Introduction

As an engineer in controls work, one bumps into problems with timing and task planning on a regular basis. It is never expected but there are times when the problems of computer timing seem to beg to be considered. Usually, the problem is simple and only takes a few minutes of our day. Other times, the problems of tasks being shared and I/O read are a problem worth giving some time to.

The M.T.A.
The Kingston Trio
These are the times that try men's souls
In the course of our nation’s history the people of Boston have rallied bravely whenever the rights of men have been threatened
Today a new crisis has arisen
The Metropolitan Transit Authority, better known as the M.T.A.
Is attempting to levy a burdensome tax on the population in the form of a subway fare increase
Citizens, hear me out, this could happen to you!
Well, let me tell you of the story of a man named Charlie
On a tragic and fateful day
He put ten cents in his pocket, kissed his wife and family
Went to ride on the MTA
Well, did he ever return?
No he never returned and his fate is still unlearned (what a pity)
He may ride forever 'neath the streets of Boston
He's the man who never returned
Charlie handed in his dime at the Kendall Square station
And he changed for Jamaica Plain
When he got there the conductor told him, "one more nickel"
Charlie couldn't get off of that train!
But did he ever return?
No he never returned and his fate is still unlearned (poor old Charlie)
He may ride forever 'neath the streets of Boston
He's the man who never returned
Now, all night long Charlie rides through the station
Crying, "what will become of me?
How can I afford to see my sister in Chelsea
Or my cousin in Roxbury?"
But did he ever return?
No he never returned and his fate is still unlearned (shame and scandal)
He may ride forever 'neath the streets of Boston
He's the man who never returned
Charlie's wife goes down to the Scollay Square station
Every day at quarter past two
And through the open window she hands Charlie a sandwich
As the train comes rumbling through!
But did he ever return?
No he never returned and his fate is still unlearned (he may ride forever)
He may ride forever 'neath the streets of Boston
He's the man who never returned
Pick it Davey
Kinda hurts my fingers
Now, you citizens of Boston, don't you think it's a scandal
How the people have to pay and pay?
Fight the fare increase, vote for George O'Brian
Get poor Charlie off the MTA!
Or else he'll never return
No he'll never return and his fate is still unlearned (just like Paul Revere)
He may ride forever 'neath the streets of Boston
He's the man who never returned
He's the man who never returned
He's the man who never returned
Et tu, Charlie?
Source: LyricFind

One simple problem encountered some years ago had to do with a high speed counter card. The card was purchased to count the pulses from a pulse output flow meter dealing with flow to a mixer. The flow was to turn off within .1 gallon of total flow from the liquid’s feed line. This was not happening. Every time the ingredient was called, the accuracy was off by .7 or .8 gallon, either plus or minus and never predictable. The time spent on this was of great concern because there was an obvious waste of each batch that had to be scrapped since the liquid was so far out of specification. It was also pointed out that the original flow controller was able to fill the batch to within .05 gallon accuracy repeatedly.

Everything looked to be in order. The pulses were coming into the high-speed card accurately and the PLC was scanning the card often enough to read and turn on an output every 10 ms or so. What was the problem?

Not until a program was written to show the delay between the update from the high-speed counter card and the CPU did the problem become obvious. The high-speed counter card was updating but not updating the count to the CPU more than about every .75 seconds, enough to cause the .7 or .8 gallon error. The program, while simple to write, caused concern for the engineer since the card was obviously flawed and needed to be replaced. The question became one of was this for this card alone or was the problem with all cards of this design. That question was answered by calling the manufacturer and asking that they put the card on a bench to test for the same problem. When this occurred, it was found that the problem was across the board and the card was essentially not able to control the flow of liquid accurately enough for the fill. Another solution would need to be found. Later, a solution was found that was accurate enough and the problem was solved. But the manufacturer was embarrassed by this obvious flaw in the design of his I/O card and the problem was only exposed by writing a program to show the delay between updates of the pulsed input to the cpu.

The solution found was to create a frequency divider for the pulsed input from the flow controller and read the inputs into the cpu directly at a slower rate that was able to be updated by the discrete I/O card on the cpu.

Both A-B and Siemens have methods of solving problems such as the one above. The manufacturer was not one of these two but rather another PLC vendor from a few years ago. Both have aggressive methods to deal with timing issues.
Both use a method to read I/O on an immediate basis rather than from the scan’s own I/O update. If necessary, they allow an instruction to have an immediate read from an input or an immediate write to an output. This is necessary for the processor to guarantee some of the simple high-speed problems of a process. The rest of the chapter outlines some of the more sophisticated methods for problem solving of timing issues and diagnosis of problems when they arise. In Chapter 19, it will be obvious that a scheduled program is needed to help in providing a best possible performance from the PID loops being programmed.

**Allen-Bradley - Managing Tasks**

The default RSLogix 5000 project provides a single task for all logic. While this is sufficient for many applications, some need more than a single task to perform the program well.

A Logix5000 controller supports multiple tasks to schedule and prioritize the execution of your programs based on specific criteria. This balances the processing time of the controller.

- The controller executes only one task at one time
- A different task can interrupt a task that is executing and take control
- In any given task, only one program executes at one time

A Logix5000 controller supports three types of tasks:

<table>
<thead>
<tr>
<th>If you want to execute a section of your logic</th>
<th>Then use this type of task</th>
<th>Description</th>
</tr>
</thead>
</table>
| All the time                                  | Continuous Task           | The continuous task runs in the background. Any CPU time not allocated to other operations (such as motion, communication, and periodic or event tasks) is used to execute the programs within the continuous task.  
  - The continuous task runs all the time. When the continuous task completes a full scan, it restarts immediately  
  - A project does not require a continuous task. If used, there can be only one continuous task |
| At a constant period (example, every 100 ms)   | Periodic Task             | A periodic task performs a function at a specific period. Whenever the time for the periodic task expires, the periodic task:  
  - interrupts any lower priority tasks  
  - executes one time  
  - returns control to where the previous task left off  
  You can configure the time period from 0.1 ms to 2000 s |
| Multiple times within the scan of your other logic | Event Task               | An event task performs a function only when a specific event (trigger) occurs. Whenever the trigger for the event task occurs, the event task:  
  - interrupts any lower priority tasks  
  - executes one time  
  - returns control to where the previous task left off  
  The trigger can be a:  
  - change of a digital input  
  - new sample of analog data  
  - certain motion operations  
  - consumed tag  
  - EVENT instruction |
Table 18-1
Examples of types of tasks and a suggested method for dealing with them:

<table>
<thead>
<tr>
<th>For this example situation</th>
<th>Use this type of task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill a tank to its maximum level and then open a drain valve</td>
<td>Continuous task</td>
</tr>
<tr>
<td>Collect and process system parameters and send them to a display</td>
<td>Continuous task</td>
</tr>
<tr>
<td>Complete step 3 in a control sequence – reposition the bin diverter</td>
<td>Continuous task</td>
</tr>
<tr>
<td>Your system must check the position of a field arm each 0.1 s and calculate the average</td>
<td>Periodic task</td>
</tr>
<tr>
<td>rate of change in its position. This is used to determine braking pressure</td>
<td></td>
</tr>
<tr>
<td>Read the thickness of a paper roll every 20 ms</td>
<td>Periodic task</td>
</tr>
<tr>
<td>A packaging line glues boxes closed. When a box arrives at the gluing position, the</td>
<td>Event task</td>
</tr>
<tr>
<td>controller must immediately execute the gluing routine</td>
<td></td>
</tr>
<tr>
<td>In a high-speed assembly operation, an optical sensor detects a certain type of reject.</td>
<td>Event task</td>
</tr>
<tr>
<td>When the sensor detects a reject, the machine must immediately divert the reject</td>
<td></td>
</tr>
<tr>
<td>In an engine test stand, you want to capture and archive each analog data immediately</td>
<td>Event task</td>
</tr>
<tr>
<td>after each sample of data</td>
<td></td>
</tr>
<tr>
<td>Immediately after receiving new production data, load the data into the station</td>
<td>Event task</td>
</tr>
<tr>
<td>In a line that packages candy bars, you have to make sure that the perforation occurs in</td>
<td>Event task</td>
</tr>
<tr>
<td>the correct location on each bar. Each time the registration sensor detects the</td>
<td></td>
</tr>
<tr>
<td>registration mark, check the accuracy of an axis and perform any required adjustment</td>
<td></td>
</tr>
<tr>
<td>A gluing station must adjust the amount of glue it applies to compensate for changes in</td>
<td>Event task</td>
</tr>
<tr>
<td>the speed of the axis. After the motion planner executes, check the command speed of an</td>
<td></td>
</tr>
<tr>
<td>axis and vary the amount of glue, if needed.</td>
<td></td>
</tr>
<tr>
<td>In a production line, if any of the programs detect an unsafe condition the entire line</td>
<td>Event task</td>
</tr>
<tr>
<td>must shut down. The shutdown procedure is the same regardless of the unsafe condition</td>
<td></td>
</tr>
</tbody>
</table>

