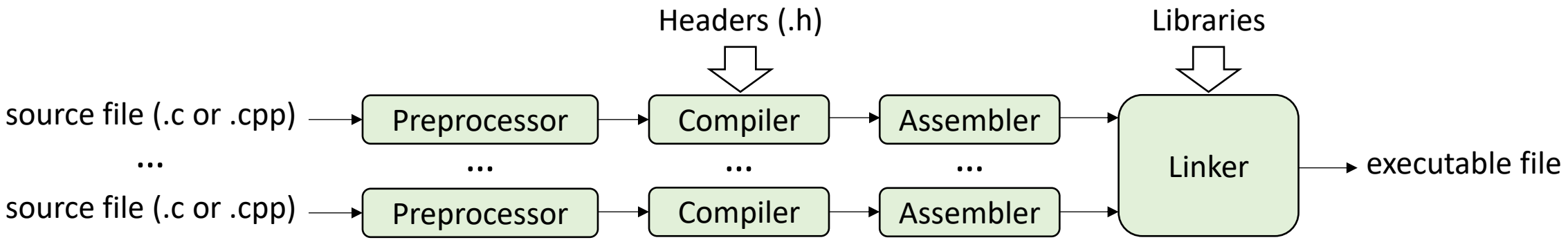
Lancaster University

SCC.131: Digital Systems
The micro:bit radio module

Ioannis Chatzigeorgiou
(i.chatzigeorgiou@lancaster.ac.uk)

The Big Picture for Weeks 7 to 12

Lancaster University



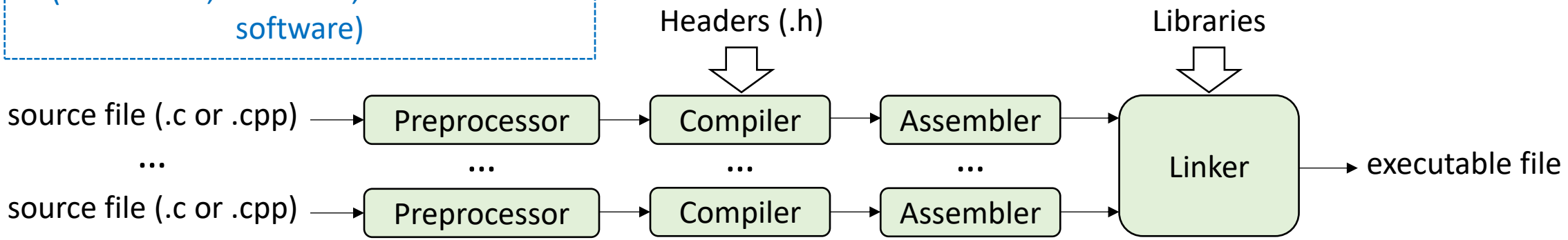
The Big Picture for Weeks 7 to 12

Lancaster University

Example of a digital embedded system

Week 7, Lectures 1 and 2

(motivation, hardware, firmware and software)



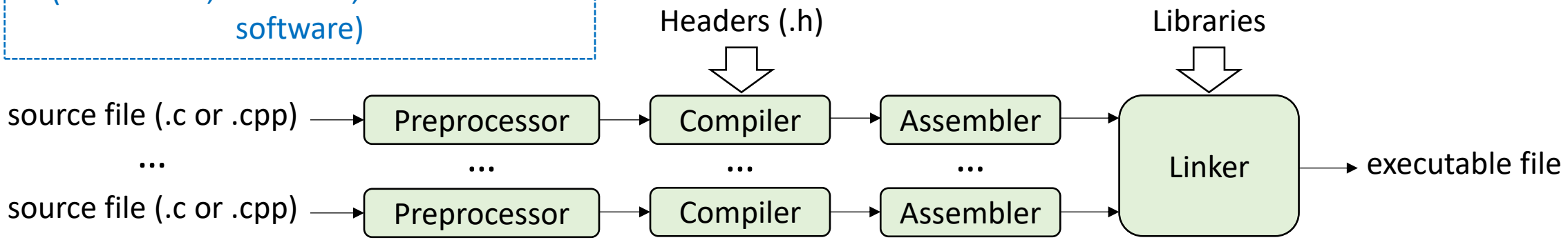
The Big Picture for Weeks 7 to 12

Lancaster University

Example of a digital embedded system

Week 7, Lectures 1 and 2

(motivation, hardware, firmware and software)



