Daily Progress (Siemens R&D Internship)

Note:

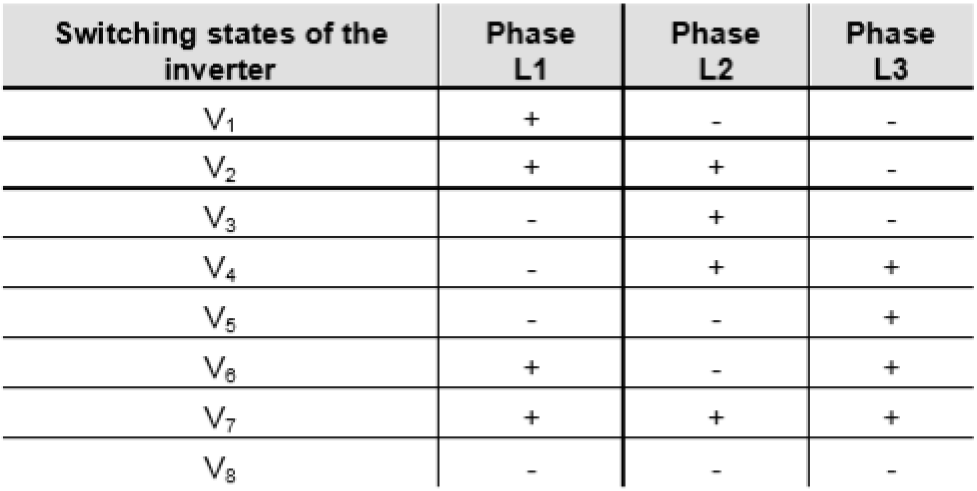
i) This file contains Week 2 (14-16/05, 24 & 25/05) and Week 3(28/05 to 01/06) progress as I went to home and was absent on 17/05, 18/05, 22/05 and 23/05.

ii) This internship has two integral parts: Theory Learning and Practical Application (TIA Portal) This document doesn't segregate the two parts but the final report of two months will track progress in terms of those parts.

iii) This document doesn't separate day-wise learning like the last document but instead follows a concept wise approach. For example, Drives' working and PLC Theory was done over several days so it can't be segregated thus there's no division as such.

Week 2:

- **Working of PWM**: If we consider all three phases, there is a total of $2^3 = 8$ switching states in the inverter, and the effect of these states in the motor can be defined by voltage phasors.



Switching states of the inverter	Phase L1	Phase L2	Phase L3
V_1	+	-	-
V_2	+	+	-
V_3	-	+	-
V_4	-	+	+
V_5	-	-	+
V_6	+	-	+
V_7	+	+	+
V_8	-	-	-

- If, for example, phase L1 is connected to the positive DC link voltage, and phases L2 and L3 to the negative voltage so as to produce switching state V1, the resultant voltage phasor points in the direction of motor phase L1 and is designated phase I. The length of this phasor is determined by the DC link voltage.

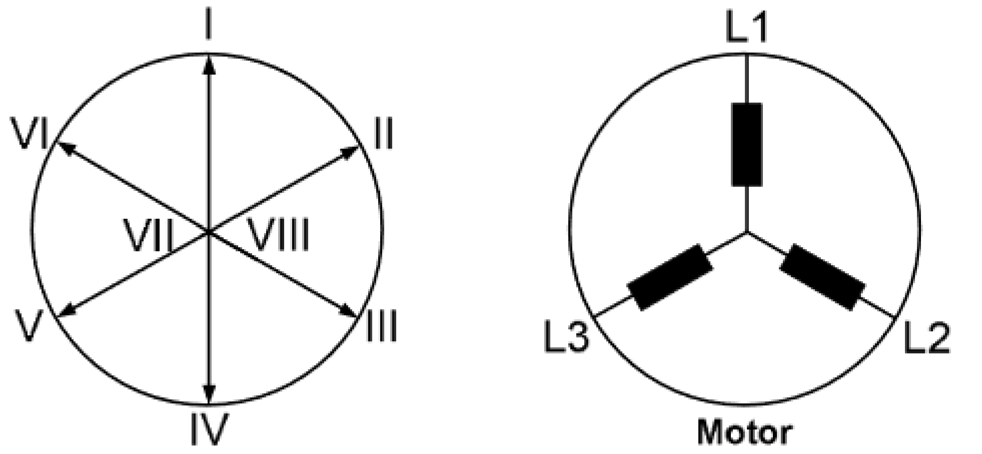
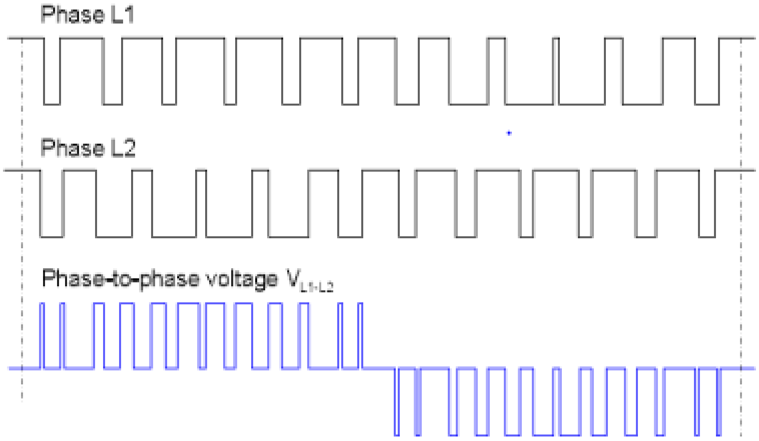


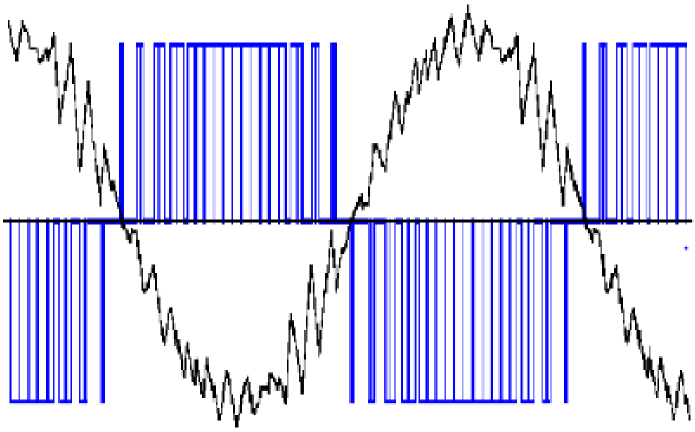
Fig: Representation of resultant motor voltages as phasor

- If the switching state changes from V1 to V2, then the voltage phasor rotates clockwise by an angle of 60°el. due to the change in potential at terminal L2. The length of the phasor remains unchanged. In the same way, the relevant voltage phasors are produced by switching combinations V3 to V6. Switching combinations V7 and V8produce the same potential at all motor terminals. These two combinations therefore produce voltage phasors of "zero" length (zero voltage phasor).

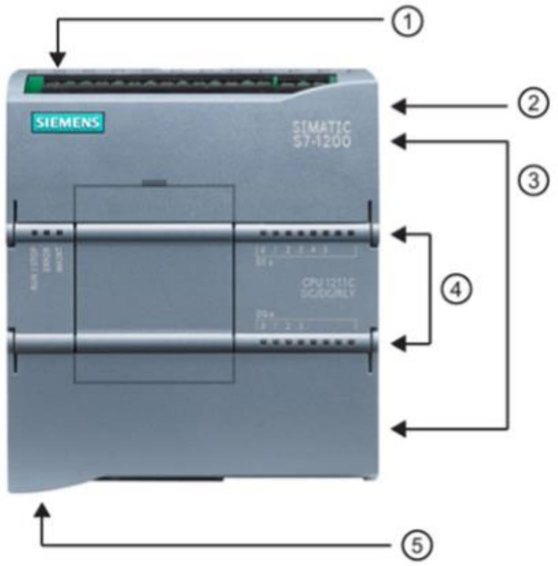
- Voltage and frequency must be specified in a suitable way for a certain operating state of the motor, characterized by speed and torque. Ideally, this corresponds to control of the voltage vector V(wt) on a circular path with the speed of rotation w = 2*p *f and adjusted absolute value. This is achieved through modulation of the actual settable voltage space vectors (pulse-width modulation). In this way, the momentary value V(wt) is formed by pulses of the adjacent, actual settable voltage space vectors and the voltage zero. The solid angle is set directly by varying the ratio of the ON durations (pulse-width) of adjacent voltage vectors, the desired absolute value by varying the ON duration of the zero voltage vector. This method of generating gating signals is called space vector modulation SVM. It is used in all units described in this engineering manual. Space vector modulation provides sine-modulated pulse patterns. The following diagram illustrates how the voltages in phases L1 and L2 plus output voltage V_{L1-L2} (phase-to-phase voltage) are produced by pulse-width modulation or space vector modulation and shows their basic time characteristics. The frequency with which the IGBTs in the inverter phases are switched on and off is referred to as the pulse frequency or clock frequency of the inverter.



- The diagram below shows the time characteristic (in blue) of one of the three output voltages of the inverter (phase-to-phase voltage) and the resulting current (in black) generated in one of the three motor phases when a standard asynchronous motor with a rated frequency of 50 Hz or 60 Hz is used and the inverter is operating with a pulse frequency of 1.25 kHz. The diagram shows that the smoothing effect of the motor inductances causes the motor current to be virtually sinusoidal, despite the fact that the motor is supplied with a square-wave pulse pattern.



- Having already studied about the basics of PLC, now was the time to take a closer look at S7-1200, one of the popular PLCs SIEMENS offers. **S7-1200 Instruction Manual** was used for the same purpose.
- S7-1200 PLC is Siemens solution for controlling a variety of devices to satisfy your automation needs. The CPU combines a microprocessor, an integrated power supply, input and output circuits, built-in PROFINET, high-speed motion control I/O, and on-board analog inputs in a compact housing to create a powerful controller.