Table 18-2

To assign a priority to a task, use these guidelines:

<table>
<thead>
<tr>
<th>If you want</th>
<th>Then</th>
<th>Notes</th>
</tr>
</thead>
</table>
| This task to interrupt another task | Assign a priority number that is less than (higher priority) the priority number of the other task | • A higher priority task interrupts all lower priority tasks  
• A higher priority task can interrupt a lower priority task multiple times |
| Another task to interrupt this task | Assign a priority number that is greater than (lower priority) the priority number of the other task |                                   |
| This task to share controller time with another task | Assign the same priority number to both tasks | The controller switches back and forth between each task and executes each one for 1 ms |

Table 18-3
Additional considerations

As you estimate the execution interrupts for a task, consider the following:

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion planner</td>
<td>The motion planner interrupts all user tasks, regardless of their priority&lt;br&gt;<strong>•</strong> The number of axes and coarse update period for the motion group effect how long and how often the motion planner executes&lt;br&gt;<strong>•</strong> If the motion planner is executing when a task is triggered, the task waits until the motion planner is done&lt;br&gt;<strong>•</strong> If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute</td>
</tr>
<tr>
<td>I/O Task</td>
<td>CompactLogix, FlexLogix and DriveLogix controllers use a dedicated periodic task to process I/O data. This I/O task:&lt;br&gt;<strong>•</strong> Does not show up in the Tasks folder of the controller&lt;br&gt;<strong>•</strong> Does not count toward the task limits for the controller&lt;br&gt;<strong>•</strong> Operates at priority 8&lt;br&gt;<strong>•</strong> Executes at the fastest RPI you have scheduled for the system&lt;br&gt;<strong>•</strong> Executes for as long as it takes to scan the configured I/O modules&lt;br&gt;As you assign priorities to your tasks, consider the I/O task&lt;br&gt;<strong>If you want a task to Interrupt or delay I/O processing</strong>&lt;br&gt;Then assign one of these priorities 1...5&lt;br&gt;<strong>Share controller time with I/O processing</strong>&lt;br&gt;6&lt;br&gt;<strong>Let I/O processing interrupt or delay the task</strong>&lt;br&gt;7...15</td>
</tr>
<tr>
<td>System overhead</td>
<td>System overhead is the time that the controller spends on unscheduled communication&lt;br&gt;<strong>•</strong> Unscheduled communication is any communication that you do not configure through the I/O configuration folder of the project, such as Message (MSG) instructions and communication with HMIs or workstations.&lt;br&gt;<strong>•</strong> System overhead interrupts only the continuous task&lt;br&gt;<strong>•</strong> The system overhead time slice specifies the percentage of time (excluding the time for periodic or event tasks) that the controller devotes to unscheduled communication.&lt;br&gt;<strong>•</strong> The controller performs unscheduled communication for up to 1 ms at a time and then resumes the continuous task.</td>
</tr>
<tr>
<td>Continuous Task</td>
<td>You do not assign a priority to the continuous task. It always runs at the lowest priority. All other tasks interrupt the continuous task.</td>
</tr>
</tbody>
</table>

Table 18-4

Leave enough time for unscheduled communication. Unscheduled communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that the scan times and execution intervals leave enough time for unscheduled communication. Use these methods to plan enough unscheduled communication time.

1. Verify that the execution time of a highest priority task is significantly less than its specified period.

2. Verify that the total execution time of all tasks is significantly less than the period of the lowest priority tasks.
For example, the following is true in this configuration.

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>Execution Time</th>
<th>Period Specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Higher</td>
<td>20 ms</td>
<td>80 ms</td>
</tr>
<tr>
<td>2</td>
<td>Lower</td>
<td>30 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total execution time: 50 ms</td>
<td></td>
</tr>
</tbody>
</table>

Table 18-5

The execution time of the highest priority task (Task 1) is significantly less than its specified period (20 ms is less than 80 ms).

The total execution time of all tasks is significantly less than the specified period of the lowest priority task (50 ms is less than 100 ms).

This generally leaves enough time for unscheduled communication. One should adjust the period of the task as needed to get the best trade-off between execution of the logic and servicing unscheduled communication. If the project has a continuous task, unscheduled communication occurs as a percentage of controller time (excluding the time for periodic or event tasks).

An overlap is a condition where a task (periodic or event) is triggered while the task is still executing from the previous trigger. If an overlap occurs, the controller disregards the trigger that caused the overlap. In other words, you may miss an important execution of the task.

The example below shows how this can happen:

At circle 1, the task trigger occurs. The task executes.

At circle 2, the task trigger occurs a second time. The task executes again.

At circle 3, the task trigger occurs a third time. The task executes a third time.

At circle 4, the task from the third trigger is still executing and the trigger for the fourth trigger does not restart the task. The trigger is ignored.

If this occurs, and the task is periodic, increase the period of the task. If the task is an event task, adjust the configuration of the system to trigger the task less frequently.
Manually check for overlaps using the following steps:

1. **In the Controller Organizer**, right-click **Main Task** and choose **Properties**.

   ![Diagram of Controller Organizer](image)

   *The Task Properties dialog box appears.*

2. Click the **Monitor** tab.

   ![Task Properties - Task](image)

   Number of overlaps since the counter was last reset: 0

3. Click **OK**.

   **Fig. 18-2**

Module Input Data State Change Trigger:

To trigger an event task based on data from an input module, use the module Input Data State Change trigger shown below:

![Task Properties - Task](image)

**Fig. 18-3**
Some A-B I/O has the ability to send the I/O status to multiple processors. This is referred to as an I/O Module Trigger to an event Task. We will not discuss this topic further here since the I/O is not present in the lab and we do not need this capability to process any information here. However, do remember that the more sophisticated processors have this capability and it should be considered when planning a project.

**Siemens**

Siemens, like Allen-Bradley, has a sequence of events that occur when the processor starts up and then services the I/O and solves logic during the scan of the processor. It is roughly described in the following diagram:

**Operating modes of the CPU**

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, and you can download a project.
- In STARTUP mode, the CPU executes any startup logic (if present). Interrupt events are not processed during the startup mode.
- In RUN mode, the scan cycle is executed repeatedly. Interrupt events can occur and be processed at any point within the program cycle phase. Some parts of a project can be downloaded in RUN mode.

The CPU supports the warm restart method for entering the RUN mode. Warm restart does not include a memory reset, but a memory reset can be commanded from the programming software.
A memory reset clears all work memory, clears retentive and non-retentive memory areas, and copies load memory to work memory. A memory reset does not clear the diagnostics buffer or the permanently saved IP address. All non-retentive system and user data are initialized at warm restart.

Processing the scan cycle in RUN mode:

For each scan cycle, the CPU writes the outputs, reads the inputs, executes the user program, updates communication modules, and responds to user interrupt events and communication requests. Communication requests are handled periodically throughout the scan.

These actions (except for user interrupt events) are serviced regularly and in sequential order. User interrupt events which are enabled are serviced according to priority in the order in which they occur.

The system guarantees that the scan cycle will be completed in a time period called the maximum cycle time; otherwise a time error event is generated.

- Each scan cycle begins by retrieving the current values of the digital and analog outputs from the process image and then writing them to the physical outputs of the CPU, SB, and SM modules configured for automatic I/O update (default configuration). When a physical output is accessed by an instruction, both the output process image and the physical output itself are updated.

- The scan cycle continues by reading the current values of the digital and analog inputs from the CPU, SB, and SMs configured for automatic I/O update (default configuration), and then writing these values to the process image. When a physical input is accessed by an instruction, the value of the physical input is accessed by the instruction, but the input process image is not updated.

- After reading the inputs, the user program is executed from the first instruction through the end instruction. This includes all the program cycle OBs plus all their associated FCs and FBs. The program cycle OBs are executed in order according to the OB number with the lowest OB number executing first.

Communications processing occurs periodically throughout the scan, possibly interrupting user program execution. Self-diagnostic checks include periodic checks of the system and the I/O module status checks. Interrupts can occur during any part of the scan cycle, and are event-driven. When an event occurs, the CPU interrupts the scan cycle and calls the OB that was configured to process that event. After the OB finishes processing the event, the CPU resumes execution of the user program at the point of interruption.

**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. Specific events in the CPU trigger the execution of an organization block. OBs cannot call each other or be called from an FC or FB. Only a start event, such as a diagnostic interrupt or a time interval, can start the execution of an OB. The CPU handles OBs according to their respective priority classes, with higher priority OBs executed before lower priority OBs. The lowest priority
class is 1 (for the main program cycle), and the highest priority class is 26 (for the time-error interrupts).

OBs control the following operations:

1 - Program cycle OBs execute cyclically while the CPU is in RUN mode. The main block of the program is a program cycle OB. This is where you place the instructions that control your program and where you call additional user blocks. Multiple program cycle OBs are allowed and are executed in numerical order. OB 1 is the default. Other program cycle OBs must be identified as OB 200 or greater.

