Using ARM-based boards in a second year course

by

Amanpreet Dhaliwal
B.Sc., Punjab University Chandigarh (India)1998
M.Sc. (Physics), Punjabi University Patiala (India)2004
M.Tech., Punjabi University Patiala (India) 2007

Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

© Amanpreet Dhaliwal 2015
University of Victoria

All rights reserved. This report may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.
Supervisory Committee

Using ARM-based boards in a second year course

by

Amanpreet Dhaliwal
B.Sc., Punjab University Chandigarh (India) 1998
M.Sc. (Physics), Punjabi University Patiala (India) 2004
M.Tech., Punjabi University Patiala (India) 2007

Dr. M. Serra, Supervisor
Department of Computer Science

Dr. J. C. Muzio, Member
Department of Computer Science
Abstract

The Raspberry Pi and the Arduino have emerged as very interesting platforms to learn about the ARM processor and its programming environment, and to develop small systems. They are also fairly inexpensive and could be bought directly by students. In this project we investigate the suitability of using the Raspberry Pi as a platform for the assignments in Computer Science 230, a second year course which introduces computer architecture and uses low-level programming to provide hands-on experience with registers and CPU components. To accomplish this goal, all assignments for the last few years are implemented using the Raspberry Pi. An analysis of the choices of tools is given and documentation for future course use. A short comparison to the Arduino environment is also included.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>viii</td>
</tr>
<tr>
<td>Dedication</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 The Current Environment</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 The ARMSim# simulator</td>
<td>5</td>
</tr>
<tr>
<td>2.2 The Embest Board</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Board Projects</td>
<td>9</td>
</tr>
<tr>
<td><strong>3 The Raspberry Pi</strong></td>
<td>11</td>
</tr>
<tr>
<td>3.1 Hardware</td>
<td>12</td>
</tr>
<tr>
<td>3.2 How to use the Board</td>
<td>16</td>
</tr>
<tr>
<td>3.2.1 Write the Image on the SD card using Win32DiskImager</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2 Configuration Setting</td>
<td>20</td>
</tr>
<tr>
<td>3.2.3 Executing the CSC 230 assignments as case studies</td>
<td>22</td>
</tr>
<tr>
<td><strong>4 The Arduino Board</strong></td>
<td>25</td>
</tr>
<tr>
<td>4.1 Hardware</td>
<td>26</td>
</tr>
<tr>
<td>4.2 Software</td>
<td>28</td>
</tr>
<tr>
<td><strong>5 Case Studies Implementation on the Raspberry Pi</strong></td>
<td>29</td>
</tr>
</tbody>
</table>
5.1 Assignment Case Study: the Sieve of Eratosthenes
5.1.1 The Specifications for the problem to be solved
5.1.2 The Program Design
5.1.3 The first implementation using C
5.1.4 The second implementation using ARM
5.1.5 The third implementation using multiple files in ARM
5.1.6 The fourth implementation using both C and ARM

5.2 Assignment Case Study: the Kolakoski Sequence
5.2.1 The Specifications for the problem to be solved
5.2.2 Algorithm, PseudoCode and Output
5.2.3 The Program Design
5.2.4 The implementation using C
5.2.5 The second implementation using ARM
5.2.6 The third implementation using both C and ARM
5.2.7 The recurrence and the plot for the Kolakoski Sequence

5.3 Assignment Case Study: the Catalan Numbers [15]
5.3.1 The Specifications for the problem to be solved
5.3.2 The Program Design
5.3.3 The first deliverable: the implementation using C
5.3.4 The second deliverable: the implementation using ARM
5.3.5 The third deliverable: the implementation using multiple files in ARM
5.3.6 The fourth deliverable: the implementation using both C and ARM
5.3.7 Some tricks and hints

6 Analysis and Conclusions
6.1 Raspberry Pi: advantages and disadvantages
6.1.1 Highlighting the advantages of the Raspberry Pi
6.1.2 Highlighting the disadvantages of the Raspberry Pi

6.2 Opinions and thoughts on Raspberry Pi
6.3 Arduino: advantages and disadvantages
6.3.1 Highlighting the advantages of the Arduino
6.3.2 Highlighting the disadvantages of the Arduino

6.4 Opinions and thoughts on Arduino

6.5 Comparisons of various boards

6.6 Conclusions
Bibliography

A Assignment Case Study: the Sieve of Eratosthenes on the Raspberry PI 56
   A.1 The implementation for the Raspberry PI .......................... 56
      A.1.1 The C program ........................................... 56
      A.1.2 The ARM Program ....................................... 58

B Assignment Case Study: the Kolakoski Sequence on the Raspberry PI 61
   B.1 The implementation for the Raspberry PI ....................... 61
      B.1.1 The C program ........................................... 61
      B.1.2 The ARM Program ....................................... 64

C Assignment Case Study: the Catalan Numbers 66
   C.1 The implementation for the Raspberry PI ..................... 66
      C.1.1 The C program ........................................... 66
      C.1.2 The ARM Program ....................................... 68
List of Tables

Table 2.1 Some of the Peripheral Components of Embest University Board . . . . 7

Table 5.1 The Kolakoski Sequence .................................................. 34
Table 5.2 Examples for Kolakoski sequence A000002 ......................... 35
Table 5.3 The Catalan Numbers ....................................................... 40
List of Figures

Figure 2.1 The Layout of the Embest University Board ........................................ 7
Figure 2.2 The Structure of the Embest University Board ................................... 8
Figure 2.3 Embedding ARM assembly code into a pre-prepared project .................... 9

Figure 3.1 Block Diagram of the Raspberry Pi .................................................. 13
Figure 3.2 The Raspberry Pi Board ................................................................. 13
Figure 3.3 The Raspberry Pi Board Diagram ..................................................... 14
Figure 3.4 Connection to the USB Keyboard and SD card .................................... 15
Figure 3.5 Connection to power .......................................................................... 15
Figure 3.6 Wheezy Raspbian Image ................................................................. 17
Figure 3.7 Win32DiskImager Unzipped folder .................................................... 18
Figure 3.8 Win32DiskImager UI ........................................................................ 18
Figure 3.9 Progress bar while writing on the SD card .......................................... 19
Figure 3.10 Configuration Settings Screen .......................................................... 20
Figure 3.11 Resizing the SD card with expand_rootfs ........................................ 21
Figure 3.12 Graphical Session Initial Screen ...................................................... 21
Figure 3.13 Opening the Command line for the Gcc compiler .............................. 22
Figure 3.14 The boot/home folder .................................................................... 23
Figure 3.15 The boot/home folder ..................................................................... 23
Figure 3.16 Output and execution of the Sieve program ...................................... 24
Figure 3.17 Output and execution of the Kolakoski program ............................... 24
Figure 3.18 Output and execution of the Catalan program .................................. 24

Figure 5.1 The Kolakosky Recurrence ................................................................ 39
Figure 5.2 The Kolakoski Plot .......................................................................... 39
Acknowledgements

I would like express my sincere gratitude toward my supervisor, Dr. Micaela Serra, for providing her assistance, support, guidance and encouragement throughout my graduate program and thesis.
Dedication

I would like to dedicate this thesis to my family. There is no doubt in my mind that without my family’s continued support and counsel I could not have completed my M.Sc.
Chapter 1

Introduction

"Computer Science 230 - Introduction to Architecture and Assembly Language" is a core second year course for all students in computer science or other related degrees. This course approaches the topics related to computer organization and architecture in two manners: the "what", and the "how". To answer the what, the course presents the fundamental principles of computer organization and architecture. This leads to an introductory understanding of the design of processors, the structure and operations of memory and virtual memory, cache, storage, and pipelining, system integration, and peripherals. Furthermore, the course introduces the issues of system performance evaluation and the relations of architecture to system software.

Regarding the how, the course teaches basic programming in assembly language with calls from a high level language. This all leads to a direct and practical understanding of the inner working stages of a processor in relation to the rest of the system, including memory and cache management, interrupt processing and pipelining. The connections between assembly language to high level languages and system software is explained, through the processes of compilation, linking, and execution cycles. The labs and assignments make use of both software, namely programming and use of IDE's and hardware, including downloading to a board with an embedded processor. The sample architecture used for practice is based on the ARM, one of the major RISC processor, widely used in industry especially for embedded systems.

Currently three of the four assignments use the ARMSim simulator, the gcc compiler for the C language and the Code Sourcery tool chain to mix the two. The mini-project (3rd assignment) is worth 10% of the final grade, it is written all in ARM assembly and implements

1 Integrated Development Environment,
a small embedded system with peripherals. The projects used so far (and explained in more
detail in chapter 2) are: a traffic light controller; a treadmill controller; an elevator controller;
a cellular phone simulator. These projects are developed and tested using the ARMSim# simulator and then downloaded for execution on a hardware platform, namely the Embest board (all details are in chapter 2).

This has been an excellent learning environment for a few years, except that the require-
ments of the Embest board are now coming into conflicts with upgrades in both operating
systems and hardware. The two major stumbling blocks to maintaining the same context
are:

1. The XP OS for windows platforms is the only one supported by the Embest IDE. XP
   is no longer supported by Microsoft after April 2014 and is available at UVic only in
   the lab where the Embest board is used.

2. The connection from a Windows-based PC to the Embest board is through a parallel
   port, which is no longer available as standard in new machines. The newest Embest
   board are sold with a USB port connection as an alternative, but efforts to acquire
   only the new connections have failed, as the manufacturer insists on providing it only
   with the purchase of new boards. When machines in the lab are upgraded, it is still
   possible to get some with a parallel port, but extra costs are involved - useless, since
   USB is a newer and better standard.

The issues above have led to an investigation of alternate hardware platforms for the
course. One search has involved a straight alternative by acquiring something similar to the
current board. This topic is not covered in this project which instead focuses on changing
to a completely different paradigm by analyzing the possibility of using a Raspberry Pi or
an Arduino Board. The former is analyzed here in more depth.