① Power connector

② Memory card slot under top door

③ Removable user wiring connectors (behind the doors)

④ Status LEDs for the on-board I/O

⑤ PROFINET connector (on the bottom of the CPU)

- *Different CPU models* of the 1200 series are available and summed up in the table given on next page.
- A signal board (SB) provides additional I/O for your CPU. The SB connects on the front of the CPU. A communication board (CB) allows you to add another communication port to your CPU. A battery board (BB) allows you to provide long term backup of the real time clock.
- STEP 7 provides user friendly environment for building, editing and monitoring the logic made in your program. It provides two different views of the project: a task-oriented set of portals that are organized on the functionality of the tools (Portal view), or a project-oriented view of the elements within the project (Project view).
- The CPU has *three modes of operation*: STOP mode, STARTUP mode, and RUN mode. Status LEDs on the front of the CPU indicate the current mode of operation.
  - In **STOP** mode, the CPU is not executing the program. You can download a project.
  - In **STARTUP** mode, the start-up OBs (if present) are executed once. Interrupt events are not processed during the start-up mode.
  - In **RUN** mode, the program cycle OBs are executed repeatedly. Interrupt events can occur and be processed at any point within the RUN mode. Some parts of a project can be downloaded in RUN mode
- The CPU supports a warm restart for entering the RUN mode. Warm restart does not include a memory reset. All non-retentive system and user data are initialized at warm restart. Retentive user data is retained.
- **Program Blocks:**
  - ❖ Organization blocks (OBs): OBs control the execution of the user program. Each OB must have a unique OB number. The default OB numbers are reserved below 200. Other OBs must be numbered 200 or greater. E.g. Program Cycle, Start-up, Time Delay Interrupt, Cyclic Interrupt, Hardware Interrupt, Time Error Interrupt, Diagnostic Error Interrupt, Pull or Plug of modules, Rack or Station Failure, Time of Day, Status, Update, Profile, MC-Interpolator, MC-Servo, MC-PreServo, and MC-PostServo.

Table 1- 1 Comparing the CPU models

Feature		CPU 1211C	CPU 1212C	CPU 1214C	CPU 1215C
Physical size (mm)		90 x 100 x 75	90 x 100 x 75	110 x 100 x 75	130 x 100 x 75
User memory	Work	30 Kbytes	50 Kbytes	75 Kbytes	100 Kbytes
	Load	1 Mbyte	1 Mbyte	4 Mbytes	4 Mbytes
	Retentive	10 Kbytes	10 Kbytes	10 Kbytes	10 Kbytes
Local on-board I/O	Digital	6 inputs/4 outputs	8 inputs/6 outputs	14 inputs/10 outputs	14 inputs/10 outputs
	Analog	2 inputs	2 inputs	2 inputs	2 inputs / 2 outputs
Process image size	Inputs (I)	1024 bytes	1024 bytes	1024 bytes	1024 bytes
	Outputs (Q)	1024 bytes	1024 bytes	1024 bytes	1024 bytes
Bit memory (M)		4096 bytes	4096 bytes	8192 bytes	8192 bytes
Signal module (SM) expansion		None	2	8	8
Signal board (SB), Battery board (BB), or communication board (CB)		1	1	1	1
Communication module (CM) (left-side expansion)		3	3	3	3
High-speed counters	Total	3 built-in I/O, 5 with SB	4 built-in I/O, 6 with SB	6	6
	Single phase	3 at 100 kHz SB: 2 at 30 kHz	3 at 100 kHz 1 at 30 kHz SB: 2 at 30 kHz	3 at 100 kHz 3 at 30 kHz	3 at 100 kHz 3 at 30 kHz
	Quadrature phase	3 at 80 kHz SB: 2 at 20 kHz	3 at 80 kHz 1 at 20 kHz SB: 2 at 20 kHz	3 at 80 kHz 3 at 20 kHz	3 at 80 kHz 3 at 20 kHz
Pulse outputs[1]		4	4	4	4
Memory card		SIMATIC Memory card (optional)			
Real time clock retention time		20 days, typ. / 12 day min. at 40 degrees C (maintenance-free Super Capicator)			
PROFINET		1 Ethernet communication port			2 Ethernet communication ports
Real math execution speed		2.3 μs/instruction			
Boolean execution speed		0.08 μs/instruction			

[1] For CPU models with relay outputs, you must install a digital signal board (SB) to use the pulse outputs.

Table 1- 2 Blocks, timers and counters supported by S7-1200

Element		Description
Blocks	Type	OB, FB, FC, DB
	Size	30 Kbytes (CPU 1211C) 50 Kbytes (CPU 1212C) 64 Kbytes (CPU 1214C and CPU 1215C)
	Quantity	Up to 1024 blocks total (OBs + FBs + FCs + DBs)
	Address range for FBs, FCs, and DBs	1 to 65535 (such as FB 1 to FB 65535)
	Nesting depth	16 from the program cycle or start up OB; 4 from the time delay interrupt, time-of-day interrupt, cyclic interrupt, hardware interrupt, time error interrupt, or diagnostic error interrupt OB
	Monitoring	Status of 2 code blocks can be monitored simultaneously
OBs	Program cycle	Multiple: OB 1, OB 200 to OB 65535
	Startup	Multiple: OB 100, OB 200 to OB 65535
	Time-delay interrupts and cyclic interrupts	4[1] (1 per event): OB 200 to OB 65535
	Hardware interrupts (edges and HSC)	50 (1 per event): OB 200 to OB 65535
	Time error interrupts	1: OB 80
	Diagnostic error interrupts	1: OB 82
Timers	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, 16 bytes per timer
Counters	Type	IEC
	Quantity	Limited only by memory size
	Storage	Structure in DB, size dependent upon count type • SInt, USInt: 3 bytes • Int, UInt: 6 bytes • DInt, UDInt: 12 bytes

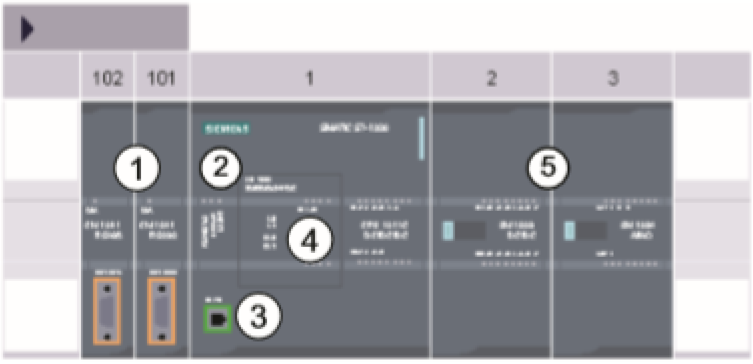
- A function block (FB) is a subroutine that is executed when called from another code block (OB, FB, or FC). The calling block passes parameters to the FB and also identifies a specific data block (DB) that stores the data for the specific call or instance of that FB. Changing the instance DB allows a generic FB to control the operation of a set of devices. For example, one FB can control several pumps or valves, with different instance DBs containing the specific operational parameters for each pump or valve.

- A function (FC) is a subroutine that is executed when called from another code block (OB, FB, or FC). The FC does not have an associated instance DB. The calling block passes parameters to the FC. The output values from the FC must be written to a memory address or to a global DB.

- **Memory management** : The CPU provides the following memory areas to store the user program, data, and configuration:
  - Load memory is non-volatile storage for the user program, data and configuration. When a project is downloaded to the CPU, it is first stored in the Load memory area. This area is located either in a memory card (if present) or in the CPU. This non-volatile memory area is maintained through a power loss. The memory card supports a larger storage space than that built-in to the CPU.
  - Work memory is volatile storage for some elements of the user project while executing the user program. The CPU copies some elements of the project from load memory into work memory. This volatile area is lost when power is removed, and is restored by the CPU when power is restored.
  - Retentive memory is non-volatile storage for a limited quantity of work memory values. The retentive memory area is used to store the values of selected user memory locations during power loss. When a power down or power loss occurs, the CPU restores these retentive values upon power up.
- **Data Types:**
  - Bit and Bit sequences: Bool (Boolean or bit value), Byte (8-bit byte value), Word (16-bit value), DWord (32-bit double-word value)
  - Integer: USInt (unsigned 8-bit integer), SInt (signed 8-bit integer), – UInt (unsigned 16-bit integer), Int (signed 16-bit integer), UDInt (unsigned 32-bit integer), DInt (signed 32-bit integer)
  - Floating-point Real : Real (32-bit Real or floating-point value), LReal (64-bit Real or floating-point value)
  - Time and Date : Time (32-bit IEC time value), Date (16-bit date value), TOD (32-bit time-off-day value), DT (64-bit date-and-time value)
  - Character and String : Char (8-bit single character), String (variable-length string of up to 254 characters)
  - Array
  - Data structure: Struct
  - PLC Data type
  - Pointers : Pointer, Any, Variant
- Accessing a "slice" of a tagged data type PLC tags and data block tags can be accessed at the bit, byte, or word level depending on their size. The syntax for accessing such a data slice is as follows:
  - "<PLC tag name>".xn (bit access)
  - "<PLC tag name>".bn (byte access)
  - "<PLC tag name>".wn (word access)
  - "<Data block name>".<tag name>.xn (bit access)
  - "<Data block name>".<tag name>.bn (byte access)

- "<Data block name>".<tag name>.wn (word access)

- One can create the device configuration for his PLC by adding a CPU and additional modules to project. ”Add Device” in Devices and Networks was done.