2 - Startup OBs execute one time when the operating mode of the CPU changes from STOP to RUN, including powering up in the RUN mode and in commanded STOP-to-RUN transitions. After completion, the main "Program cycle" OB will begin executing. Multiple startup OBs are allowed. OB 100 is the default. Others must be OB 200 or greater.

3 - Cyclic interrupt OBs execute at a specified interval. A cyclic interrupt OB will interrupt cyclic program execution at user defined intervals, such as every 2 seconds. You can configure up to a total of 4 for both the time-delay and cyclic events at any given time, with one OB allowed for each configured time-delay and cyclic event. The OB must be OB 200 or greater.

4 - Hardware interrupt OBs execute when the relevant hardware event occurs, including rising and falling edges on built-in digital inputs and HSC events. A hardware interrupt OB will interrupt normal cyclic program execution in reaction to a signal from a hardware event. You define the events in the properties of the hardware configuration. One OB is allowed for each configured hardware event. The OB must be OB 200 or greater.

5 - A time error interrupt OB executes when either the maximum cycle time is exceeded or a time error event occurs. The OB for processing the time error interrupt is OB 80. If triggered, it executes, interrupting normal cyclic program execution or any other event OB. The events that trigger the time error interrupt and the reaction of the CPU to those events are described below:

- Exceeding the maximum cycle time: You configure the maximum cycle time in the properties of the CPU. If OB 80 does not exist, the reaction of the CPU for exceeding the maximum time is to change to STOP.
- Time errors: If OB 80 does not exist, the reaction of the CPU is to stay in RUN. Time errors occur if the time of day event is missed or repeated, the queue overflows, or an event OB (time delay event, time of day event, or cyclic interrupt) starts before the CPU finishes the execution of the first. The occurrence of either of these events generates a diagnostic buffer entry describing the event. The diagnostic buffer entry is generated regardless of the existence of OB 80.

6 - Diagnostic error interrupt OBs execute when a diagnostic error is detected and reported. A diagnostic OB interrupts the normal cyclic program execution if a diagnostics-capable module recognizes an error (if the diagnostic error interrupt has been enabled for the module). OB 82 is the only OB number supported for the diagnostic error event. You can include an STP instruction (put CPU in STOP mode) inside your OB 82 if you desire your CPU to enter STOP mode upon receiving this type of error. If there is no diagnostic OB in the program, the CPU ignores the error (stays in RUN).
**Event execution priorities and queuing**

The CPU processing is controlled by events. An event triggers an interrupt OB to be executed. You can specify the interrupt OB for an event during the creation of the block, during the device configuration, or with an ATTACH or DETACH instruction. Some events happen on a regular basis like the program cycle or cyclic events. Other events happen only a single time, like the startup event and time delay events. Some events happen when there is a change triggered by the hardware, such as an edge event on an input point or a high speed counter event. There are also events like the diagnostic error and time error event which only happen when there is an error.

The event priorities and queues are used to determine the processing order for the event interrupt OBs.

The program cycle event happens once during each program cycle (or scan). During the program cycle, the CPU writes the outputs, reads the inputs and executes program cycle OBs. The program cycle event is required and is always enabled. You may have no program cycle OBs, or you may have multiple OBs selected for the program cycle event. After the program cycle event is triggered, the lowest numbered program cycle OB (usually OB 1) is executed. The other program cycle OBs are executed sequentially (in numerical order) within the program cycle.

The cyclic interrupt events allow you to configure the execution of an interrupt OB at a configured scan time. The initial scan time is configured when the OB is created and selected to be a cyclic interrupt OB. A cyclic event will interrupt the program cycle and execute the cyclic interrupt OB (the cyclic event is at a higher priority class than the program cycle event).

Only one cyclic interrupt OB can be attached to a cyclic event. Each cyclic event can be assigned a phase shift so that the execution of cyclic interrupts with the same scan time can be offset from one another by the phase shift amount. The default phase shift is 0. To change the initial phase shift, or to change the initial scan time for a cyclic event, right click on the cyclic interrupt OB in the project tree, click "Properties", then click "Cyclic interrupt", and enter the new initial values. You can also query and change the scan time and the phase shift from your program using the Query cyclic interrupt (QRY_CINT) and Set cyclic interrupt (SET_CINT) instructions. Scan time and phase shift values set by the SET_CINT instruction do not persist through a power cycle or a transition to STOP mode; scan time and phase shift values will return to the initial values following a power cycle or a transition to STOP. The CPU supports a total of four cyclic and time-delay interrupt events.

The startup event happens one time on a STOP to RUN transition and causes the startup OBs to be executed. Multiple OBs can be selected for the startup event. The startup OBs are executed in numerical order.

The time delay interrupt events allow you to configure the execution of an interrupt OB after a specified delay time has expired. The delay time is specified with the SRT_DINT instruction. The time delay events will interrupt the program cycle to execute the time delay interrupt OB. Only one time delay interrupt OB can be attached to a time delay event. The CPU supports four time delay events.

The hardware interrupt events are triggered by a change in the hardware, such as a rising or falling edge on an input point, or a HSC (High Speed Counter) event. There can be one interrupt OB selected for each hardware interrupt event. The hardware events are enabled in Device configuration. The OBs are specified for the event in the Device configuration or with an ATTACH instruction in the user program. The CPU supports several hardware interrupt events. The exact events are based on the CPU model and the number of input points.
The time and diagnostic error interrupt events are triggered when the CPU detects an error. These events are at a higher priority class than the other interrupt events and can interrupt the execution of the time delay, cyclic and hardware interrupt events. One interrupt OB can be specified for each of the time error and diagnostic error interrupt events. The time and diagnostic error interrupt events are triggered when the CPU detects an error. These events are at a higher priority class than the other interrupt events and can interrupt the execution of the time delay, cyclic and hardware interrupt events. One interrupt OB can be specified for each of the time error and diagnostic error interrupt events.

**Understanding event execution priorities and queuing**

The number of pending (queued) events from a single source is limited, using a different queue for each event type. Upon reaching the limit of pending events for a given event type, the next event is lost. Refer to the following section on "Understanding time error events" for more information regarding queue overflows. Each CPU event has an associated priority. You cannot change the priority of an OB. In general, events are serviced in order of priority (highest priority first). Events of the same priority are serviced on a "first-come, first-served" basis.

**OB Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>OB number</th>
<th>Quantity allowed</th>
<th>Start Event</th>
<th>OB Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program cycle</td>
<td>OB1, OB 200 to OB 65535</td>
<td>1 program cycle event</td>
<td>• Startup OB ends</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple OBs allowed</td>
<td>• Last program cycle OB ends</td>
<td></td>
</tr>
<tr>
<td>Startup</td>
<td>OB 100, OB 200 to OB 65535</td>
<td>1 startup event</td>
<td>STOP to RUN transition</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple OBs allowed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>OB 200 to OB65535</td>
<td>Up to 4 time events</td>
<td>Time-delay OB event is scheduled Cyclic OB event is scheduled</td>
<td>3 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 OB per event</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>OB 200 to OB 65535</td>
<td>Up to 50 process events</td>
<td>Edges: Rising edge events – 16 max</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 OB per event</td>
<td>Falling edge events – 16 max</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For HSC: CV=PV: 6 max</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Direction changed: 6 max</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>External reset: 6 max</td>
<td></td>
</tr>
<tr>
<td>Diagnostic error</td>
<td>OB 82</td>
<td>1 event (only if OB 82 was loaded)</td>
<td>Module transmits an error</td>
<td>9</td>
</tr>
<tr>
<td>Time error</td>
<td>OB 80</td>
<td>1 event (only if OB 80 was loaded)</td>
<td>Maximum cycle time was exceeded A second time interrupt (cyclic or time-delay) started before the CPU had finished execution of the first interrupt</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 18-6
Interrupt latency

The interrupt event latency (the time from notification of the CPU that an event has occurred until the CPU begins execution of the first instruction in the OB that services the event) is approximately 175 μsec, provided that a program cycle OB is the only event service routine active at the time of the interrupt event.

Understanding time error events

The occurrence of any of several different time error conditions results in a time error event. The following time errors are supported:

- Maximum cycle time exceeded
- Requested OB cannot be started
- Queue overflow occurred

The maximum cycle time exceeded condition results if the program cycle does not complete within the specified maximum scan cycle time. See the section on "Monitoring the cycle time in the S7-1200 System Manual" (Page 80) for more information regarding the maximum cycle time condition, how to configure the maximum scan cycle time, and how to reset the cycle timer.

The requested OB cannot be started condition results if an OB is requested by a cyclic interrupt, a time-delay interrupt, or a time-of-day interrupt, but the requested OB is already being executed. The queue overflow occurred condition results if the interrupts are occurring faster than they can be processed. The number of pending (queued) events is limited using a different queue for each event type. If an event occurs when the corresponding queue is full, a time error event is generated. All time error events trigger the execution of OB 80 if it exists. If an OB 80 is not included in the user program, then the device configuration of the CPU determines the CPU reaction to the time error:

- The default configuration for time errors, such as starting a second cyclic interrupt before the CPU has finished the execution of the first, is for the CPU to stay in RUN.
- The default configuration for exceeding the maximum time is for the CPU to change to STOP.

