LASER SYSTEM INTEGRATION

Practical advice for manufacturing managers and process engineers

Manufacturing with Light (MWL)

A Great Laser Machine Isn't Just a Laser

Making motion, laser, optics, controls, utilities, safety, and software work together as one production system.

AI note: AI tools were used to help draft, organize, and edit this article. The technical guidance still requires application-specific engineering, current vendor documentation, and an appropriate laser and machine safety review.

A good laser source does not automatically make a good laser machine.

The same is true of the motion platform, processing head, optics, gas system, chiller, safety system, controls, vision system, or HMI. You can select excellent components from excellent suppliers and still end up with a machine that is hard to commission, inconsistent in production, or frustrating to support.

The real challenge is getting all of those components to behave like one machine.

For a manufacturing manager, that means thinking beyond the purchase price and individual datasheets. For a process engineer, it means understanding the interfaces between subsystems well enough that the process remains stable when speed, part geometry, recipe, material, or operating conditions change.

A good production machine should also remain understandable and recoverable years after commissioning. Software maintainability belongs in the machine specification just as much as spare parts, drawings, and service access.

A useful way to look at integration

Most difficult commissioning problems do not live inside a single component. They live at the boundary between two components: motion and laser timing, head and workpiece, gas supply and nozzle, chiller and laser, safety and process control, or machine software and operator workflow.

1. Start with the process - then choose the architecture

It is tempting to begin a new system by building a component list: laser, stage, controller, head, chiller, gas panel, vision, and so on. Those decisions matter, but the process requirements should come first.

Before selecting the architecture, write down the real manufacturing problem. What are we cutting, welding, drilling, marking, or modifying? What feature size and tolerance matter? What is the acceptable heat input? What is the desired cycle time? What changes from one part or recipe to another?

The most important question is often not the maximum speed of a stage or the maximum power of a laser. It is how the whole process behaves while the machine accelerates, decelerates, changes direction, pierces, crosses small features, or moves between process and non-process areas.

For early design reviews, I would want the team to agree on at least these items:

- Process envelope: materials, thicknesses, feature sizes, expected recipes, and future variants.
- Quality targets: dimensional accuracy, cut or weld quality, allowable heat input, spatter, taper, roughness, or other process-specific measures.

- Throughput targets: not just peak travel speed, but realistic part-to-part cycle time including loading, gas delays, pierce time, inspection, and recovery.
- Synchronization requirements: which events are time-based, position-based, velocity-dependent, or tied to particular motion segments.
- Operating modes: production, setup, service, dry run, alignment, recipe development, and recovery from interrupted cycles.

Manager takeaway

A faster component is not automatically a faster machine. Ask for a cycle-time model that includes the process transitions and supporting equipment, not just the motion profile.

2. Decide who owns each machine function

A modern laser system may contain a machine controller, PLC, laser controller, safety controller, vision computer, HMI, and several intelligent peripheral devices. There is no rule that every machine needs the same architecture.

On many precision laser systems, a high-performance platform such as **ACS Motion Control** or **Aerotech** can function as the primary machine controller rather than simply sitting below a PLC. The same controller may handle coordinated motion, I/O, machine states, process parameters, trajectory generation, and laser-synchronized events.

A PLC may still be the right place for utilities, auxiliary equipment, supervisory sequencing, plant interfaces, or other functions. On some systems the PLC will have a large role; on others it may have little or no role. The point is not to defend one architecture. The point is to make ownership obvious.

Function	Typical owner	Integration question
Coordinated trajectory	Machine/motion controller	Who generates the path and owns feed hold/recovery?
Laser process command	Machine controller + laser interface	Who decides when power, gating, or pulse mode changes?
Utilities	PLC or machine controller	Who verifies gas, cooling, extraction, and pressure permissives?
Safety permission	Safety system	What independent conditions must be true before hazardous energy is permitted?
Operator workflow	HMI + machine controller	Where are mode, recipe, alarm, and recovery states defined?

Function	Typical owner	Integration question
Diagnostics	All layers, coordinated	Which controller owns the root-cause message shown to maintenance?