The Raspberry Pi has emerged as a very interesting platform to learn about the ARM
processor and its programming environment, and to develop small systems. It is also fairly
inexpensive and could be bought directly by students. The main goal of this project is the
investigation of the suitability of the Raspberry Pi as a platform for the assignments in CSC
230. The secondary goal is to compare this scenario to the possibility of using the Arduino
Board. To accomplish this, some research has been done in the requirements, costs and
tools connected to the Raspberry Pi. Yet the main focus is much more practical: can all
assignments for the last few years be implemented effectively using the Raspberry Pi and
still provide a comparable learning experience? Moreover, one must add the analysis of the
lab support and documentation needed.
This document is organized as follows:

**Chapter 2** describes briefly the current *CSC 230* lab environment, the *Embest* board and the assignments/mini-projects used in the last few years.

**Chapter 3** describes the *Raspberry Pi* and the tools used for the *Raspberry Pi* to implement programs in C and ARM.

**Chapter 4** describes briefly the *Arduino Board* and the tools which are available for it.

**Chapter 5** presents the results of implementing all assignments as case studies for the evaluation of the suitability of the *Raspberry Pi* in this context.

**Chapter 6** contains the evaluation of the two boards and some conclusions.
Chapter 2

The Current Environment

The period up to 2015 has seen the CSC 230 course adopt two main platforms for its implementation labs: the ARMSim# simulator for the ARM processor and the *Embest* board as a hardware platform with peripherals. Describing the overall dynamics of the course through its assignments and labs may be the best way to explain the environment.

- Assignment 1 (and initial labs) are focused on C programming, at a very elementary level, simply to take students away from the object-oriented paradigm of Java and inside a lower level of memory managements for variables and function calls. Standard C compilation tools and possibly IDEs are used and some interesting programming task is developed.

- Assignment 2 (and attached labs) focus on ARM programming. The ARMSim# simulator is used and the same problem programmed in assignment 1 in C is now re-developed in ARM assembly language. Having already designed and implemented a solution at a high level, students can now completely focus on the new language and rather strange (for them) ways of implementing solutions, using the previous C program as pseudo-code.

- Assignment 3: the project. This is worth 10% of the final mark and includes a one-on-one demo of the running solution plus a one-on-one session with the instructor to analyze the *quality* of the code. The ARMSim# simulator with a plug-in to simulate the *Embest* board is used for development and debugging and then the final tested code is ported to the board itself and executed there for the demo. This avoids having to learn a new debugging tool on the board itself, yet still gives the experience of downloading code to hardware using a complex IDE for the setup and execution.
• Assignment 4 focuses on interfacing ARM and C code, using all the tools learned so far.

Thus the lab and assignment environment is based on two tools: the simulator and the Embest board. A brief overview of both is given below.

2.1 The ARMSim# simulator

ARMSim# [9] is a desktop application running in a Windows environment. It allows users to simulate the execution of ARM assembly language programs on a system based on the ARM7TDMI processor.

ARMSim# includes both an assembler and a linker and it also provides features not often found in similar applications. They enable users both to debug ARM assembly programs and to monitor the state of the system while a program executes. The monitoring information includes both cache states and timing information. ARMSim# is also able to link object files produced by other applications.

An important feature not found elsewhere is the addition of graphical views, programmed as plugins, allowing execution of interactive interfaces and supporting interrupt processing. The main view included is that of a board including an ARM processor and a set of peripherals - buttons, LED lights, keyboard, small LCD screen. Other plugins are also included - for example, the simulation of a traffic light controller. Instructions on how to design and implement new ones are included.

ARMSim# has been developed by members of the Department of Computer Science at the University of Victoria, in Victoria, British Columbia, Canada. It is distributed for free for academic use. A full description of the simulator is beyond the scope of this project. The full description of all details is more easily found in the web pages at [9]. A User Guide prepared for both students and instructor is also available at [8]. ARMSim# is implemented in C#. It requires the .NET Framework to be present. It also runs on Mac OS with Parallel and on Mac OS and Linux with the Mono implementation of the .NET framework.

ARMSim# was developed as an effective simulator for the ARM processor to be used initially as a tool in academia, both for teaching and for research. It is similar in spirit to the SPIM software for the MIPS architecture. ARMSim# has been expanded to include features not normally found in such applications, such as cache simulation, detailed timing information and extensions to the ISA and user interface.

ARMSim# has been tested and used extensively in an introductory course to architecture
in a Computer Science department where the ARM processor and its Assembly language are integrated as part of the learning experience (more information about course material can be obtained from the authors). It can also be used in more advanced architecture courses, where programming and simulating a processor are part of the scope of learning. The attached interactive graphic views of peripherals, with interrupts enabled, provide also an excellent platform for the simulation of embedded systems.

Similar to the SPIM software (a MIPS32 Simulator [10]), ARMSim# has a built in assembler that assembles and links the user code directly when loaded. ARMSim# can also load and link to object files and libraries that have been pre-built allowing the user to utilize code from other sources, such as the C runtime libraries.

An important improvement available in ARMSim# is the ability to extend its functionality through the use of plugins. New hypothetical instructions can be implemented and tested in simulation. Hardware can be simulated by intercepting the accesses to the memory map and this hardware can be visualized through the user interface extensions. The main example is a plugin that simulates the Embest S3CEV40 development board for ARM. It simulates the major components of the board such as the LED’s, buttons, LCD screen, keyboard and Eight segment display. Users can write code targeting the development board and simulate this code in ARMSim# before deploying. ARMSim# is available for download, together with several other plugins including a full hardware board view (emulating an Embest board) with many peripherals, a traffic light and an eight-segment display.

2.2 The Embest Board

The Embest University board [5] is a development board for embedded systems which is used with the Embest IDE (Integrated Development Environment). The board is based on the ARM 7TDMI core processor. It is particularly suitable for use in university courses at both the undergraduate and graduate levels in computer science, computer engineering and electrical engineering. The IDE supports embedded software development on the Embest board. Through its graphical user interface, it facilitates managing and building projects, as well as allowing applications to be run and debugged.

Integrating an ARM assembly language source file into a previously prepared project for the Embest IDE can be accomplished in these four stages:

1. Open the previously prepared project using the Embest IDE.

2. Add the assembly source file to the project.
3. Build the project.

4. Execute the project on the Embest University board.

All the details are in the User Guide prepared for the course [17].

The actual board consists of two parts: the Main Board, and the board with the LCD and Keyboard as shown in Figure 2.1.

![Figure 2.1: The Layout of the Embest University Board](image)

The Main Board includes several peripheral components, listed in Table 2.1.

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>External IDE port</td>
</tr>
<tr>
<td>EarPhone</td>
<td>Earphone Output</td>
</tr>
<tr>
<td>MicroPhone</td>
<td>Microphone Input</td>
</tr>
<tr>
<td>Ethernet</td>
<td>10M Ethernet Interface Connector</td>
</tr>
<tr>
<td>USB</td>
<td>USB Connector</td>
</tr>
<tr>
<td>LCD</td>
<td>320*240 mono LCD Screen Display</td>
</tr>
<tr>
<td>POWER</td>
<td>External Power Button</td>
</tr>
<tr>
<td>RESET</td>
<td>Reset Button</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
</tbody>
</table>

Table 2.1: Some of the Peripheral Components of Embest University Board
The structure of the Main Board is shown in Figure 2.2. Students normally use the Embest board as a last phase in their large project, after having designed it, implemented it and tested with the ARMSim# simulator. They then need to learn how to download the ARM code to the board by embedding it within a larger project which contains extra functions to interface with the peripherals. Those functions have calls which work the same way as the Plug-In available in the ARMSim# simulator and thus the expectation is that the porting of the code should be almost seamless. Moreover, they have to do so within the context of the Embest IDE, which is very similar to the environment and interface provided by Visual Studio [3]. On one hand this is an advantage for familiarity, but in this second year course it is quite a struggle at times, as it is often the first IDE they experience. Most of all, it a complex environment, as it can take a while to learn all its facets. To help, a manual has been developed stating the basic necessary steps and listing all the settings. Finally a pre-prepared project is supplied to the students, so that they need only insert their ARM code within it correctly, without having to write the complex code for the interface to each peripheral. The major steps are summarized schematically in figure 2.3.
There are still always problems, mostly of timing as the clock frequencies are different, but it is part of the educational experience to go through the "frustrations" of learning how to pay attention to a myriad of details. Students find a great satisfaction, according to their feedback, in eventually executing their programs on a real board, touching real buttons with clicks and bounce problems, and observing actual LEDs and writing on an LCD screen. The major steps involved in the downloading of a program onto the Embest board are summarized in figure 2.3.

2.3 Board Projects

Over the years a number of very interesting projects using most of the peripherals on the board have been developed. A full description of them is not necessary here and can be found, upon request, from the material of CSC 230. Only a summary of their functionality and requirements is given below. The projects have not been implemented fully using the Raspberry Pi, as described in the case studies to follow, as the choice of possible peripherals and their various manufacturers which can be bought is too large to be of use for a focused evaluation.

- Traffic Light Controller.

  **Functionality** A traffic light controlling an intersection between a major road and a side road, with possible pedestrian button interrupts.

  **Peripherals Used** LEDs, LCD screen, black buttons.

- Cellular Phone.

  **Functionality** Local and long distance calls receiving and sending, with calculations of different levels of costs.

  **Peripherals Used** LEDs, LCD screen, black buttons, blue buttons.
• Elevator Controller.

**Functionality** A 4 floor elevator with requests for up and down runs at each floor and control from requests from inside the elevator cab.

**Peripherals Used** LEDs, LCD screen, black buttons, blue buttons.

• Treadmill Controller.

**Functionality** Controls for starting, setting speed and inclination, stopping, emergency interrupts, and calculations for calories, distance and time.

**Peripherals Used** LEDs, LCD screen, black buttons, blue buttons.
Chapter 3

The Raspberry Pi

The Raspberry Pi is a series of credit-card-sized single-board computers developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. Although it does not have a huge computational power, it can run various GNU/Linux and BSD distributions and be employed with success in computational task such as multimedia media centres and networking.

What is exactly the Raspberry Pi? The best background can be found by paraphrasing the entries in Wikipedia [16]. The Raspberry Pi is a single board computer of very small dimensions, comparable to a credit card. Notwithstanding its small size and low price it contains all the electronic components and I/O capabilities that can be usually found in a basic personal computer. Since its introduction in the consumer market in 2012 it has earned a lots of success either among the power users and amateurs, selling over five million units during these years.

