
</tr>
<tr>
<td>Over all accuracy</td>
<td>±1% (for the range of -10V to +10V)</td>
<td>±1% (for the range of -20mA to +20mA)</td>
</tr>
<tr>
<td>Conversion speed</td>
<td>15ms/channel (Normal speed), 6ms/channel (High speed)</td>
<td></td>
</tr>
</tbody>
</table>

Analog Inputs continued...

Preset 0 (-10V to +10V)  Preset 1 (+4mA to +20mA)  Preset 2 (-20mA to +20mA)

NOTE: Preset ranges are selected by an appropriate setting in buffer memory of the analog block. Current/Voltage input selection must match the correct input terminal connections.

Miscellaneous

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation</td>
<td>Photocoupler isolation between analog and digital circuits. DC/DC converter isolation of power from FX2N MPU. No isolation between analog channels.</td>
</tr>
<tr>
<td>Number of occupied I/O points</td>
<td>8 points taken from the FX2N expansion bus (can be either inputs or outputs)</td>
</tr>
</tbody>
</table>
### 3.4 Allocation of buffer memories (BFMs)

<table>
<thead>
<tr>
<th>BFM</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td>Channel initialization. Default = H0000.</td>
</tr>
<tr>
<td>#1</td>
<td>Channel 1. Contains the number of samples (1 to 4096) to be used for an averaged result.</td>
</tr>
<tr>
<td>#2</td>
<td>Channel 2. The default setting is 8-normal speed. High speed operation can be selected with a value of 1.</td>
</tr>
<tr>
<td>#3</td>
<td>Channel 3.</td>
</tr>
<tr>
<td>#4</td>
<td>Channel 4.</td>
</tr>
<tr>
<td>#5</td>
<td>Channel 1. These buffer memories contain the averaged input values for the number of samples entered for the channel in buffer memories #1 to #4 respectively.</td>
</tr>
<tr>
<td>#6</td>
<td>Channel 2.</td>
</tr>
<tr>
<td>#7</td>
<td>Channel 3.</td>
</tr>
<tr>
<td>#8</td>
<td>Channel 4.</td>
</tr>
<tr>
<td>#9</td>
<td>Channel 1. These buffer memories contain the present value currently being read by each input channel.</td>
</tr>
<tr>
<td>#10</td>
<td>Channel 2.</td>
</tr>
<tr>
<td>#11</td>
<td>Channel 3.</td>
</tr>
<tr>
<td>#12</td>
<td>Channel 4.</td>
</tr>
<tr>
<td>#13-#14</td>
<td>Reserved.</td>
</tr>
<tr>
<td>#15</td>
<td>Selection of A/D conversion speed (see note). When set to 0, a normal speed is selected at 15ms/ch (default). When set to 1, a high speed is selected at 5ms/ch.</td>
</tr>
</tbody>
</table>

#### BFM #16-#19

<table>
<thead>
<tr>
<th>BFM</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>Reset to Defaults and Preset. Default = 0.</td>
</tr>
<tr>
<td>#21</td>
<td>Offset, Gain Adjust. Default = (0, 0).</td>
</tr>
<tr>
<td>#22</td>
<td>Offset, Gain Adjust. Default = (0, 0).</td>
</tr>
<tr>
<td>#23</td>
<td>Offset Value. Default = 0.</td>
</tr>
<tr>
<td>#24</td>
<td>Offset Value. Default = 0.</td>
</tr>
<tr>
<td>#25-#28</td>
<td>Reserved.</td>
</tr>
<tr>
<td>#29</td>
<td>Error status.</td>
</tr>
<tr>
<td>#30</td>
<td>Identification code = #2610.</td>
</tr>
<tr>
<td>#31</td>
<td>Cannot be used.</td>
</tr>
</tbody>
</table>

#### BFM #32

- **Reserved**
- **b7**
- **b6**
- **b5**
- **b4**
- **b3**
- **b2**
- **b1**
- **b0**

### (1) Channel Selection

Channel initialization is made by a 4 character HEX number H0000 in buffer memory BFM #0. The least significant character controls channel 1 and the 4 character controls channel 4.

Setting each character is as follows:

- **O = 0:** Preset range (-10V to +10V) OR **O = 2:** Preset range (-20mA to +20mA)
- **O = 1:** Preset range (+4mA to +20mA) OR **O = 3:** Channel OFF

Example: H0310

- CH1: Preset range (-10V to +10V)
- CH2: Preset range (+4mA to +20mA)
- CH3: CH4: Channel OFF

### (2) Analog to Digital Conversion Speed Change

By writing 0 or 1 into BFM #16 of the FX-14D, the speed at which A/D conversion is performed can be changed. However, the following points should be noted:

- To maintain a high speed conversion rate, use the FROM/TO commands as seldom as possible.

**NOTES:** When a conversion speed change is made, BFM #1-#4 are set to their default values immediately after the change. This is regardless of the values they held originally. Bear this in mind if a speed change will be made as part of the normal program execution.

