UNIVERSITY OF CAMBRIDGE

Raspberry Pi – Why, What, How

Alan Mycroft

Computer Laboratory, University of Cambridge

http://www.cl.cam.ac.uk/users/am/

and

Raspberry Pi Foundation

http://www.raspberrypi.org

15 June 2012

What's gone wrong with school-age computing?

My initial perspective: interviewing 18-year-old applicants for the Computer Science BA at Cambridge.

Question: tell us about the most interesting program you've written.

Answer c. 1990: used a BBC micro to write a program which ...

Answer c. 2010: written a web page and used a package to ...

During 2000–2010 applications for Computer Science halved – not just at Cambridge.

We can't produce enough *Computer Science* graduates for industry demand. [But beware that some degrees have low employment rates.]

What's gone wrong with school-age computing (2)

UNIVERSITY OF CAMBRIDGE

Reason 1:

- The bête noir: ICT Key Skills (a new name for the subject called "Typing" when I was at school).
  Focus on presentation. Bright kids knew it already.
  Gove/Schmidt "not fit for purpose".
  Royal Society: Digital Literacy vs CS vs IT.

But *teachers* have tried their best with the prescribed syllabus (with some seditious heroes doing "proper CS").

Delicate issue: moving sensitively from "The ICT teacher was appointed because they already edited the school glossy" to where we want to be. Management (and Government) underestimate this.

What's gone wrong with school-age computing (3)

Reason 2:

- Over-packaged and fragile hardware and software raises the "activation energy" for doing anything.
  [Poorer families] "You're not taking the back off the computer"
  "You installing Linux has lost all my photos."
  [Richer families] "Where are the screws to enable school-age children to take their expensive Macbook Air to bits?"
  [Schools] "How clever of XXX causing all the computers in the school computer suite to show that amusing picture."
  [Software] "It's all so complicated nowadays that I can't see how to start doing anything."
  Windows lacks pre-installed software too.

The fashions for "Computer Programming"

In the UK (and Europe and North America) "Computer Programming" in the popular mind goes in and out of fashion on a 25-year cycle – 1960's, late 1980's, 2010's – often brought about by some change in technology.

Important not to repeat the boom-and-bust "everyone should be a programmer".

- separation into "Digital Literacy" and "Computer Science" is useful
- but important to realise that these are just parts of a spectrum – and children's talents are also a spectrum.

Danger of over-emphasising 'programming'

Dictionary.net etc:

"Programming. The most fun you can have with your clothes on."

While many people here agree with this, it's not generally held, also:

"Programming. A pastime similar to banging one's head against a wall, but with fewer opportunities for reward."

- Important to teach "Computational Thinking" (Wing 2006, also YouTube)
- Computational thinking – "as fundamental as reading, writing and arithmetic"
- Incestuous: "Computing and Computers enable the spread of Computational thinking"

E.g. how does one sort a pack of playing cards? An algorithm.

See A-level introduction on `www.cl.cam.ac.uk/users/am/teaching`

"But you're from Cambridge – what about the real world?"

Well, as a University we're certainly at one extreme.

But we suffer from the same forces that affect us all.

It's important for the community to get children of all abilities interested in computing by doing 'fun' things – not just 'proper academic things'. E.g. programming filters which 'adjust' the view of the school web pages when selected .... E.g. games programs.