Two models have been developed so far. The first one was based on a 700Mhz single core ARM1176JZF-S equipped with 256MB/512MB (rev1/rev2) RAM memory and a price of 25$ \footnote{\textsuperscript{1}}. In early February 2015, the next-generation Raspberry Pi, Raspberry Pi 2, was released. The main improvement of the second model is basically the 900MHz quad-core ARM Cortex-A7 CPU. Besides a memory upgrade to 1GB of RAM, it shares the same layout and hardware of the first model and costs 35$. The full specifications of the Raspberry Pi 2 model B are as follows:

- 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

\footnote{\textsuperscript{1}US dollars}
- 4 USB ports
- 40 GPIO pins
- Ethernet 100Mbit/s
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

The new computer board is currently available in only one configuration. Crucially, the Raspberry Pi 2 retains the same US$35 price point of the previous model.

Thanks to the ARMv7 computer architecture, it can run a full range of GNU/Linux distributions as well as FreeBSD, NetBSD, Plan9 and even Windows 10 with the second model. The Raspberry Pi foundation provides its own GNU/Linux distribution, called Raspbian, which is based on Debian Wheezy. It also provides Noobs, an operating system installer for non-expert users. In addition there are also many third party distributions such as: Ubuntu Mate, Snappy Ubuntu core; some of them are media-center-based such as OpenElec.

The original Raspberry Pi is available in several configurations instead, sold mainly online. While the hardware is the same for all manufacturers, there are some subtle differences in their versions. The board used here is based on the Broadcom BCM2835 system on a chip (SoC), which includes an ARM1176JZF-S 700 MHz processor and VideoCore IV GPU. Normally now RAM is 512 MB.

The Raspberry Pi Foundation provides Debian and Arch Linux ARM distributions for download. Tools are available for Python as the main programming language, with support for BBC BASIC \cite{14} (via the RISC OS image or the Brandy Basic clone for Linux), C, C++, Java, Perl and Ruby.

As of 18 February 2015, over five million Raspberry Pis have been sold \cite{13}. It is the fastest selling British personal computer.

### 3.1 Hardware

The block model of figure 3.1 provides a high level view of the architecture.
Though the original models do not have an Ethernet port, they can be connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the later models the Ethernet port is provided by a built-in USB Ethernet adapter. One of the main strengths of the board is that generic USB keyboards and mice are compatible with it. The complete list of specifications can be found on the Wikipedia page and is not repeated here.

Figure 3.2 shows a photo of the actual board used here.

Figure 3.3 shows the diagram of the board used here.

A separate monitor is needed to display information. The monitor is normally connected via HDMI. The Raspberry Pi only supports USB keyboards. It also does not have a built-in
hard disk and one must use an SD card for data storage. The connections are shown in Figure 3.4.

An external power source is required and its connection is shown in Figure 3.5.
Figure 3.4: Connection to the USB Keyboard and SD card

Figure 3.5: Connection to power
3.2 How to use the Board

This section is written rather pedantically as a mini-tutorial for a new user, so that its information may be carried over in the future to an educational environment.

While the efficiency of the hardware, availability of peripherals and the low cost are important factors, the *ease of use* is becoming more the crucial aspect of any new product. In the educational environment this is even more critical. The ease of use must be pitched at the proper level for a course - not too difficult to avoid frustrations, and yet challenging enough to stimulate learning. The ease of use also has another important side aspect, namely how much effort it takes for an instructor to set up a lab environment and exercises, together with the necessary wrapper infrastructure, interface to system software and appropriate tutorials, both for the students and for the teaching assistants. All this is predicated on the availability of ready-made tools for the product and how much open-source the accompanying software might be, in order to customize it.

Finally, for each course, the learning objectives are often local and integrated in a bigger vision of the curriculum. Thus a particular product, excellent under many aspects, may still not fulfill such objectives. The main example of such in this case is the balance between the usage of Assembly level language and a high level language such as C. There is a need to teach the low level construct in order to fulfill the general goals of the course. Yet one does not want to go back to the custom of years ago when every single port signal had to be set, making the process extremely tedious, error prone and unrealistic in terms of what practically goes on in an industrial setting. Conversely, moving completely towards C-based programming only, similarly to senior courses in embedded system, is also not desirable as students would miss the low-level architectural experience.

In this section an overview is given of the steps to be taken to deploy the *Raspberry Pi* effectively. It is explained in a detailed and pedantic fashion, so as not to leave any doubts as to what a novice user would face.

3.2.1 Write the Image on the SD card using Win32DiskImager

The Raspbian Wheezy disk image is needed for the task of writing, compiling and running a C program. The image is available at the following link: [www.raspberrypi.org/downloads](http://www.raspberrypi.org/downloads). The Raspbian Wheezy disk image needs to be downloaded and unzipped to extract the relevant files.

The next step is to write the image to the SD card. As mentioned on the Raspberry Pi official website [6], a Windows user should use the *Win32DiskImager* to write an image on
the card. The folder of the wheezy-raspbian image downloaded from the official Raspberry Pi website should look as shown in Figure 3.6.

![Figure 3.6: Wheezy Raspbian Image](image)

The **Win32DiskImager** to write an image on the SD card can be found at this link: http://www.raspberry-projects.com/pi/pi-operating-systems/win32diskimager. The unzipped folder of **Win32DiskImager** is shown in Figure 3.7.

After unzipping the **Win32DiskImager**, clicking on the **Win32DiskImager** file starts the writing of the Raspbian image on the SD card. The screen shown in Figure 3.8 pops up after the **Win32DiskImager** file is clicked.

Select the device where the image should be written, in this case the SD card. A progress bar dialog should pop up as shown in Figure 3.9.
Figure 3.7: Win32DiskImager Unzipped folder

Figure 3.8: Win32DiskImager UI
Figure 3.9: Progress bar while writing on the SD card
3.2.2 Configuration Setting

At this point all the hardware can be connected and the power can be turned on. The first time the Raspberry Pi goes directly to the configuration setting as shown in Figure 3.10.

![Figure 3.10: Configuration Settings Screen](image)

The menu options are selected by using the arrow keys, the tab key and the Enter key on the keyboard.

- **expand_rootfs** is shown in Figure 3.11. The SD card after using the linux image appears to have only a 2GB capacity even though the actual capacity of the card is 4GB. To ensure that the Raspberry Pi can use the full capacity, the `expand_rootfs` is used.

- **overscan** controls the size of the border around the screen image. It may not need to be adjusted.

- **configure_keyboard.** A list of all the available keyboards appears and one needs to be selected together with its layout and its language (US English may not appear in which case selecting “other” or “default” should work). Additional options for the keyboard configuration should also come up. Here the selection was made of ”No compose key” and ”NO” to the use of Control+Alt+Backspace to terminate the X server.

- **change_pass** allows an user to change the default password. A new one needs to be entered twice.

- **change.locale** and **change.timezone** enable an user to set the local time zone and region and city.
• After the last option of `memory_split` is selected, a reboot will be requested and it is necessary.

The reboot screen now pops up after the settings are completed and one needs to login with user and password. Then the command `startx` is used to launch a graphical session as shown in Figure 3.12.
3.2.3 Executing the CSC 230 assignments as case studies

The gcc compiler is built on the Wheezy image that already installed. It can be run through the command line directly by selecting the LXterminal from the start button as shown in Figure 3.13. It is always best to check the gcc compiler version using the command gcc v.

The code for the existing CSC 230 assignments was loaded on the SD card in the boot folder. It needs to be copied onto the Raspberry Pi desktop. Clicking on the start button and selecting the File Manager option, opens the Raspberry Pi file folders. After choosing the PI folder, selecting its parent folder will show the boot/home folder, which should look similar to Figure 3.14.

By clicking on the boot/home folder to open it, one can copy/paste the assignment folder onto the Raspberry Pi desktop. Now the editor can be started to work on the code. A Leafpad text editor is already built into the Raspberry Pi. Once the C program is ready, it can be compiled using gcc. The examples in Figures 3.15, 3.16, 3.17 and 3.18 also call external ARM Assembly language functions contained in .s files and its steps and outputs are shown. All the corresponding code is found in the Appendices.
Figure 3.14: The boot/home folder

Figure 3.15: The boot/home folder
Figure 3.16: Output and execution of the Sieve program

Figure 3.17: Output and execution of the Kolakoski program

Figure 3.18: Output and execution of the Catalan program
Chapter 4

The Arduino Board

Arduino is an open-source computer hardware and software company, also a project and user community, that designs and manufactures micro-controller kits. A large collection of models has been developed ranging from entry level Atmel 8-bit AVR micro-controller to more powerful Atmel 32-bit ARM processor, along with a great variety of packaging that suite any kind of needs. Moreover modules, called Shields, that can be plugged onto a board to give extra functionality are provided. Beyond the hardware boards, Arduino supplies also an open-source IDE which runs on GNU/Linux, Mac OS X and Windows, to facilitate the software development and uploading to the board.

Arduino ultimately is a very simple product. It is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for anyone intending to implement interactive objects that can sense and control the physical world based upon a design. It can be used for simple projects such as blinking a small light or for complex projects such as to interact with a cellular phone. Basically it is a board that puts together a micro-controller and all the components needed to power, upload code and provide the clock frequency. In addition it exposes in an handy way its I/O pins. All the different models shares this common principle except replacing the microcontroller that could be more or less powerful or have variable amounts of memory and I/O pins, or owning a particular boards configuration.

The Arduino board consists of microcontroller and software or IDE (Integrated Development Environment) which includes support for C and C++ programming languages. The boards can be purchased preassembled, or can be assembled from scratch following a template (Eagle CAD) for the Arduino hardware, also available on the Arduino official website. The Arduino software is available for the Windows, Macintosh OS X, and Linux environments and it is free. The Arduino project is designed to read data from sensors, perform the
appropriate desirable computations and output results through the selected devices, such as LEDs or LCD screens. A project can be stand-alone, or can be combined with other software running on a computer, such as Flash, Processing, MaxMSP [2].