### (3) Adjusting Gain and Offset values

#### (a) When buffer memory BFM #20 is activated by setting it to K1, all settings within the analog special function block are reset to their default settings. This is a very quick method to erase any undesired gain and offset adjustments.

#### (b) If [b1, b0] of BFM #21 is set to (1, 0), gain and offset adjustments are prohibited to prevent inadvertent changes by the operator. In order to adjust the gain and offset values, bits [b1, b0] must be set to (0, 1). The default is (0, 1).

#### (c) Gain and offset values of BFM #23 and #24 are sent to non-volatile memory gain and offset registers of the specified input channels. Input channels to be adjusted are specified by the appropriate G=2 (gain-offset) bits of BFM #22.

- **Example:** If bits G1 and G2 are set to 1, input channel 1 will be adjusted when BFM #22 is written to by a TO instruction.

#### (d) Channels can be adjusted individually or together with the same gain and offset values.

#### (e) Gain and offset values in BFM #23 and #24 are in units of mV or µA. Due to the resolution of the unit the actual response will be in steps of 5mV or 2µA.

---

Dr. J. McGrory, DIT Kevin Street. Version 2.0, File: plc_manual_laboratory_advanced_v1 Page26 of 95
### (4) Status Information BFM #29

<table>
<thead>
<tr>
<th>BFM devices of BFM #29</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0: Error</td>
<td>When any of b1 to b4 is ON, A/D conversion of all the channels is stopped</td>
<td>No error</td>
</tr>
<tr>
<td>b1: Offset / gain error</td>
<td>Offset/Gain data in EEPROM is corrupted on adjustment error.</td>
<td>Offset/Gain data normal</td>
</tr>
<tr>
<td>b2: Power source abnormality</td>
<td>24V DC power supply failure</td>
<td>Power supply normal</td>
</tr>
<tr>
<td>b3: Hardware error</td>
<td>A/D converter or other hardware failure</td>
<td>Hardware Normal</td>
</tr>
<tr>
<td>b10: Digital range error</td>
<td>Digital output value is less than -2048 or more than 2047</td>
<td>Digital output value is normal.</td>
</tr>
<tr>
<td>b11: Averaging error</td>
<td>Number of averaging samples is 4097 or more or 0 or less (default of 8 will be used)</td>
<td>Averaging is normal, (between 1 and 4096)</td>
</tr>
<tr>
<td>b12: Offset / gain adjust prohibit</td>
<td>Prohibit (b1, b0) of BFM #21 is set to (1, 0). Permit (b1, b0) of BFM #21 is set to (0, 1).</td>
<td>NOTE: b4 to b7, b9 and b13 to b15 are undefined.</td>
</tr>
</tbody>
</table>

### (5) Identification Code BFM #30

The identification (or ID) code number for a Special Function Block is read using the FROM command.

This number for the FX2n-4AD unit is K2010.

The user's program in the PLC can use this facility in the program to identify the special function block before commencing data transfer from and to the special function block.

**CAUTION**

- Values of BFM #0, #23 and #24 are copied to EEPROM memory of the FX2n-4AD. BFM #21 and BFM #22 are only copied when data is written to the gain/offset command buffer BFM #22. Also, BFM #23 causes writing to the EEPROM memory. The EEPROM has a life of about 10,000 cycles (changes), so do not use programs which frequently change these BFM.
- Because of the time needed to write to the EEPROM memory, a delay of 300 ms is required between instructions that cause a write to the EEPROM.
- Therefore, a delay timer should be used before writing to the EEPROM a second time.

### 4. DEFINING GAIN AND OFFSET

**Gain**

- Gain value
- Analog
- +1000
- Digital

*Gain determines the angle or slope of the calibration line, identified at a digital value of 1000.*

- (a) Small gain
- (b) Zero gain
- (c) Large gain

**Offset**

- Offset value
- Analog
- 0

*Offset is the "Position" of the calibrated line, identified at a digital value of 0.*

- (d) Negative offset
- (e) Zero offset
- (f) Positive offset

*Offset and gain can be set independently or together. Reasonable offset ranges are -5 to 15V or 20mA to 20mA, and gain values 1V to 10V or 4mA to 20mA. Gain and offset can be adjusted by software in the FX2n main unit (see program example 2).*

- Bit devices 1, 2 of the gain/offset BFM #21 should be set to 0, 1 to allow adjustment.
- Once adjustment is complete these bit devices should be set to 1, 0 to prohibit any further changes.
- Channel initialization (BFM #5) should be set to the nearest range, i.e., voltage/current etc.
5. EXAMPLE PROGRAM

5.1. Basic Program

In the following example channels CH1 and CH2 are used as voltage inputs. The FX1n-4AD block is connected at the position of special function block No.0. Averaging is set at 4 and data registers D0 and D1 of the PC receive the averaged digital data.