Week 9, Lecture 1 – Compiler, assembler, linker and loader

Week 12, Lecture 2 – Build automation

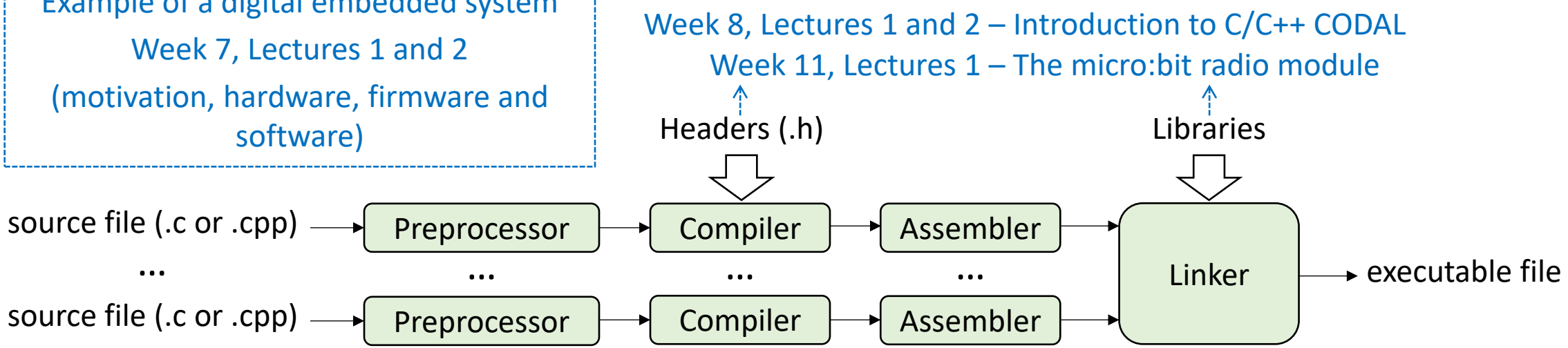
The Big Picture for Weeks 7 to 12

Lancaster University

Example of a digital embedded system

Week 7, Lectures 1 and 2

(motivation, hardware, firmware and software)