4.1 Hardware

There are number of Arduino boards available, and they can be adapted based upon individual needs. The original Arduino was designed for one specific task, while later on the company created more designs with different capabilities to fulfil the different requirements. The initial board can be considered the backbone of the Arduino hardware, namely the Arduino Uno board [11].

Here is the list of the range of boards provided by Arduino [7].

UNO The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button.

PRO The Arduino Pro is a microcontroller board based on the ATmega168 or ATmega328. The PRO comes in both 3.3V / 8 MHz and 5V / 16 MHz versions. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a battery power jack, a power switch, a reset button, and holes for mounting a power jack, an ICSP header, and pin headers. A six pin header can be connected to an FTDI cable or Sparkfun breakout board to provide USB power and communication to the board.

MEGA The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

ZERO The Arduino Zero is a simple and powerful 32-bit extension of the platform established by Arduino UNO. The board is powered by Atmels SAMD21 MCU, which features a 32-bit ARM Cortex M0+ core.

DUE The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core μC . It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog
inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.

**YUN** The Arduino Yn is a microcontroller board based on the ATmega32u4 and the Atheros AR9331. The Atheros processor supports a Linux distribution based on OpenWrt named OpenWrt-Yun. The board has built-in Ethernet and WiFi support, a USB-A port, micro-SD card slot, 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and a 3 reset buttons.

Arduino also has available a set of **Shields**. These are elements that can be plugged onto a board to give it extra features such as: step-motor driver, ethernet and GSM interface. This is a wonderful feature as they expand the capability of the Arduino interface. The shields need to be placed directly on Arduino board, and communicate through various pins on the board. The shields are available in many different shapes and sizes. The Arduino platform has grown considerably lately and a lot more shields have been created by the wider community [1]. Following is a list of the most popular or useful shields.

**Motor** The Arduino Motor Shield is based on the L298, which is a dual full-bridge driver designed to drive inductive loads such as relays, solenoids, DC and stepping motors. It lets one drive two DC motors with the Arduino board, controlling the speed and direction of each one independently.

**Proto** The Arduino Prototyping Shield makes it easy to design custom circuits. One can solder parts to the prototyping area to create a custom project, or use it with a small solderless breadboard (not included) to quickly test circuit ideas without having to solder.

**Ethernet** The Arduino Ethernet Shield connects the Arduino to the internet in mere minutes. Just plug this module onto the Arduino board, connect it to the network with an RJ45 cable (not included) and follow a few simple instructions to start controlling the world through the internet.

**GSM** The Arduino GSM Shield connects the Arduino to the internet using the GPRS wireless network. One can also make/receive voice calls (an external speaker and microphone circuit are needed) and send/receive SMS messages.
Arduino furthermore proposes some new items which can be classified within the realm of wearables computing.

**GEMMA** The Arduino Gemma is a microcontroller board made by Adafruit based on the ATtiny85. It has 3 digital input/output pins (of which 2 can be used as PWM outputs and 1 as analog input), an 8 MHz resonator, a micro USB connection, a JST connector for a 3.7V battery, and a reset button.

**LILYPAD** The LilyPad Arduino is a microcontroller board designed for wearables and e-textiles. It can be sewn to fabric and similarly mounted power supplies, sensors and actuators with conductive thread. The board is based on the ATmega168V (the low-power version of the ATmega168) or the ATmega328V.

### 4.2 Software

In addition to the collection of hardware boards, Arduino provides as well the IDE for the software development. The Arduino IDE contains a text editor for writing code, a message area and a text console. It supports the Arduino board for uploading the programs and communicate with them. There is also a large availability of libraries, some of them already included with the Arduino software, others which can be downloaded from a variety of sources.

The IDE is based on the Processing programming language and shares some common ideas with it. Processing is an open source programming language and IDE with the purpose of teaching fundamental programming in a visual context and to serve as the foundation for electronic sketchbooks. For this reason software written using Arduino are called sketches. The Arduino language provides an easy, ready-to-go way to write software that wraps around all the low level details usually needed to program a microcontroller. Beginners, who most likely don’t know the internals of a microcontroller and, as a consequence, don’t know how to program it in the traditional way, with assembler or C, are able to effectively write software without particular limitations.
Chapter 5

Case Studies Implementation on the Raspberry Pi

In this chapter a description is given of the case studies implemented on the Raspberry PI to test the environment and its suitability for use in a classroom. Notwithstanding all the positive claims made in the literature, there is no substitution to being hands-on and testing every details directly.

The case studies shown are the final assignments given in CSC 230, as they are the ones which include both C programming and ARM programming. While the ARM technical portion is not extremely complex, the goal here is to test how well integrated all the tools are and how easy to use, as opposed to test tricky programming constructs. Moreover it was kept to computational-based algorithms, instead of involving the use of various peripherals. There are just too many possibilities for peripherals to be attached in order to implement a class project as done in CSC 230, and yet they all have the same programming interface. Thus this hands-on part is left for the future.

Three programming assignments were chosen and their specifications are presented here in their complete form, as they were given to students. All the code is presented in the Appendices. The case studies are:

- The Sieve of Eratosthenes
- The Kolakoski Sequence
- The Catalan Numbers
5.1 Assignment Case Study: the Sieve of Eratosthenes

5.1.1 The Specifications for the problem to be solved

In order to generate all prime numbers up to 1,000,000, the Sieve of Eratosthenes is quite an efficient algorithm. The algorithm filters the prime numbers by deleting from a list all multiples of those numbers that have already been discovered to be primes. For example, from a list from 1 to N, one would keep 2, but reject 4, 6, 8, etc. as they are all multiples of the initial seed 2. Similarly one would keep 3 and reject 6, 9, 12, etc.

The pseudocode below defines the solution you need to program, for a case where the maximum range of integers is 0 to 100 and one wants to find all prime number up to a MAX of 50.

Define MAXLIST = 100
Declare Max as integer
Declare sieve[MAXLIST] integers
Declare primes[MAXLIST] integers
Max = GetMax - prompt user and read Max (in range from 1 to 100)
Initialization step:
    int i;
    sieve[0] = 0;
    sieve[1] = 0;
    k = 0;
    for (i=2, i<Max; i++) sieve[i] = 1;  
Find primes:
    int i,j;
    for (i=2; i<Max; i++) {  
        if sieve[i] = 1 { /*new prime found*/
            primes[k++] = i
            for (j=i*2; j<Max; j+=i) /*flag all multiples*/
                sieve[j] = 0;
        }
    }
Print primes:
    for (i=0; i<number of primes k;i++)
print list of primes at 15 per line

The output of the program should be the list of prime numbers found. Test the program by listing all the prime numbers less than Max as stored them in an array called primes (let the maximum size be MAXLIST = 100). Simply print out primes to stdout in a pleasing and professional format. For example, if MAX were given to be 15, then the output should be similar to:

There are 6 prime numbers up to 15 and they are:
2 3 5 7 11 13

Make sure the program can work properly for many different values of MAX.

5.1.2 The Program Design

The structure of the program must include at least three subroutines called by main, namely: GetMax, Initialize, FindPrimes, and PrintPrimes. Do not forget to add appropriate messages at the beginning and at the end of your program, and whenever output is expected. The interface to the functions should be as follows:

Max = GetMax ( )
Initialize (address of sieve)
Howmany = FindPrimes(address of sieve; address of primes)
PrintPrimes(number of primes; address of primes)

5.1.3 The first implementation using C

Draw a precise and correct flow chart which includes all portions of your program and shows the modular design. Following your design from the flowchart, implement your program using C, in a file called A4sieveC.c, compiled using:

```
gcc -Wall -ansi A4sieveC.c
```

Test your code for many instances of Max from 10 to 100.
5.1.4 The second implementation using ARM

For the implementation in ARM, delete the input from the user for Max and instead encode its value using an EQU. This implies that you will need to assemble the program every time you need to test with a new value for Max, but it will avoid you having to code for user input (not a normal practice at this low level). Draw a precise and correct flow chart which includes all portions of your program and shows the modular design. Following your design from the flowchart, implement your program using ARM, in a file called A4sieveA.s, tested using ARMSim#. Test your code for many instances of Max from 10 to 100 simply by reassembling it with different initial values (you do not need to read values from a file or from stdin). Submit the program with a pre-encoded value of Max=50.

5.1.5 The third implementation using multiple files in ARM

It is good to get experience in compiling, assembling and executing programs which are split into multiple files. The important issue is to make sure that the separate module are interfacing properly and can communicate. Here you will place the function FindPrimes in a separate file such that it will be called as an external one by the main function in your program. The structure should be as shown below.

@ File: A4sieve-withExtern.s
[a] code and comments as before
[b] .global _start
[c] .extern FindPrimes
[d] .global Max
[e] .text
    _start:
@ code as before without FindPrimes
@ and only with its call to it
[g] .data
code as before
[h] .end
@ File: FindPrimes-Extern.s
code and comments as before
[i] .global FindPrimes
[j] .text
The program starts from file A4sieve-withExtern.s where the code for main, for Initialize and for PrintPrimes remains. The code for FindPrimes is moved to another file called FindPrimes-Extern.s. The connection between the files must be stated explicitly through the use of symbols being exported or imported as necessary, using the .global and .extern commands. At [b] the label _start is exported through the use of the .global Assembler command as usual. This signals to any other portion of the program where the entry point is as it must be unique. At [c] the command .extern states that the label FindPrimes, when invoked later, will be found externally to this file (hopefully in a module jointly assembled). At [d], the label Max is similarly exported as the function FindPrimes will need to use it. Note the difference in use between .global and .extern. The rest of the program at [f] remains the same except that the portion for the function FindPrimes is no longer here, only its call (using BL). Note that the parts for data and end at [g] and [h] are the same. The file FindPrimes-Extern.s contains the function FindPrimes and thus its name (label) is immediately exported for all to see at [i] through the .global command. The rest of the code for FindPrimes remains the same in [j] to [m], and it can even have its own data (not necessary here) and it must have its own end command. Once the .global and .extern declarations have been properly set, the program can be executed in ARMSim# by using the File | Load Multiple to load both assembly files, through the dialogue box. Step through the execution. Notice the transition between modules when BL FindPrimes is invoked.