The I/O code for the special function block at position “0” is read from BFM #30 of that block and stored at D4 in the main unit. This is compared to check the block is a FX1n-4AD, if OK, M1 is turned ON. These two program steps are not strictly needed to perform an analog read. They are, however, a useful check and are recommended as good practice.

The analog input channels (CH1, CH2) are set up by writing H0002 to BFM #0 of the FX1n-4AD.

The number of averaged samples for CH1 and CH2 is set to 4 by writing 4 to BFM #1 and #2 respectively. Do not execute the pulse.

The operational status of the FX1n-4AD is read from BFM #29 and output as bit devices at the FX1n main unit.

If there are no errors in the operation of the FX1n-4AD, then the averaged data (BFM’s) are read.

In the case of this example BFM #5 and #6 are read into the FX1n unit and stored at D0 and D1. These devices contain the averaged data for CH1 and CH2 respectively.

5.2. Using gain and offset in a program

The gain and offset of the FX1n-4AD can be adjusted using push-button switches on the input terminal of the PC. It can also be adjusted using software settings sent from the PC.

Only the gain and offset values in the memory of the FX1n-4AD need to be adjusted. A voltmeter or an ammeter for the analog input is not needed. A program for the PC will be needed however.

The following is an example of changing the offset value on input channel CH1 to 0V and the gain value to 2.5V.

The FX1n-4AD block is in the position of block No.0 (i.e. closest to the FX1n main unit).

Example: Adjusting gain/offset via software settings.

- Adjustment start.
- Adjustment end.
Module 12, PID Instruction

FX Series Programmable Controllers

Applied Instructions 5

5.9.8 PID (FNC 88)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Receives a data input and calculates a corrective action to a specified level based on PID control</td>
</tr>
<tr>
<td>FNC 88</td>
<td></td>
</tr>
<tr>
<td>(PID control loop) register each</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Operands</th>
<th>Program steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>D83</td>
<td></td>
<td>D83</td>
</tr>
<tr>
<td>Note: S1 and S2 use a single data register</td>
<td>Note: S3 uses 25 consecutive data registers</td>
<td>Note: D uses a single data register</td>
</tr>
<tr>
<td>PULSE-P</td>
<td>16 BIT OPERATION</td>
<td>32 BIT OPERATION</td>
</tr>
<tr>
<td>[ ] [ ] [ ] [ ]</td>
<td>[ ] [ ] [ ] [ ]</td>
<td>[ ] [ ] [ ] [ ]</td>
</tr>
<tr>
<td>X10</td>
<td>[S1] [S2] [S3] [D]</td>
<td></td>
</tr>
</tbody>
</table>

**Operation:**

This instruction takes a current value (S2) and compares it to a predefined set value (S1). The difference or error between the two values is then processed through a PID loop to produce a correction factor which also takes into account previous iterations and trends of the calculated error. The PID process calculates a correction factor which is applied to the current output value and stored as a corrected output value in destination device (D). The setup parameters for the PID control loop are stored in 25 consecutive data registers S3+0 through S3+24.

**Points to note:**

a) Every PID application is different. There will be a certain amount of "trial and error" necessary to set the variables at optimal levels.

b) The PID instruction is only available on FX and FX2c Main Processing Units fitted with CPU versions 3.11 or greater.

c) On FX2N MPUs a Pre-tuning feature is available that can quickly provide initial values for the PID process. Refer to page 10-28 for more details.

d) As 25 data register are required for the setup parameters for the PID loop, the head address of this data stack cannot be greater than D975. The contents of this data stack are explained later in this section. Multiple PID instructions can be programmed, however each PID loop must not have conflicting data registers.

e) There are control limits in the PLC intended to help the PID controlled machines operate in a safe manner. If it becomes necessary to reset the Set Point Value (S1) during operation, it is recommended to turn the PID command Off and restore the command after entering the new Set Point Value. This will prevent the safety control limits from stopping the operation of the PID instruction prematurely.

f) The PID instruction has a special set of error codes associated with it. Errors are identified in the normal manner. The error codes associated with the PID loop will be flagged by M8067 with the appropriate error code being stored in D8067. These error devices are not exclusive to the PID instruction so care should be taken to investigate errors properly. Please see chapter 6, ‘Diagnostic Devices’ for more information.

g) A full PID iteration does not have to be performed. By manipulation of the setup parameters P (proportional), I (Integral) or D (Derivative) loops may be accessed individually or in a user defined/selected group. This is detailed later in this section.
**Chapter 7, Questions to give you experience**

**Exercise 1, Orange Concentrate Plant**

You have been asked to develop a system to manufacture orange juice from concentrate. Tank 1 contains water and Tank 2 contains concentrated orange juice. Both tanks have low level probes which should stop the respective pumps when the tank is empty (indicated by the loss of signal of the low level probe). Both pumps are to be automated so they can only work together (at the same time) and never alone or the resultant product would be too strong or weak. Tank 3 contains the diluted orange juice and has a low level probe and a high level probe. When the high level is actuated both pumps feeding Tank 3 should stop. The conveyor should be connected to a stop/start switch arrangement. When a container is present the conveyor should stop and Pump 3 should operate for 10 seconds then stop and allow the conveyor move the container on.