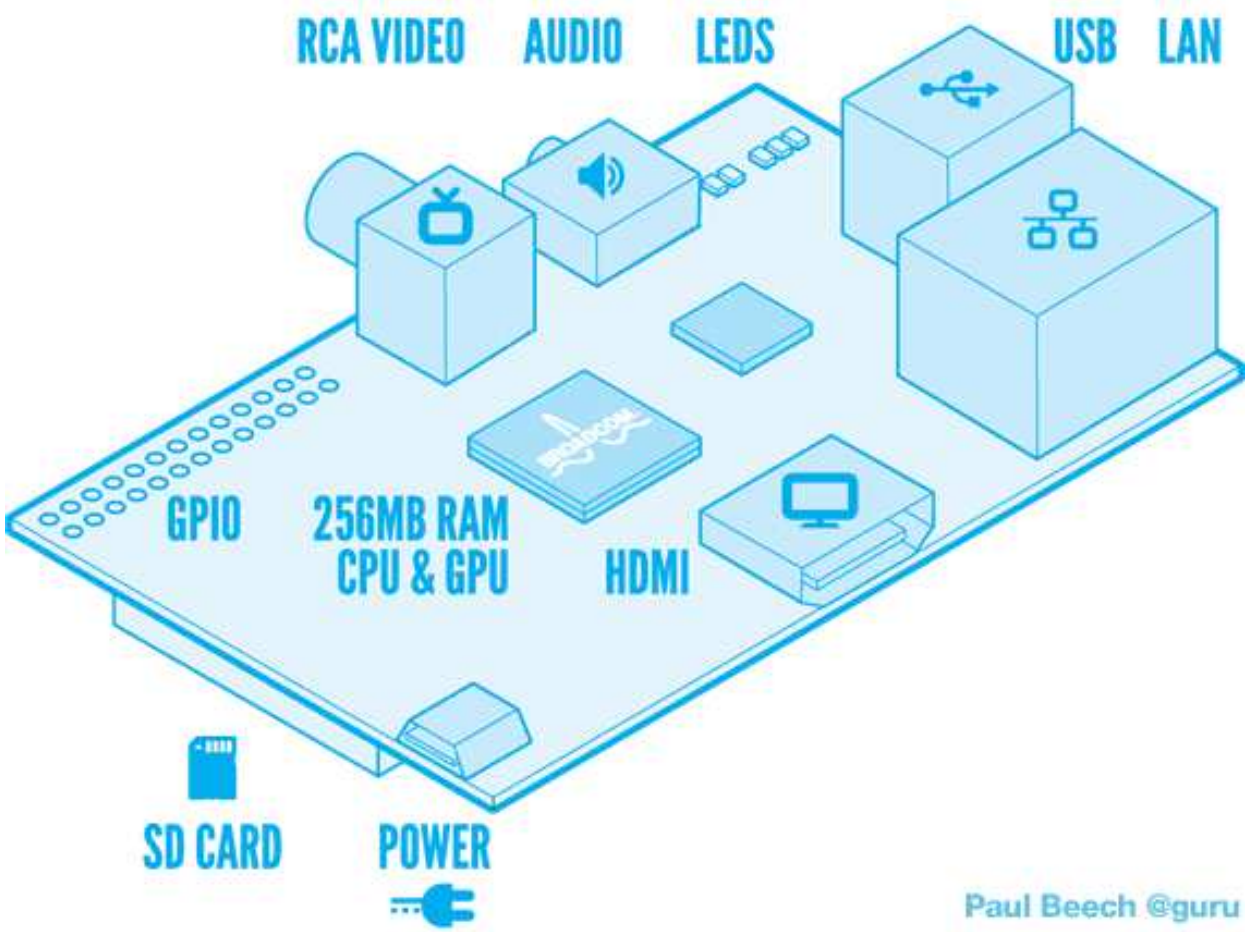
We want schools to initiate Computations Thinking, and open opportunities to *all* pupils. Of course, over years some pupils will develop more than others and we'll still be looking to recruit the best, but we want to raise the standard of all – this benefits UK plc too.

Spiral development – both engineering and teaching

- Programmers often talk about the spiral development method – write a full system with unsophisticated components and gradually refine these into the final system.
- There's a danger of over-developing one component prematurely.
- Claim: this also applies to teaching a subject like Computing.
- Teach principles/ideas over detail. E.g. not "the half-dozen different ways of drawing boxes round things in Microsoft Word".
- Someone later can teach more detail – but only for students who find it interesting (e.g. degree level). E.g. modern processor design as a refinement of 'fetch-execute cycle'.