You can use the RE_TRIGR instruction to reset the maximum cycle time. However, if two "maximum cycle time exceeded" conditions occur within the same program cycle without resetting the cycle timer, then the CPU transitions to STOP, regardless of whether OB 80 exists. OB 80 includes startup information that helps you determine which event and OB generated the time error. You can program instructions inside OB 80 to examine these startup values and to take appropriate action.

Monitoring the cycle time

The cycle time is the time that the CPU operating system requires to execute the cyclic phase of the RUN mode. The CPU provides two methods of monitoring the cycle time:

- Maximum scan cycle time
- Fixed minimum scan cycle time
Scan cycle monitoring begins after the startup event is complete. Configuration for this feature appears under the "Device Configuration" for the CPU under "Cycle time". The CPU always monitors the scan cycle and reacts if the maximum scan cycle time is exceeded. If the configured maximum scan cycle time is exceeded, an error is generated and is handled one of two ways:

- If the user program does not include an OB 80, then the CPU generates an error and goes to STOP. (You can change the configuration of the CPU to ignore this time error and stay in RUN. The default configuration is for the CPU to go to STOP.)
- If the user program includes an OB 80, then the CPU executes OB 80

The RE_TRIGR instruction (Re-trigger cycle time monitoring) allows you to reset the timer that measures the cycle time. However, this instruction only functions if executed in a program cycle OB; the RE_TRIGR instruction is ignored if executed in OB 80. If the maximum scan cycle time is exceeded twice within the same program cycle with no RE_TRIGR instruction execution between the two, then the CPU transitions to STOP immediately. The use of repeated executions of the RE_TRIGR instruction can create an endless loop or a very long scan.

Typically, the scan cycle executes as fast as it can be executed and the next scan cycle begins as soon as the current one completes. Depending upon the user program and communication tasks, the time period for a scan cycle can vary from scan to scan. To eliminate this variation, the CPU supports an optional fixed minimum scan cycle time (also called fixed scan cycle). When this optional feature is enabled and a fixed minimum scan cycle time is provided in ms, the CPU will maintain the minimum cycle time within 1 ms for the completion of each CPU scan.

In the event that the CPU completes the normal scan cycle in less time than the specified minimum cycle time, the CPU spends the additional time of the scan cycle performing runtime diagnostics and/or processing communication requests. In this way the CPU always takes a fixed amount of time to complete a scan cycle.

In the event that the CPU does not complete the scan cycle in the specified minimum cycle time, the CPU completes the scan normally (including communication processing) and does not create any system reaction as a result of exceeding the minimum scan time.

Recording of measured values with the trace function:

The trace and logic analyzer function can be called in the device folder in the project navigator under the name “Traces”. You record device tags and evaluate the recordings with the trace and logic analyzer function. Tags are for example drive parameters or system and user tags of a CPU. The maximum recording duration is limited by the memory size. How much memory is available for the recording depends on the hardware used.
We will begin a review of the configuration of the PLC at start-up of a project and include the most important tasks to accomplish:

Getting Started with the Hardware – Configuration
The Hardware and Network Editor
One Editor – Three Different Views

Fig. 18-7

Network View

Fig. 18-8

Device View

Fig. 18-9
The TIA Portal has a single editor for performing tasks associated with configuring both the automation system hardware and any associated networks in the project. The Hardware and Network Editor offers three different views, each one offering access to perform specific tasks.

**Topology View** – allows configuring and editing of Ethernet network topology.

**Network View** – allows editing of the physical and logical network interconnections between devices, adding new networks, and assigning a device to an existing network in the project.

**Device View** - provides a graphical representation of the station selected and provides access to the Properties of a device for configuration.

**Topology View**

![Topology View](image)

The Topology View is used to both edit and display the Profinet topology of the project. The editor can also be used to do an offline and online comparison, and the offline can be adopted to match the online.

In a PLC, the overall program can be organized into several different execution routines:

- Main Program – the major portion of the program that is executed every scan.
- Initialization Routines
- Error Handling Routines
- Time-based Routines
- Process-initiated Routines
- Hardware Fault Routines

In addition, there are typically many other subroutines that perform specific functions, and are called by these routines.
Overview of the Organization Blocks

![Diagram of Organization Blocks](image)

**Startup:**

A startup program is carried out before the cyclic program execution after a power recovery or a change of operating mode (through the CPU’s mode selector or by the PG). OB 100 is available in every PLC model, whereas 101 and 102 are only available in the S7-400. In these blocks you can, for example, preset the communications connections or execute initialization routines.

**Cyclic Program Execution:**

The program that is to be continuously executed is stored in the Organization Block OB1. After the user program has been completely executed in OB1, a new cycle begins with the updating of the process images and the processing of the first statement in OB1. The scan cycle time and the response time of the system is a result of these operations. The response time is the total of the executing time of the CPU’s operating system and the time it takes to execute all of the user program. The response time, that is, how quickly an output can be switched dependent on an input signal, is equal to scan cycle time x 2. In the S7-1500, a program can contain up to 100 cyclic execution OBs and 100 startup OBs. The additional OBs are added at or above address 123, and they are executed in numerical order.

**Periodic Program Execution:**

With periodic program execution, you can interrupt the cyclic program execution at fixed intervals. With cyclic interrupts, an OB 30 to OB 38 organization block is executed after a preset timing code has run out, every 100 ms for example. Control-loop blocks with their sampling interval are called, for example, in these blocks. With time-of-day interrupts, an OB is executed at a specific time, for example every day at 17:00 hours (5:00 p.m.), to save the data.
**Interrupt-driven Program Execution:**

The hardware interrupt can be used to respond quickly to a process event. After the event occurs, the cycle is immediately interrupted and an interrupt program is carried out. The time-delay interrupt responds after a delayed time period to a process event. With the error OBs you can determine how the system is to behave, for example, if the backup battery fails.

**OB Execution and Priorities**

![Diagram](image)

Priorities are assigned to the OBs ranging from 0 to 28 – the higher the priority number, the higher the relative importance of the OB. If the operating system calls an OB, the OB will be executed to completion unless interrupted by an OB with a higher priority. As shown in the slide, when the operating system calls another OB, it interrupts the cyclic program because OB1 has the lowest priority. Afterwards, OB1 resumes execution at the point of interruption.

If an OB with a higher priority than the one currently being executed is called, the lower priority OB is interrupted after the current statement has been completed. The higher priority OB is executed to completion, then the lower priority OB is resumed from its point of interruption. OBs of the same priority do not interrupt each other, but are started one after the other in the sequence they are recognized. In the S7-1200, 1500, and 400, priorities of OB’s can be manually assigned and changed, if desired.
An Article: Programming a PLC vs Industrial PC: Which is Best? How to decide between ladder logic and Visual Basic

By Mike Bacidore, editor in chief
Jun 29, 2017

A Control Design reader writes: I'm a controls designer at an OEM and need to program a new automated machine we will be selling as a product. It's an assembly machine with about 80 I/O points, mostly discrete using mostly pneumatic actuators and a few servo motors for a build, test and package application. The debate is PLC programming vs. VB programming. A programmer here, who has never seen a PLC or worked with automation, wants to use an industrial PC (IPC) and program it all in VB.net. I strongly want to use a PLC and program the 21-step sequence using ladder logic. For industrial applications, especially machine control, I think PLC programming is the way to go. Can you help me settle this debate? Which programming language and control should I use for this medium-size discrete machine application?

Answers

More than one answer
Shaun Derrick, senior applications support engineer, Triangle Research International: There is probably no one right answer, given the fairly limited information available. We believe that the best we can do is to give guidance to an approach to the answer under the circumstances. There are too many factors that have to be considered and the relative weights of each factor, given the specifics of the application and the backgrounds of the programmer, the machine users and the equipment maintenance crew.

Some of the dependencies are discussed here.

1. Programming: Which language is the programmer most familiar with? Which of these options is best is highly dependent on the programmer’s comfort and skill set.

2. Application needs: What is the scope of the application and intended use of the equipment? If the intent of the controls is purely to automate a process with limited user input and there is no need for an advanced interface or data collection/reporting, then a PLC is the best way to go. At the same time, an industrial PC could be a good option if the application and machine operation extend beyond simple process control. Some examples of why you might need the power of an industrial PC are:
   - intensive calculations need to be performed locally
   - the machine is linking in to other external systems, such as enterprise resource planning (ERP) or building automation system (BAS), for data logging, analysis and reporting
   - a high-end user interface needs to be layered on top of the process control locally at the machine and possibly remotely, as well.