Use an I/O list, sketch the ladder logic and written description.

---

**Diagram**

Tank 1

<table>
<thead>
<tr>
<th>Low Level Probe</th>
</tr>
</thead>
</table>

Tank 2

| Low Level Probe |

Tank 3

| Low Level Probe |
| High Level Probe |

Pump 1

Pump 2

Pump 3

Conveyor

Product

Conveyor Drive Motor
Chapter 9, HMI, MMI and OMI
As you have seen from the laboratory session so far the PLC is a very powerful piece of equipment. The PLC itself however has a downfall. It displays very little information on the operation it’s controlling. Without the monitoring software running on a PC or Laptop we would have to depend on the Input and Output LED’s which tell us only if the signal is on or off (analogue values are non existent) but nothing specific about the process itself.

One way of overcoming this is by using the operator panel usually called any of the three titles below.
HMI Human Machine Interface
MMI Man Machine Interface
OMI Operator Machine Interface

A digital image of the E200 HMI as used in the Laboratory is shown below.

Don’t worry about all the different names as they all in essence mean the same thing. The HMI is a component to assist the processing automation system that allows the operator view the process and interact with it. Some of the standard features of the HMI is as follows;

- Viewing and changing of I/O
- Displaying of analogue values
- Alarms
- Bar Graphs
- Information text
- Full graphical screens (similar to monitors) are being used in industry.
To design a good system it's always a good idea to construct a storyboard from which the system will be developed.

It is important to note at this point that the HMI and the PLC are both connected to the one port at the back of the computer. It is therefore impossible to have the PLC in monitoring mode and upload the E-Designer files at the same time. You will see an error appear stating that the port is being used by another package.
It would be advisable to set up a simple ladder logic code running on the PLC before running the E-Designer Software. In this case X007 the green button starts the conveyor Y001. X006 the red button stops the conveyor. Of course the universal starting switch will keep the conveyor running for us. X004 will be used for the Counter C1. Every time the X004 button is pressed it will increment the C1 counter by one. If you wish to put in a reset for the counter that is up to you but not needed for this example.
Step 1: How to load the E-Designer Programme
From the main desktop screen press the “Start” button using the mouse pointer and progress up to “Program” then across to “E-Designer” and finally to the “E-Designer” icon.

From there the following screen should be loaded on to the computer screen
Step 2: Start a new project
From the “file” menu on the top bar choose “New” and click on the text.

The following screen will be displayed and you will be able to choose the model of
the HMI unit installed in the Laboratory.
Remember the Terminal is a E200 6.0x and the Controller system is a FX0(S)CPU.
When the above is completed you will then be shown the screen below:

**Step 3: Getting Started**
Instead of jumping in and programming loads of screens let’s take a little time to explore the functions available to us using the HMI.
If you double click on the “Main” block in the middle of the screen the following screen will appear.
The screen displays an emulation of what the user will see when their code is uploaded to the HMI.

Start by clicking the large white area and type in Main Conveyor. If the program was now uploaded the text Main Conveyor would be displayed. But let’s add a little to system before we do that. Click on the Main Conveyor and move the cursor to the end of the “r” and press return. Type in something like “Conveyor” followed by a space and using the small O/I box in the Icon selection set as shown to the right and the following screen appears.
Enter in something like Y1 (Which we know is the conveyor running signal, on the PLC’s provided) and when it is at off we want the Stopped word to appear and when it is on we want the Running words to appear. Then press Apply and OK.

You can then see the software would have placed in a piece of text as shown below.
Transfer the code to the HMI unit by choosing “Transfer” from the top menu and “Project” from the sub menu.

The following screen will appear and allow you to send your programme up to the HMI.
When “Send” is pressed the you will be asked to confirm if you want to upload and when yes is pressed the code is then transferred across to the HMI.

At the same time you will see the HMI screen change indicating that the code is changing.
When this is completed you will see.

Main Conveyor
Conveyor Off

If you now start the conveyor by pressing the green button the following text is displayed:

Main Conveyor
Conveyor On

This simple program shows us two important things about HMI’s.
1. We can add descriptive text about processes for the operator to see.
2. We can have a dynamic changing of text linked to a changing process.
Step 4: More involved HMI, Display Analogue Data

It is not only possible to have a descriptive text and dynamic changing of text in two states, but we can also display data contained within registers such as the Timer or Counter. If we return to the Main screen on the E-Developer and continue to a new line we can add a little more text. Type in Counter and after that press the 0.3 Icon from the right menu and the following will be displayed.

Type in C1 (at the beginning of this section I asked you to upload a simple ladder logic program to the PLC, C1 was the counter, a number between 1 and 20 linked to X004) and leave the rest the same as shown.
Transfer the code to the HMI unit by choosing “Transfer” from the top menu and “Project” from the sub menu.