① Communication module (CM) or communication processor (CP): Up to 3, inserted in slots 101, 102, and 103

② CPU: Slot 1

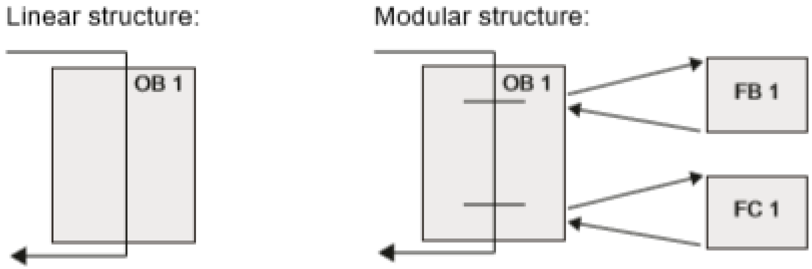
③ Ethernet port of CPU

④ Signal board (SB), communication board (CB) or battery board (BB): up to 1, inserted in the CPU

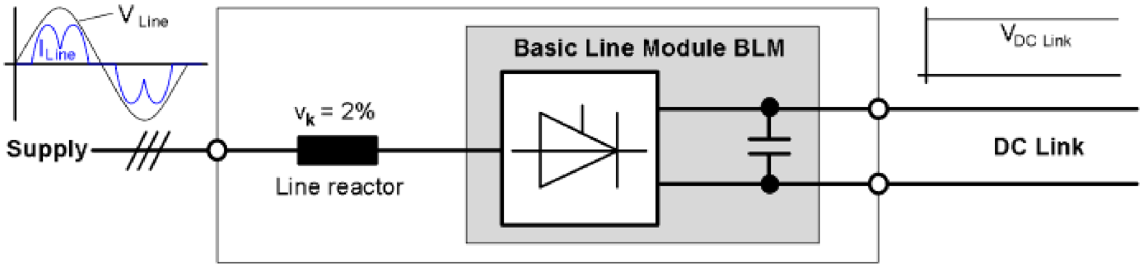
⑤ Signal module (SM) for digital or analog I/O: up to 8, inserted in slots 2 through 9
(CPU 1214C and CPU 1215C allow 8, CPU 1212C allows 2, CPU 1211C does not allow any)

- Adding modules to the configuration: Use the hardware catalog to add modules to the CPU:
  - Signal module (SM) provides additional digital or analog I/O points. These modules are connected to the right side of the CPU.
  - Signal board (SB) provides just a few additional I/O points for the CPU. The SB is installed on the front of the CPU.
  - Battery Board 1297 (BB) provides long-term backup of the real-time clock. The BB is installed on the front of the CPU.
  - Communication board (CB) provides an additional communication port (such as RS485). The CB is installed on the front of the CPU.
  - Communication module (CM) and communication processor (CP) provide an additional communication port, such as for PROFIBUS or GPRS. These modules are connected to the left side of the CPU.
- **CPU Insertion**: One can create device configuration by inserting a CPU into the project. Select the CPU in the "Add a new device" dialog and click "OK" to add the CPU to the project.
- To insert a module into the device configuration, select the module in the hardware catalog and either double-click or drag the module to the highlighted slot. One must add the modules to the device configuration and download the hardware configuration to the CPU for the modules to be functional.
- To configure the operational parameters for the CPU, select the CPU in the Device view and use the "Properties" tab of the inspector window:
  - PROFINET IP address and time synchronization for the CPU
  - Start-up behaviour of the CPU following an OFF to-ON power transition
  - Local (on-board) digital and analog I/O, high speed counters (HSC), and pulse generators
  - System clock (time, time zone and daylight saving time)
  - Read/write protection and password for accessing the CPU
  - Maximum cycle time or a fixed minimum cycle time and communications load
  - Web server properties

- Choosing type of **structure of a program:**
  - A _linear program_ executes all of the instructions for one's automation tasks in sequence, one after the other. Typically, the linear program puts all of the program instructions into the OB for the cyclic execution of the program (OB 1).
  - A _modular program_ calls specific code blocks that perform specific tasks. To create a modular structure, one can divide the complex automation task into smaller subordinate tasks that correspond to the technological functions of the process. Each code block provides the program segment for each subordinate task. One can structure his/her program by calling one of the code blocks from another block



- Line Infeeds: There are three infeeds namely; Basic, Smart and Active.
  - The **Basic Infeed** is a robust, unregulated Infeed for two-quadrant operation (i.e. the energy always flows from the supply system to the DC link). This Infeed is not designed to regenerate energy from the DC link back to the supply system. If regenerative energy is produced for brief periods by the drive, e.g. during braking, it must be converted to heat by a Braking Module connected to the DC link combined with a braking resistor. The Basic Infeed consists of a line-commutated, 6-pulse, three-phase rectifier equipped with thyristors or diodes. A line reactor with a relative short-circuit voltage of 2 % is generally connected on the line side

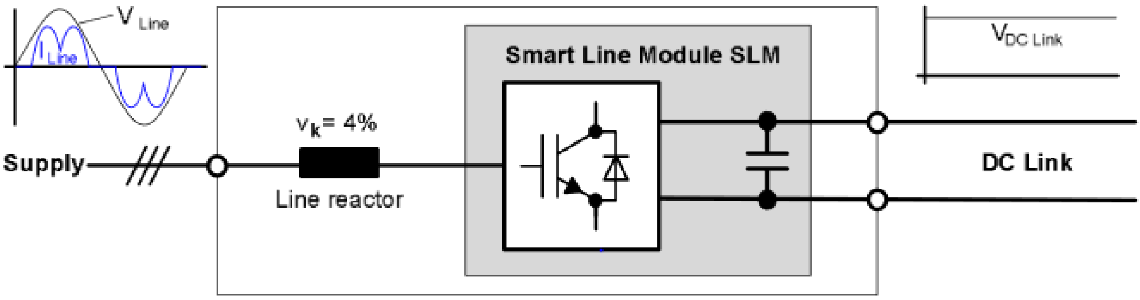


SINAMICS S120 Basic Infeed comprising a Basic Line Module with thyristors and a line reactor with v_k = 2%

The Basic Infeed is a line-commutated rectifier which, from the three-phase line voltage V_{Line}, produces an unregulated, load-dependent DC link voltage V_{DCLink}. Under no-load conditions, the DC link is charged to the peak line voltage value, i.e. V_{DCLink} = 1.41 • V Line. When loaded the DC link voltage decreases. When partially loaded the DC link voltage will be $V_{DCLink} \approx 1.35$ • V_{Line} and at full load, $V_{DCLink} \approx 1.32$ • V_{Line}.

As the DC link voltage is unregulated, line voltage fluctuations cause the DC link voltage to fluctuate correspondingly.

- The **Smart Infeed** is a stable, unregulated rectifier / regenerative unit for four-quadrant operation, i.e. the energy flows from the supply system to the DC link and vice versa. The current values stated in the catalogs are available in both, rectifier and regenerative operation. The Smart Infeed consists of an IGBT inverter, which operates on the line supply as a line-commutated 6-pulse bridge rectifier / regenerative unit. In contrast to the Active Infeed, the IGBTs are not active pulsed using the pulse-width modulation method. In rectifier operation (motor operation) the current flows via the diodes integrated into the IGBT modules from the line supply to the DC link, so that a line-commutated, 6-pulse diode bridge circuit is present in motor operation. In regenerative operation the current flows via the IGBTs, which are synchronised at the line frequency. Thus, a line-commutated, 6-pulse IGBT bridge circuit is present at regenerative operation. As IGBTs, in contrast to thyristors, can be switched off at any time, inverter shoot-through during regenerative operation caused by supply system failures cannot occur in contrast to rectifier / regenerative units equipped with thyristors. On the line side, the Smart Infeed is normally equipped with a line reactor having a relative short circuit voltage of v_k= 4 %.



SINAMICS S120 Smart Infeed comprising a Smart Line Module and a line reactor with v_k = 4%

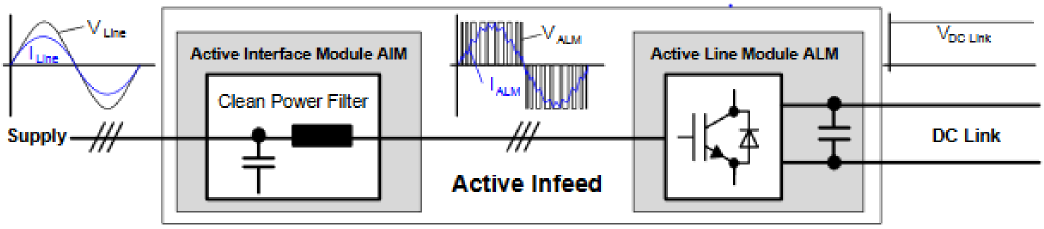
The IGBTs for regenerative operation of the Smart Infeed are always switched on at the natural firing point and switched off after 120° (electr.) independently from the direction of the energy flow. As a result of this, a current resp. energy flow from the supply to the DC link or vice-versa is possible at any time. The direction of the actual current resp. energy flow is only determined by the voltage ratios between the supply and the DC link. In steady-state motor operation, the DC link voltage during the possible current-flow phase is always smaller than the supply voltage, so that the current flows from the supply to the DC link via the diodes. In steady-state regenerative operation, the DC link voltage during the possible current-flow phase is always larger than the supply voltage, so that the current flows from the DC link to the supply via the IGBTs. This control principle offers the advantage that the Smart Infeed can react relatively fast to load variations and can also change the direction of the current resp. energy flow at any time. However, a characteristic of the control principle described is that a harmonic reactive current flows on the line side in no-load operation. The cause is the sinusoidal supply voltage on the one hand, and an almost perfectly smoothed DC voltage in the DC link during no-load operation at the other hand. Consequently, directly after the firing of the IGBTs, a short-term current flows from the DC link to the supply because, at this time, the supply voltage is slightly lower than the DC link voltage. If the supply voltage then reaches its peak value, the voltage ratios are reversed and thus the current direction is also reversed. This reactive current decreases as the load on the Smart Infeed increases and disappears completely under full load.

