1.60 Basic programme design

A guideline to the design and structure of embedded system programmes using the superloop architecture

EE108 – Computing for Engineers

1

Overview

Aims

- Learn a simple but general approach to designing programmes

Learning outcomes – you should be able to…

- Design short programmes using the superloop architecture
- Structure the superloop into input-process-output (sometime called the sense-think-act) steps

2

Programme organisation

3

Decomposition of large problem into smaller problems…

- To make code readable and understandable, it is best to divide it up into self contained chunks of about 25 lines or so
- The self-contained chunks are functions

General outline of programme

- For simple embedded systems (as used in this module), we use the superloop architecture which generally has the following outline

```
…
// Code that energia uses
// behind the scenes
int main(void) {
  // setup and initialization
  setup();

  // superloop
  for (;;) { // repeat forever
    loop();
  }
}
```

```
…
void setup(void) {
  // your setup and initialization code goes
  // here
}

void loop(void) {
  // add code that should run repeatedly
  // until the system fails or powers down
  // here
}
```

20 October 2020

3

Programme design

4

Once we have decided to use the superloop architecture, designing the programme mostly involves designing the superloop body

The basic strategy is to divide the superloop body (i.e. the loop function) into the following 3 steps

- **Input (including sensing and events)**
- **Processing**
- **Output (including actuation and side-effects)**
- We refer to these steps as **Input-process-output** or **sense-think-act**

System setup (i.e. the setup function) should be implemented according to the needs of the superloop body

20 October 2020

4

Example


```
…
// STEP 0: setup
void setup() {
  // set up LEDs and whatever else is needed by superloop
  …
}

void loop() {
    static int prevAdcValue = 0;
    int adcValue;
    int smoothAdcValue;

    // STEP 1: input/sensing
    adcValue = analogRead(POT);

    // STEP 2: processing
    // average the current and previous sample for basic smoothing
    smoothAdcValue = (adcValue + prevAdcValue) / 2;
    prevAdcValue = adcValue; // remember adc value for next time function runs

    // STEP 3: output/actuation
    showValueOnLeds(smoothAdcValue);
}
```


Programme design contd.

Input/sensing/events

- In general input involves anything that originates outside the system and needs to be read or detected
- E.g. button press, analogue values read via Analogue to Digital Converter (ADC) peripheral, the receiving side of (serial) communications, etc.

Processing

- Any transformation of the input (e.g. converting between the raw numeric representation – obtained using the ADC – of temperature sensor's analogue voltage level into a temperature in celcius)
- Any calculations using the input data and/or "state" of the programme

Output/actuation/side-effects

- Providing output to something external to the system, e.g. lighting a LED (digital on/off), actuating a fading LED or a motor drive (PWM output), transmitting (serial) communications, etc.
- Side effects such as delaying for a certain period of time

Contd.

It is often desirable to do all input first, followed by all processing, followed by all output/actuation. However, this is not always possible and, it does not always result in the easiest to understand programmes.

Instead you should consider input-process-output to be a guideline which helps you to figure out what should be done and in what order.

For example, it might be more natural to design the programme to consist of a sequence of input-processing-output stages, i.e. input-processing-output-1, followed by input-processing-output-2, etc.

In general, the main processing stages in your programme depend on some input and the output in your programme depends on some processing.

Whether there is one input, processing, and output stage or multiple input, processing, and output stages is a consequence of whether the input, or processing, or output can be decomposed into somewhat independent pieces.

7

Examples – general/abstract version

One each of input, processing, output

```
void loop() {
  doAllInput();
  doAllProcessing();
  doAllOutputs();
  …
}
```

Multiple input-processing-output stages that have some interdependencies

```
void loop() {
  doInput1();
  doProcessingDependsOnInput1();
  doOutput1DependsOnProcessing1();

  doProcessing2DependsOnInput1();
  doOutput2DependsOnProcessing2();
  …
}
```

Multiple independent input-processing-output stages in sequence.

```
void loop() {
  doInput1();
  doProcessingDependsOnInput1();
  doOutput1DependsOnProcessing1();

  doInput2();
  doProcessing2DependsOnInput2();
  doOutput2DependsOnProcessing2();
  …
}
```

Finite state machine to allow different combinations of input-processing-output stages on each repetition of the loop

```
See later slide…
```


8

Examples – concrete versions

One each of input, processing, output

```
void loop() {
  inputFromADC();
  processADCValue();
  showADCValueOnLeds();
  …
}
```

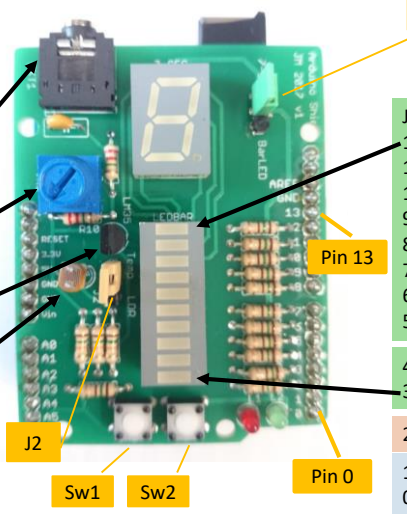
Multiple input-processing-output stages that have some interdependencies

```
void loop() {
  inputFromButton();
  processButtonToIdentifyPress();
  showButtonPressOnLed1();

  processButtonToIdentifyLongPress();
  showLongPressOnLed2();
  …
}
```

Multiple independent input-processing-output stages in sequence.