Now your system will operate as before and you will see.

- Main Conveyor
- Conveyor Off
- Counter 0

As you start the system you will then be able to have the conveyor change from Off to On using X007 and X006 and increment the counter by pressing X004.

**Step 5: Bar Charts**

If you wish to place a bar chart it can also be completed using the HMI. This would be used where a number is just not enough. Consider the following example. A tank of liquid ranges between 0 and 10 meters in height. By just stating that the height is 2 to an unfamiliar operator would mean nothing but when the graph shows 2 out of 10 it would mean more.

Return to the E-Designer Main screen and at the end of the counter number information press return. Then click on the “--#” icon.
When the dialogue box appears put C1 in the Analogue Signal section and click on Calc. to fill in the upper and lower values for this case choose lower 0 and upper 20 as that is what we have put in the PLC ladder logic. Of course this could be changed showing a scale if that is necessary.
You will see a “#-------------------------” appears on the HMI. As shown below.

Transfer the code to the HMI unit by choosing “Transfer” from the top menu and “Project” from the sub menu.

Now your system will operate as before and you will see.
  Main Conveyor
  Conveyor Off
  Counter 10
  XXXXXXXXXXXXXXXXXX

As you start the system you will then be able to have the conveyor change from Off to On using X007 and X006 and increment the counter by pressing X004. As you get closer to the 10 the more panels will be darkened.
As a way of reminding you, this simple program shows us three important things about HMI’s.

1. We can add descriptive text about processes for the operator to see.
2. We can have a dynamic changing of text linked to a changing process.
3. We can have Graphical features.

**Step 6: Force Outputs**

The particular screen used in the laboratory has five Text spaces, Five LEDs and Five Buttons. If we want to edit any of these Items all we have to do is double click on the element needed. If it is the Text then you can fill in what you want the operator to see. If you wish to use the LEDs for something then you can assign them.

The five buttons below the screen can be used to force Outputs.

If you double click on any of the buttons a dialogue box appears asking you do you want this to be a local or global key. A Global Key would be something that will appear on every screen designed and using a Local Key would mean it is only for this screen. For our case choose Local Key.

![Diagram of HMI screen with buttons and text spaces.](image-url)
The screen dialogue box can be filled in as shown below.

**Button 1 is Unused**
**Button 2 is The same as pressing X7**
**Button 1 is The same as pressing X4**
**Button 1 is The same as Forcing On Y1**
**Button 1 is Unused**

**IMPORTANT:** I cannot stress enough the danger of using **Button 4 to Force On/Off the Y1 coil to drive the conveyor as the system has now no protection in place.**

Transfer the code to the HMI unit by choosing “Transfer” from the top menu and “Project” from the sub menu.

Now your system will operate as before and you will see.

- Main Conveyor
- Conveyor Off
- Counter 10
- XXXXXXXXXX

©Dr. J.McGrory, DIT Kevin Street. Version 2.0, File: plc_manual_laboratory_advanced_v1
Reminding you again, this simple program shows us four important things about HMIs.

1. We can add descriptive text about processes for the operator to see.
2. We can have a dynamic changing of text linked to a changing process.
3. We can have Graphical features
4. We have a way of providing output control.

**Step 7: Now connecting this together, Storyboard**

As you have experienced the above features of a HMI allow for interaction of the operator and the previous inaccessible PLC. But to use this tool effectively some planning is needed.

I would suggest making a storyboard of how you feel the system should be designed and then implement and test it. In this way the operator can drill through relevant screens, making the best of the limited space (screen size, buttons and LED’s) on the HMI.

Firstly identify the locations on the HMI you wish to use. For this illustration I assume the following:

---

**Screen Title**

![Main Screen](image)

<table>
<thead>
<tr>
<th>No.</th>
<th>Text</th>
<th>LED</th>
<th>Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

©Dr. J.McGrory, DIT Kevin Street. Version 2.0, File: plc_manual_laboratory_advanced_v1
The detail of the screens can now be written in typed text as shown in the sample below. You fill in the detail you wish to enter into the HMI. Using this method it is possible to develop a system that flows, allowing the operator access the information in a meaning full way (a fully documented way as well) and avoid any possible pitfalls or delays when coding. The testing of this is following the story board and seeing if it works.