3. Maintenance/troubleshooting: Is the programmer who writes the control program also the one to maintain and improve the software for the foreseeable future? If so, then the programmer should choose the programming language that he or she feels most comfortable with. If, on the other hand, someone else such as a shop-floor technician is required to take over the maintenance and, at some point, troubleshooting of the program code, then it is very important that the chosen
A programming language is in line with what a typical shop-floor tech is trained in or has exposure to. On most shop floors, it’s more likely that the technicians may know something about PLC ladder-logic programming but have little or no knowledge of Visual Basic.

4. **Cost**: Industrial PCs are relatively expensive compared to a wide range of choices of low-cost PLCs with about 80 I/Os.

5. **Stability of operating system**: If the machine is to run 24/7 unattended, then the designer should be concerned with whether an office-based operating system, such as Windows, is suitable for the task. How tolerant is the application to stoppages and resets?

In summary, we do not believe that there is one right answer, and the final right decision is a weighted consideration of all the above factors.

**Compromise**

*Don Fitchett, President, Business Industrial Network*: My advice to the controls designer when speaking to the computer programmer is to explain two very important points and differences—the end-user-friendly and the machine-safety points of using a PC vs. PLC/PAC.

As an OEM, the number-one objective should be customer satisfaction. With machine control design, it is understood. Safety, reliability and flexibility are rolled in to the customer-satisfaction requirements. Unlike with computer software design, a machine customer will expect the ability to access control code for troubleshooting and future minor modifications. Also, unlike computer software, the machine’s lifecycle will be many time longer than that of typical software.

The end user of an OEM’s equipment often varies from customer to customer. All will have electricians; most will have industrial engineers; but few will have full-time computer programmers on staff who are fluent in VB.net. So, the best control-system programming language by far is ladder logic. Downtime of a machine can cost the customer thousands to millions, so they cannot wait until they get a computer programmer to travel to their location. Remote support is an option, but that leads into higher safety risk, as I will touch on more.

Also, unlike most computer software, machines act in the real world where damage to man or machine can occur. The equipment the programming will be controlling affects great amounts of power, such as 480 V; speed, such as 3,2000 rpm; physical weights and pressures, such as tons; and other dangerous forces external to the programming control.

So, where safety is not even a concern with most software-program designs, you would use VB.net or C++, for example. With machine control, it is or should be the number-one concern. When using a PC, vs. a PLC, the PC has much greater risk of being compromised by malicious code. The PC at some point in the future could have other software installed on it or operating-system update issues that could not only jeopardize reliability, but also safety, to those who work around the machine, especially with the industrial sector’s rush to networking without all individuals being properly trained in network security. This great risk gets greater with some moving to the cloud/Internet and cyber warfare increasing exponentially.

So, in summary, offer your computer-programmer coworker a compromise—a process automation controller (PAC) using ladder logic as the primary programming language. Where a PLC is the safest, most reliable and easiest for the customer or machine end user to work with, a
PAC is also much better than an industrial PC. A PAC is an industrial PC but has a customized operating system, making it less vulnerable to malicious code than a PC is. And the customer cannot install unrelated software programs on it that might degrade its reliability and safety.

Gimme 3 points

William Nieves, product manager, Industrial Automation Group, Industrial Devices Sales, Panasonic of North America: When Mr. Dick Morley introduced the PLC in 1968, the industrial process was forever changed. Since then, the discussion about which platform is better is still ongoing. PLCs and industrial PCs (IPCs) have narrowed the gap that used to make them so different. Even so, many engineers are still choosing PLCs over IPCs. To answer this about using PLCs and IPCs in packaging applications, there are three points to consider:

1. Machine flexibility
2. Human-machine interaction

Machine flexibility: A packaging machine must be first and foremost flexible. It must be able to handle various products at different speeds following complex motion profiles. For instance, a servo axis on the roll of film is used to maintain tension in the process so the exact amount of film material can be used with minimal waste. Feeding the correct length of the film must keep aligned with other types of processes such filling, labeling and weight control. Here, the PLC shows robustness when processing multiple signals with minimum code, native math functions and fast response time. The PLC has no moving parts, so it can withstand harsh environments for millions of cycles. An IPC, however, requires the use of fans or some other cooling system to mitigate heat issues. They are susceptible to vibrations, demanding a level of care not seen in PLCs.

Part of the PLC’s flexibility comes from its ability to work with many types of devices at minimal cost, thus expanding its control capabilities and functionality. An IPC requires additional components, even for the simplest of functions. This includes discrete, analog or motion PCI I/O cards, networks cards and other types of signal processing peripheral devices. In contrast, the integration with a vision system is often used to validate the limitations of a PLC, thus encouraging the user to use an IPC. Here you should inquire whether the cost of the IPC and its peripheral devices are justifiable.
In addition, an IPC demands attention, care and expertise to maintain its operation. This is often tested during the downtime of the machine. Many process and control engineers have commented in the past about PC registry issues after updating a program or adding a PCI driver. When compared to an IPC, PLCs have reduced machine downtime, thanks in part to built-in functions like backup programs or early-failure-detection subroutines.

As for the PLC, the programming environment is clear, concise and consistent. For instance, the definition of a variable is done beyond the actual PLC program; this minimizes the number of lines required for your code to operate (Figure 1).

PLCs offer a graphical method for the development and control of the machine’s behavior. It is called ladder logic, and it is simple and amazingly effective. This is achieved by graphically mimicking an electrical circuit, emulating how a machine operates. However, PLCs have more to offer. Since the definition and implementation of the IEC 61131 standard, PLCs have shared a common logic and program foundation, regardless of the manufacturer. PLC programs that are structured under this standard can be debugged efficiently and effectively. On the contrary, IPCs do not follow a recognized standard, making it difficult to decrypt the actual logic defined by the programmer which makes the transfer of IP difficult in systems designed for long life or to be sold. The IEC 61131-3 standard promotes the use of five different programming languages in the PLC:

- instruction list (IL)
- functional block diagram (FBD)
- ladder logic (LL) for digital inputs and outputs logic
- structured text (ST) for advanced mathematical formulas, sequence of events (FOR or WHILE loops) and decision making (IF THEN ELSE, CASE) statements
- sequential function chart (SFC) for complex process iterations or branching.

PLCs also offer built-in PID functionality which provides accurate control of the film unwind mechanism by controlling the servo’s speed or start-stop routines, as in the case of the packing machine mentioned earlier. How to use a PLC’s PID strictly implies the proper definition of its inputs, such as sensor, thermocouples or encoder, and outputs, such as ON-OFF control or pulse with modulation.

The PLC’s PID executes cyclic tuning drills, finding optimal PID constants. Consequently, the controls engineer’s attention remains focused on the process and not the PLC program. On the contrary, an IPC programming platform, such as Visual Basic, demands a workaround to create the algorithm that defines the PID function. Such workarounds lack the industry exposure time that’s required to vet its efficiency and work out any bugs in the function. This adds delays during the testing and troubleshooting of the program and limits the precision of the algorithm to the experience of the programmer.

To deliver maximum performance while requiring minimum time, the PLC uses what is known as the reduced instruction set computer (RISC) structure. Typically, PLCs have 32-bit microprocessors with a clock frequency around 100 Mhz. Designed with the intent to be operating during most of its lifetime, a PLC requires less maintenance than an IPC. IPCs,
however, with complex hardware architecture, often require updates to the operating system, which in many occasions requires the program to be altered.

Another example is related with the use of stepper motors. The interaction between the IPC’s CPU and a PCI motion card will require advanced instructions or ActiveX apps (.Net platform) to operate properly. Consequently, a set of constraints will limit how the IPC’s program executes a motion profile, corrects high positioning deviations and controls interrupts for emergency stops or rapid speed changes. The PLC has evolved around motion, making its implementation much easier. Dedicated PLC motion cards provide friendly user interfaces, such as motion tables, to quickly build complex motion profiles. IPCs do offer similar solutions. The difference is that, with the PLC, the programmer can read and write over the PLC motion card’s shared memory. This allows for deeper access to hardware features, which helps with the debugging of motion issues. In addition, PLC motion cards offer hardware and software interrupts, alleviating headaches for machine builders who require the use of sensors to detect label edges, gaps, registration marks and more (Figure 2).

**Human-machine interaction:** IPCs can host SCADA programs. The issue is that SCADA programs come with higher price tags and utilize a large amount of resources to implement and run. SCADA programs also require the addition and integration of an OPC server. This adds a high level of complexity that may impact your project efficiency. However, more is less when thinking of effective human-machine interaction; this is the case when using PLCs coupled with an HMI or touchscreen. Connected through serial ports, the PLC and HMI can perform at the same level as that of a SCADA package. HMIs are simple to program and provide an intuitive
feeling when using them. The HMI itself is a self-contained piece of hardware that offloads the graphic requirements of the PLC. In this case, the user just needs to deal with the right PLC driver and incorporate memory addressing and/or tags. Limitations typical of HMI are the available widgets and the parts library. PLCs and HMIs are the on the front line of packaging machines, thanks to industrial protocols and PC-oriented hardware.