5.1.6 The fourth implementation using both C and ARM

Finally write the last version of the Sieve program using a mixture of C and ARM. Use the instructions given in the lab (and soon to be posted on the website) to combine multiple object files and to make sure that all parameters are appropriate (be careful about what the compiled C code may change!). Use the following template with functions and procedures as indicated.

Main: in ARM assembly, in a file A4sieveAC.s.
Initialize: in ARM assembly, in the same file as main.
FindPrimes: in C, in a file A4SieveCsub.c.
PrintPrimes: in ARM assembly, in the same file as main.

Basically you are repeating the third implementation while substituting the `FindPrimes-Extern.s` with the C function for `Findprimes` in an external C source file, `A4SieveCsub.c`, which needs to be cross compiled to ARM code. Then `ARMSim#` will be able to link it in with the assembled `A4sieveAC.s` and the program should run as it did both in the second full ARM implementation and the third one with multiple files.

5.2 Assignment Case Study: the Kolakoski Sequence

5.2.1 The Specifications for the problem to be solved

The Kolakoski sequence (A006928) is an infinite list whose initial 108 entries are shown in Table 5.1.

<table>
<thead>
<tr>
<th>elements 1 to 18</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements 19 to 36</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>elements 37 to 54</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>elements 55 to 72</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>elements 73 to 90</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>elements 91 to 108</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.1: The Kolakoski Sequence

The only numbers allowed are 1 and 2. Each number states how many numbers need to be appended to the sequence. There can be no more than two of the same number in sequence. Every time a new number is considered (read), one must alternate between writing 1 and 2. There are actually a few permutations of the Kolakoski sequence (with different number labels). The easiest English explanation is found for the development of the Kolakoski sequence A000002, shown in Table 5.2.

Kolakoski’s self-generating sequences are of interest to both computer scientists and mathematicians (see *A New Kind of Science*, p. 895, by Stephen Wolfram). One might expect that the density of 1’s and 2s is 50 for each, but this conjecture has never been formally proved (see the *On-Line Encyclopedia of Integer Sequences (OEIS)* at http://oeis.org/Seis.html). Further references can be found in N. J. A. Sloane and Simon Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press, 1995.
write 1;  
read 1 as the number of 1's to write before switching to 2;  
write 2;  
read 2 as the number of 2's to write before switching back to 1;  
read the new 2 as the number of 1's to write;  
read the new 1,1 as stating one needs to write one 2's and one 1's  
continue generating forever.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write 1;</td>
<td>$$[1]$$</td>
</tr>
<tr>
<td>read 1 as the number of 1's to write before switching to 2;</td>
<td>$$[1]$$</td>
</tr>
<tr>
<td>write 2;</td>
<td>$$[1 2]$$</td>
</tr>
<tr>
<td>read 2 as the number of 2's to write before switching back to 1;</td>
<td>$$[1 2 2]$$</td>
</tr>
<tr>
<td>read the new 2 as the number of 1's to write;</td>
<td>$$[1 2 2 1 1]$$</td>
</tr>
<tr>
<td>read the new 1,1 as stating one needs to write one 2's and one 1's</td>
<td>$$[1 2 2 1 1 2 1]$$</td>
</tr>
</tbody>
</table>

Table 5.2: Examples for Kolakoski sequence A000002

5.2.2 Algorithm, PseudoCode and Output

While the steps shown above appear to be a simple enough explanation, it is still far from being a proper algorithm ready for programming. It would be a great learning experience to leave the development of the algorithm and pseudo code as part of this assignment, but it may take away from the real focus, which is to program in both C and ARM and link the code together. Thus the pseudo code is given to you below for the production of the Kolakoski sequence A006928, for which a closed form algorithm can be found (as opposed to the one shown above which is the A000002 sequence, used only for explanation). Please note carefully the language (in case you do not have much experience). When it is stated that the range of a variable is from \( x \) to \( y \) this implies that both \( x \) and \( y \) are included in the possible values.

In main:

Let \( Kseq \) be the array to contain the Kolakoski sequence

Initialize the beginning elements as:

\( Kseq[0] \) is not used
\( Kseq[1] = 1 \)
\( Kseq[2] = 2 \)

In function AppendKseq

Let \( n \) be the number of run steps in producing the sequence

where \( n \) ranges from 2 to infinity. In a practical program, \( n \) should range from 2 to a given maximum number of runs, here denoted by \( \text{MAXRUNS} \)

for (\( n=2 \) to \( \text{MAXRUNS} \)) {
    for (\( i=1 \) to \( i=Kseq[n] \)) {
        append the element (1+(\( n \) mod 2)) to \( Kseq \)
    }
}
The output of a program should be the list of entries of the Kolakoski sequence up to a maximum number of runs, together with a count of the total number of entries, the count of the number of elements equal to 1 and the corresponding count of the number of elements equal to 2. A sample output for a given maximum run of n=50 is shown below, where elements are printed 10 per row.

Jean Luc Picard V00123456
Kolakoski Program starts
Kolakoski sequence of length 75 with 50 Runs
   Number of 1's is = 38
   Number of 2's is = 37
1 2 1 1 2 1 2 2 1 2
2 1 1 2 1 1 2 2 1 2
1 1 2 1 2 2 1 1 2 1
1 2 1 2 2 1 2 2 1 1
2 1 2 2 1 2 1 1 2 1
1 2 2 1 2 2 1 1 2 1
2 2 1 2 2 1 1 2 1 1
2 2 1 2 2
Kolakoski Program ends

Make sure your program can work properly even if the maximum number of runs (MAXRUNS) is a different number, as, in marking, we may change your code and run it again with different values! DO NOT, however, make MAXRUNS a variable input from the user, or from a command line, or a parameter to a function (instead use EQU and #define).

5.2.3 The Program Design

The structure of the program must include at least two functions called by main, namely: AppendKseq and PrintKseq. Do not forget to add appropriate messages at the beginning and at the end of your program, and whenever output is expected. The interface of these two functions should be as follows (no changes allowed):

AppendKseq. int AppendKseq(int Kseq[],int startindex,int *Num1)
Description: Append the next elements in the Kolakoski sequence to the initial array.
Inputs: address of initial Kseq array, index = next position to be filled; Num1 = address of variable for the number of entries equal to 1.
Output: int= next position to be filled.
Side effects: update number of entries equal to 1 in its variable Num.
Expectation: Kseq array has been initialized for the first 3 entries at least (i.e. index > 2)

PrintKseq. void PrintKseq(int Kseq[], int index)
Description: Print the Kolakoski sequence.
Inputs: address of Kseq array, index = next position to be filled.
Output: none.
Side effects: none. However the elements of Kseq are printed in rows, possibly 10 elements per row.
Expectation: Kseq is not empty.

MAXRUNS. Use a #define in C and an .EQU in ARM.
Maximum size of Kseq array. It should be MAXRUNS*2 in C and similarly, in bytes for ARM.

5.2.4 The implementation using C
Following your design from a flowchart or precise pseudo-code (do it!) implement your program using C, in a file called A4KolakoskiC.c, compiled using:

```bash
gcc -Wall A4KolakoskiC.c
```
Test your code for any number of MAXRUNS from 10 to 100 simply by recompiling it with different initial values (you must not read values from a file or from the command line or from stdin). Submit the program with a pre-encoded value of MAXRUNS=50.

5.2.5 The second implementation using ARM
Implement your program using ARM, in a file called A4KolakoskiA.s, tested using ARMSim#, for any number of MAXRUNS from 10 to 100 simply by reassembling it with different initial values. Submit the program with a pre-encoded value of MAXRUNS=50. Important to note at this point is that in the next deliverable you will substitute the ARM function AppendKseq with the C function. Thus, before you code the ARM one, make sure that the parameter passing in registers follow the C expectations of the compiler, otherwise you may find yourself rewriting code later on! Read below before doing this part.
5.2.6 The third implementation using both C and ARM

Finally write the last version of the Kolakoski program using a mixture of C and ARM. Use the instructions posted on the website to combine multiple object files and to make sure that all parameters are appropriate (be careful about what the compiled C code may change!). Note carefully: the goal here is that you do not write any new code. You should be able to insert your original C code for `AppendKseq` in a separate file called by the ARM program. We will check if any changes were made between the two versions! The trick is that the function should be called in ARM passing the parameters as expected by the C compiler.

Use the following template with functions and procedures as indicated.

Main: in ARM assembly, in a file `A4KolakoskiAC.s`.
AppendKseq: in C, in a file `A4AppendKseqCsub.c`.
PrintKseq: in ARM assembly, in the same file as main.
Any other function: in ARM assembly, in the same file as main.

Basically you are repeating the second implementation while substituting the C code for `AppendKseq` in an external C source file, `A4AppendKseqCsub.c`, containing the C source code for `AppendKseq`, which needs to be cross compiled to ARM code. Then `ARMSim#` will be able to link it in with the assembled `A4KolakoskiAC.s` and the program should run as it did both in the second full ARM implementation.

5.2.7 The recurrence and the plot for the Kolakoski Sequence

A recurrence plot of the Kolakoski sequence is illustrated in Figure 5.1 and its point plot in Figure 5.2.
Figure 5.1: The Kolakosky Recurrence

Figure 5.2: The Kolakoski Plot
5.3 Assignment Case Study: the Catalan Numbers [15]

5.3.1 The Specifications for the problem to be solved

In combinatorial mathematics, the Catalan numbers form a sequence of natural numbers that occur in various counting problems, often involving recursively defined objects. They are named after the Belgian mathematician Eugène Charles Catalan (1814–1894). They are also called Segner numbers. The initial entries are shown in Table 5.3.

The $n$th Catalan number, for $n \geq 0$ is given directly in terms of binomial coefficients by:

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n} \frac{n+k}{k}$$


Following are some examples, with illustrations for small cases.