The reactive current under no load can be prevented by a regenerative operation disable command (parameter p3533: Infeed, inhibit generator mode). It can also be helpful to inhibit regenerative operation of the Infeed to obtain greater stability in operation when the unit is operating in motor mode if one or more of the following conditions apply:

- ✓ Operation of the Smart Infeed on a weak, unstable supply system with frequent line voltage dips.
- ✓ Operation of the Smart Infeed within a DC configuration with very high DC link capacitance C_{DCLink}
- ✓ 12-pulse operation of the Smart Infeed on a three-winding transformer.
- ✓ Mixed Smart Infeed/Basic Infeed operation.

Regenerative operation can be inhibited and enabled by the higher-level automation system, for example, if the process steps in which regenerative operation can occur are clearly defined in the process control system. Since the Smart Infeed is a line-commutated Infeed, it generates an unregulated, load-dependent DC link voltage V_{DCLink} from the three-phase line voltage V_{Line}. In motor operation, the DC link voltage decreases by a slightly larger amount than on the Basic Infeed, because the voltage drop across the 4% reactor is higher than across the 2% reactor on the Basic Infeed. Under partial load in motor operation, $V_{DCLink} \approx 1.32 \bullet V_{Line}$, while the figure for full load in motor operation is $V_{DCLink} \approx 1.30 \bullet V_{Line}$ (motor mode). The DC link voltage is higher in regenerative operation than in motor operation, because the direction of current flow reverses and thus also the voltage drop across the 4% reactor. Under partial load in regenerative operation, $V_{DCLink} \approx 1.38 \bullet V_{Line}$, while the figure for full load in regenerative operation is $V_{DCLink} \approx 1.40 \bullet V_{Line}$ (regenerative). Because the DC link voltage is not regulated, fluctuations in the line voltage and changes in the operating state (motor mode / regenerative mode) cause it to fluctuate accordingly.

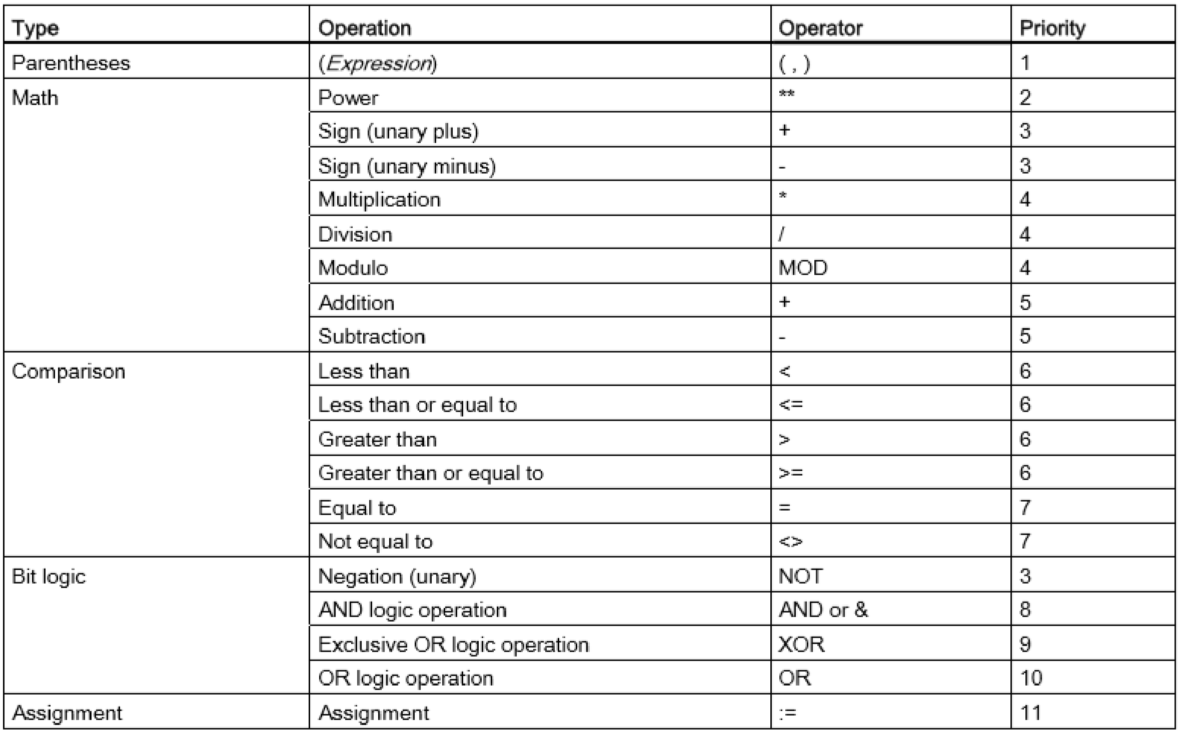
- ❖ The **Active Infeed** is an actively pulsed, stable, regulated rectifier / regenerative unit for four-quadrant operation, i.e. the energy flows from the supply system to the DC link and vice versa. The current values stated in the catalogs are available in both rectifier and regenerative operation The Active Infeed comprises a self-commutated IGBT inverter (Active Line Module ALM), which operates on the supply system via the Clean Power Filter (Active Interface Module). The Active Line Module operates according to the method of pulse-width modulation and generates a constant, regulated DC link voltage V_{DCLink} from the three-phase line voltage V_{Line}. The Clean Power Filter which is installed between the Active Line Module and the supply system, filters out, as far as possible, the harmonics from the Active Line Module's pulse-width modulated voltage V_{ALM}, thereby ensuring a virtually sinusoidal input current on the line side and, therefore, minimal harmonic effects on the supply system.



SINAMICS S120 Active Infeed comprising an Active Interface Module and an Active Line Module

The Active Infeed is a self-commutated rectifier/regenerative unit and produces from the three-phase line voltage V_{Line} a regulated DC link voltage V_{DCLink}, which remains constant independently from line voltage variations and supply voltage dips. It operates as a step-up converter, i.e. the DC link voltage is always higher than the peak value of the line voltage ($V_{DCLink} > 1.41 \bullet V_{Line}$). The value can be parameterized (1.42 to 2.0) and its factory setting is $V_{DCLink} = 1.50 \bullet V_{Line}$.

- When *designing a PLC system*, one can choose from a variety of methods and criteria. The following *general guidelines* can apply to many design projects and of course company specifications can change the procedure according to the varying needs:
  - Partition the process or machine: Dividing process or machine into sections that have a level of independence from each other.
  - Create the functional specifications: Writing the descriptions of operation for each section of the process or machine, such as the I/O points, the functional description of the operation, the states that must be achieved before allowing action for each actuator etc.
  - Design the safety circuits: circuits: Identifying any equipment that might require hard-wired logic for safety. Remember that control devices can fail in an unsafe manner, which can produce unexpected startup or change in the operation of machinery. Where unexpected or incorrect operation of the machinery could result in physical injury to people or significant property damage, consider the implementation of electromechanical overrides (which operate independently of the PLC) to prevent unsafe operations.
  - Plan system security: Determining what level of protection one requires for access to the process. So one can password-protect CPUs and program blocks from unauthorized access.
  - Specify the operator stations
  - Create the configuration drawings: Overview, mechanical and electrical drawings.
  - Create a list of symbolic names



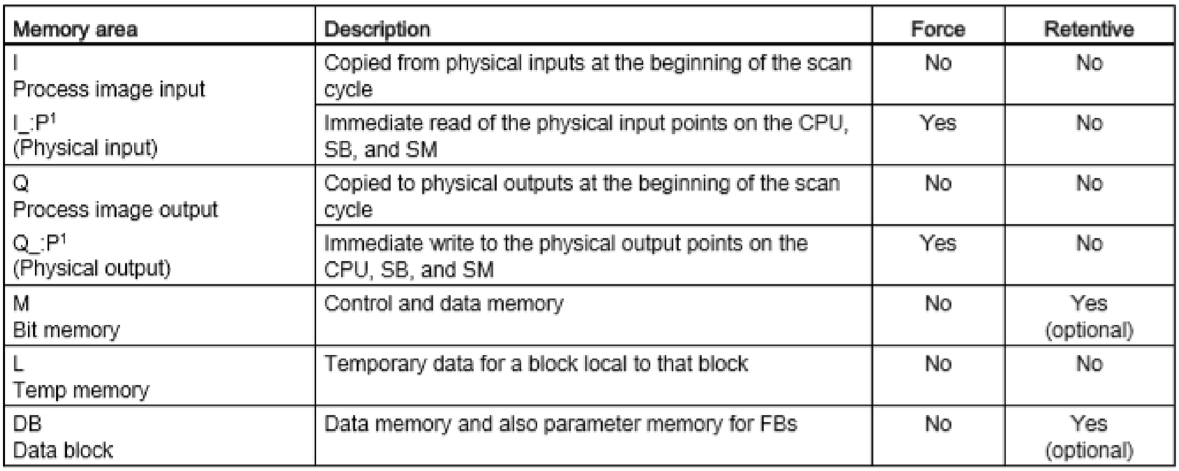
Type	Operation	Operator	Priority
Parentheses	(*Expression*)	(,)	1
Math	Power	**	2
	Sign (unary plus)	+	3
	Sign (unary minus)	-	3
	Multiplication	*	4
	Division	/	4
	Modulo	MOD	4
	Addition	+	5
	Subtraction	-	5
Comparison	Less than	<	6
	Less than or equal to	<=	6
	Greater than	>	6
	Greater than or equal to	>=	6
	Equal to	=	7
	Not equal to	<>	7
Bit logic	Negation (unary)	NOT	3
	AND logic operation	AND or &	8
	Exclusive OR logic operation	XOR	9
	OR logic operation	OR	10
Assignment	Assignment	:=	11

Table: Priority Order for SCL expressions

- The CPU provides the following **memory areas** to store the user program, data, and configuration:
  - *Load memory* is non-volatile storage for the user program, data and configuration. When a project is downloaded to the CPU, it is first stored in the Load memory area. This area is located either in a memory card (if present) or in the CPU. This non-volatile

memory area is maintained through a power loss. You can increase the amount of load memory available for data logs by installing a memory card.

