Stop, Code, Go – Intro to STOP:bit

In this lesson, students will learn how to code the Kitronik STOP:bit - a physical traffic light add-on for the BBC micro:bit. They'll explore how real-world systems like traffic lights use sequences, timing, and logic to keep people safe.

Students will create their own working traffic light sequence using Microsoft MakeCode, then test and debug their program on the STOP:bit. They can then complete a challenge that extends the skills and context of the prior learning.

Curriculum links

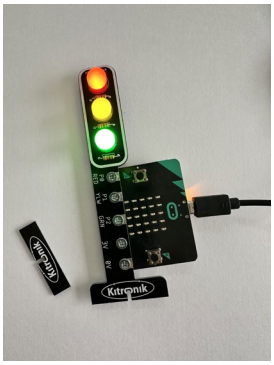
This activity and the suggested challenges are aligned with elements of the following learning area. We encourage teachers to adapt the content and focus to suit the needs and interests of their students. They can be adapted for students between **Year 3 and Year 6**. More details are provided in the supporting materials (p. 5.)

Digital Technologies

Required resources

You will need the following materials which are available in our CSER lending library kit:

- BBC micro:bit device attached to a Kitronik STOP:bit (one per group of students)
- USB cable and battery pack
- Laptop or tablet with Microsoft MakeCode editor



Pre-requisite knowledge

Prior to this session, students should:

- have a basic knowledge of how to code using block-based programming language
- understand how to download code to a micro:bit.

We suggest discussing and reviewing the features of the micro:bit including the pins using the following videos

- micro:bit basics
- micro:bit pins

Technologies Core concepts	Systems – inputs, processes and outputs	Interactions and impacts
	Systems thinking	Computational thinking

For more information about the Australian Curriculum Technologies core concepts Understand this learning area - Technologies

Suggested steps

Whole class introduction

Discuss the sequence of lights in a traffic light system. Why is the order important? What happens if it's wrong or too fast? As students respond, write the algorithm on the board.

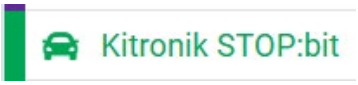
1. Red light ON
2. Red and amber OFF, green ON
3. Green OFF, amber ON
4. Amber OFF, red ON

The Basic sequence tutorial from Kitronik provides a scaffold for introducing the required skills. Strategies for incorporating this tutorial into the classroom include:

 **Teacher demonstration with students following along**	 **Students working in groups to follow online tutorial**	PRIMM **PRIMM activity Student given sample code (see Supporting materials (p. 5) for code and further information)**

The sequence of learning in the tutorial begins with using the buttons on the micro:bit to turn the STOP:bit lights on and off, progressing to a full traffic light sequence using a forever loop and a 1 second pause. Next, students refine their program by inserting varied pause blocks between each light change. They are then prompted to make the code more efficient by removing unnecessary blocks.

Note: if creating a new project, students will need to add the extension for the STOP:Bit to their MakeCode editor. **Click +Extensions** and search the STOP:Bit and select. The extension will now be available in your toolbar



More information: Free Online MakeCode Tutorials For STOP:bit for micro:bit – Kitronik Ltd

Pose questions

Use these example open-ended questions to prompt student reflection, check for understanding, and encourage discussion to help students explain their thinking and build on each other's ideas.

- Why do traffic lights follow a set sequence? Why is it important that traffic lights are reliable and consistent?
- What might happen in real life if the traffic light order was incorrect or changed too quickly?
- How did adding pauses between each light change/affect how the lights behaved?
- If you were explaining your code to someone else, what would be important for them to understand?
- What challenges did you face while coding the sequence? How did you overcome these challenges?
- How did you make your code more efficient or easier to read?
- What other features could you add to your traffic light program?

Challenges

With the skills and understanding of the basic of coding a traffic light sequence, students could explore the following challenges based on their ideas and interests. These challenges encourage students to extend their understanding of inputs, outputs, and control logic while designing interactive systems that model real-world scenarios or solve classroom problems.

For each challenge, students should create an algorithm before starting to code, using either plain English or flowcharts to plan the sequence of actions and decisions their program will follow.

Design a pedestrian crossing

Add a pedestrian button using micro:bit's A or B button. When pressed, it should pause the traffic lights and give pedestrians time to cross. Add a countdown on the micro:bit LED screen. Program flashing amber to simulate pelican crossings.