Week 9, Lecture 1 – Compiler, assembler, linker and loader

Week 12, Lecture 2 – Build automation

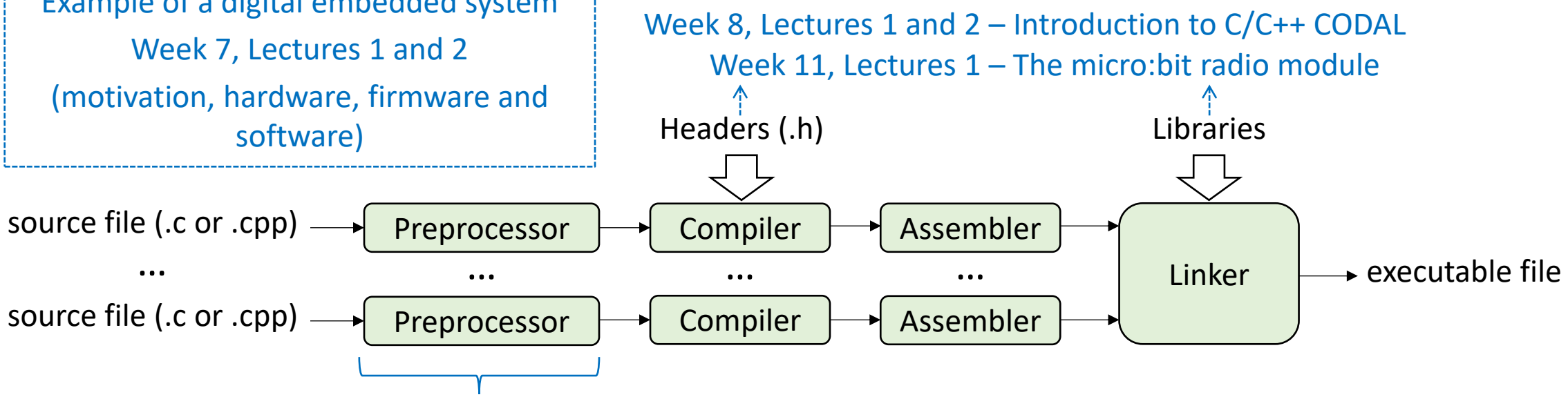
The Big Picture for Weeks 7 to 12

Lancaster University

Example of a digital embedded system

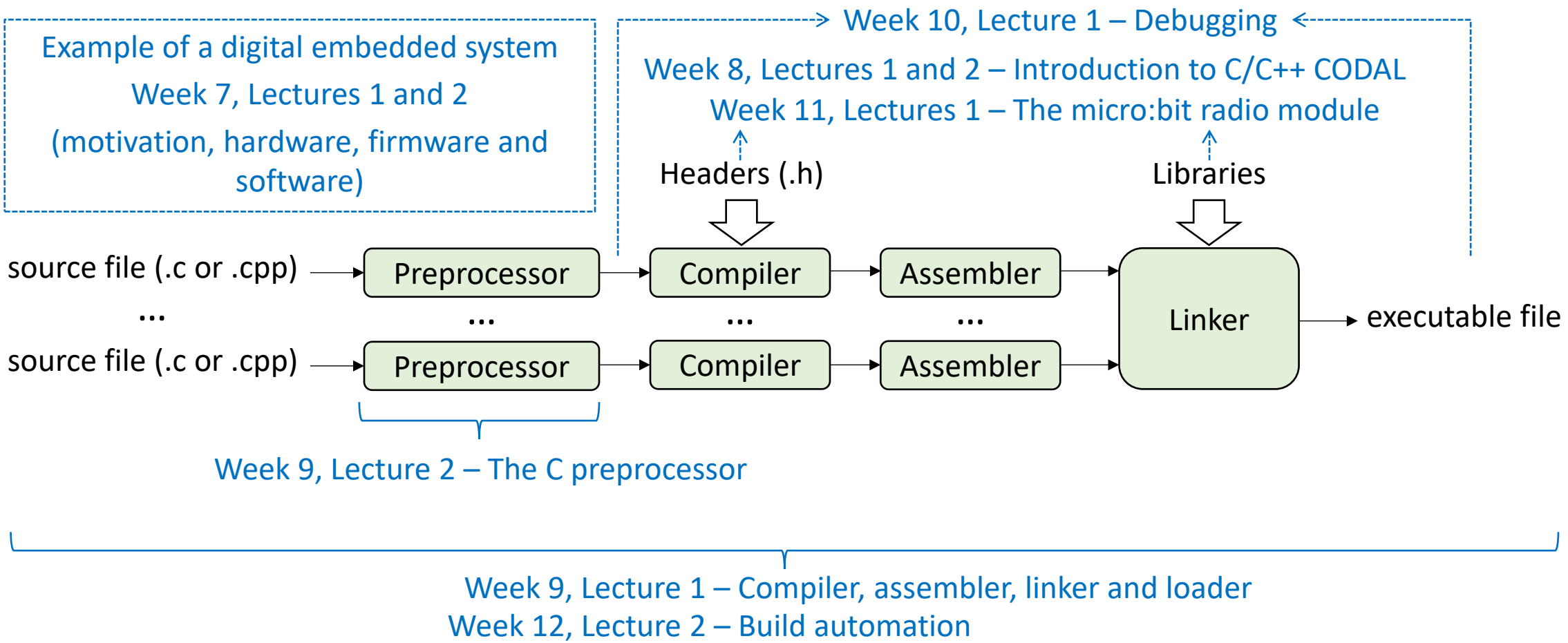
Week 7, Lectures 1 and 2

(motivation, hardware, firmware and software)