- *Work memory* is volatile storage for some elements of the user project while executing the user program. The CPU copies some elements of the project from load memory into work memory. This volatile area is lost when power is removed, and is restored by the CPU when power is restored.
- *Retentive* memory is non-volatile storage for a limited quantity of work memory values. The retentive memory area is used to store the values of selected user memory locations during power loss. When a power down or power loss occurs, the CPU restores these retentive values upon power up.



Memory area	Description	Force	Retentive
I Process image input	Copied from physical inputs at the beginning of the scan cycle	No	No
I_:P[1] (Physical input)	Immediate read of the physical input points on the CPU, SB, and SM	Yes	No
Q Process image output	Copied to physical outputs at the beginning of the scan cycle	No	No
Q_:P[1] (Physical output)	Immediate write to the physical output points on the CPU, SB, and SM	Yes	No
M Bit memory	Control and data memory	No	Yes (optional)
L Temp memory	Temporary data for a block local to that block	No	No
DB Data block	Data memory and also parameter memory for FBs	No	Yes (optional)

- **Pulse Outputs**: The CPU or signal board (SB) can be configured to provide four pulse generators for controlling high-speed pulse output functions, either as pulse-width modulation (PWM) or as pulse-train output (PTO). The basic motion instructions use PTO outputs. One can assign each pulse generator to either PWM or PTO, but not both at the same time.
  Pulse outputs cannot be used by other instructions in the user program. When one configures the outputs of the CPU or SB as pulse generators, the corresponding output addresses are removed from the Q memory and cannot be used for other purposes in the user program. If the user program writes a value to an output used as a pulse generator, the CPU does not write that value to the physical output.

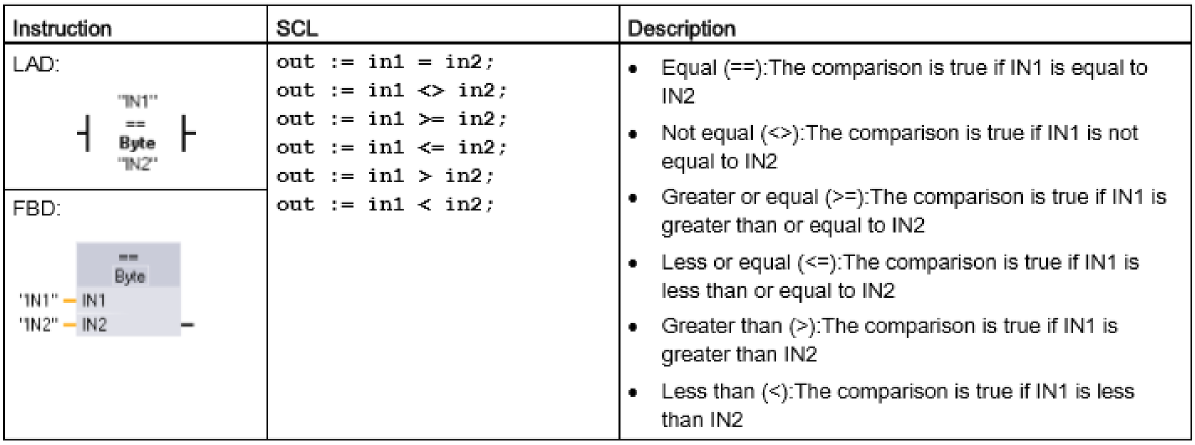
- After reviewing FBD and Ladder Logics in last week's document lets move to SCL overview:
  Structured Control Language (SCL) is a high-level, PASCAL-based programming language for the SIMATIC S7 CPUs. SCL supports the block structure of STEP 7. One can also include program blocks written in SCL with program blocks written in LAD and FBD. SCL uses standard PASCAL program control operations, such as IF-THEN-ELSE, CASE, REPEAT-UNTIL, GOTO and RETURN. One can use any PASCAL reference for syntactical elements of the SCL programming language. Many of the other instructions for SCL, such as timers and counters, match the LAD and FBD instructions. Because SCL, like PASCAL, offers conditional processing, looping, and nesting control structures, one can implement complex algorithms in SCL more easily than in LAD or FBD.

  The following examples show different expressions for different uses:

"C" := #A+#B;	Assigns two local variables to a tag
"Data_block_1".Tag := #A;	Assignment to a data block tag
IF #A > #B THEN "C" := #A;	Condition for the IF-THEN statement
"C" := SQRT (SQR (#A) + SQR (#B));	Parameters for the SQRT instruction As

- Addressing of global variables (tags): "<tag name>" (Tag name or data block name enclosed in double quotes)
- Addressing of local variables: #<variable name> (Variable name preceded by "#" symbol)
- Absolute addressing: %<absolute address>, for example %I0.0 or %MW10

- In a code block you can declare following parameters:
  - Input, Output, InOut, and Ret_Val: These parameters define the input tags, output tags, and return value for the code block. The tag name that is entered here is used locally during the execution of the code block. One typically would not use the global tag name in the tag table.
  - Static (FBs only): Static tags are used for storage of static intermediate results in the instance data block. Static data is retained until overwritten, which may be after several cycles. The names of the blocks, which are called in this code block as multi-instance, are also stored in the static local data.
  - Temp: These parameters are the temporary tags that are used during the execution of the code block.
- **Assignment** operations:

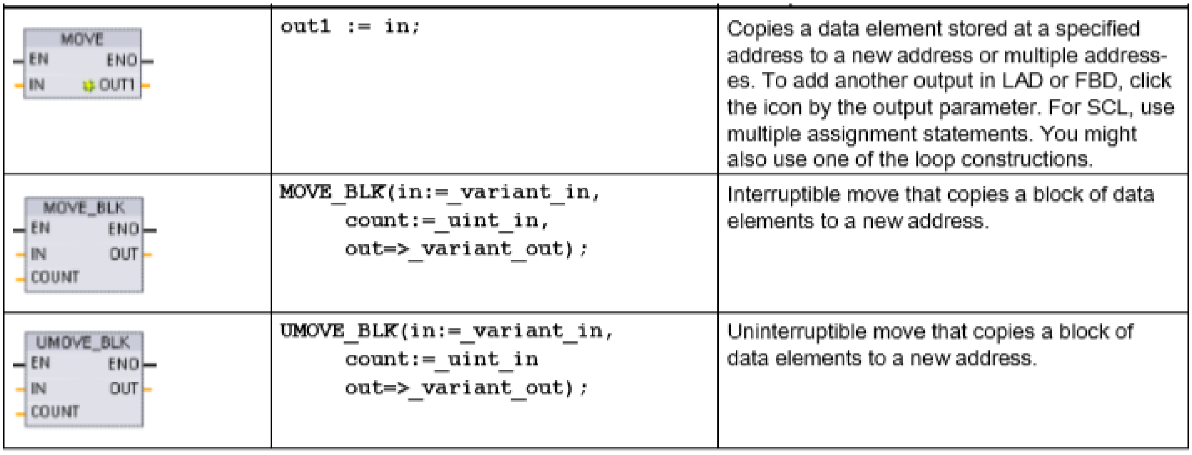

"IN1"
==
Byte
"IN2"
==
Byte
"IN1" IN1
"IN2" IN2

Instruction	SCL	Description

LAD:	out := in1 = in2;
	out := in1 <> in2;
	out := in1 >= in2;
	out := in1 <= in2;
	out := in1 > in2;
	out := in1 < in2;
		• Equal (==):The comparison is true if IN1 is equal to IN2
		• Not equal (<>):The comparison is true if IN1 is not equal to IN2
		• Greater or equal (>=):The comparison is true if IN1 is greater than or equal to IN2
		• Less or equal (<=):The comparison is true if IN1 is less than or equal to IN2
		• Greater than (>):The comparison is true if IN1 is greater than IN2
		• Less than (<):The comparison is true if IN1 is less than IN2

FBD:		

- **MOVE** Operations:

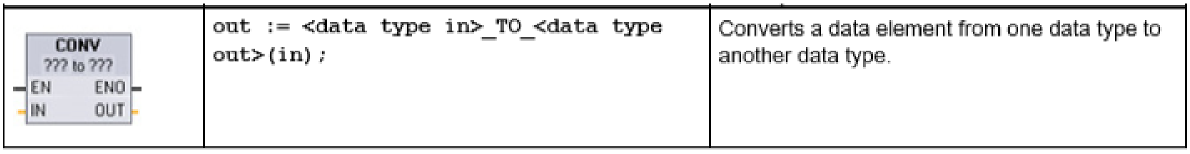

MOVE
EN ENO
IN OUT1
MOVE_BLK
EN ENO
IN OUT
COUNT
UMOVE_BLK
EN ENO
IN OUT
COUNT

MOVE	out1 := in;	Copies a data element stored at a specified address to a new address or multiple addresses. To add another output in LAD or FBD, click the icon by the output parameter. For SCL, use multiple assignment statements. You might also use one of the loop constructions.

MOVE_BLK	MOVE_BLK(in:=_variant_in,
	count:=_uint_in,
	out=>_variant_out);
		Interruptible move that copies a block of data elements to a new address.

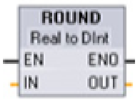
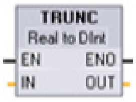
UMOVE_BLK	UMOVE_BLK(in:=_variant_in,
	count:=_uint_in
	out=>_variant_out);
		Uninterruptible move that copies a block of data elements to a new address.

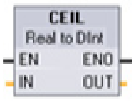
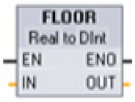
- **Conversion** Operation:

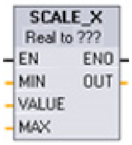

CONV
??? to ???
EN ENO
IN OUT

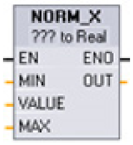
CONV	out := <data type in>_TO_<data type out>(in);	Converts a data element from one data type to another data type.