- $C_n$ is the number of Dyck words of length $2n$. A Dyck word is a string consisting of $n$ X’s and $n$ Y’s such that no initial segment of the string has more Y’s than X’s. For example, the following are the Dyck words of length 6:
  
  XXXYYY XYXXYY XXYXYX XXXYYX XXXYY

- Re-interpreting the symbol X as an open parenthesis and Y as a close parenthesis, $C_n$ counts the number of expressions containing $n$ pairs of parentheses which are correctly matched.

<table>
<thead>
<tr>
<th>elements</th>
<th>$C_5$ to $C_9$</th>
<th>$C_{10}$ to $C_{14}$</th>
<th>$C_{15}$ to $C_{19}$</th>
<th>$C_{20}$ to $C_{22}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_5$ to $C_9$</td>
<td>42</td>
<td>132</td>
<td>429</td>
<td>1430</td>
</tr>
<tr>
<td>$C_{10}$ to $C_{14}$</td>
<td>16796</td>
<td>58786</td>
<td>208012</td>
<td>742900</td>
</tr>
<tr>
<td>$C_{15}$ to $C_{19}$</td>
<td>9694845</td>
<td>35357670</td>
<td>129644790</td>
<td>477638700</td>
</tr>
<tr>
<td>$C_{20}$ to $C_{22}$</td>
<td>6564120420</td>
<td>24466267020</td>
<td>91482563640</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 5.3: The Catalan Numbers
\(((())) \ (())\ (())\ (())\ (())\ (())\)

You can see how this is useful in an editor!

• \(C_n\) is the number of different ways \(n + 1\) factors can be completely parenthesized (or the number of ways of associating \(n\) applications of a binary operator). For \(n = 3\), for example, we have the following five different parenthesizations of four factors:
  
  
\((ab)c)d \ (a(bc))d \ (ab)(cd) \ a((bc)d) \ a(b(cd))

The original expression stated above to compute the Catalan numbers contains a binomial coefficient which can be tricky to implement (efficiently). Fortunately there exists a recurrence relation for the Catalan numbers which is much easier to program, as:

\[
C_0 = 1 \quad \text{and} \quad C_n = \frac{2(2n - 1)}{n + 1} \ C_{n-1} \quad \text{for} \ (n \geq 0)
\]

You will be using the above formula in your program. However some thinking is still required. The only operations you have available are for integers and yet you have a division. Consider the order of computation before you write your code! (Feel free to come and consult). The output of the program should be the list of entries of the Catalan sequence up to a maximum number of entries, in a nice professional format (all your choice!). Make sure your program can work properly even if the maximum number of entries (MAX) is a different number, as, in marking, we may change your code and run it again with different values! DO NOT, however, make MAX a variable input from the user, or from a command line (instead use EQU and #define).

5.3.2 The program design

The structure of the program must include at least two functions called by main, namely: CatSeq and PrintSeq. Do not forget to add appropriate messages at the beginning and at the end of your program, and whenever output is expected. The interface of these two functions should be as follows (no changes allowed).

CatSeq. int CatSeq(int CatArray[], int n, int top)

Description: Insert the elements in the Catalan sequence in the array given initialized only for \(C_0\).

Inputs: address of array; \(n = \) next position to be filled; \(top = \) maximum number of entries to be computed.

Output: \(\text{int} = \) number of elements in the array.

Side effects: update the elements of the array.
Expectation: CatArray array has been initialized for the first entry with \( C_0 = 1 \) in CatArray[0].

**PrintSeq.** void PrintSeq(int Seq[], int top)

*Description*: Print the sequence in the array.
*Inputs*: address of array, \( \text{top} = \) number of elements to be printed.
*Output*: none.
*Side effects*: none. However the elements of \( \text{Seq} \) are printed in rows, normally 5 elements per row.
*Expectation*: \( \text{Seq} \) is not empty.

*Note*: the number of elements printed per row (here 5 is suggested) should not be hard-coded here. The suggestion would be to use something like: `#define SEQ_PER_LINE 5` in C and a corresponding `.EQU` in ARM.

**MAX.** Use a `#define` in C and an `.EQU` in ARM. Submit with `MAX=11`.

**Maximum size of array.** Statically allocated as a function of `MAX`. in C and similarly, in bytes for ARM.

### 5.3.3 The first deliverable: the implementation using C

Write your program using C, in a file called `A4CatalanALLC.c`, compiled using:

```
gcc -Wall A4CatalanALLC.c
```

Test your code with `MAX` from 5 to 11 simply by recompiling it with different initial values (you must not read values from a file or from the command line or from `stdin`). Submit the program with a pre-encoded value of `MAX=11`.

### 5.3.4 The second deliverable: the implementation using ARM

Following your design the C program, implement your program using ARM, in a file called `A4CatalanALLA.s`, tested using `ARMSim#`. Test your code with `MAX` from 5 to 11 simply by reassembling it with different initial values. Submit the program with a pre-encoded value of `MAX=11`. 
Technical detail: division

As you should know by now, ARM does not include an instruction to do division. You have to write your own division function. However, if you look back in your lecture notes (ARM-programming Part 2) I used the code for a function doing division as an example when teaching the language. You can use that code and optimize it. Moreover you may have used a division function in a previous assignment.

5.3.5 The third deliverable: the implementation using multiple files in ARM

It is good to get experience in compiling, assembling and executing programs which are split into multiple files. The important issue is to make sure that the separate module are interfacing properly and can communicate. Here you will place the function \texttt{CatSeq} in a separate file such that it will be called as an external one by the main function in your program. The structure should become as shown below.

\@
\begin{verbatim}
File: A4CatalanA-withExtern.s
[a] code and comments as before
[b]  .global _start
[c]  .extern CatSeq
[d]  .global MAX
[e]  .text
    _start:
    code as before
[f]  without CatSeq
[g]  .data
    code as before
[h]  .end
\end{verbatim}
\@

\@
\begin{verbatim}
File: A4CatSeqA-Extern.s
code and comments as before
[i]  .global CatSeq
[j]  .text
@int CatSeq(.parameters..)
\end{verbatim}
\@
The program starts from file \texttt{A4CatalanA-withExtern.s} where the code for the main routine and for \texttt{PrintSeq} remains. The code for \texttt{CatSeq} has been moved to another file called \texttt{A4CatSeqA-Extern.s}. The connection between the files must be stated explicitly through the use of symbols being exported or imported as necessary, using the \texttt{.global} and \texttt{.extern} commands. At \([b]\) the label \texttt{.start} is exported through the use of the \texttt{.global} Assembler command as usual. This signals to any other portion of the program where the entry point is as it must be unique. At \([c]\) the command \texttt{.extern} states that the label \texttt{CatSeq}, when invoked, will be found externally to this file (hopefully in a module jointly assembled). At \([d]\), the label \texttt{MAX} is similarly exported as the function \texttt{CatSeq} will need to use it. Note the difference in use between \texttt{.global} and \texttt{.extern}. The rest of the program at \([f]\) remains the same except that the portion for the function \texttt{CatSeq} is no longer here. Note that the parts for data and end at \([g]\) and \([h]\) are the same. The file \texttt{A4CatSeqA-Extern.s} contains only the function \texttt{CatSeq} and thus its name (label) is immediately exported for all to see at \([i]\) through the \texttt{.global} command. The rest of the code for \texttt{CatSeq} remains the same in \([j]\) to \([m]\), and it can even have its own data (not necessary here) and it must have its own \texttt{end} command.

Once the \texttt{.global} and \texttt{.extern} declarations have been properly set, the program can be executed in \texttt{ARMSim#} by using the \texttt{File \mid Load Multiple} to load both assembly files, through a dialogue box. Step through the execution. Notice the transition between modules when \texttt{BL CatSeq} is invoked.

### 5.3.6 The fourth deliverable: the implementation using both C and ARM

Finally write the last version of the Catalan program using a mixture of C and ARM. Use the instructions posted on the website to combine multiple object files and to make sure that all parameters are appropriate (be careful about what the compiled C code may change in registers!). Note carefully: the goal here is that you do not write any new code. You should be able to insert your original C code for \texttt{CatSeq} in a separate file called by the ARM program. We will check if any changes were made between the two versions! Use the following template with functions and procedures as indicated.
Main: in ARM assembly, in a file A4CatalanAC.s.
CatSeq: in C, in a file A4CatSeqCsub.c.
PrintSeq: in ARM assembly, in the same file as main.
Any other function: in ARM assembly, in the same file as main.

Basically you are repeating the third implementation (all in ARM with external function) while substituting the C code for CatSeq with an external C source file, which needs to be cross compiled to ARM code. Then ARMSim# will be able to link it in with the assembled A4CatalanAC.s and the program should run as it did both in the second full ARM implementation and the third one with multiple files.

5.3.7 Some tricks and hints

1. You may think your program has gone into an infinite loop when you test it to produce more than 10 Catalan numbers in ARM. This is mainly due to the slow execution of the division function (there are more efficient versions, feel free to find them and code them). Thus test your program step by step with \( \text{MAX} = 5, 6, 7, \text{etc.} \) up to 15 at least, and then submit it with \( \text{MAX} = 11 \). We will change it ourselves.

2. The program gets even more inefficient when the CatSeq function is external. You may have to wait quite a bit for it to return when \( \text{MAX} = 15 \), so it is best to test up to \( \text{MAX} = 12 \) at most.

3. There is no division in ARM so in the second program all in ARM you have to code yourself a division function. You do not need to do this in the CatSeq version of C in the C first program. However when you make CatSeq in C external to the calling main program in ARM, the compilation to object code will produce non-available instructions for the operator in C, and ARMSim# will reject your code. Solution? Write a function for dividing in C (you can use the same repeated subtraction pseudo code), include it in the external file for CatSeq and call it instead of using the operator.
Chapter 6

Analysis and Conclusions

It is time to attempt to give some answers to the main questions which have been the objectives of this report.

- Is the Raspberry Pi suitable for CSC 230? Is the Arduino suitable for CSC 230?
- What are the main requirements?
- What are the advantages and disadvantages?
- Is it possible to make a final recommendation?

6.1 Raspberry Pi: advantages and disadvantages

First of all it would appear logical that such a successful device must have lots more advantages than disadvantages. However the main disadvantages arise when the Raspberry Pi is employed in heavy computation. Of course this is not a proper use of the device because is not meant to do such tasks, although it has a quad core and it is pretty powerful compared to its dimension and power consumption.