Radio-controlled intersection

Use two micro:bits with STOP:bits to simulate traffic lights at two nearby intersections. Use radio signals to stop both lights from going green at the same time.

Create a new use for STOP:bit

Brainstorm and prototype a new use for the STOP:bit beyond traffic lights. For example:

- A countdown timer for classroom activities
- A visual cue for quiet time or noise levels
- A tool for students to indicate if they require assistance with their work. Green = ok, red = need help.

Use STOP:bit in a game

Design a game that uses the features of a STOP:bit. The following examples could be provided as stimulus.

Quick reaction challenge - The STOP:bit randomly turns green. Players must press Button A only when the green light shows. If they press too early (red or amber), they're out or lose a point.
Add-ons could include: Display "Too early!" or "Great job!" messages on the LED screen

Traffic light dice – When the STOP:bit is shaken, it randomly lights up one colour. Each colour is linked to an action or rule. For example: Green = Jump 3 times, Amber = Balance on one leg, Red = Do a slow-motion walk

Why is this relevant? (Real world connections)

Through these activities, students are building simplified versions of embedded systems which is the same kind of technology used in traffic lights, pedestrian crossings, public transport, smart appliances, and vehicles. By coding the STOP:bit, students are learning how real-world devices are programmed to respond to inputs and control outputs safely and predictably.

Assessment

Observation can be used to check students' ability to carry out tasks aligned to the Australian Curriculum. We have included some suggested questions for teachers to reflect on and to guide these observations.

Checking for understanding

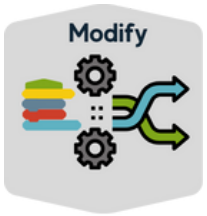
- Were the students able to describe and explain their algorithm and code, including iteration, branching and variables?
- Did their program reflect their algorithm?
- Can they explain what changes they made to the program to make their code more efficient? For example, using a variable instead of repeating code blocks.
- Has the student tested their program and identified any bugs? Can they describe what didn't work and how they fixed it?
- Have they considered an improvement or new feature they could add?

For more assessment resources we recommend the Assessment resources on the Digital Technologies Hub.

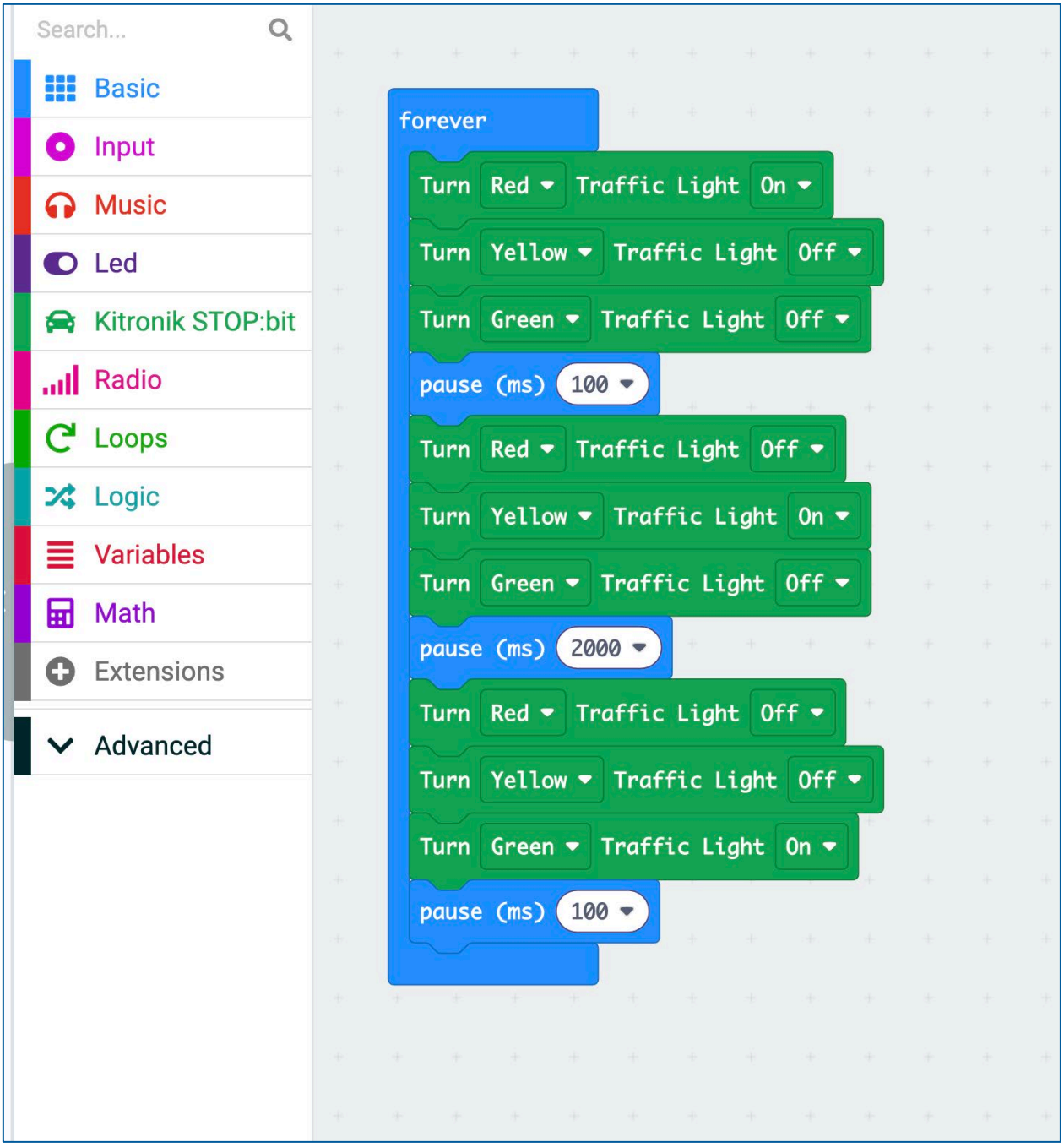
Supporting materials – PRIMM Programming – traffic lights

