Teachers’ notes for BOOKLET_76543_QB00700_BRANCH_Q_ art_3

Activity 1: Task planning and system design changes – see **TASK_PLAN_76543_QB00700_BRANCH_Q_Part_3.pdf** which is the ongoing plan.

Activity 3: System Design -
For this I used a low fat, low salt version of a methodology that’s been around since the early to mid 1980’s. I left the sugar in – otherwise it’s too hard to swallow. Ed Yourdan came up with a graphical method to describe system design in the late 1970’s. Later extensions to cope with realtime systems better were added by Ward-Mellor. There’s lots of books and information on the internet, tools for drawing the diagrams etc. You don’t need anything more than a pencil and paper though. There’s newer methodologies around like UML, but I find that the Yourdan / Ward-Mellor methodology is a better match for small microcontrollers running Flowcode, C or assembler. That’s just my opinion, some younger programmers may have an alternative view.

In brief, it’s a top-down design method. You start with a context diagram where a circle represents the system, inputs (generally on the left) represent the, usually hardware inputs to the system. Outputs (generally on the right) represent the usually hardware outputs from the system. Data flows to and from the system are represented by lines with arrows showing the direction of data flow.

The system design starts with the next level down. At this level and below, where a circle represents a process (think of it as a macro in Flowcode). As before, lines with arrows represent data flows. The only other symbol I use is two horizontal lines representing a data store (think of it as the global variables in Flowcode).

There’s a numbering system that tells you where you are in the hierarchy of the design. Number ‘1’ is the top level (your context diagram), below that, processes are numbered 1.something (say 1.5 for LED_task). If the LED_task needs expanding out to smaller processes, they become 1.5.something (say 1.5.1 for LED_set_LED_ON) etc. This will become clearer as the design gets implemented.

As I said it’s a stripped down version of the methodology and purists will scream about the lack of arrows with double heads, dotted lines, dotted circles and loads of other stuff. It’s fine for our purposes though. It’s also great to explain how the system works to non-technical people with no knowledge of programming.

It’s a lot like flowcharts – they are meant to be easy to read and they are. Making them takes a bit of practice and there’s a few tips and tricks that help. This is something for the “underpinning knowledge” topic that will come later.

The remainder of the design uses the condition/action tables described in the Pearson books.

Activity 3: System Assembly and programming -
System assembly is trivial – just plugging a few components into the breadboard and connecting it to the Matrix HP4988 board.

On the programming part – the second program we always write for a new system is the “hardware test” program (the first program is the “flashing LED” at one flash per second program). There’s two reasons for this:

(1) We make sure that all the pieces of hardware do what we expect.
(2) If, later on during development, the system misbehaves, it could be due to a bug in the program or a hardware fault that has developed in the system.

The advantage of having a “hardware test” program that we know works is that we can download it to the system and run it. If it behaves as expected – the problem is not hardware related – it is most likely to be a bug in our new program.

It is not unusual for hardware to fail due to static discharge or mechanical damage due to something being dropped on the PCB or the PCB being dropped etc. You can waste a lot of time hunting problems that you think is a program bug when something in the hardware has failed.

The program I used for testing the egg timer system is a simple “hardware test” program written earlier with the addition of a test for the new LED and piezo sounder.