- **Rounding off** Operations:

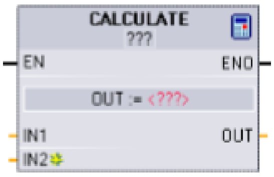
	`out := ROUND (in);`	Converts a real number (Real or LReal) to an integer. The instruction rounds the real number to the nearest integer value (IEEE - round to nearest). If the number is exactly one-half the span between two integers (for example, 10.5), then the instruction rounds the number to the even integer. For example: • ROUND (10.5) = 10 • ROUND (11.5) = 12 For LAD/FBD, you click the "???" in the instruction box to select the data type for the output, for example, "DInt". For SCL, the default output data type is DINT. To round to another output data type, enter the instruction name with the explicit name of the data type, for example, ROUND_REAL or ROUND_LREAL.
	`out := TRUNC(in);`	Converts a real number (Real or LReal) to an integer. The fractional part of the real number is truncated to zero (IEEE - round to zero).

	`out := CEIL(in);`	Converts a real number (Real or LReal) to the closest integer greater than or equal to the selected real number (IEEE "round to +infinity").
	`out := FLOOR(in);`	Converts a real number (Real or LReal) to the closest integer smaller than or equal to the selected real number (IEEE "round to -infinity").

 SCALE_X
Real to ???
EN
ENO
MIN
OUT
VALUE
MAX
	`out := SCALE_X(`
	`min:=_in_,`
	`value:=_in_,`
	`max:=_in_);`
		Scales the normalized real parameter VALUE where (0.0 <= VALUE <= 1.0) in the data type and value range specified by the MIN and MAX parameters:
		OUT = VALUE (MAX - MIN) + MIN

 NORM_X
??? to Real
EN
ENO
MIN
OUT
VALUE
MAX
	`out := NORM_X(`
	`min:=_in_,`
	`value:=_in_,`
	`max:=_in_);`
		Normalizes the parameter VALUE inside the value range specified by the MIN and MAX parameters:
		OUT = (VALUE - MIN) / (MAX - MIN),
		where (0.0 <= OUT <= 1.0)

- **CALCULATE** Instruction:

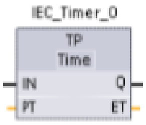
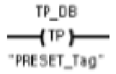
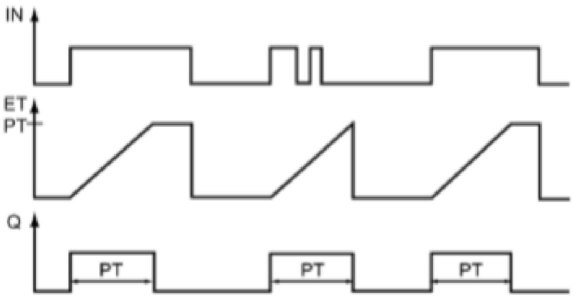
 CALCULATE ??? EN ENO OUT := <???> IN1 OUT IN2	Use the standard SCL math expressions to create the equation.	The CALCULATE instruction lets you create a math function that operates on inputs (IN1, IN2, .. INn) and produces the result at OUT, according to the equation that you define. • Select a data type first. All inputs and the output must be the same data type. • To add another input, click the icon at the last input.

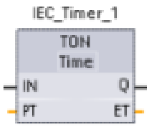
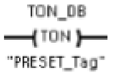
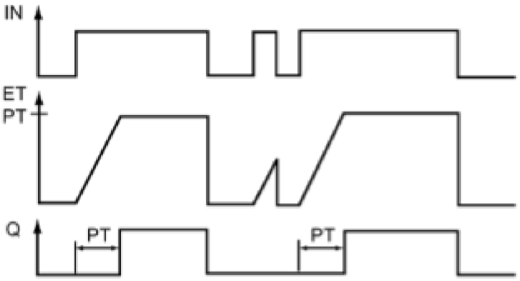
- **TIMER** Operations:

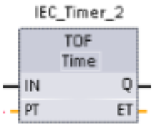
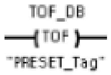
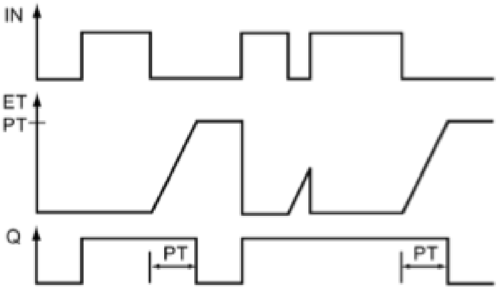
  The S7-1200 supports the following timers:

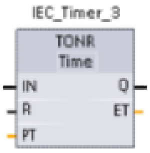
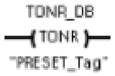
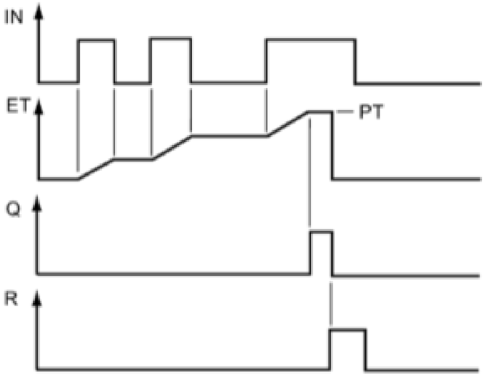
  - The TP timer generates a pulse with a preset width time.
  - The TON timer sets the output (Q) to ON after a preset time delay.
  - The TOF timer sets the output (Q) to ON and then resets the output to OFF after a preset time delay.
  - The TONR timer sets the output (Q) to ON after a preset time delay. The elapsed time is accumulated over multiple timing periods until the reset (R) input is used to reset the elapsed time.
  - The PT (preset timer) coil loads a new preset time value in the specified timer.
  - The RT (reset timer) coil resets the specified timer.

LAD / FBD	SCL	Timing diagram

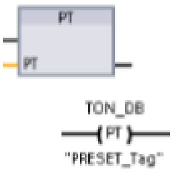


	"timer_db".TP(
	IN:=_bool_in_,
	PT:=_time_in_,
	Q=>_bool_out_,
	ET=>_time_out_);
		

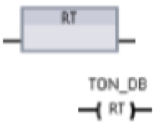


	"timer_db".TON(
	IN:=_bool_in_,
	PT:=_time_in_,
	Q=>_bool_out_,
	ET=>_time_out_);
		



	"timer_db".TOF(
	IN:=_bool_in_,
	PT:=_time_in_,
	Q=>_bool_out_,
	ET=>_time_out_);
		



	"timer_db".TONR(
	IN:=_bool_in_,
	R:=_bool_in_,
	PT:=_time_in_,
	Q=>_bool_out_,
	ET=>_time_out_);
		

LAD / FBD	SCL	Description

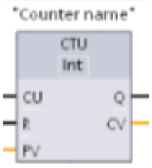
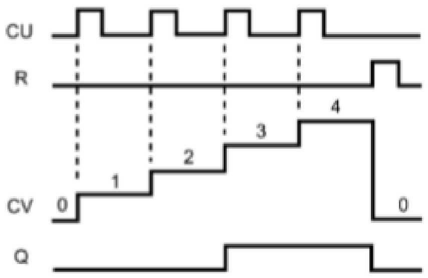
	PRESET_TIMER(
	PT:=_time_in_,
	
	TIMER:=_iec_timer_in_)
	;
		Use the Preset timer -(PT)- and Reset timer -(RT)- coil instructions with either box or coil timers. These coil instructions can be placed in a mid-line position. The coil output power flow status is always the same as the coil input status.
		• When the -(PT)- coil is activated, the PRESET time element of the specified IEC_Timer DB data is set to the "PRESET_Tag" time duration.
		• When the -(RT)- coil is activated, the ELAPSED time element of the specified IEC_Timer DB data is reset to 0.

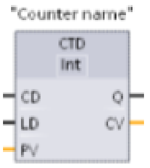
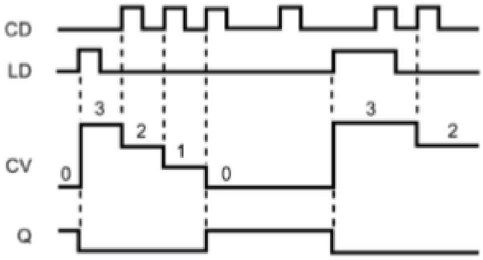
	RESET_TIMER(_iec_timer_in_);	

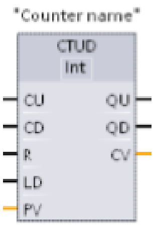
Timer	Changes in the PT and IN box parameters and the corresponding coil parameters
TP	• Changing PT has no effect while the timer runs. • Changing IN has no effect while the timer runs.
TON	• Changing PT has no effect while the timer runs. • Changing IN to FALSE, while the timer runs, resets and stops the timer.
TOF	• Changing PT has no effect while the timer runs. • Changing IN to TRUE, while the timer runs, resets and stops the timer.
TONR	• Changing PT has no effect while the timer runs, but has an effect when the timer resumes. • Changing IN to FALSE, while the timer runs, stops the timer but does not reset the timer. Changing IN back to TRUE will cause the timer to start timing from the accumulated time value.

- COUNTER Operations:
  - One uses the counter instructions to count internal program events and external process events: The "count up" counter (CTU) counts up by 1 when the value of the input parameter CU changes from 0 to 1.
  - The "count down" counter (CTD) counts down by 1 when the value of input parameter CD changes from 0 to 1.
  - The "count up and down" counter (CTUD) counts up or down by 1 on the 0 to 1 transition of the count up (CU) or count down (CD) inputs.
  - S7-1200 also provides high-speed counters (HSC) for counting events that occur faster than the OB execution rate.