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Screen Title:</strong></td>
<td>Main Screen</td>
</tr>
<tr>
<td><strong>Main Screen:</strong></td>
<td>Conveyor System</td>
</tr>
<tr>
<td><strong>No.1</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
<td>Return to Main Menu</td>
</tr>
<tr>
<td><strong>LED:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Button:</strong></td>
<td>Link to Main Screen Block 1</td>
</tr>
<tr>
<td><strong>No.2</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
<td>Alarms</td>
</tr>
<tr>
<td><strong>LED:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Button:</strong></td>
<td>Link to Alarms Block 2</td>
</tr>
<tr>
<td><strong>No.3</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
<td>Trends</td>
</tr>
<tr>
<td><strong>LED:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Button:</strong></td>
<td>Links to Trends Block 3</td>
</tr>
<tr>
<td><strong>No.4</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>LED:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Button:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>No.5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Text:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>LED:</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Button:</strong></td>
<td>N/A</td>
</tr>
</tbody>
</table>
When each screen is developed it can be arranged using the Block Manager which allows the flow from one block to another using arrows. The following screen shows what a fully implemented system would look like before it is uploaded to the HMI.
Chapter 10, OPC Server
For the data from the PLC to be accessed by the SCADA system it is necessary to employ the features of an OPC Server. OPC stands for OLE for Process control and OLE stands for Object Linking and Embedding. In simple terms the address and data table in the PLC is copied.
How to set up the OPC server
Step 1: Load up the software
In a similar fashion to the MELSOFT and the E-Designer we also load up the OPC server by clicking on Start – Programs – Kepware and then on the Kepware serverEX text. The screen shown below will appear.

You can see that every event is date and time stamped so the operator is kept full aware of what is happening at all times.
**Step 2: Add a Channel**
The channel refers to the communication link from the PC to the PLC.

You will need to enter in all the details about the serial port, baud rate and so on.
The channel name will appear in the dialogue box on the left of the screen showing that it is available.

**Step 3: Add a Device**

Then you will be asked to confirm the type of device running on the Channel (in our case it is a Mitsubishi FX, other units available by using the scroll down menu).
You will be asked to call it a name so you could identify it over the network. For example this could be PLC_MMC1_Builidng_1.

As the FX PLC we use has a number of different models FX0S, FX1N etc. you need to confirm the exact Device model.
The timeouts and number of fails in the communications should also be entered so the SCADA system can be alerted of any error when they happen.

If finishes up with a device summary and finish ends this part.
Step 4:  **Add a Group**

You can see as we perform changes to the program it lists the date and time of each event with a brief summary of it.

Now if we choose a Tag Group to keep respective tags together. An example of this would be Air Handling Unit 1 controls should be kept together and so should Air Handling Unit 2 and Air Handling Unit 3.
For us we will enter a group called K044 which is of course the laboratory you are in at the moment.

**Step 5: Add a Tag**

The tag group is similar to a folder but we need now to select tags on the PLC. We type in a Name we wish to use, the PLC address and a brief description of what it does. Remember when we uploaded a program to the PLC and read it back from the PLC it lost all its statements and notes. All that came back was the Address and the ladder logic nothing else. The same happens here. So this tag entry is where we fill in this detail again.
Once entered the following line appears on the OPC server giving the Tag Name, Address, Data type, DDE Scan cycle, Scaling if any and lastly the description.

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Address</th>
<th>Data Type</th>
<th>DDE Scan</th>
<th>Scaling</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button29</td>
<td>20000</td>
<td>Boolean</td>
<td>100</td>
<td>None</td>
<td>Switch NO for starting conveyor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>User Name</th>
<th>Source</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/15/02</td>
<td>2:51:12 PM</td>
<td>Default User</td>
<td>KEPserveEx</td>
<td>KEPware OPC/DDE Server V4.0</td>
</tr>
<tr>
<td>10/15/02</td>
<td>2:56:04 PM</td>
<td>Default User</td>
<td>KEPserveEx</td>
<td>Mitsubishi PX device driver loaded successfully</td>
</tr>
<tr>
<td>10/15/02</td>
<td>2:56:04 PM</td>
<td>Default User</td>
<td>KEPserveEx</td>
<td>Starting Mitsubishi driver driver</td>
</tr>
<tr>
<td>10/15/02</td>
<td>2:56:04 PM</td>
<td>Default User</td>
<td>Mitsubishi PX</td>
<td>Mitsubishi PX Device Driver V4.00.0.27</td>
</tr>
</tbody>
</table>
Step 6: Run OPC Client

Now the hard work is done and we can start the OPC client running by choosing the Hammer from the Icon menu.

A new screen appears which shows the KEPware Server on the left and all the channel and device and group items. Click on the DeviceK044 and you will then see the I/O entered into it.
You can now see the data stored at the address which in this case is a 0 or a 1.

**Step 7: Genesis**

This information can now be accessed through the SCADA package Genesis using the OPC Universal Tag and Browser.
Chapter 11, SCADA

The lecture notes need to be viewed in relation to the following.

**What does a SCADA package do?**
Most SCADA packages have the following:
- Graphical representation of the process to be controlled
- Trend data against itself or time or another data source
- Alarm if a condition has been achieved.
- Data logging
- Historical information

**Step1: Getting Started**
Click on “Start”, “Programs”, “ICONICS Genesis32” and “Graphwor32”
Step2: Graphical representation
The following screen appears. This screen is the configuration screen for the graphical side to this package.