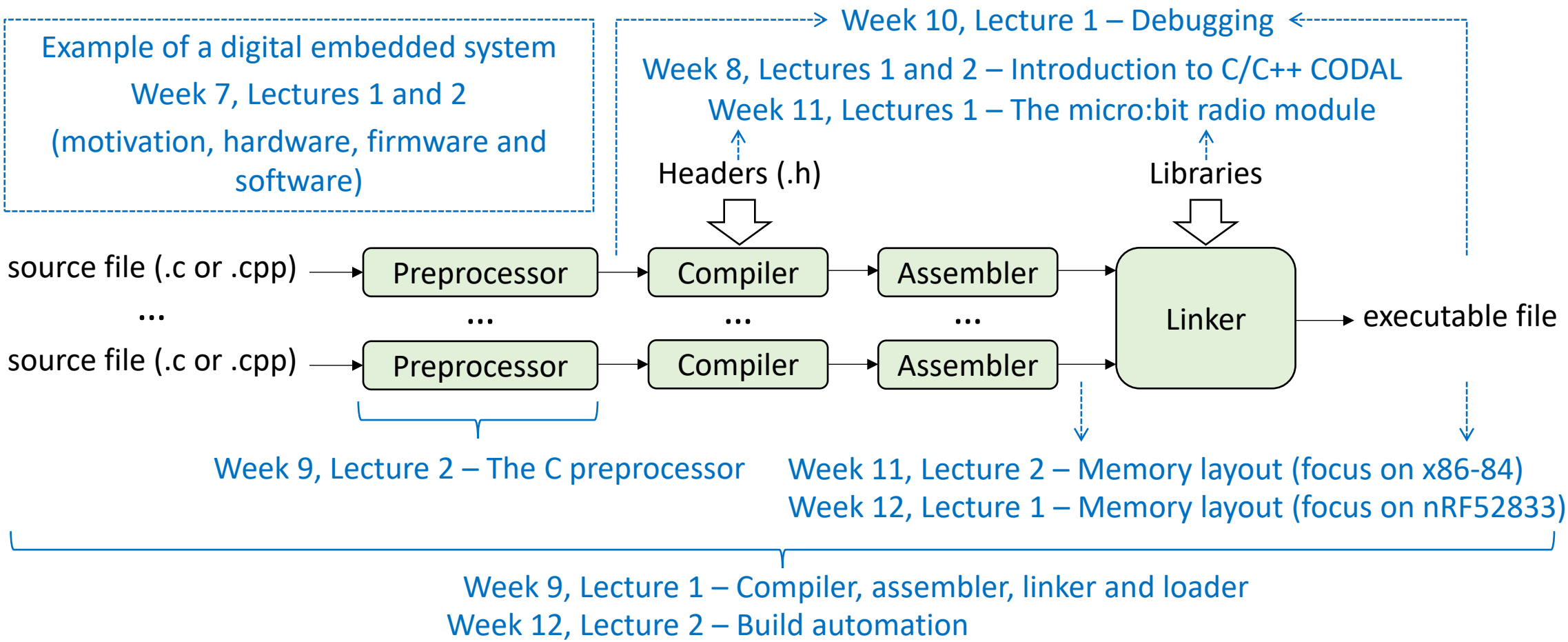
The Big Picture for Weeks 7 to 12

Lancaster University



The Big Picture for Weeks 7 to 12

Lancaster University



Reminder: Using **myLab**

Lancaster University

- How to connect remotely to the Lancaster University network using **GlobalProtect**: https://portal.lancaster.ac.uk/ask/vpn
- How to access **myLab**: https://portal.lancaster.ac.uk/ask/mylab/
- Which virtual machine to choose: **SCC Lab**
- How to access your **personal** (H:) **drive** in Windows: https://portal.lancaster.ac.uk/ask/personal-filestore/
- Who to ask if you experience issues with the above: https://portal.lancaster.ac.uk/ask/contact/ask-team/

Reminder: The MicroBit class

Lancaster University

- Classes of MicroBit that have been covered:

```
MicroBit uBit;
```

- `uBit.i2c`
- `uBit.storage`
- `uBit.serial`
- `uBit.MessageBus`
- `uBit.buttonA`
- `uBit.buttonB`
- `uBit.buttonAB`
- `uBit.display`
- `uBit.accelerometer`
- `uBit.compass`
- `uBit.thermometer`
- `uBit.io`
- `uBit.log`
- `uBit.radio`

: Week 8, Lecture 1

: Week 8, Lecture 2

: Week 10, Lecture 1

: Week 11, Lecture 1 (this one!)