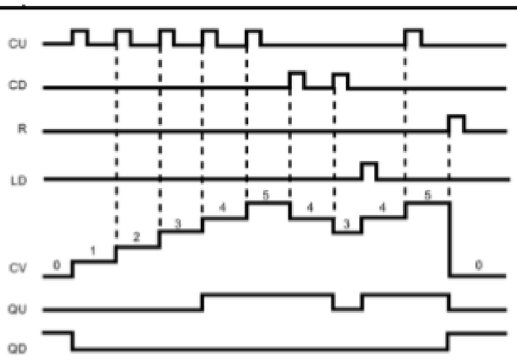
LAD / FBD	SCL	Operation

	"ctu_db".CTU(
	CU:=_bool_in,
	R:=_bool_in,
	PV:=_in_,
	Q=>_bool_out,
	CV=>_out_);
		

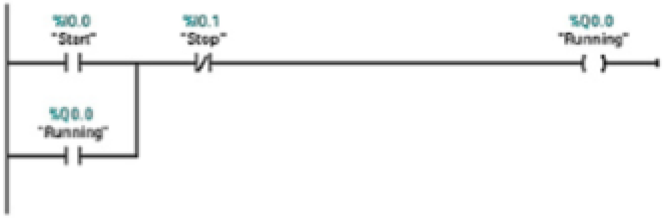
	"ctd_db".CTD(
	CD:=_bool_in,
	LD:=_bool_in,
	PV:=_in_,
	Q=>_bool_out,
	CV=>_out_);
		



```
"ctud_db".CTUD(
      CU:=_bool_in,
      CD:=_bool_in,
      R:=_bool_in,
      LD:=_bool_in,
      PV:=_in_,
      QU=>_bool_out,
      QD=>_bool_out,
      CV=>_out_);
```



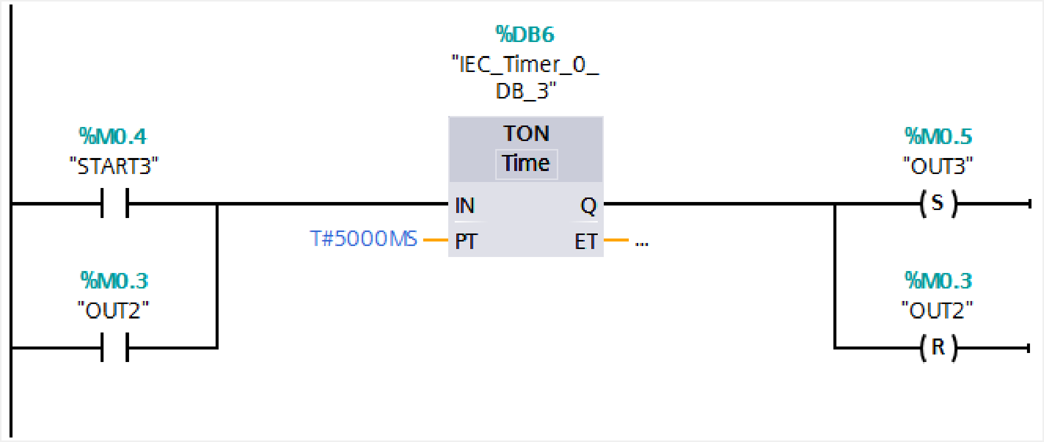
- There are many more instructions yet to study in TIA portal yet. Still we did go through the basic of the instructions and now ready for more practical applications of the concepts and hands on experience on TIA Portal. SIMATIC Field PG M4 was used for the same purpose with TIA Portal v14 loaded on it.
- Firstly a simple implementation of a starting a motor using Ladder Logic was done:



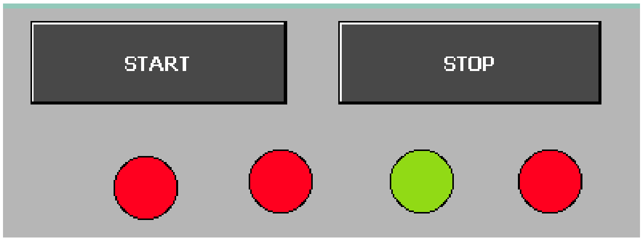
In this, force tables were used to vary the values of the input parameters because we can't change them while monitoring since we didn't use memory bits. After which the use of memory bits was carried out always because we are dealing with only simulations in our learning phase.

- **HMI-to-PLC communication**: The CPU supports PROFINET communications connections to HMIs. The *following requirements* must be considered when setting up communications between CPUs and HMIs:
    - The PROFINET port of the CPU must be configured to connect with the HMI.
    - The HMI must be setup and configured.
    - The HMI configuration information is part of the CPU project and can be configured and downloaded from within the project.
    - No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network
- **Steps to connect HMI**:
    1. Establishing the hardware communications connection A PROFINET interface establishes the physical connection between an HMI and a CPU. Since AutoCross-Over functionality is built into the CPU, one can use either a standard or crossover Ethernet cable for the interface. An Ethernet switch is not required to connect an HMI and a CPU.
    2. Configuring the devices.
    3. Configuring the logical network connections between an HMI and a CPU.
    4. Configuring an IP address in your project Use the same configuration process; however, one must configure IP addresses for the HMI and the CPU.
    5. Testing the PROFINET network. One must download the configuration for each CPU and HMI device.
- After HMI basics were covered. Second task was to carry out an implementation of *synchronized lighting* of 4 LEDs. If LED1 is switched on then after 5 seconds it switches off and LED2 starts. Again after 5 seconds it switches off and LED3 switches on. It goes in same way with LED4. And after LED4 switches off, LED1 should start. Stop button should stop the entire cycle.

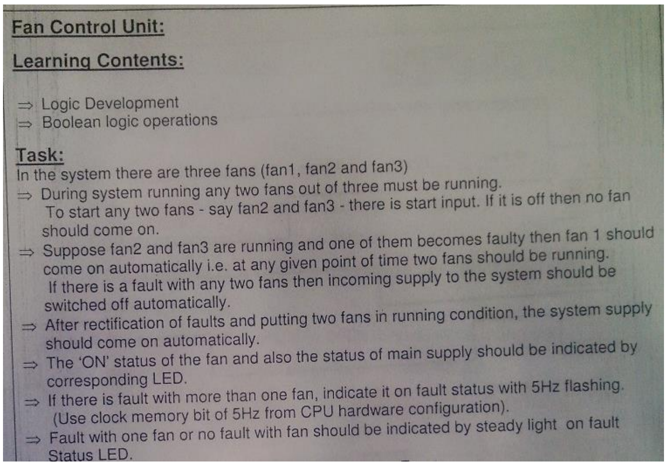
Here’s a basic ladder logic of the third LED. Rest were implemented in same manner:



Implemented the entire setup on a working HMI Panel:



- Thirdly a question on *fan control unit* was given as below:



Fan Control Unit:

Learning Contents:

⇒ Logic Development
⇒ Boolean logic operations

Task:
In the system there are three fans (fan1, fan2 and fan3)

⇒ During system running any two fans out of three must be running. To start any two fans - say fan2 and fan3 - there is start input. If it is off then no fan should come on.

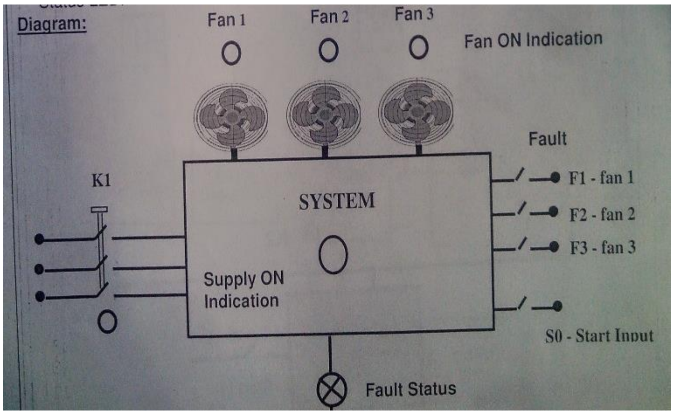
⇒ Suppose fan2 and fan3 are running and one of them becomes faulty then fan 1 should come on automatically i.e. at any given point of time two fans should be running. If there is a fault with any two fans then incoming supply to the system should be switched off automatically.

⇒ After rectification of faults and putting two fans in running condition, the system supply should come on automatically.

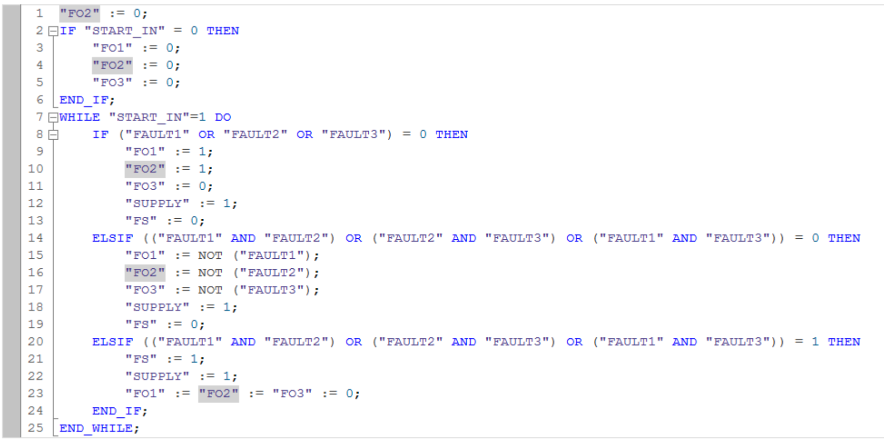
⇒ The ‘ON’ status of the fan and also the status of main supply should be indicated by corresponding LED.

⇒ If there is fault with more than one fan, indicate it on fault status with 5Hz flashing. (Use clock memory bit of 5Hz from CPU hardware configuration).

⇒ Fault with one fan or no fault with fan should be indicated by steady light on fault Status LED.

