LESSON 98

Coding Inputs, Outputs, and System Responses

240 minutes (6 periods) | Applied Learning Tasks (ALTs) | Outcome 3.11 | Outcome 3.12 | Outcome 3.13 | Outcome 3.14 | Outcome 2.14 | Outcome 2.20 | Outcome 1.22

WHAT THIS LESSON DOES

In this lesson, you'll code the inputs, outputs, and system responses for your embedded system project. Follow step-by-step guidance to review your plan, set up your environment, program interactions, debug issues, and document your work for a functional prototype.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
3.11	use and control digital inputs and outputs within an embedded system
3.12	measure and store data returned from an analogue input
3.13	develop a program that utilises digital and analogue inputs
3.14	design automated applications using embedded systems
2.14	describe the difference between digital and analogue input
2.20	identify and fix/debug warnings and errors in computer code and modify as required
1.22	read, write, test, and modify computer programs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the relationship between inputs, outputs, and system responses in embedded systems.
- Develop skills to code and integrate input handling and output control in a functional prototype.
- Apply debugging techniques to identify and resolve issues in embedded system programming.
- Demonstrate the ability to set up and configure a development environment for coding embedded systems.
- Recognise the importance of documenting code for clarity and future reference.

BEFORE THE LESSON

- Decide and test the exact platform your class will use, physical microcontroller with cables or an online simulator, and confirm it installs and runs behind the school filter well before the class arrives.
- Run a blinking-LED 'hello world' yourself on that platform first, so the environment setup step is a check and not a first attempt.
- Confirm any sensor or component libraries and drivers install, and that each student can reach their own design document from a previous lesson.

Coding Inputs, Outputs, and System Responses

HOW THE LESSON RUNS

Introduction
Review System Plan
Set Up Development Environment
Code Basic Input Handling
Implement Output Control
Integrate Inputs and Outputs
Debug and Iterate
Document Code

TEACHING MOVES

- 2. Review System Plan. Have each student say aloud whether each input is digital or analogue before writing any code. This one distinction shapes every read they will write, so name it now rather than letting them discover it mid-loop.
- 3. Set Up Development Environment. Do not move on until every student has a blinking LED or its simulator equivalent running. A working baseline is the whole point of this step, and it saves you chasing environment faults during the coding steps.
- 4. Code Basic Input Handling. Get them printing the raw input value before doing anything with it. Seeing the actual numbers coming off an analogue sensor, and mapping them to a real unit, prevents guessing at thresholds later.
- 5. Implement Output Control. Test the output in isolation with a fixed condition first, before any real input drives it. If the LED will not blink on a timer, it will not blink on a sensor either, so isolate the fault here.
- 6. Integrate Inputs and Outputs. Watch for blocking waits. A long wait inside the main loop freezes the system so it misses inputs. Point out the short debounce delay and explain why the loop must keep cycling.
- 7. Debug and Iterate. Insist on print statements to check what the code actually reads, not what students assume it reads. Have them record one bug and its fix in words, which is exactly the reflective evidence a Coursework Project needs.
- 8. Document Code. Ask them to comment the why, not the what. 'Turn on LED' restates the obvious; 'threshold set to 512, half of the sensor range' is the comment that helps them in three weeks.

WHAT TO WATCH FOR

- Students use assignment where they mean comparison inside the if-statement that checks an input, for example writing a single equals instead of a comparison. Read the condition aloud as a question, 'is the input high?', and check that the operator asks rather than sets. Show how the wrong operator makes the branch always run.
- Students assume that because the output looks right once, the logic is correct, without testing the else branch. Have them deliberately trigger the off case, release the button or drop the sensor value, and confirm the output actually turns off. A prototype must be tested in both states.
- Students put a long wait inside the loop and then wonder why the system misses their button press. Explain that the code can only read the input when the loop reaches the read line. During a long wait it is not looking. Reduce the delay and re-test responsiveness.

DIFFERENTIATION

- Emerging: If the environment is fighting a student, pair them onto a partner's working setup so they can code rather than lose the class to installation. Give them the fully worked digital button-to-LED path and let analogue input wait until that runs.
- Developing: Ask them to explain, before running it, what their integrated loop will do when the input is high and when it is low. If they can predict both, they understand the logic.
- Proficient: Push them to add error handling or a smoothed average of noisy analogue readings, and to justify their threshold choice in comments, all strong evidence for the Coursework Project. Have them add a second input or output and manage the combined logic cleanly in one loop.