Using the micro:bit for wireless communication

Lancaster University

- Micro:bit devices are equipped with a Nordic Semiconductor nRF52833 System on Chip (SoC). This chip contains a built-in **2.4GHz radio module**.
- The radio module has been primarily designed to run the **Bluetooth Low Energy** (BLE) protocol. It also supports the **802.15.4-2006 standard**, which is the basis (physical layer and medium access control) for ZigBee, WirelessHART, 6LoWPAN and other low-rate wireless personal area networks.
- It can also be placed into a **simpler proprietary mode of operation** that allows a micro:bit device to broadcast general purpose data packets to other micro:bit devices.
- For **privacy**, all devices look identical. If you want to be able to identify yourself, you need to add information to your transmitted data.

Capabilities of proprietary radio mode

Lancaster University

- **Bandwidth and frequency**: 1MHz narrowband, typically 2.407 GHz (configurable in the 2.400 GHz - 2.499 GHz band).
- **Transmission rate**: 1Mbps.
- **Maximum transfer unit**: Typically, 32 bytes (reconfigurable, up to 1024 bytes).
- **Encryption**: None
- **Error detection**: 16-bit hardware cyclic redundancy check (CRC) coding.
- **Transmission power**: Eight user-configurable settings from 0 (-30 dBm) to 7 (+4 dBm).
- **Transmission range**: Approximately 20 m at 0 dBm.

Definition of dBm

Lancaster University

- The strength of the received signal is expressed in decibels (dB) with respect to 1 mW.
- If P is the received power in Watts (W), the ratio $\frac{P}{1\ \mathrm{mW}}$ indicates how much stronger than 1 mW is P.
- To go from W to dBm, and from dBm to W, use:

$$P\ (\text{in dBm}) = 10 \times \log_{10}\left(\frac{P\ (\text{in W})}{1\ \mathrm{mW}}\right)$$

$$P\ (\text{in W}) = 1\ \mathrm{mW} \times 10^{\frac{P\ (\text{in dBm})}{10}}$$

4 dBm	2.5 mW
0 dBm	1 mW
-10 dBm	100 µW
-20 dBm	10 µW
-30 dBm	1 µW
-40 dBm	100 nW
-50 dBm	10 nW
-60 dBm	1 nW

Datagrams

Lancaster University

- A micro:bit device can transmit a *datagram* at a time, that is, a packet that can be up to 32 bytes long (default value).
- The datagram is a *sequence of bytes*, for instance:
  - An array of bytes, e.g., `uint8_t myArray[10];`
  - A sequence of characters, e.g., `ManagedString s("HELLO");`
  - A packet buffer, e.g., `PacketBuffer b(16);`
- Let `uBit` be an object of type `MicroBit`. To transmit a datagram, use `uBit.radio.datagram.send(`*`datagram`*`)`, where *`datagram`* can be one of the aforementioned types.
- To receive a datagram, use `uBit.radio.datagram.recv()`

Transmitting and receiving

Lancaster University

- Before transmission or reception can take place, the **radio** module should be **enabled** using `uBit.radio.enable()`
- The transmitting micro:bit can use `uBit.radio.datagram.send` to broadcast a datagram but the **receiving** micro:bit should use `uBit.radio.datagram.recv` only after a `MICROBIT_RADIO_EVT_DATAGRAM` event has been raised:

```
uBit.messageBus.listen(MICROBIT_ID_RADIO, MICROBIT_RADIO_EVT_DATAGRAM, onRx);
```

Monitor the radio component

Raise an event when a datagram is received

Call the event handler (choose any name) to read the datagram

ManagedString and PacketBuffer (1/4)

Lancaster University

- Both `ManagedString` and `PacketBuffer` are **managed types**. This means that they will automatically reserve and release memory, as needed – i.e., you do not need to explicitly allocate and free memory.
- Variables of type `ManagedString` are **immutable**, i.e., once created, they cannot be changed. However, they can be compared and joined to create other strings.

```
// Strings can be compared
ManagedString part1("HELLO");
ManagedString part2("micro:bit");
if (part1 == part2) uBit.display.scroll("SAME");
if (part1 < part2) uBit.display.scroll("LESS");
if (part1 > part2) uBit.display.scroll("MORE");
```