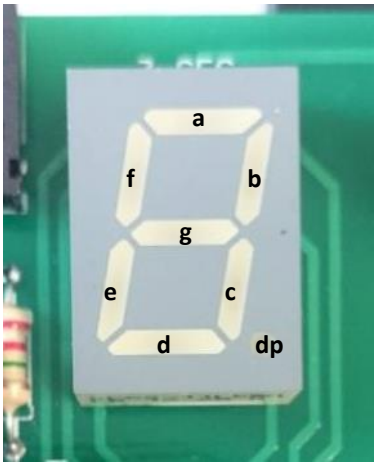
```
void loop() {
  inputFromButton();
  processButtonToIdentifyPress();
  showButtonPressOnLed1();

  if (buttonClicked) {
    inputFromADC();
    processADCValue();
    showADCValueOnLeds();
  }
  …
}
```


Reminder: EE108 daughterboard pin assignments



EE108 7 segment pins

11



Jumper J1 selectable
12 – BarLED10 / 7seg dp
11 – BarLED9 / 7seg a
10 – BarLED8 / 7seg b
9 – BarLED7 / 7seg c
8 – BarLED6 / 7seg d
7 – BarLED5 / 7seg e
6 – BarLED4 / 7seg f
5 – BarLED3 / 7seg g

Worked example – print click duration
(using input-process-output pattern)

12

Develop a short programme to do the following:

- We'll be using serial output and SW1 button input
- At programme start up print the programme name
- Thereafter, repeatedly do the following [i.e. in the body of the loop function]…
- Whenever SW1 is clicked (i.e. pressed and released), take note of the click duration (time between press and release) and increment a click counter.

  [Hint: if the button is first pressed note the time and then check the time again when the button is released]
- Print out the current time in ms, the number of clicks to date, and the duration of the most recent click
- Mark the input, processing, and output steps clearly in your solution

Worked example – light daughterboard LEDs
(using input-process-output pattern)

13

Develop a short programme to do the following:

- We'll be using LED1, LED2, the Bar LEDs and SW1 button input. We can optionally use serial output for debugging
- [Optional DEBUG] At programme start up print the programme name
- Initially BarLED1 should be on
- Thereafter, repeatedly do the following…
- When SW1 is pressed, light LED1 as long as it is held pressed down.
- Each time SW1 is clicked (i.e. pressed and released), increment the selected BarLED number (you'll need to keep track of this between calls to loop function).
- [Optional DEBUG] print out the selected LED number
- Switch on only the selected daughterboard LED. (This means that you must ensure that all other LEDs are off.)
- Mark the input, processing, and output steps clearly in your solution – there may be multiple input-process-output steps

13

Worked example – flash daughterboard LEDs
(using input-process-output pattern)

14

Develop a short programme to do the following:

- We'll use the Bar LEDs and SW1 for input
- Increment a counter and flash a Bar LED each time SW1 is clicked (pressed and released)
  - At first, just flash the BarLED1 every time.
  - Then modify the programme so that the number of the LED to flash is incremented after every 5 button clicks. Limit the range of LED numbers used to 0 through 3 inclusive – numbers outside this range should be wrapped back into the valid range.
- Write functions where appropriate to decompose the problem and simplify the loop function
- Mark the input, processing, and output steps clearly in your solution

14

Self test programme exercise

15

Q1. Develop a short programme to do the following:

- *When SW1 clicked choose a random bar LED number (1-10), choose a random repeat count (1-7 incl).*
- *Print out the randomly chosen LED number and repeat count*
- *Flash the randomly chosen led the specified randomly chosen number of times. (Use a function to help with this)*
- *Use comments to indicate clearly the INPUT, PROCESSING, and OUTPUT stages of the programme*

15

Self test programme exercise

16

Q2. Develop a short programme to do the following:

- *When SW1 clicked choose a random starting bar LED number (1-5), choose a random ending bar LED number (6-10), choose a random repeat count (1-7 incl).*
- *Print out the randomly chosen start and end LED number and repeat count*
- *Flash all LEDs from the start to end LED number the specified randomly chosen number of times. (Use a function to help with this)*
- *Use comments to indicate clearly the INPUT, PROCESSING, and OUTPUT stages of the programme*

16