As for the case for processing information, this topic will shift favorably toward the IPC. IPCs can process vast amounts of data in a short amount of time, due to their processors, and IPCs have the capability to deliver data in any accessible network. Memory capacity of IPCs exceeds that of the PLC in several orders of magnitude. If an application is highly sensitive to data management, then the argument is in favor of IPCs. Nevertheless, PLCs have been making significant improvements in this arena. PLCs offer memory cards for data collection, high-end mathematical instructions for error detection, data encryption for safety data management, client-server broadcasting protocols, Web server functions and much more. All of this is accomplished with minimum hardware and programming effort.

**Product quality:** This topic has already been validated at this point. A flexible machine that can be operated efficiently, with minimal setup time and for a long period should produce a high-quality product. Although a true statement, it is not without its issues. It is good to remember that not all PLCs are created equal, and not all PLCs have the same performance record. Recall that an IPC has the capability to perform any industrial task it’s programmed to perform. The issue extends from the costs associated with the software and hardware required to perform the task.

In the end, PLCs are cheaper and easy to maintain but this does not ensure quality. Major PLC brands deserve our gratitude and trust, but not our loyalty. Your product quality depends of choosing the right hardware and software, technical support and resources to encourage creativity and positive ROI. PLCs have been used for so long in this field that process and controls engineers rely on this knowledge to deploy solutions in shortened timeframes. Currently IPCs lack such valuable resources.

**The best of neither world**

**Kenn Anderson, President, Nova Systems, Milwaukee, Wisconsin, member of Control System Integrators (CSIA):** In my opinion, neither of the options is ideal.

VB.Net has pros and cons.

Pros:
- appears to be the only tool in the programmer’s toolbox.

Cons:
- appears to be the only tool in the programmer’s toolbox
- difficult to debug; more startup time
- difficult to reload if CPU needs replacement
- almost impossible for a user to troubleshoot
- requires special I/O hardware
- difficult to access motion (EtherCAT)
- hardware will likely be more costly.

PLC with ladder editor has pros and cons, too.

Pros:
- economical hardware
- easy debug if you understand the code
- user access for troubleshooting
- quick download if processor dies
- standard off-the-shelf hardware
easy to integrate third-party evaluated safety (must for machinery these days).

Cons:
- ladder not the right editor for machinery sequence
- using the ladder editor, machinery sequences can lead to complicated, hard-to-understand code.

Our recommendation is a PLC with sequential function chart editor.

Pros:
- can implement PackML state machine for control
- widely accepted; easy-to-follow architecture; ISA 88 standard
- easy-to-debug state machine using step mode
- uses off-the-shelf hardware
- runs fast (only one state is executing at a time)
- easy to incorporate motion blocks
- easy user access for troubleshooting
- easy to integrate third-party evaluated safety (must for machinery these days).

Cons:
- have to learn new editor but able to use VB-like structured text in state blocks.

An experienced system integrator (SI) can help in this process, but it will add cost. However, the project will get to market more quickly and the in-house controls engineers can learn from the SI at an accelerated rate to be able to support the program after the first one or two machines are out and then be mostly self-sufficient.

Compete on price

**Don Pham, product manager, IDEC:** Either programming language will work, but an IPC will almost always be too expensive for an 80-I/O-point application, as compared to a PLC. The exception would be if there are extraordinary requirements for features typically found in an IPC and not in a PLC, which doesn’t seem to be the case here. The cost of automation components is particularly important for OEM machine builders, who often must compete on price, so cost is a big factor and will probably override any perceived programming issues.

In terms of programming, modern PLCs have many features included with their free or very low-cost PC-based PLC programming software packages to simplify ladder-logic programming, making it simpler to program than VB for machine control applications. VB is a general-purpose programming environment, whereas ladder logic is specifically intended for machine control, making it a better fit in this application.

Finally, very few OEM customers would object to ladder logic, whereas many might have an issue with VB. In the worst case, this could cause OEM customers to reject the bid upfront. In other cases, the OEM customer might accept the VB programming, but then expect the OEM to provide free support due to a lack of familiarity.

Who’s running the machine?

**Max Gorbunov, control and automation engineer III, performance materials, Laird:** I understand where your programmer is coming from. To a person trained in object-oriented programming, ladder logic can seem very daunting and impenetrable. Thinking in object-oriented programming is a tough habit to break. It took me several years and several PLC software suites before I found my groove with ladder logic and saw it as a good alternative to PC-based systems. PCs absolutely have their place in automation control, but they need to be used with care. There are a number of big advantages to using a PC; it is much more computationally powerful, compatible with a larger number of interfaces and protocols and is often times less expensive. Microsoft’s .NET is a very powerful platform to build on. It can be made compatible with any number of network protocols to communicate with devices such as servo controllers. It has libraries available for advanced industrial applications such as true parallel thread routines,
machine vision, point cloud analysis or HMI bells and whistles such as animations or remote support. However, there are some shortcomings to take into account when designing a system. The first is reliable timing. A running PLC will cycle through every ladder rung in a program at a very consistent speed. This makes sequence timing very reliable. A PC on the other hand will have an operating system with dozens of services running in the background. These services will ultimately compete with your control application for system resources. If you were to write a program in .NET which required the use of a 500-ms delay, for example, you could see the real delay time sway 50-75 ms in either direction, while on a PLC you would see much more stable results. This becomes especially apparent on multi-threaded applications. To avoid any timing errors, it is a good practice to not rely on timing alone and instead make sequencing more event-and feedback-driven. Another big drawback to PC-based systems is system startup. When you power up a PLC-based system, you can have a dedicated run once startup routine and then your system boots to its main routine within milliseconds. A PC system startup is a much longer process. The system has to boot into the operating system and then launch the control application. In those few minutes, you will have no control over any of your I/O. A good practice would be to interlock your software application with the safety controller to make sure the system boots safely.

Ultimately, the decision should come down to the person who is going to develop and support the program. Learning ladder logic and learning to use a PLC software package for a particular PLC brand from scratch takes a lot of time and work. If your programmer feels they are better equipped to use a PC-based solution and have no need to learn ladder logic, this may be the best way to go. The hardware itself would be plenty capable of running the machine provided that all of the proper precautions are taken.

**Implementation, support, acceptance**

_Branch Bissette, control designer and programmer, Solvere, Belmont, North Carolina, member of Control System Integrators Association (CSIA):_ A larger portion of our business is to design and build control systems for OEM machinery builders. The majority of these systems are similar in size and scope to what you are asking about. We almost exclusively use PLCs for the core portion of the control system for reasons I will outline below. While we may utilize PCs in conjunction with PLCs for some systems, the PCs are mainly used for data collection and to provide a bridge between a plant network and the outside world. The machinery control functions are dedicated to the PLC. Our preference for PLCs over PCs is based on the following.

PLC are a more rugged and stable control product, designed for industrial environments. While there are some industrial PCs that hold up well in harsh environments, we have found PLCs to be much more reliable than PCs. PLCs are also a very stable platform since the software is specific to the hardware by that manufacturer. We have had PLC-based systems running for more than 20 years. During this time, modules may have been replaced many years after original installation. It is rare to encounter problems with incompatibility when modules or the program is changed on older PLCs. With PCs you are more likely to encounter problems where operating systems or drivers have to be upgraded when changes are made.

PLCs have better acceptance by industrial end users. Often a factory will specify a type of PLC it wants as part of the machinery. Usually this is to make it easier for them to support spare parts and training of personnel. If your control system is PC-based, you will have a difficult time convincing a customer with a PLC specification to accept your PC-based control system. If your system is PLC-based but a different brand than specified, you would have an easier time gaining acceptance, and, if not, it will be much easier to port your PLC program to the other PLC brand than if your system is PC-based.
PLCs are a more cost-effective control solution, compared to PCs. When using PCs for industrial control, you are usually just replacing the CPU portion of the PLC and still must use PLC I/O modules or a similar product for connection to digital and analog signals on the machine. PCs often require an additional module with the I/O to serve as an interface between the PC and the I/O, which is not required by the PLC.

Most of our PLC-based machinery control systems are programmed with ladder logic. We find this the easiest to implement and certainly to support, and it has the best industry acceptance. For a machinery control system, the time it takes to write the original program is just one of many factors to be considered. The program should be written so others can easily follow and make future changes. This may not only involve other programmers at the OEM, but technical personnel at the end user, often many years after the original program was written. We have found the control programming language best suited for long-term support by others of varying levels of training is ladder logic.

If the application requires a certain feature which is better suited to another form of programming, such as statement list or STL for Siemens PLCs, we will write most of the program in ladder and call an STL subroutine for the special feature.

**What creates value?**

**Dan Cole, controls engineer, Pearson Packaging Systems:** Machine control has been addressed using PLC programming for many years. It is the industry standard. The PLC is a stable device that will continue running long after the machine has mechanically failed. However, there are some downsides to using a PLC. If the machine is going to be connected to a network, the PLC will require additional equipment, such as switches. A PLC is also a pricey solution, sometimes twice as much as a PC. Visual Basic (VB) programming is also a simple way to program machine control, but it comes with the added benefit of being easily connected to a plant-level network. Using a PC with an embedded operating system protects the controller from the dreaded Windows updates, maintaining solid performance for the controller. As a developer, I could argue either solution. However, I prefer the flexibility that VB provides.