A simple test

Pick any important state - READY, LASER ENABLED, GAS READY, PART CLAMPED, CYCLE COMPLETE. Can everyone on the team explain exactly which controller owns it, what makes it true, what makes it false, and what happens if communication is lost?

3. Treat interfaces like engineering deliverables

A common source of late-project pain is assuming that a communications protocol is the interface specification. It is not. Knowing that two components both support EtherCAT, EtherNet/IP, PROFINET, analog I/O, or digital I/O only tells you that they can exchange information.

The real interface is the meaning of that information. What does READY mean? Is a fault latched? What clears it? How quickly can a setpoint change? What happens after a power cycle? Can a laser request be accepted while the source is warming up? What state is safe if communications disappear?

For every major subsystem, I like the idea of a short interface-control document. It does not need to be a hundred-page specification. A few well-maintained pages can save days during commissioning.

At minimum, define:

- Commands and their valid operating states.
- Status signals and exactly what each one means.
- Permissives and interlocks required before a command is accepted.
- Parameter units, scaling, valid ranges, defaults, and update timing.
- Timeouts, retry behavior, and what happens after loss of communication.
- Fault/reset behavior and which faults require operator action versus automatic recovery.
- Version compatibility: firmware, option codes, protocol revisions, and configuration files.

Helpful commissioning habit

If someone says "the device is ready," ask, "Ready for what?" Ready for communication, ready to accept a command, ready to emit laser power, and ready for production can be four different states.

4. Communication is not the same as synchronization

This is especially important on precision laser equipment. Sending a number over an industrial network is very different from making an event occur at an exact location on a contour.

ACS describes **motion-to-process synchronization** as synchronizing process events such as laser firing or inspection triggering with motion. Its G-code support is also specifically positioned for high-precision laser processing and can be combined with ACSPL+ for machine-specific functions.

Aerotech's **Position Synchronized Output (PSO)** is another example of the same general principle: generate process events from position or distance rather than relying only on elapsed time.

For a process engineer, the practical question is: what variable actually controls process quality? If equal pulse spacing matters, position may matter more than time. If energy per unit length matters, velocity may need to influence power or pulse rate. If the laser should only be active on selected portions of a contour, segment or path-based gating may be appropriate.

For a manufacturing manager, this is worth discussing early because a machine can meet its axis accuracy specification and still produce inconsistent parts if the process events are not synchronized correctly.

Questions worth asking

Does the process trigger from commanded position or measured position? How much latency and jitter are acceptable? What happens during acceleration and deceleration? Does the process need fixed-distance pulses, velocity compensation, segment gating, or only simple on/off control?

5. Utilities are process components, not accessories

Some of the hardest commissioning problems come from equipment that never appears on the glossy laser or motion datasheet: gas supply, cooling, compressed air, vacuum, extraction, electrical power, and facility interfaces.

Take cutting gas. Saying "I need 300 psi nitrogen" still leaves a lot undefined. What pressure is required at the machine inlet? What pressure is required at the nozzle while flowing? What is the flow rate? How much pressure drop is created by the regulator, valve, hose, fitting, swivel, and nozzle? How long does it take pressure to stabilize after the valve opens?

A useful specification should describe the condition under which the number is required. Static pressure is not the same as dynamic pressure. Peak flow is not the same as continuous flow. And the process may care about rise time just as much as steady-state pressure.

Cooling deserves the same attention. A chiller can have enough nominal capacity and still be poorly integrated if the system does not handle flow, dew point, warm-up, temperature stability, hose routing, filtration, alarms, and restart behavior correctly.

For each utility, capture three types of requirements:

- Capacity: pressure, flow, temperature, electrical load, vacuum level, or extraction volume.
- Dynamic behavior: stabilization time, response time, allowable droop, startup sequence, and recovery after interruption.
- Machine response: what the machine should do if the utility is marginal, missing, or unstable.

6. Mechanical and optical integration can quietly limit performance

Controls software cannot compensate for every mechanical problem. The process head has to remain where the process expects it to be, and the workpiece has to be presented repeatably.