ManagedString and PacketBuffer (2/4)

Lancaster University

- Both `ManagedString` and `PacketBuffer` are **managed types**. This means that they will automatically reserve and release memory, as needed – i.e., you do not need to explicitly allocate and free memory.
- Variables of type `ManagedString` are **immutable**, i.e., once created, they cannot be changed. However, they can be compared and joined to create other strings.

```
// Strings can be joined to create a new string
ManagedString greeting("HAPPY NEW YEAR ");
ManagedString year(2024); // a value can be passed too!
ManagedString msg = greeting + year;
uBit.display.scroll(msg);
```

ManagedString and PacketBuffer (3/4)

Lancaster University

- Elements in arrays of type PacketBuffer can be changed at any time. A byte can be read or written to the buffer by simply dereferencing it with square brackets.

```
#include "MicroBit.h"           // The MicroBit header file
MicroBit uBit;                  // The MicroBit object

int main() {                    // C CODE for the TRANSMITTER
  uBit.init();                  // Initialise the device
  uBit.radio.enable();          // Enable the radio component
  PacketBuffer b(2);            // Create a sequence of two bytes
  b[0] = 255;                   // Set the value of the first byte
  b[1] = 10;                    // Set the value of the second byte
  uBit.radio.datagram.send(b); // Transmit packet as a datagram
}
```

ManagedString and PacketBuffer (4/4)

Lancaster University

```
#include "MicroBit.h"                                   // The MicroBit header file
MicroBit uBit;                                          // The MicroBit object

void onData(MicroBitEvent e) {                          // The event handler
    PacketBuffer b = uBit.radio.datagram.recv(); // Store the received datagram
    uBit.display.scroll(b[0]);          // Display the first byte of the datagram
}

int main() {                                            // C CODE for the RECEIVER
    uBit.init();                                        // Initialise the device
    uBit.radio.enable();                                // Enable the radio component
    uBit.messageBus.listen(MICROBIT_ID_RADIO, MICROBIT_RADIO_EVT_DATAGRAM, onData);
    release_fiber();
}
```

Other methods / functions of interest

Lancaster University

```
PacketBuffer b;
b = uBit.radio.datagram.recv();
int a = b.getRSSI();
```

Method `getRSSI()` retrieves the received signal strength indicator (RSSI), which is measured in dBm, of the most recently received datagram.

```
uBit.radio.enable();
uBit.radio.disable();
```

Whereas `enable()` initialises the radio component of micro:bit for transmission/reception, `disable()` disables this component for use as a multipoint sender/receiver.

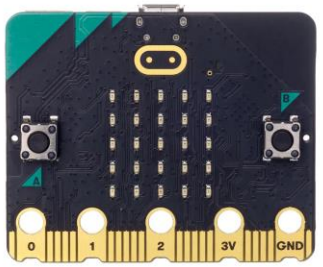
```
uBit.radio.setGroup(10);
```

Users can define a group to which their micro:bit devices belong. Datagrams sent will only be received by other micro:bits in the *same* group. If a group is not specified, the default group of 0 will be used.

Bring it all together

Lancaster University

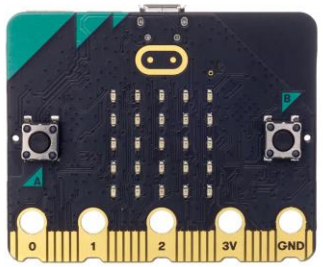
Transmitter



Select a number between 0 - 9 and send it.

- **Button A**: Decrease value by 1 (return to 9 if 0 has been reached).
- **Button B**: Increase value by 1 (return to 0 if 9 has been reached).
- **Buttons A and B**: Transmit the value.

Receiver



Print the received value on the display.

- **Button A**: Scroll the received *power* across the display.
- **Buttons A and B**: Clear the display.