This activity will follow the PRIMM pedagogical framework. The framework is a structured approach to teaching computer programming that emphasises reading and understanding existing code before writing new programs.

Provide student with the sample code below and proceed through the stages.

	Students are given a piece of code and asked to **predict** what it will do before running it. • What do you think this code is designed to do? • Which lights will turn on first? • How long do you think each light will stay on? • Are there any patterns you can spot in the code? • Will this run continuously or stop after one cycle?
	Students **run** the code to see what it actually does. • What happened when you ran the code? • Was it what you expected? Why or why not? • Which part of the code controls each part of the output?
	Students **investigate** how the code works. • What does each line/block of code do? • How is the timing of each light controlled? • What would happen if you changed this number or moved this block? • Can you find variables? What are they storing or controlling?
	Students **modify** the code to change its behaviour. **Modification ideas:** • Change the duration each light stays on • Swap the order of the lights • Add a button input to switch between normal and emergency mode • Use the micro:bit LED display to show a pedestrian signal or countdown **Guiding questions:** • What would you change to make the light stay on longer? • How could you use a button to control when the lights start? • What happens if you add a pause or change the loop? • How can you test whether your changes worked?
	Students **make** their own version or an entirely new program using what they've learned. **Project ideas:** • Design a traffic light system for a pedestrian crossing • Create a level crossing warning signal • Program a set of lights that react to varying light levels **Guiding questions:** • What problem do you want your program to solve? • What will your traffic light (or similar system) need to do? • What inputs will you use? (e.g. buttons, sensors)

Sample MakeCode project – Traffic lights



To access project

https://makecode.microbit.org/S44603-25913-26846-33359

Australian Curriculum

Depending on the challenges undertaken, these activities could be suitable for students between Year 3 and Year 6 Digital Technologies.

Digital Technologies

Students in Years 3 and 4 learn to:

- follow and describe algorithms involving sequencing, comparison operators (branching) and iteration (AC9TDI4P02)
- implement simple algorithms as visual programs involving control structures and input (AC9TDI4P04).

By the end of Year 4, students follow and describe simple algorithms involving branching and iteration and implement them as visual programs.

Students in Years 5 and 6 learn to:

- design algorithms involving multiple alternatives (branching) and iteration (AC9TDI6P02)
- implement algorithms as visual programs involving control structures, variables and input (AC9TDI6P05).

By the end of Year 6, students design algorithms involving complex branching and iteration and implement them as visual programs including variables.