Payload is more than the mass listed on a stage datasheet. The real moving system includes the head, optics, cable carrier, fiber, gas lines, sensor cables, air lines, covers, and sometimes a height-control mechanism. Those items add inertia, drag, changing forces, and possible resonances.

Optical integration adds another set of relationships: focus position, beam alignment, nozzle centering, standoff, protective window condition, contamination, back reflection, and the mechanical stability of the entire beam-delivery path.

A machine that is mechanically excellent at low speed can behave differently at production acceleration. That is why process qualification should use realistic trajectories and payloads, not only slow commissioning moves.

Process-engineering advice

When a cut or weld changes with direction, speed, or location on the table, do not immediately assume the laser recipe is wrong. Look for motion tracking, cable forces, focus/standoff changes, gas delivery, thermal growth, and fixture behavior too.

7. Keep the safety architecture independent of the process recipe

The process controller may decide that the laser should fire. The safety system determines whether the laser may fire. Those are different responsibilities.

For higher-power industrial laser systems, engineering controls such as protective housings and interlocks need to be considered as part of the machine architecture. **OSHA's laser guidance** emphasizes engineering controls as a primary method for limiting exposure to hazardous laser radiation.

From an integration standpoint, this means the process sequence should request laser operation through a controlled interface. It should not be able to defeat the safety permission chain.

Service and alignment modes deserve particular attention because the machine may intentionally operate differently from production. Those modes should be designed, documented, and risk-assessed rather than created as an afterthought during commissioning.

8. Commission in layers - and make 'beam off' a real operating mode

Trying to commission the entire machine at once makes it difficult to know where a problem originates. I prefer a layered approach that makes each step prove something before the next layer is introduced.

A well-designed dry-run mode is extremely useful. It should allow the software to execute the real sequence, generate motion, exercise interfaces, and report expected process commands while preventing hazardous or damaging outputs.

Layer	What to prove
1. Communications	Verify network topology, addressing, device versions, and loss-of-communication behavior.
2. I/O and device	Prove each command/status pair individually. Test faults and reset behavior.

Layer	What to prove
control	
3. Motion	Home, limits, coordinate systems, tuning, travel, and realistic production trajectories.
4. Machine sequencing	Run automatic cycles without the process energy enabled.
5. Process synchronization	Verify triggers, gates, analog outputs, and timing with instrumentation or safe substitutes.
6. Utilities	Run gas, cooling, extraction, vacuum, and facility interfaces under realistic demand.
7. Live process	Introduce laser energy only after the machine behavior is understood and the safety conditions are satisfied.

Do not skip fault injection

Commissioning is the best time to intentionally unplug a device, remove a permissive, trip a chiller alarm, drop gas pressure, stop an axis, and interrupt a cycle. You want to learn how the machine fails while the engineering team is standing next to it - not after it ships.

9. Diagnostics should answer 'where do I start?'

An alarm that says "LASER FAULT" may be technically correct and still be nearly useless.

The operator and maintenance team need enough information to distinguish a laser-source fault from a missing chiller permissive, open external interlock, head fault, communication problem, gas issue, or machine-state problem.

This does not mean every alarm needs a paragraph of text. It means the software should preserve the chain of cause and effect.

Useful diagnostic design usually includes:

- A clear machine state and current cycle step.
- Commanded state versus actual state for each major subsystem.
- Permissive details: not just NOT READY, but which required condition is missing.
- Timeout messages that identify the command being attempted and the feedback that never arrived.
- Communication health, device status, and useful fault codes from the component itself.
- A timestamped event history that helps reconstruct what happened before a stop.
- Service screens that expose relevant I/O and status without requiring the technician to open source code.

A good alarm should reduce the search area

The goal is not to tell maintenance exactly which part to replace. The goal is to get them to the right subsystem, condition, and next check quickly.

10. Think about maintainability before the machine ships

The best integration work often becomes most valuable years after the factory acceptance test.

A machine that is easy to support usually has organized software, meaningful names, documented interfaces, accessible backups, clear revision history, and diagnostic tools that a technician can actually use.