Building the transmitter (1/2)

Lancaster University

Create a buffer that can hold a 1-byte packet. →

Initialise and print the content of the buffer on the display. →

Set up listeners for buttons A, B and AB. →

```
#include "MicroBit.h"

MicroBit uBit;

PacketBuffer b(1);

// -> Event handlers go here (they have been included in the next slide)

int main()
{
  uBit.init();
  uBit.radio.enable();

  b[0] = 0;
  uBit.display.print(b[0]);

  uBit.messageBus.listen(MICROBIT_ID_BUTTON_A, MICROBIT_BUTTON_EVT_CLICK, onButtonA);
  uBit.messageBus.listen(MICROBIT_ID_BUTTON_B, MICROBIT_BUTTON_EVT_CLICK, onButtonB);
  uBit.messageBus.listen(MICROBIT_ID_BUTTON_AB, MICROBIT_BUTTON_EVT_CLICK, onButtonAB);

  release_fiber();
}
```

Building the transmitter (2/2)

Lancaster University

If `b[0]>0` is true, reduce the value of `b[0]` by 1, otherwise set it to 9.

```
void onButtonA(MicroBitEvent e)
{
  b[0] = (b[0]>0) ? (b[0]-1) : 9;
  uBit.display.print(b[0]);
}

void onButtonB(MicroBitEvent e)
{
  b[0] = (b[0]<9) ? (b[0]+1) : 0;
  uBit.display.print(b[0]);
}

void onButtonAB(MicroBitEvent e)
{
  uBit.radio.datagram.send(b);
}
```

If `b[0]<9` is true, increase the value of `b[0]` by 1, otherwise set it to `0`.

Transmit the packet in the buffer as a datagram.

In general:

```
variable = condition ? value_if_true : value_if_false
```

This is known as the **ternary operator**.

Building the receiver (1/2)

Lancaster University

Create a buffer but do not specify the length of the packet that it can hold.

Set up a listener for the radio component.

Set up listeners for buttons A and AB.

```
#include "MicroBit.h"

MicroBit uBit;

PacketBuffer b;

// -> Event handlers go here (they have been included in the next slide)

int main()
{
  uBit.init();

  uBit.messageBus.listen(MICROBIT_ID_RADIO, MICROBIT_RADIO_EVT_DATAGRAM, onData);
  uBit.messageBus.listen(MICROBIT_ID_BUTTON_A, MICROBIT_BUTTON_EVT_CLICK, onButtonA);
  uBit.messageBus.listen(MICROBIT_ID_BUTTON_AB, MICROBIT_BUTTON_EVT_CLICK, onButtonAB);

  uBit.radio.enable();

  release_fiber();
}
```

Building the receiver (2/2)

Lancaster University

Obtain the datagram and store it in the packet buffer. Then, display the first byte, i.e., `b[0]`.

Scroll the received signal strength indicator (RSSI) across the display.