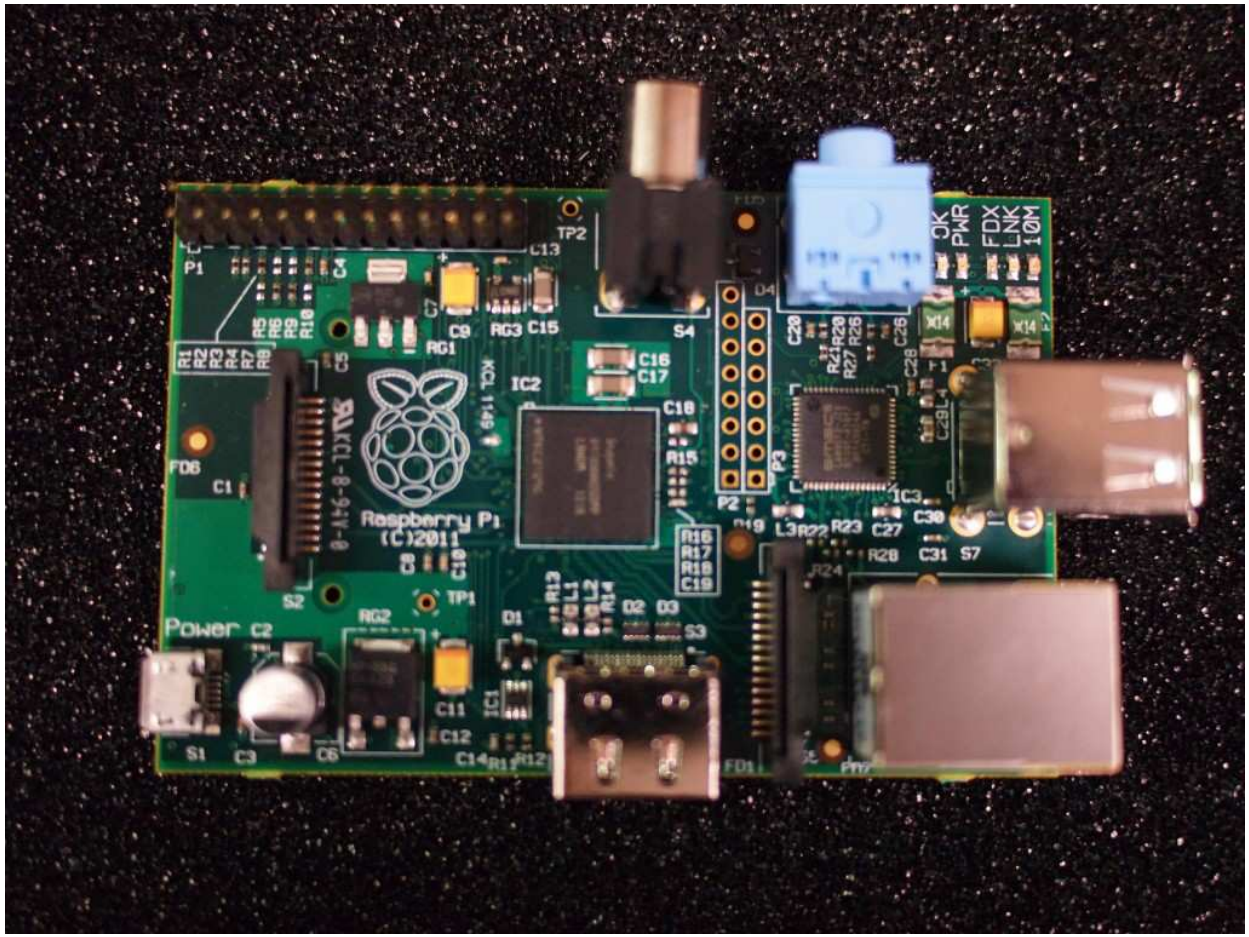
Raspberry Pi design (1)

UNIVERSITY OF CAMBRIDGE



Raspberry Pi design (2)

UNIVERSITY OF CAMBRIDGE



Why the Raspberry Pi?