6.1.1 Highlighting the advantages of the Raspberry Pi

- Low cost: this is a huge advantage. With $35 everyone can buy one and enjoy the compact micro-computer experience. This promotes its use especially in lower levels of education where students can be introduced to computer programming without the need to spend lots of money to buy standard computer and OS licenses.
• **Size**: here the benefits are two-fold, namely portability and encumbrance. Such a small device can be easily moved from one location to the other. It takes only a few minutes (if one has good instructions) to connect keyboard, mouse, monitor and power supply to be ready to go. Its small size makes it a good media player, in fact it is easy to connect it to a TV via its HDMI port and watch movies stored into an external hard drive.

• **Learning and experimenting**: the Raspberry Pi offers to everyone the possibility to learn how to write computer programs, with an easier operating system that can broaden the users’ knowledge. It is also a superb platform for experimenting on really interesting electronics projects. On the web, an endless list of projects can be found such as: radio, phone, quad-copter, cluster.

• **GPIO pins**: this I/O interface is a really useful feature since it allows the user to connect the Raspberry Pi to some other custom electronic devices and make them interact. This is the reason why there can be such a lot of interesting projects. The use of this interface is very easy because the OS handles all the low-level details of the communication while providing the user with a clean API ready to use.

• **Support**: the Linux kernel perfectly supports all components of the Raspberry Pi out of the box. One never ends up compiling custom kernel modules to fix some incompatibility.

• **VideoCore IV**: this low-power mobile multimedia processor is specifically designed to make the encoding/decoding of multimedia contents flexible and efficient, relieving the CPU to such expensive tasks.

• **HDMI**: with this high quality digital audio/video output it is possible to connect the Raspberry Pi to any kind of modern TVs and monitors enjoying the high definition of the multimedia reproduction.

### 6.1.2 Highlighting the disadvantages of the Raspberry Pi

• **Sdcard**: unfortunately it ends up being quite slow when the filesystem has to access it to read and especially write large amount of data (where large here is 50MB-100MB or more).
• **Wifi:** the lack of this interface is certainly a disadvantage and hinders its portability. Having a way to connect to the Internet without a cable is certainly a must for the Raspberry Pi, the need to rely on ethernet cable and switches is something that of course one would like to avoid. One can buy an aftermarket USB dongle, but it is something that one might prefer to find already on board.

• **Composite video:** probably the vast majority of users definitely do not need this old audio/video analog interface as nowadays this is an undoubtedly deprecated AV interface.

### 6.2 Opinions and thoughts on Raspberry Pi

In my opinion the Raspberry Pi is by all means an interesting device that can give lots of opportunity either for power users or for beginners. It is certainly worth all the $35. The Raspberry Pi can be adapted to a large variety of tasks as long as they don’t require heavy computational resources. The quad core ARM Cortex is a quite powerful CPU and represents a big advantage with respect to Raspberry Pi model 1. The four USB ports allow to extend easily the device adding a wifi dongle, USB external hard drive etc. The VideoCore GPU speeds up considerably the reproduction of compressed multimedia content permitting the Raspberry Pi to reproduce HD video content without a glitch.

Beyond the amateur electronics projects, which are no doubt very interesting but maybe limited to the hobbistic, there are also some other kind of applications from which users could benefit through the Raspberry Pi. These applications are the lightweight network servers such as: firewalls, proxies, DHCP and DNS resolver. In such cases, the lack of the wifi interface is not a disadvantage anymore. Indeed one would, in any case, connect the Raspberry Pi to a network through an ethernet port in order to avoid the high latency and typical issues of a radio communication. All is needed is an additional ethernet USB card. The most straightforward way to do it is to install a basic NIX distribution, install the required packages and configure them. There is a large variety of choice, the most common packages are **iptables** (firewall), **squid** (web cache proxy), **isc-dhcp-server** (DHCP) and **bind** (DNS). Alternatively one could install a pre-configured distribution that comes with everything installed and offer simple and intuitive GUI tools for the configuration. An example is **OpenWRT**.

Alongside these technical applications, another very important one is education and learning. This inexpensive device can largely be employed for early students education to
computer and computer programming. The Raspberry Pi shows that there no need to buy fancy expensive computers to start doing some programming. By using this device it is possible to appreciate how effective coding could be with very effective yet simple tools, showing that there is no real need of expensive proprietary IDE to get things done.

In summary: this would be an ideal platform for CSC 230.

6.3 Arduino: advantages and disadvantages

The Arduino is not as widespread and popular as the Raspberry PI and yet it has a lot of interesting features which make it a clear competitor in the field.

6.3.1 Highlighting the advantages of the Arduino

- **Inexpensive** Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50.

- **Cross-platform** The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- **Simple programming environment** The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it is conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino.

- **Open source and extensible software** The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it is based. Similarly, one can add AVR C code directly into the Arduino programs.

- **Open source and extensible hardware** The Arduino is based on Atmel’s ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons License, so experienced circuit designers can make their own version of the module, extend it and improve it. Even relatively inexperienced users
can build the breadboard version of the module in order to understand how it works and save money.

6.3.2 Highlighting the disadvantages of the Arduino

- **Debugging** The lack of a proper in-circuit debugger that allows to view the content of registers, place breakpoints, have a watchdog and so on, is in many cases going to be a real disadvantage. This is particularly true for power users who develop complex applications and need these missing functionalities.

- **Industrial application** From the industrial point of view the Arduino does not seem to have had much penetration at all.

6.4 Opinions and thoughts on Arduino

The Arduino is a really interesting item that is not only limited to provide hardware and software but tries, as well, to build a community of users that shares ideas and solution and can be an active part of it. The Arduino is an open source project, intended in the most positive interpretation of the terms, that is, it aims to target the users needs rather than the interests of any company.

Every part of the Arduino project is available to the users: hardware schematics and any kind of software. Users are encouraged to make modifications and share them in order to improve the final product. The online community is very active, there is a huge amount of users projects, information, which can be found on the Arduino forum. Furthermore some successful initiatives have already started from it such as: Arduino Atheart, Casa Jasmina. More information is available on the official web pages.

The range of Arduino boards is wide and satisfies the needs of all users from beginners to expert. Moreover Arduino provides some very useful Shields. These are elements that can be plugged onto a board to give it extra features such as: step-motor driver, ethernet and GSM interface. Finally, Arduino offers a selection of wearable devices of extremely small dimensions equipped with the ATtiny85. The whole range of offers, considering all the boards, shields and wearable items is definitely wide and valid.

Another big advantages of Arduino is the interoperability between different platforms. In fact Arduino perfectly works on GNU/Linux, Mac OS X and Windows. This is a remarkable fact that many other big microcontrollers manufacturers lack. It may not look like a big deal for the industry niche which is largely Windows-based. But it should be considered an
important feature that should be provided by all the major companies, at least at the binary level. The Arduino software development environment provides all the tools necessary to code efficiently even if it lacks some advanced features of the commercial counterparts. The Arduino programming language is extremely useful for beginners, and it no doubt represents an excellent way to expose low level programming to young students and amateurs.

Although the Arduino board has lots of advantages, it cannot handle many different processes at once. Moreover, similarly to the Raspberry PI, it is not designed for projects in a heavy-duty setting where there is need for lots of computing power. Finally it has much less memory compared to other boards [12].

In summary: it would be a great platform for CSC 230, similarly to the Raspberry Pi, except that it might require a lot more extremely low level hardware and programming intervention on the part of a student. This can be seen as both a positive and a negative feature. Within the context of the curricula, where the 3rd year course in hardware is now an optional choice, the deeper exposure to hardware could be beneficial to students who may never take any other hardware-related subjects within their computer science degree. The fact that Arduino is not an industrial success may be a bit of a disappointment to some students and instructors, since it may appear to continue the (expected?) view that academia remains removed from the "real world".

6.5 Comparisons of various boards

There are many other similar boards and microcontroller platforms available such as BeagleBone, PandaBoard, Parallax Basic Stamp as some examples. Each of these has their own strengths, weakness and certain platforms are better for certain types of projects compared to others. The Arduino simplifies the process of working with microcontrollers, and here are some advantages of Arduino over other systems:

- **Inexpensive.** Both Raspberry Pi and Arduino boards are relatively inexpensive compared to other board. The Arduino module can be assembled by hand and even the pre-assembled Arduino modules cost less than the Panda board and BeagleBone. Other boards are more expensive.

- **Cross-platform.** The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- **Simple, clear programming environment.** The Arduino programming environment is easy-to-use and flexible. The flexibility of the Arduino allows it to work with any kind
of sensor or chips. The Raspberry Pi has similar flexibility and supporting environment and may in fact provide a better set of software wrapper to contain the setting for very low levels signals and ports.

- **Size.** The size is very small and the boards are easy to use.

- **Open source and extensible software.** The Arduino software is published as open source tools, there is a vast online community and documentation making the knowledge to work and extend the system easily. The Raspberry Pi board has a large set of tools and third-party availability of both software and hardware extras, but they are not necessarily free.

- **Open source and extensible hardware.** The designs for the Arduino modules are available on the official website and published under a *Creative Commons License*. The Arduino board is easy to extend as two rows of sockets along each side of the board allows extension by using the shields. The extensibility of hardware for the Raspberry Pi is based mainly on third-party modules.

The table below shows a comparison of the specifications of the four most popular project boards.