It also has an answer to a very practical question: if a controller, drive, laser source, HMI PC, sensor, or communication module fails five years from now, what information will the service team need to replace it and restore production?

Software can become the long-term maintenance problem

Mechanical components are visible, and spare-parts planning usually gets discussed. Software can be easier to overlook because it may work perfectly for years - until a PC, controller, drive, or operating system has to be replaced.

The long-term risk is often not the source code itself. It is the collection of dependencies around it: the exact development environment, runtime, vendor SDK, communication driver, database, license, firmware revision, Windows configuration, or third-party library that was required to make the released machine software work.

A machine can be mechanically healthy and still become difficult to support if the only working copy of the application lives on an old engineering laptop or if nobody can reproduce the production build.

For long-term support, I would specifically require:

- Editable source code for the machine controller or PLC, HMI, vision system, custom PC applications, scripts, and configuration utilities used by the machine.
- A clearly identified production release tied to the machine, with revision history and enough information to determine exactly what software was running when the machine shipped.
- The required development tools, runtime versions, drivers, vendor SDKs, libraries, database versions, and firmware/option dependencies.
- License and activation ownership documented so the manufacturer can reinstall or rebuild the software without depending on an individual engineer's personal account.
- Build and deployment instructions that another qualified engineer can follow, preferably verified from a clean engineering workstation or controlled virtual environment.
- Backups or recovery images for industrial PCs and exported configuration backups for controllers, drives, HMIs, vision systems, laser interfaces, and other programmable devices.
- A plan for PC and operating-system replacement. If a custom HMI or host application depends on an older environment, document what is known to work and how it will be preserved or migrated.
- Securely managed administrative credentials and recovery information owned by the manufacturer, rather than undocumented passwords known only to the original integrator.

- A periodic backup check. A backup that has never been opened, restored, or verified is only an assumption.

This is not meant to turn a machine project into an IT project. It is simply recognizing that modern production equipment is partly a software product, and the software needs the same lifecycle thinking we apply to bearings, optics, valves, drives, and spare parts.

Before shipment, I would want a support package that includes:

- Controller, PLC, HMI, vision, robot, and laser configuration/source backups.
- Firmware and software versions, option codes, licenses, and required development tools.
- Network topology, addresses, fieldbus mappings, and device configuration files.
- Electrical and pneumatic drawings that match the as-built machine.
- A documented recovery procedure for replacing major components.
- Recommended spare parts based on lead time, failure impact, and availability - not only component cost.
- A short list of the machine's most important process and troubleshooting variables.

11. Manage process recipes and machine software as different kinds of change

One useful architectural idea is to separate reusable machine capabilities from the part or process sequence where practical.

The machine-control software defines how to perform validated functions - move, clamp, enable gas, set process power, inspect, recover from a fault. The part or process program defines when those functions are used and with what approved parameters.

That separation can make process development easier because a recipe change does not automatically become a low-level controls-software change. It can also make responsibilities clearer: controls engineers own the machine capabilities, while qualified process personnel can own approved recipe values and sequences within defined guardrails.

The important word is guardrails. A recipe should not have direct access to raw safety outputs or unrestricted machine functions. It should call validated capabilities with range checking, state checking, timeouts, and clear success/failure reporting.

Why this matters to manufacturing

As product mix grows, a machine that requires a controls programmer for every process variation can become a bottleneck. A controlled recipe/sequence layer can move routine process ownership closer to the people who understand the product.

12. Acceptance testing should prove the integrated system, not just the parts

A factory acceptance test can become a checklist of individual devices: axis moves, laser turns on, chiller runs, gas valve opens. Those checks are necessary, but they do not prove the integrated manufacturing process.

A better FAT also tests the transitions and edge cases that cause real production downtime.

Add software recoverability to the FAT

The FAT is one of the best opportunities to verify software maintainability because the builder, controls engineer, process engineer, and customer are still working from the same released machine.

You do not necessarily need to perform a full disaster-recovery exercise on every machine, but the team should prove that the delivered software package is complete enough to recover from a failed HMI PC, controller replacement, or future engineering change.