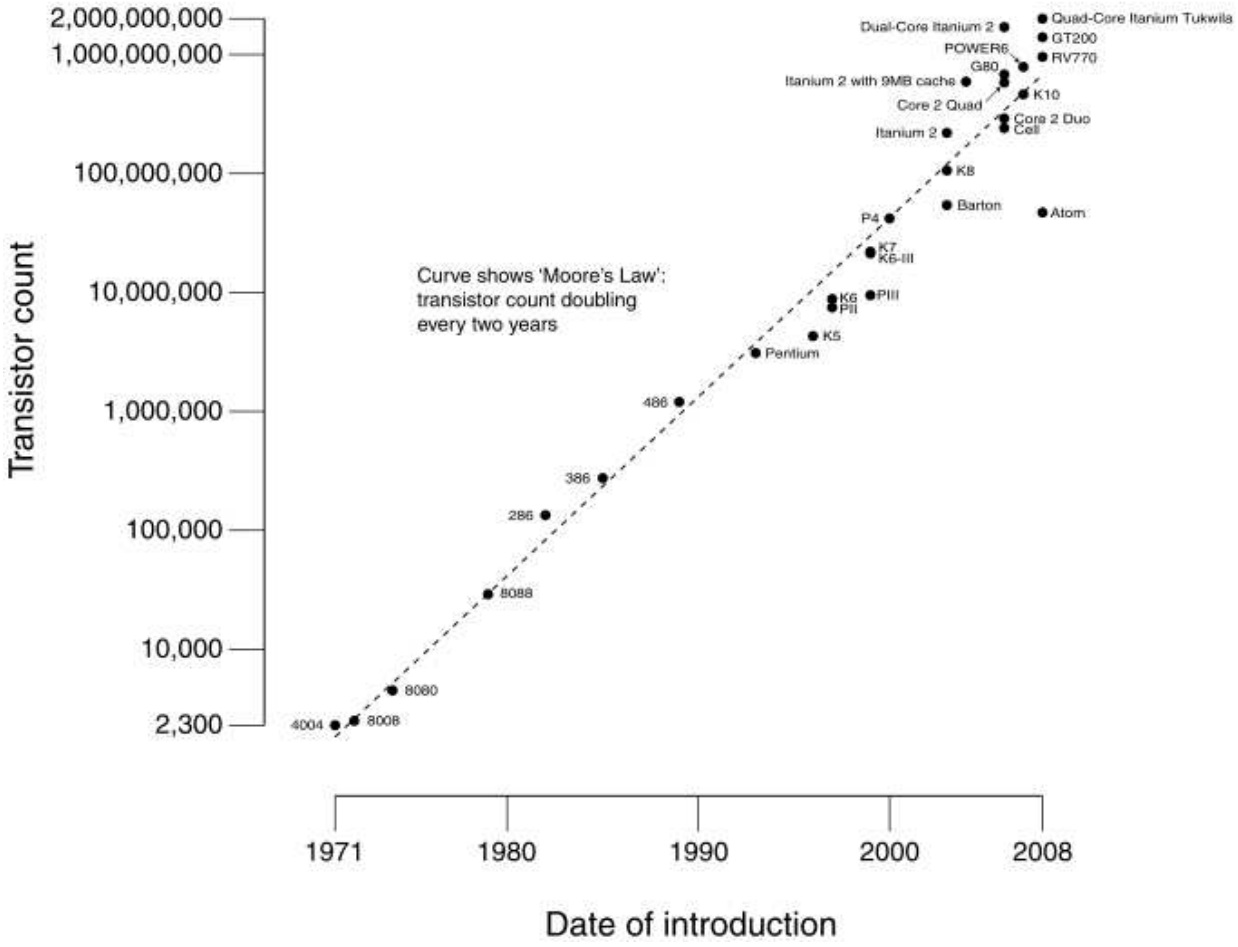
"It's just an slightly under-powered computer without a screen, and anything you can do on it you could do on a laptop"

- True, but ...
- ... it's cheap, you can't wreck it with software, it changes mindsets.
- access to physical pins.

What about Arduino?

- Similar aims, but ...
- ... Raspberry Pi runs Linux

Moore's Law


CPU Transistor Counts 1971-2008 & Moore's Law
Transistor count
2,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
Curve shows 'Moore's Law': transistor count doubling every two years
Dual-Core Itanium 2
Quad-Core Itanium Tukwila
POWER6
GT200
G80
RV770
Itanium 2 with 9MB cache
K10
Core 2 Quad
Core 2 Duo
Cell
Itanium 2
K8
Barton
Atom
P4
K7
K6-III
K6
PIII
PII
K5
Pentium
486
386
286
8088
8080
4004
8008
1971
1980
1990
2000
2008
Date of introduction

Moore's law (2)

Quoting Wikipedia:

> "Moore's law is a rule of thumb . . . whereby the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years"

I.e. twice as powerful every two years. Alternatively:

> "For a given design the size of chip needed (and hopefully the cost) halves every two years".

So, we can make a chip which has a 2005-state-of-the-art graphics card and a 700MHz ARM (and 256MB memory) for a very small number of dollars.

$25 computers

The BBC micro was more expensive than $25, but relied on the existing television for display. Everyone has as TV or a spare monitor.

Eben Upton (then also doing admissions at Cambridge) realised in 2006 that he could make a computer using an Atmel chip for around $25, using external keyboard and display.

Probably too geeky.

But Moore's law caught up, and repeat the process with an ARM-with-graphics-card processor and put Linux on it.

Industry standard.

Who would make Raspberry Pi chips?

Remember: fabricating ('fabbing') a new chip costs tens of $M.