Let’s start with a simple screen to let the as shown below. It looks complex but its only make up of a number of components connected together.
In the sketch below you can see some of the components used to construct the screen. The elbow is drawn once and copied and rotated. The same is for the pipe and switches and valve. In fact these objects can be imported from the symbol library in the package and if you develop a great drawing you could save it to the library as well. The graphical display is only limited by your imagination.
Step3: Dynamics
You can use the dynamics feature of the SCADA package to make objects move, flash change color, rotate and change size etc. If you take the sample of the mixer shown in the sample “batch” it is simply made up of four separate images which are displayed in an animated sequence.
Toggle between the two displays
Any image or component in the process can be made dynamic. By choosing the “Dynamics” menu from the top bar and choosing “Action” you have a choice of different aspects that can be used.
Let’s take a simple example. If you draw the following components and select all you will be automatically shown the following icons

![Diagram](image)

Selector, Analogue and Animator

By clicking on each one you will be given a number of screens which can be used to put dynamics on to the screen.
**Step 4: Standard Gauges**

Within the Genesis SCADA package there are standard gauges that you can use. These gauges are features that only need the I/O data tag and give you various options on how the gauge will look. Because they are standard modules the code to display them was only written once and each instance of the gauge is a clone of the original thus saving space compared to a personalized gauge designed by the programmer.
**Step 5: Check Boxes**

Instead of having a button you may wish to use the check box function. It could be printing out reports or ticking that an alarm has been accepted.
**Step 6: Graphical representation**

Graphical Features

The highlighted section below is used to align selected items.

Features to help line up selected items

If you wish to change the order in which items appear you can use the highlighted section below.

Select the order in which items appear

Front, Back, Layered

If you wish to rotate items you can using the highlighted section below.

Rotate
**Step7: Digital Display**

By choosing the icon as shown below you can place a Digital display on the screen. The Property Inspector on the right asks for the colors, fonts and title of the digital display. The Property Inspector on the left is involved with the OPC tag (i.e. its reference to the OPC server data). Note both screens cannot be seen at the same time, although shown below but are accessed via the toggle taps at the top of the box.
**Step 8: Date and Time display**

If the designer wishes to display the date and time on a graphical screen all they need to do is press the clock icon as shown and the dialogue boxes appear allowing most variation of the date and time to be displayed.

---

**Digital Display**

**Toggle between the two displays**
**Step 9: Buttons**

Buttons in a good SCADA system can be used for many things. In this package, the list of what can be done is displayed when you choose the down scroll arrow on the action.

As you choose a certain aspect, the dialogue box changes to allow:

- **Toggle between the two displays**
- **Digital Display**
**Step10: Symbol Library**

The Genesis SCADA package has a symbol library where you can store standard images of pumps, pipes etc and use them whenever you want. You can also use the existing images for your own screens.
Step 11: Trending

Trending Chart

Toggle between the ten displays

Choice of Chart
Step12: Multi function
On occasion it is necessary to have a number of different things happen an element in the SCADA package. In the example given below we started with a rectangle, then added a size change, then flash on/off and finally pick. All of these were added to the rectangle and could be added to almost any item developed on the SCADA system.

- Started as a Rectangle
- Then Size changes was added
- Then Flash on/off was added
- Then Pick was added
**Step 13: Arithmetic**

If your are able to get the height in a tank and wish to display the volume, it is possible by using the arithmetic feature.

![Diagram of a tank with height and volume measurements](Image)

Volume would be height * area

Expression: 

\[ V = \pi r^2 h \]
Joining the screens together

As explained in the buttons section above it is possible to link to another screen by pressing on a button and loading up another screen. This is a very powerful tool as it allows the system to be much more than one screen to having many screens. Remember all the Genesis files have a .gdf extension.

The screen displayed below is VBATanks.gdf. The Bean Factory button on the top right is linked to the VBABeanRoaster.gdf which is shown on the next page.
Another very good example of using the buttons or other diagrams is the Notebook1.gdf example. The folder tabs are used to load up the screens from No1, 2, 3, 4, 5 and also back to No.1 if available.
Good Working Practice

Consistency
It would be a good idea to set up a template at the beginning of the systems development so consistency of the end product is evident. On the example screen below you can see a line and under the line you have the date, the DIT logo and buttons to take you to the common screens. Windows applications have been consistent from product to product and once you have been trained or familiar with one you can then use all of them.
Storyboard

Another good design tool when using SCADA systems is the use of storyboards showing where one screen would be linking on to another. Using the folder example you can see how it works. It identifies what is going to be displayed on each screen and how it links from one to the other.
Exercise Laboratory 1

Aim: Construct a graphical user interface as shown above.