- Record the approved production version of every major software and firmware component before shipment.
- Create and verify backups from the actual production machine - not only from the programmer's development folder.
- Confirm that the controller/PLC, HMI, vision, laser-interface, drive, and network configuration files can be opened with the documented engineering tools.
- Where practical, open or build the main projects on a clean engineering PC or controlled virtual environment to prove that the required dependencies and licenses are available.
- Verify that the manufacturer has access to required installers, runtime packages, licenses, entitlement information, and vendor accounts needed for future service.
- Prove the HMI or industrial-PC recovery method, including network settings, device drivers, startup behavior, and the location of machine-specific configuration files.
- Power-cycle the complete machine and verify that it returns to a known, supportable state without undocumented engineering steps.
- Document secure ownership and recovery of administrative credentials needed for maintenance.
- Include a software/firmware compatibility matrix in the handoff package when multiple controllers or intelligent devices must remain at compatible revisions.

A machine that runs correctly at FAT but cannot be restored from the delivered software package still carries a long-term production risk.

Examples I would include:

- Run representative parts at realistic production acceleration and speed.
- Verify recipe changes, unit conversions, parameter limits, and invalid-value handling.
- Interrupt a cycle and prove a controlled recovery path.
- Test low gas pressure, missing cooling, communication loss, and subsystem faults.
- Power-cycle the machine and verify that every device returns to a known state.
- Confirm that service/dry-run modes cannot accidentally enable the real process.
- Measure actual cycle time and compare it with the commercial/process target.
- Capture a known-good set of diagnostics and process data for future service comparison.

Questions to ask before design freeze

Manufacturing / management	Process / engineering
☐ Have we defined realistic cycle time, including non-process time?	☐ What variable actually controls quality: time, position, velocity, energy/length, standoff, pressure, or something else?
☐ Who will support each major component after installation?	☐ Which process events must be synchronized with the trajectory?
☐ What are the long-lead or production-critical spares?	☐ How are focus, nozzle/standoff, gas delivery, and workholding verified?
☐ Can maintenance get the required software, backups, and documentation?	☐ Which recipe parameters can change, and what limits are enforced?
☐ Are supplier responsibilities clear at subsystem boundaries?	☐ What diagnostics will tell us why a process did not start or complete?
☐ Does the acceptance test prove recovery from faults, not only normal operation?	☐ What data should be saved as a known-good process baseline?
☐ Do we have editable source, installers, development-tool access, licenses, and documented software versions?	☐ Are recipe files and process data backed up and versioned independently from the controller/HMI software where practical?
☐ Could we recover from a failed HMI PC or controller without needing the original programmer's laptop?	☐ After a firmware or software change, how will we verify recipe compatibility and protect the known-good process baseline?

The goal is one understandable production machine

A successful laser system is rarely the result of one magic component. It comes from understanding how the laser, motion, mechanics, optics, controls, utilities, safety, software, and process interact - and defining those interactions before they become commissioning problems.

The people who add the most value during integration are often the ones willing to ask the inconvenient questions early: Who owns this state? What happens if this signal disappears? What does READY actually mean? How quickly does this command take effect? How will someone troubleshoot this in five years?

The bottom line

Taking good components and making them behave like one reliable, maintainable production machine is where laser system integration becomes real engineering.

Selected sources and further reading

- **ACS Motion Control - Laser Processing Systems** - High-precision motion, servo control, and motion-to-process synchronization for laser OEMs.
- **ACS Motion Control - G-Code Programming** - G-code support developed for laser processing, including user-definable G/M codes, ACSPL+ integration, and simulator support.

- **ACS Motion Control - Motion-to-Process Synchronization** - Position- and event-synchronized process control for laser firing, inspection triggering, and other precision operations.
- **Aerotech - Position Synchronized Output (PSO)** - Position/distance-based process triggering for lasers and other devices.
- **Aerotech - Automation1 Software-Based Machine Controller** - Example of a motion platform designed to control both motion and process functions.
- **OSHA - Laser Hazards** - Workplace laser hazard information and links to engineering-control guidance.
- **Manufacturing with Light** - Practical articles on laser manufacturing, motion control, automation, and machine building.