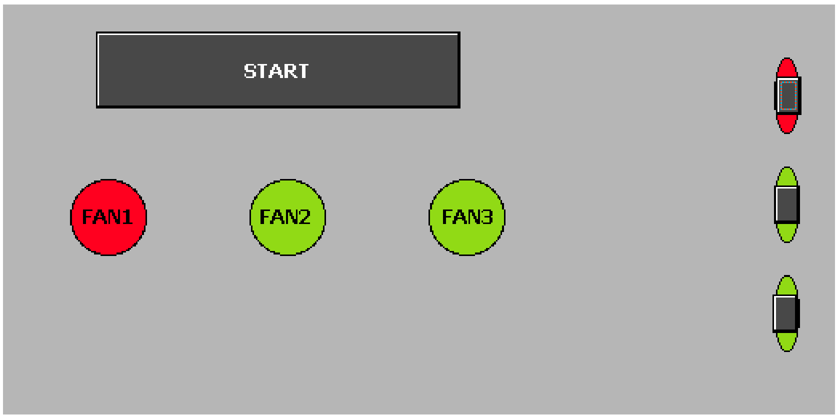


Here the faults were taken on HMI. They actually occur and there is no button to create faults. These side buttons on right side were created only for the sakes of simulation. Code:-

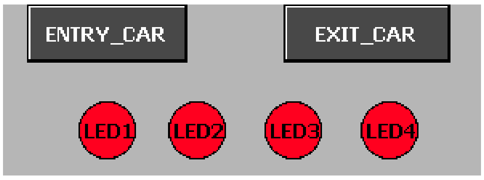


```
"FO2" := 0;
IF "START_IN" = 0 THEN
    "FO1" := 0;
    "FO2" := 0;
    "FO3" := 0;
END_IF;
WHILE "START_IN"=1 DO
    IF ("FAULT1" OR "FAULT2" OR "FAULT3") = 0 THEN
        "FO1" := 1;
        "FO2" := 1;
        "FO3" := 0;
        "SUPPLY" := 1;
        "FS" := 0;
    ELSIF (("FAULT1" AND "FAULT2") OR ("FAULT2" AND "FAULT3") OR ("FAULT1" AND "FAULT3")) = 0 THEN
        "FO1" := NOT ("FAULT1");
        "FO2" := NOT ("FAULT2");
        "FO3" := NOT ("FAULT3");
        "SUPPLY" := 1;
        "FS" := 0;
    ELSIF (("FAULT1" AND "FAULT2") OR ("FAULT2" AND "FAULT3") OR ("FAULT1" AND "FAULT3")) = 1 THEN
        "FS" := 1;
        "SUPPLY" := 1;
        "FO1" := "FO2" := "FO3" := 0;
    END_IF;
END_WHILE;
```

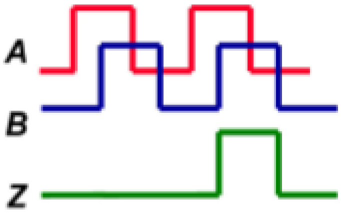
HMI Implementation:-



- Lastly a *parking system* implementation was to be carried out for indicating whether it was empty, 1-5, 5-9 or full for n floors. Was implemented using CTUD.
HMI:-



- There were many other small things that were observed over browsing TIA portal. Like Traces, HMI Trends etc. These will be explored in detail in upcoming week along with Technology objects.
- Let's move to theory part again. **Encoders** now. The spectrum of rotary position and speed feedback devices on the market is increasing in breadth on a daily basis. The question is no longer just whether you should go with an encoder or a resolver. The number of questions can be mind boggling. Beginning with "Should I go absolute or incremental?" Should I go magnetic or optical? There are other manufacturers out there that make capacitive and inductive. Should I be considering them as well? Do I need commercial, industrial, heavy, extreme, or maximum duty systems? Answer all these questions, and its easy street right? Wrong. When it comes to resolvers, you need to know size, accuracy, mounting, speed(s), and electrical interface. With encoders, you need to know resolution, accuracy, mounting, output, and connection.
- **Incremental Encoders** rely on external electronics to interpret the position based on the count of the events that occurred on that device. The outputs for incremental encoders can come in the form of a single square wave (A), phased square waves (A and B) to determine direction of rotation, or phased square waves and an index or one pulse per revolution (A, B, and Z).



The concept of phasing square waves to determine rotational direction is often referred to as "quadrature". The means of achieving an incremental signal are typically referred to as the encoder engines. The two primary encoder engine categories are optical and magnetic. In both engines, similar sensor alignment is performed to provide output compatibility. In the optical design, light is generated by an LED and detected by a chip-level sensor. Between the two is a code disc, typically made of glass, metal, or plastic. In an incremental encoder, the code disc is etched, coated, or punched (if metal) with a fine grating of similar lines around the circumference.

In the magnetic design, there is a magnetic wheel or disk, a magneto-resistive sensor, and a conditioning circuit. The disk or wheel is magnetized with several poles. The sensor converts the sinusoidal change in magnetic field to an electrical signal as the disk or wheel rotates. That electrical signal is multiplied, divided, or interpolated by the conditioning circuit to produce the desirable square wave output. Incremental encoders continue to lead in the area of speed feedback with no signs of slowing. Incremental encoders are the most widely used of all rotary feedback devices. With the ease of solid state circuit and software design, devices that accept incremental encoder input are widely available. One can find drives, panel meters, counters, and PC cards that interface with incremental encoders. Optical encoders can be found in office environment applications such as copiers, and industrial applications such as automated guided vehicles (AGV's), magnetic encoders are typically needed in harsh conditions where optical encoders may show significant performance decrease. These applications can be over-head cranes, off highway vehicles, and paper mills.

- **Absolute Encoders**: Absolute devices provide a means of knowing the exact angle of the rotation with respect to a fixed device. An absolute encoder uses a unique binary pattern that doesn't repeat itself within the revolution, giving the encoder its absolute attributes. The feedback will also change when the rotational position is changed when power is removed from that device. When a gear train is used to track the number rotations of an encoder, it is a multi-turn encoder.

  In an optical absolute, a disc rotates between the LED and sensor, light is either allowed to pass to several sensors or blocked, based on the disc's pattern. This, ultimately, is what provides the "on-or-off" of each bit in the digital signal from the encoder.

  At the cost of precision manufacturing and an optimal application environment, optical encoders provide for the highest resolution and highest accuracy of all the feedback options. There are several means of transmitting the absolute position. The first absolute feedback devices transmitted their position by using parallel data. In parallel data, each binary digit has its own wire which is interpreted by a controller. Next, there are serial encoders where absolute position is transmitted in sync with a clock pulse. SSI or Synchronous Serial Interface is the most common protocol of serial encoders. BiSS encoders manipulate the clock pulses to provide bidirectional communication.

  Bus encoders are now on the market. They allow for several encoders to be wired in line or taped off of a single transmission cable. DeviceNet, Profibus, and Interbus are among the most popular bus protocols.

  In magnetic absolute encoders, absolute information can be obtained by rotating a magnet axially above a sensor network as shown in Figure 1.

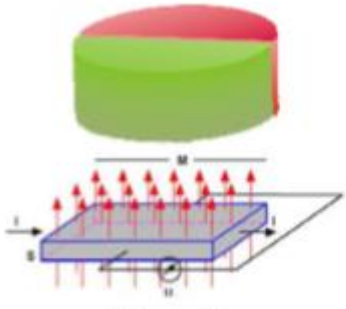


Figure 1

The sensor is typically either a Hall-effect chip type or magneto-resistive sensor circuit.

In terms of environment, the same rules would be followed for absolute or incremental. However, absolute encoders use is growing worldwide due to increased complexity in machine design that requires multiple axis to be synchronized and operate efficiently and safely. Incremental encoders rely on secondary devices, such as limit switches, to operate functionally for accurate position feedback.

- **Resolvers**: Similar to encoders, resolvers are also electromechanical devices that convert mechanical motion into an electronic signal. However, unlike an encoder, a resolver transmits an analog signal rather than digital. It is essentially a rotary transformer with one primary winding and two secondary windings that are phased 90 mechanical degrees as shown in Figure 2. The resolver output requires control inputs and logic that can interpret analog signals.

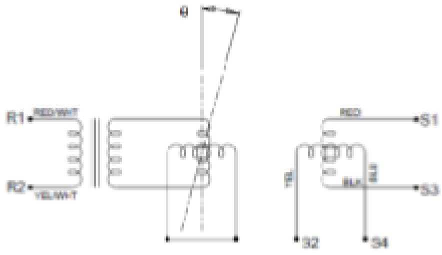


Figure 2

One of the specifications in a resolver is its number of speeds. The output in Figure 3 is the output of a resolver with a single speed output. The number of speeds is equivalent to the number of amplitude modulated sinusoidal cycles in one revolution of the resolver. Multiple speed resolvers are achieved by increasing the number of magnetic poles in the rotor and stator equally. The maximum number of speeds is limited to the size of the resolver, and is typically done to increase the accuracy. However, a single speed resolver is essentially a single turn absolute device. By increasing the speeds of a resolver, the absolute information is lost. If space allows, mounting a single speed resolver on top of a multiple speed resolver will provide the higher accuracy and absolute benefits.

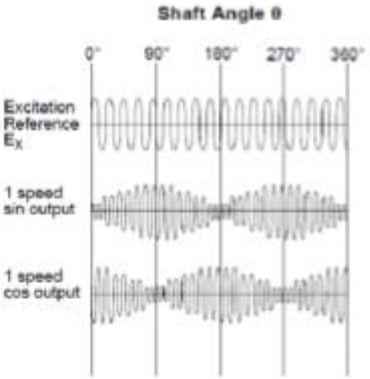


Figure 3

Resolvers lend themselves to maximum duty applications because of their simple component similarity to electric motors (windings, laminations, bearings, and carrier). The lack of optics and precision alignment increases shock and vibration capabilities. The lack of both optics and solid state electronics allows for use in high radiation environments.
Resolvers have been time tested and proven, but the analog output limits the options that are available. The most popular use of resolvers is in permanent magnet brushless ac servo motors, military, aerospace applications.