Procedure:

1. Start the Genesis GraphWorks module. Move the mouse arrow to the windows start button and press with the left button. Go into programs, Iconics Genesis 32, Graphworks and finally Graphworks32.
2. Press OK on the Licence and note the time on when the package expires.
3. Set the background to white.
4. Using the symbol library given, start constructing the screen as shown above. Remember to save the screen as you go along.
5. Make sure you use the zoom-in and zoom-out command to ensure that all the images are joined together.
6. The button, switches and level indicators and be put into the mimic using the bottom icon bar of the Graph works package.
7. The text can also be put on to the mimic using the icon bar provided.
8. When the graphics are in place and completed you may begin to add animation. Use the top tool menu and choose Dynamics and then Actions.
9. Ensure that the Dynamic actions are linked to the OPC Server as demonstrated by the instructor.
10. Use the tools, Runtime menu to start the system mimic and use the configure to return to the GraphWorks editor.
Exercise Laboratory 2

Aim: Demonstrate the ability to convert from one graphical user interface to another as shown above and introduce the standard gauges.

Procedure:

1. Start the Genesis GraphWorks module. Move the mouse arrow to the windows start button and press with the left button. Go into programs, Iconics Genesis 32, Graphworks and finally Graphworks32.
3. Change the background to white.
4. Load up the batch file as before.
5. Add in a button and call it Batch 2.
6. Now make changes to the file and save the file as Batch 1.
7. Make more changes to the Batch file and call it Batch 2.
8. Enter Batch 1 again and ensure the button when clicked loads up batch 2.
9. Save Batch 1 again.
10. Enter Batch 2 again and ensure the button when clicked loads up batch 1.
11. Runtime should interchange between the files.
12. Check out the features of GENESIS like Dials and gauges and switches.
Exercise Laboratory 3

Aim: Produce a dynamic animated image.

Toggle between the two displays

Slide 1  Slide 2  Slide 3  Slide 4
Exercise Laboratory 4

Aim: Calculate the volume in a tank knowing the height.
Chapter 12, A Generic View of Integration Engineering.

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities. Regardless of the entity that is to be engineered, the following questions must be asked and answered:

(1) What is the problem to be solved?
(2) What are the characteristics of the entity that is used to solve the problem?
(3) How will the entity (and the solution) be realised?
(4) How will the entity be constructed?
(5) What approach will be used to uncover errors that were made in the design and construction of the entity?
(6) How will the entity be supported over the long term, when corrections, adaptations and enhancements are requested by users of the entity?
Exercise 1, Packaging Plant
You have been asked to develop a system to correlate product boxes. The boxes contain special PCB’s in a similar package to a Maths Coprocessor would use. When the boxes come off the production line they are in single units and as shown below. They are required to be delivered in packs of six (6) as this is the only way the company wished to transport them.

Develop a system that can correlates the boxes into groups of six ready for loading into the flow packing machine.

Use good working practice and document all your decisions.
Exercise 2, Screws Packing Plant
You have been asked to develop a system to pack approximately 1000 screws in a circular cup.
Use good working practice and document all your decisions.
Exercise 3, Correlation
You have been asked to develop a system to pack software into boxes. The box must be filled with two manuals (each manual is sealed in cling film) and either a CD or floppy disk. Sealed and labelled.

Use good working practice and document all your decisions.
Exercise 4, PBC Cutting
You have been asked to develop a system to cut PBC from waste material as shown in the circuit below. The 10 Red sections are the only locations that need to be cut.
Use good working practice and document all your decisions.
Exercise 5, SIMM Testing
You have been asked to develop a system test the SIMM Memory Module shown below prior to it being packed and sent to the client. The circuit needs eight tests completed before it can be released. If it fails any of the eight tests then the unit should be rejected.

Use good working practice and document all your decisions.
Chapter 13, Laboratory Conveyor Rig

The Input/Output List for the Laboratory Conveyor Rig is given below:

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0</td>
<td>Box Detect Sensor</td>
</tr>
<tr>
<td>X1</td>
<td>Long Box Detect Sensor</td>
</tr>
<tr>
<td>X2</td>
<td>Box at Kicker Sensor</td>
</tr>
<tr>
<td>X3</td>
<td>Option Switch at bottom of Operator Control Station</td>
</tr>
<tr>
<td>X4</td>
<td>Black Push Button on Operator Control Station</td>
</tr>
<tr>
<td>X5</td>
<td>Option Switch at top of Operator Control Station</td>
</tr>
<tr>
<td>X6</td>
<td>Red Push Button on Operator Control Station (Normally Closed)</td>
</tr>
<tr>
<td>X7</td>
<td>Green Button on Operator Control Station (Normally Closed)</td>
</tr>
<tr>
<td>Y0</td>
<td>Kicker</td>
</tr>
<tr>
<td>Y1</td>
<td>Conveyor Belt</td>
</tr>
<tr>
<td>Y2</td>
<td>Blue Lamp</td>
</tr>
<tr>
<td>Y3</td>
<td>Red Lamp</td>
</tr>
<tr>
<td>Y4</td>
<td>Yellow Lamp</td>
</tr>
<tr>
<td>Y5</td>
<td>Green Lamp</td>
</tr>
</tbody>
</table>

![Image of conveyor rig with labeled components]
Input Terminals

Input LED Indicators

The Switch for Run/Stop

Output LED Indicators

Output Terminals

Power, Run/Stop and CPU LEDs