The best way to settle this debate is by considering the application and the customer. Most customers are comfortable with PLCs. As a machine builder, you need to be prepared to have a solid case for using a PC-based controller when selling your product. I am of the belief that you should do what is best for the customer, even at the sacrifice of your own hardware and software preference. Our job as OEMs is to create value using innovation—value that can be transferred to our customers.

**PLC vs. VB programming**

**Eelco van der Wal, managing director, PLCopen:** There are always two aspects to selecting a programming environment:

1. What programming language do you want to use? This depends in many cases on what you are used to.
2. Which components, such as sensors and actuators, that I want to use are supported in the programming system?

You mention that you will be using mostly pneumatic actuators and a few servo motors. Most PLC environments will have direct support for this, and PLCopen has taken care that the motion-
control programming is harmonized across the different systems, giving you the possibility to choose between suppliers.

If a programmer wants to use VB, he or she probably has to program these interfaces to the separate actuators completely. This can be a tedious job, wasting human resources as well as money. Moreover, the resulting program can be difficult to maintain within your organization if this person is the only one who programs in VB.

Also, if a programmer is not familiar with industrial automation, I certainly would advise to stay within a PLC development tool since the tools are much more oriented to the industrial environment.

You mention that the program comprises a 21-step sequence. Excellent decomposing done here. For this, I would use sequential function chart (SFC) to map this directly and easily to SFC steps, including the transitions between these steps, and then use any of your other known languages to specify the different actions at the action blocks. This gives an excellent overview and a fast track to your application program. I do not think that VB can give you a result here that quickly. So, I would propose to use PLC programming since, especially due to your decomposition, it helps to create your program more quickly with less errors and with a better lifecycle support, regarding future updates.

**Why reinvent the wheel?**

*Donald Labriola P.E., President, QuickSilver Controls:* A big question between the PLC and the PC, at least to me, is how long is this configuration going to be maintained, and does the company have multiple programmers to maintain it? The code for the PLC can more easily be handled by the next PLC programmer. Specialized Visual Basic or C# or C++, unless there is a lot of discipline to put the time into documentation, are much more difficult to pass to the next person, especially if the original programmer is no longer available. Mature software groups have the methods in place to succeed in passing on information; ad hoc will lead to issues.

Another issue is how hard of real-time is required. Does the PC operating system support real-time? The PLCs normally have a lighter overhead and will specify the update rate according to the number of rungs being processed.

There is also more of an issue of maintaining the product line in the face of changing PCs. Sometimes that really good price was because the manufacturer was closing out that product line. Significant effort may be needed to re-qualify the new PC more frequently than desired, especially if there is much regulatory compliance involved. Much of this is normally hidden from the customer by the PLC manufacturer. If you decide to go for an industrial PC, be sure to see how long they will make that particular configuration available.

Be aware of whether the operating system will be going through regular updates and how this affects any testing that may be needed when the update happens. This problem typically is less of an issue with the PLC.

The PC route may be justified where the complexity and interfaces are just not available in an affordable PLC, but be sure to be careful of the pitfalls. Getting all of the interface protocols up is not always as easy as one would guess, and you may need to get closer to the operating system than you would prefer. If you don’t really need to reinvent the wheel, try to avoid it.
Don’t roll your own

John Kowal, director, business development, B&R Industrial Automation: I’m more concerned that the programmer in question has no automation experience. Industrial controls suppliers invest great sums of dollars in their software development environments to provide IEC 61131-3 compliance, documentation and useful functionalities, enable best practices and promote structured, modular code. “Rolling your own” puts that responsibility on the inexperienced programmer and will likely result in a machine that your other programmers will find difficult to troubleshoot or modify in the future.

This doesn’t mean that the PLC code has to be limited to ladder. It may be a combination of IEC ladder diagram (LD), function blocks (FBs) containing code written in ST, and/or FBD and SFC. The purpose of programming in IEC 61131-3 languages does not mean that the end users should be encouraged to modify the source code. Rather, I would suggest a state-of-the-art development environment where your code is compiled to protect the integrity of your software, with configuration tools, a software "sandbox" in which the user can apply familiar ladder logic to perform tasks such as handshaking or adding timers and sensors, without touching your code, and using the ISA TR.88.00.02 (PackML) and OPC-UA standards.

Machines like ladder

Cindy Hollenbeck, vice president, SoftPLC: For machine control, ladder logic offers a number of advantages. Ladder is interpreted, not compiled, which means that run-mode programming and online monitoring of machine states are inherently built in to the PLC software tools. So, whether the end users will be maintaining/troubleshooting the machine or you as an OEM will be doing this, especially remotely, this is a huge advantage. This is especially true of sequential operations.

If someone needs to take over maintenance of the logic in future, ladder logic is familiar to electricians and controls engineers, not just to programmers. PLCs are not subject to Microsoft's development schedule of updates/obsolescence or to IT department automatic updates of the operating system based on generic policies for office computers. PLCs are deterministic; Windows and VB are not. This means that, from scan to scan, the I/O processing will be consistent on a PLC.

PLCs include well-tested, field-proven and efficient communications to I/O, HMIs and computers. Using a PLC eliminates the need to develop drivers and/or incorporate third-party drivers or to use OPC, which is not really a good solution for I/O processing vs. HMI type data exchange.

Put the end user first

Deana Fu, senior product manager, Mitsubishi Electric Automation: This is a tough debate. Once people's minds are made up, it is hard to sway them to the other side. Whether it is ease-of-programming, maintenance, scalability or parts availability, it's probably safe to assume that both of you already have a long list of pros and cons. This is natural. When the stakes are high we tend to default to what we are most experienced and confident in. Yet, with minimal effort, both of you could probably become well-versed in either programming method and complete this machine.

Let's put your preferences aside, and consider who are your end users and what do they want? Ultimately, they are the ones who will be making the purchasing decision. Do they have the skill
sets to learn and support VB programming with minimal training? If the assembly machine stops during their busiest season, would they be able to troubleshoot and quickly get it running again? Or will they be at a standstill and call you for emergency on-site support? And, when the control hardware becomes obsolete, how much time and effort will it take to convert the program to work with next generation hardware? High-level programming is the norm at some select industries, and they tend to have dedicated programmers on staff to support any custom coding needs. In that case, maybe VB programming is the way to go. But, if your end users simply want an industrial control system they can operate and maintain with minimal effort, then that 21-step ladder sequence sounds like a very good option. Most PLCs use ladder programs and other IEC 61131-3 languages that are designed to do exactly that—keep the program simple and straightforward.

**How about a hybrid?**

Clark Kromenacker, product marketing manager—PLCs, HMI, RFID, Omron: The biggest advantage a PLC brings to the table and why they continue to be used over straight out Windows PC is something that the industry calls determinism. We have all seen our own Windows-based PC applications slow down when other programs are running in the background or when we start to run a resource-intensive application. If the running of your VB.Net program started slowing down, the I/O could update at different rates, meaning actuators and indicators would vary in their response times. In other words, they would not be deterministic. We all know that with a PLC, the program executes every line of code and updates the I/O at a constant and repeatable rate, no matter what else is going on in the PLC—communications, alarm generation. The industry has tried to isolate the Windows operating system with things such as Windows Extensions, which bypass much of the Windows operating system and function at a lower level, closer to the PC BIOS. Then machine control would be run on top of these extensions. So, not sharing many of the Windows resources with other programs allows some sense of determinism. However, since this is still part of Windows, when Windows crashes, the machine control goes with it, or, if there is a problem with Windows at that lower level, close to the BIOS system, the machine control gets affected.

The real solution if a PC is needed to manipulate data is to have totally separated operating systems on the same platform. The high ground in industrial-PC technology is to have half of the machine running a conventional operating system from Windows and the other half a true real-time operating system (RTOS). Then a communications path from Windows to the RTOS allows true determinism in the machine control and full Windows functionality together. In a hybrid of this type of industrial PC, the BIOS are partitioned so the Windows and machine control RTOS operate independently with a hypervisor link to coordinate activity. Only this type of architecture can be truly reliable.

If you need Windows functionality and machine control, I would recommend using a hybrid industrial PC with a Windows RTOS system. If you only need deterministic and reliable control, then the stand-alone PLC would be the most reliable solution.

**All in real time**

Bill Faber, director, product management & new business development, Delta Products: Selecting a programming language is certainly an ongoing debate because there is a lot more at stake as the factors of deployment are considered. This is more of a platform question, and the questions that an end-user production facility would ask are eventually the same questions that an OEM machine designer would need to ask around maintenance, support,
compatibility, production level, application complexity and uniqueness, reusability and scalability.