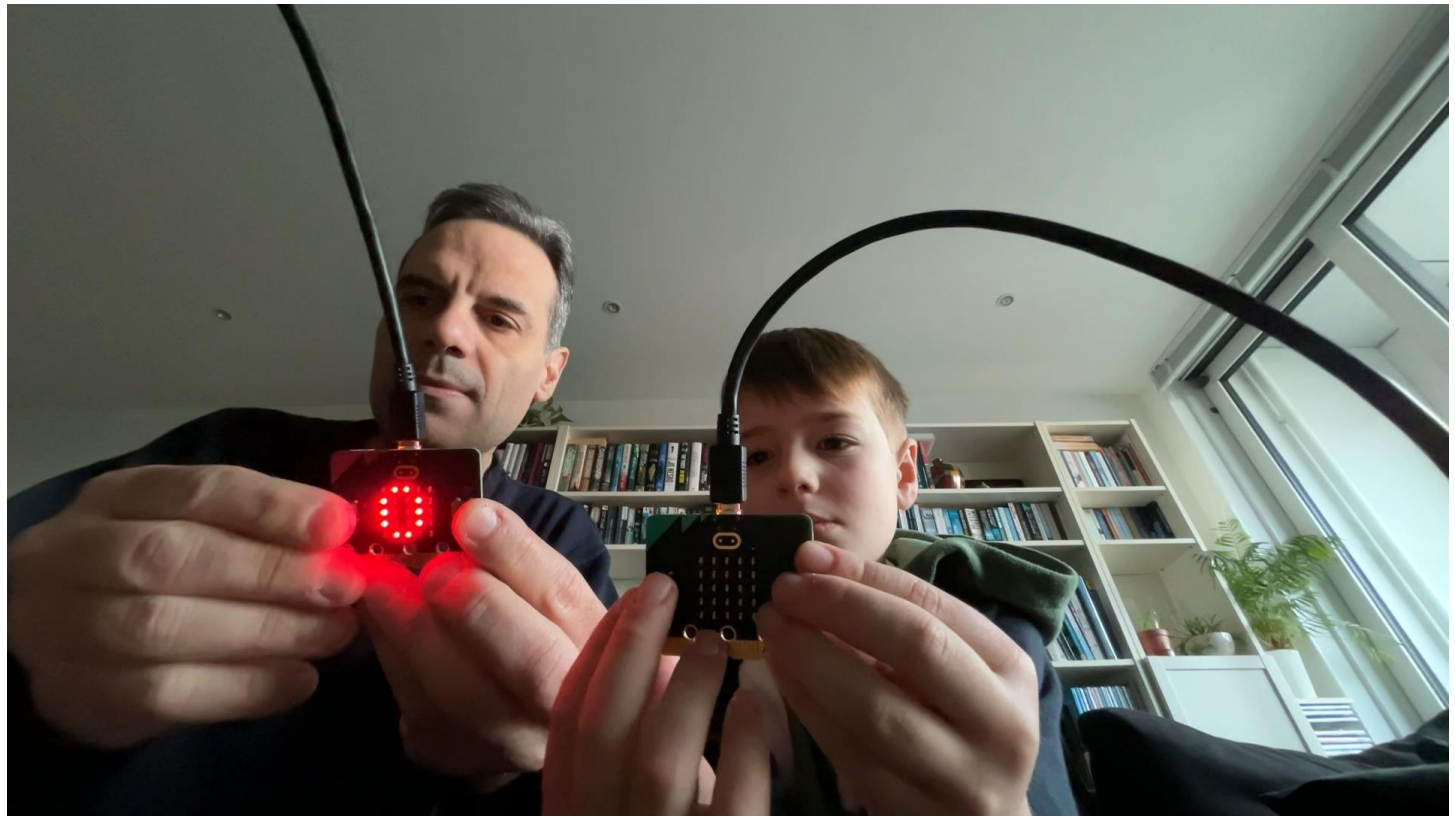
```
void onData(MicroBitEvent e)
{
  b = uBit.radio.datagram.recv();
  uBit.display.print(b[0]);
}

void onButtonAB(MicroBitEvent e)
{
  uBit.display.image.clear();
}

void onButtonA(MicroBitEvent e)
{
  uBit.display.image.clear();
  uBit.display.scroll(b.getRSSI());
}
```

Event handler triggered upon receipt of a datagram.

Recorded demonstration



The video is available on eStream. Log in via “SSO Login”.

Summary

Lancaster University

Today we learnt about:

- The capabilities of micro:bit for wireless communication.
- The characteristics of the proprietary radio mode at 2.4 GHz.
- The concept of a datagram and key instructions for transmitting and receiving datagrams using variables of type `PacketBuffer`.
- How to set up a listener for the radio component and how to call an event handler when a datagram is received.
- How to create groups and how to measure the received signal strength.
- Steps on how to build a simple one-way wireless communication system.

Resources

Lancaster University

- Using the micro:bit radio: https://lancaster-university.github.io/microbit-docs/ubit/radio/
- API documentation for:
  - The datagram: https://lancaster-university.github.io/microbit-docs/ubit/radiodatagram/
  - The `ManagedString` type: https://lancaster-university.github.io/microbit-docs/data-types/string/
  - The `PacketBuffer` type: https://lancaster-university.github.io/microbit-docs/data-types/packetbuffer/
- Morse transmitter: https://www.i-programmer.info/programming/148-hardware/14390-microbit-morse-transmitter.html