But: people already had built similar chips for HDTV set-top boxes.

All Linux needed was Upton to convince Broadcom that virtual memory hardware was just what we needed in a chip ...

[Broadcom have been extremely supportive.]

Commercial stuff

We decided to found the Raspberry Pi Foundation as a *charity*.

- lots of support from various sources who'd have wanted shares if we were a company.
- synergistic with the University being a charity.
- aggressively drove down the BOM ("bill of materials") for the Raspberry Pi hardware.
- Just enough profit margin to get big suppliers (Farnell, RS) to build and pay royalties to the Foundation.

But: success was even greater than anticipated – demand crashed their websites.

But what now?

Where we are:

- We've designed and are delivering cheap hardware.
- There's a fashion 'rush' to get one.
- But what happens next?
- Danger: parents play with it for a few hours (recovering their lost youth with a BBC Micro) and then it gets put on a shelf.

Luckily, the Government (Gove), Google (Schmidt), and exam boards have amplified the "Computer Science" theme.

But we need to avoid the "everyone should be a programmer" boom-and-bust, so need tools which give friendly introduction to programming (and Computational Thinking) ideas.

Critical needs: software, teaching materials and sharing community.

What programming software?

We like:

- Scratch
- Python

But we don't think suitable:

- C/C++ [too many ways to learn bad habits]. 'chainsaw'.

Java is OK, but there's a big learning curve centred around "design your class hierarchy first".

What user interface?

A desktop-style WIMP (Windows, Icons, Menus, Pointer) is attractive.

But the BBC-micro-style "BBC Basic is the command line" had an immediacy – or is this rose-coloured glasses?

Teaching materials

It's very difficult to find and share suitable teaching materials – there is a plethora of sites which individuals have created.

But there is too much fragmentation if everyone downloads different tools or creates their own.

Need a repository and advice for suitable course materials (probably community based). STEMNET (www.stemnet.org.uk)?

Andrew Hague's session "Educational User Manual for the Raspberry Pi" here.

Students sharing their work

It would be nice to recreate the 1980's 'buzz' whereby students could write computer games in their bedroom and sell them for a profit.

Money gives kudos to what can be seen a 'geeky' activity.

Do we want to support this?

If so, then do we help establish web-sites with upload/download for fee?

Or even an "App Store" model – perhaps even with an altruistic marker on each bit of software noting the percentage donated to charity?

Conclusions

We've seen

- *why* we designed the Raspberry Pi
- *what* is it *how* we created it.

It's created a buzz and offers a "do what you want, program on bare metal" platform.

What we are lacking (and the Raspberry Pi Foundation cannot provide) is getting a coherent set of materials we can use with it to teach Computational Thinking and principled ideas from Computer Science.

Abstract

This talk discusses the motivation and history of the Raspberry Pi Computer and speculates about future developments and uses. Highlighted drivers include the post-2000 loss of 'Computer Science knowledge' among applicants for the Computer Science degree at Cambridge, and a non-conventional interpretation of Moore's Law. As ever serendipity played a significant part (e.g. Broadcom's support for Eben Upton's use of their processor).

The big remaining issue is making sure the hardware is fully exploited educationally, and this depends on software. Does a standard Linux desktop serve as a good basis? Is it over-packaged for those who "really want to program on bare metal"? Scratch and Python offer good support for programming for certain groups – but was the convenience of BBC Basic just a rose-coloured view of history or did it offer a now-lost additional immediacy? How can sharing of material, both by teachers and students, best be achieved for Raspberry Pi, bearing in mind the disparate needs of different age groups? Do web sites suffice or would the "App Store" model work better?

Additional slides: algorithms

UNIVERSITY OF CAMBRIDGE

How do we sort a list of numbers (printed on cards) into order?

- Just do it
- Type them into Excel, click Data/Sort ...
- Break it down into simpler parts (Computational Thinking)

E.g. we can

- Find and remove the largest (and output it)
- Repeat until list is empty
- But how do we find the largest?
- Break it down into simpler parts (Computational Thinking)

Additional slides: algorithms (2)

- For older children we can work out how many "comparisons of two items" we make in total.
- But, is this the best possible?
- No – alternative algorithms can work faster
- Easy maths can get $n^2/2$ for the first sort, but $n \log_2 n$ for a better one.
- This has practical conseqences – these differ by a factor of 50 when $n = 1000$ and $25\,000$ when $n = 1\,000\,000$.