Will this programming platform allow for long-term sustainable support as a machine product, if machine operation stops, or if it needs to be upgraded? Manufacturing facilities purchasing production machinery will factor in how easy it will be to support this machine, not just in installation, but also in the machine’s lifetime in production. Standards such as IEC-61131-3 and PLCopen were established to address that point. Multiple PLC vendors and IPC vendors both allow for use of it; the installed base is already high, and we are seeing the adoption rate of the standard continue to increase in manufacturing. This standard supports multiple programming environments including structured text and function encapsulation which also helps to retain code reusability and intellectual-property (IP) protection. VB.net has open programming benefits, but, if it is not a common programming environment for the OEM machine builders target market, then using the IEC standard will put purchasers at ease when selecting capital equipment. How many engineers and technicians will be supporting this as a machine product and as a production machine? Whether they sustainably have the right training is a good litmus test of this. Additionally, it is very easy to create a state machine in ladder logic which is also simple for a technician to understand, as well as trap error states and conditions, making it easy for anyone with basic IEC standard training to troubleshoot.

What are the production goals, and how critical is downtime? The robustness of the target CPU will need to be considered, especially if the environment is a continuous process or a critical discrete component of a manufacturing line. Assembly machines could fall into that, if in-line. If the end-use facility is concerned about operational equipment efficiency, then uptime, yield and throughput will be of great concerns, as well. A lab environment might be able to sustain a reboot from a Windows PC, but a production environment will need to ensure the system is always in control to reduce downtime costs, reduce material waste and meet volume production needs. It may be possible to split the HMI and control, but of course costs of control systems need to be kept down; a PLC certainly provides that separation and scalability.

Is a PC required? If the machine requires extensive local processing, data collecting or a specific program to compliment the machine solution, then a PC might be necessary or most efficient if tight integration is an absolute need. This could be the case in a machine that has a vision system; however, there are smart-camera systems and fast networks on the market today that can accomplish the same task. Additionally, with edge-computing devices and cloud-platform solutions becoming more available, there are more flexible ways to off-load PC tasks in a hybrid or distributed control architecture and make it more accessible.

How unique is the machine solution? The more unique the machine solution is over the lifecycle of it as a marketed product, the more willing an end user will be to accept whatever programming environment is provided, but the competitive advantage would need to be substantial. It’s also important to look at the application complexity. If the machine requires centralized deterministic motion control and real-time synchronization with strict I/O, then a PLC has historically provided that more robustly, even in an upgrade or component replacement, due to auto-config modules. It’s possible to deploy an IPC with a tightly integrated RTOS, but to be able to maintain that over the lifecycle of the machine asset could prove to be a major challenge when considering upgrades and backward-compatibility. This could be overcome if the IPC supplier provides support for the complete package. Most PLC manufacturers have long lifecycles for their products because they were designed for long-lifecycle industrial applications.
Stick with the proven standard

Jeff Clonts, product manager, Allied Electronics: For machine control applications, I would strongly encourage the PLC over the industrial computer option. PLCs are a proven industrial standard that will provide a platform for application development and troubleshooting. Most PLCs offer preconfigured instruction sets that will allow the user to drag and drop popular instructions such as timers, counters, and compare functions into the logic. Some even provide motion-control instructions that could simplify your servo application. For this project, you could create individual subroutines for each of your 21-step sequences. This would provide the end user with a clean, organized and easy-to-interpret program. Your end user will likely have technicians or engineers on staff who can monitor this code to help identify hardware failures or equipment fault conditions. However, systems that are programmed in higher-level programming languages are far more difficult for the end user to troubleshoot. If the customer is unable to provide some level of in-house support for this machine, your company’s reputation could be damaged. If your decision comes down to comfort with one language over another, many PLC suppliers offer excellent PLC training classes. Having a working knowledge of PLCs and ladder logic would be a wonderful and necessary skill set for any controls engineer to have.

It's application specific

Matt Schmidt, Engineer, Concept Systems: As you are experiencing, the PLC vs. PC debate is very much alive and well in the world of controls right now. The short answer is that the decision should be application specific.

In your case, with 80 I/O points and several servo motors, a PLC is a great fit. Handling discrete I/O is one of the things PLCs do best. Remember, they are modeled around switches and relays. You will also not find it difficult to discover many PLC-based motion control options for your servo motors.

There are many factors to take into account as an OEM when making this decision. Here are a few.

Robust and Reliable: What environment will your machine operate in? For dirty, dusty, extreme temperature, or generally harsh environments a PLC will continue to run where an IPC may not. A PLC uses a dedicated processor which provides a high degree of control system reliability. The IPC will be running a PC-based OS and will require system updates and security measures to protect against unauthorized access.

Machine Service: What is the service plan for your system? Service technicians, system integrators, and plant maintenance personnel can quickly troubleshoot a PLC with discrete I/O. Similar troubleshooting on a PC application is often more complicated and difficult for those servicing the machine.

Controller Cost: Typically, this is the consideration PC-enthusiasts will tout in their favor. Yet there are many factors influencing cost beyond the base price of the controller. For example, both the PLC and IPC will require I/O modules. The PLC will require HMI hardware/software while the GUI for the IPC can be deployed directly on the controller. Pricing and availability of hardware components now and in the future should be considered. PLCs typically have a longer support and lifecycles than PCs. Development time is often a large cost. While an elegant solution can be achieved on the IPC, it may take a fraction of the development time to write a ladder-logic program.
Hopefully this will help guide the discussion as your team determines the right technology for your application.

**Question the tool**

*Matt Newton, director of technical marketing, Opto 22:* Today, there are a lot of options for programming an industrial application. Modern industrial automation controllers are being shipped with new technology built in—like web servers, data encryption, and RESTful APIs. Industrial application developers can literally write their application in just about any language they want, from ladder logic to VB.Net, Node-RED and JavaScript. But in the end, it really comes down to choosing the right tool for the job—or more specifically, the right language for the application. Here are some things to consider.

Is this machine going to be interfacing with IT resources in ERP or MES applications? If so, it’s probably a good idea to go the VB route, since that language is designed to work in applications where IT resources like databases and web servers are being used. Traditional ladder logic wasn’t designed to easily interface with external network assets like web servers and databases.

Is the machine going to be deployed and then never touched or interfaced to other systems again? In that case, a traditional PLC programming language like ladder logic or flow chart/function block-based programming might be the better choice.

Can you easily find engineering talent to support the application? Today, there are more VB.Net and JavaScript programmers than there are ladder logic programmers—which may affect the cost and availability of supporting the application down the road.

What about future proofing the application? There’s a lot of industry buzz around Industrial Internet of Things (IIoT) technology. Traditional ladder logic doesn’t have a lot of support for handling things like edge computing, interfacing to other network assets, such as cloud-based predictive analytics, and machine-learning applications. If you’re concerned about future proofing your design to work with these types of applications as they become more prevalent, a higher-level language might be the right path to take.

In the end, traditional ladder logic is a great tool for automating simple, repeated control of basic I/O points. But if you’re looking for more advanced functionality and capabilities, not to mention a wider availability of programmers to support the application, it may be time to look at VB.Net or even some of the other languages and tools out there, like JavaScript and Node-RED.

**About the author**

Mike Bacidore is the editor in chief for *Control Design* magazine. He is an award-winning columnist, earning a Gold Regional Award and a Silver National Award from the American Society of Business Publication Editors. Email him at mbacidore@putman.net.
Summary
Problems

1. For each example, list whether the task would be best programmed as a **continuous**, **periodic** or **event** (interrupt driven) task.

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Programming Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill a tank to its maximum level and then open a drain valve</td>
<td>Continuous</td>
</tr>
<tr>
<td>Read the thickness of a paper roll every 20 ms</td>
<td>Periodic</td>
</tr>
<tr>
<td>A gluing station must adjust the amount of glue it applies to compensate for changes in the speed of the axis. After the motion planner executes, check the command speed of an axis and vary the amount of glue, if needed.</td>
<td>Event</td>
</tr>
<tr>
<td>Your system must check the position of a field arm each 0.1 s and calculate the average rate of change in its position. This is used to determine braking pressure</td>
<td>Event</td>
</tr>
<tr>
<td>In a production line, if any of the programs detect an unsafe condition the entire line must shut down. The shutdown procedure is the same regardless of the unsafe condition</td>
<td>Event</td>
</tr>
</tbody>
</table>

2. For the figure below, what happens to the event task when the fourth (circle 4) trigger event occurs?

![Event Task Diagram](image)

What term describes this event?

Does the PLC have a mechanism to detect this event?

3. For the following figure, provide logic for each one-shot. Identify in your program each signal labeled below. You may use either Siemens or A-B one-shot logic or you may create the logic entirely without one-shot instructions:

![Input Diagram](image)

4. Write the program that would accomplish the task described on the first page of this chapter.
5. From the article on PLC vs IPC, pick any one of the respondents and write why you agree or disagree with their point. Note any arguments that you may not have previously realized in your discussion.

6. Pick a commercial wheel with toothed gears and a proximity switch wired to a PLC input. Describe analytically how fast the wheel is allowed to turn in order for the program in your PLC to be guaranteed to not miss any pulses.