<table>
<thead>
<tr>
<th></th>
<th>Arduino Uno</th>
<th>Raspberry Pi</th>
<th>BeagleBone</th>
<th>PandaBoard</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>ATmega328</td>
<td>ARM11</td>
<td>AM335X</td>
<td>OMAP4430</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>2KB</td>
<td>512 MB</td>
<td>512 MB</td>
<td>1GB</td>
</tr>
<tr>
<td><strong>Ethernet</strong></td>
<td>N/A</td>
<td>10/100</td>
<td>10/100</td>
<td>10/100</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>16 MHz</td>
<td>700 MHz</td>
<td>1GHz</td>
<td>1.2 GHz</td>
</tr>
<tr>
<td><strong>IDE</strong></td>
<td>Arduino IDE</td>
<td>Linux, IDLE, Open Embedded</td>
<td>Python, Scratch, Linux, Eclipse</td>
<td></td>
</tr>
<tr>
<td><strong>Audio/Video</strong></td>
<td>N/A</td>
<td>HDMI, Analog</td>
<td>HDMI</td>
<td>HDMI, Analog</td>
</tr>
</tbody>
</table>
6.6 Conclusions

This document has given an high level description of two possible boards to be used in CSC 230: the Raspberry Pi and the Arduino. The more in depth exploration was done on the Raspberry Pi, with previous assignments tested on a sample platform and a tutorial for a beginner developed step by step. An overview of the possibilities that both may offer has also been presented.

In the first part the actual devices have been discussed with their components from a technical point of view. Although they both have small dimensions, they offer enough computational power to permit to run lightweight NIX distribution and to be employed with success in some interesting applications. The advantages and disadvantages of both the Raspberry Pi and the Arduino have been highlighted. It may appear that the Raspberry Pi has more advantages than disadvantages, and that the latter can easily be circumvented employing some extra components.

The case studies, using all the advanced assignments of CSC 230 have been developed only for the Raspberry PI which seemed at the time to be the most likely platform.

The Arduino and Raspberry Pi are very different. As a summary, one could state that the Arduino is based or focused more on the hardware, whereas the Raspberry board leans towards more software-based projects. Thus they can provide a very different focus and trend in a second year course. The Raspberry Pi is a mini-computer, running a Linux operating system, while the Arduino is a microcontroller, without the typical OS style. The Raspberry pi is not an embedded system per se, where as Arduino is truly seen and configured to become an embedded system. The Raspberry Pi board was made mainly to learn about the computer. The boards are focused on very different ideas, with the combination of both making a really powerful possibility (but unrealistic). The combined project would have the power of the Raspberry Pi and its IDE, together with the external interface of an Arduino board.
Bibliography


Appendix A

Assignment Case Study: the Sieve of Eratosthenes on the Raspberry PI

A.1 The implementation for the Raspberry PI

A.1.1 The C program

Sieve C Program:
/* File: A4sieveSu10C.c - Assignment 4 */
/* Author: Amanpreet */
/* Description: Sieve of Eratoshthenes */
/* Finds all primes from a range {0..MAXLIST} up to */
/* a given MAX and displays them */

/******* The whole program here is in C ********/

#include <stdio.h>
#include <stdlib.h>
#define MAX 50 /* MAX for current search*/
#define MAXLIST 100 /* max range */
#define PRIMES_PER_LINE 9 /* max number printed per line */

/*** Declarations of functions */
void Initialize(int sievelist[]);
extern int FindPrimes(int sievelist[], int primeslist[]);
void PrintPrimes(int numberprimes, int primeslist[]);

/*** MAIN ***/
int main() {
    int numprimes;
    int sieve[MAXLIST], primes[MAXLIST];   /*arrays for 2 lists */
    printf("Sieve Program starts\n\n");
    Initialize(sieve);
    numprimes=FindPrimes(sieve, primes);
    printf("Prime numbers up to %3d are:\n ",MAX);
    PrintPrimes(numprimes, primes);
    printf("Sieve Program ends\n");
    return (0);   /* successful completion */
}

/* =====Initialize (&sievellist)
Description: Initializes the Sieve’s array
Inputs: address of sieve array
Outputs: none
Side effects: none
Array here is integers, but could be bits or chars*/
void Initialize(int sievelist[]) {
    int i;
    sievelist[0] = 0;
    sievelist[1] = 0;
    for (i=2; i<MAX; i++) sievelist[i] = 1;
}

/* ===== PrintPrimes(numberprimes,&primeslist)
Description: Prints the prime numbers found from the primes array
    at most PRIMES_PER_LINE in each line
Inputs:   Number of primes
          address of array
Outputs: none
Side effects: none
Expectation: array contain at least 1 element */
void PrintPrimes(int numberprimes, int primeslist[]) {
    int i;
    int l=1; /*max to print per line*/
    for (i=0;i<numberprimes;i++) {
        printf("%d \t",primeslist[i]);
        l++;
        if (l==PRIMES_PER_LINE) {
            printf("\n\t");
            l=1;
        }
    }
    printf("\n");
}

A.1.2 The ARM Program

Sieve Arm Program:
@ CSc 230 Assignment #4
@ File: A4sieveA-Extern.s
@ Author: Amanpreet
@ Description: Sieve of Eratosshthenes
@ Finds all primes in a range {0..MAXLIST} and display
@ them on screen

@ =============================================
@ This file contains ONLY the function Findprimes
@ called as an external subroutine from file SieveWithExtern-ARM.s
@ Use in ARMSim with Load Multiple
@ =============================================

.global FindPrimes @exported symbol
.text

@ ===== int FindPrimes(&sieve:R0, &primes:R1)
© Description: Finds the prime numbers up to a preset MAX and places them in array
© Inputs: R0 - initial sieve array
© R1 - array for primes
© Outputs: number of prime numbers found
© Side effects: none
© Expectation: sieve array has been initialized

FindPrimes:

STMFD SP!,{R1-R9,LR} @ save local use registers
mov r8,#0 @counts number of primes found
mov r3,#MAX @max size for current search
sub r3,r3,#2 @max size -2 is search loop
mov r9,r0 @keep in r9 &sieve
add r0,r0,#8 @go to sieve[2]
mov r4,#2 @r4 is i

 tbloop:
  ldr r2,[r0] @r2 := sieve[i]
cmp r2,#0 @if sieve[i]=0 go to next entry
  beq nextelem
  str r4,[r1],#4 @found prime, store it in primes table
  add r8,r8,#1 @count the primes found
  mov r5,r4, LSL #1 @j := i*2 to set up new flags in table

  flagloop: @for (j=i*2 @ j<MAX j+=i)
    cmp r5,#MAX @end of sieve?
    bge nextelem
    mov r6,r5, LSL #2 @r6 := j*4 = number of bytes offset
    mov r7,#0 @set table[j] := 0
    str r7,[r9,r6]
    add r5,r5,r4 @j := j+i
    bal flagloop

nextelem:
  add r0,r0,#4 @go to next entry in sieve
  add r4,r4,#1 @i++
  subs r3,r3,#1 @check for end of table
  bne tbloop
FP_Ret:
    mov    r0,r8         @return number of primes found
    LDMFD   SP!, {R1-R9,PC}

@ **** Data section ***********************************************
.data
.end
Appendix B

Assignment Case Study: the Kolakoski Sequence on the Raspberry PI

B.1 The implementation for the Raspberry PI

B.1.1 The C program

Kolakoski C Code

/* File: A4kolakosky.c - Assignment 4 */
/* Author: Amanpreet */
/* Description: Kolakosky sequence */
// the Kolakoski sequence A0002 is an infinite list that begins with
// 1,2,2,1,1,2,1,2,2,1,1,2,1,1,2,2,1,1,2,1,2,2,1,.
// The only numbers allowed are 1 and 2
// Each number states how many numbers need to be appended to the sequence
// There can be no more than two of the same number in sequence
// Every time we a new number is considered (read), one must alternate
// between writing 1 and 2
// Example:
// write 1;
// ==> [1]
// read 1 as the number of 1’s to write before switching to 2;
// ==> [1]
// write 2;
//  ==> [1 2]
// read 2 as the number of 2's to write before switching back to 1;
//  ==> [1 2 2]
// read the new 2 as the number of 1's to write;
//  ==> [1 2 2 1 1]
// read the new 1,1 as stating one needs to write one 2's and one 1's
//  ==> [1 2 2 1 1 2 1]
// continue generating forever.

// ===== The Pseudo Code is for the algorithm for the Kolakoski
// sequence labelled A006928 for which a closed form algorithm can be found
// Kseq[0] = 0  ==> not part of the sequence
// Kseq[1] = 1
// Kseq[2] = 2
// assume Kseq indexes range from 1 upwards
//  for(n=2 to MAXRUNS)
//   for(i=1 to i= Kseq[n]
//     append (1+(n%2)) to Kseq
//   print Kseq from Kseq[1]

#include <stdio.h>
define MAXRUNS 7        /* MAX number runs for current sequence*/
define SEQ_PER_LINE 10  /* max numbers printed per line */

/* ==== Declarations of functions */
extern int AppendKseq(int Kseq[],int index,int *Num1);
void PrintKseq(int Kseq[],int index);
/* ==== MAIN */
int main() {
    int Kseq[MAXRUNS*2];       /*array for sequence */
    int index=3;
    int Num1;                   /*number of entries=1*/
    printf("Jean Luc Picard V00123456\n");
    printf("Kolakoski Program starts\n\n");

Kseq[0] = 0;  /*initialize*/
Kseq[1] = 1;
Kseq[2] = 2;
Num1=1;
index=AppendKseq(Kseq,index,&Num1);

printf("Kolakoski sequence of length \%d with \%d Runs\n",index-1, MAXRUNS);
printf("Number of 1's is = \%d \
",Num1);
printf("Number of 2's is = \%d \n",((index-1)-Num1));
PrintKseq(Kseq,index);
printf("Kolakoski Program ends\n");
return (0);  /* successful completion */

} /* ===== PrintKseq(Kseq,index) */

// Description:  Print the Kolakosky sequence
// Inputs:  address ofKseq array,
//   index = next position to be filled
// Output:  none
// Side effects: none
// Expectation: Kseq is not empty
void PrintKseq(int Kseq[],int index) {
    int i;
    int j=0;
    for (i=1;i<index;i++) {
        printf("\t %d",Kseq[i]);
        j++;
        if (j==SEQ_PER_LINE) {
            printf("\n");
            j=0;
        }
    }
    printf("\n");
}
B.1.2 The ARM Program

@ CSc 230 - Assignment #4
@ File: AppendKseq-Extern.s
@ Author: Amanpreet dhaliwal
@ Description: Kolakoski sequence
@ Append the next elements in the
@ Kolakosky sequence to the initial array
@ ===== int AppendKseq(R0:&Kseq,R1:index,R2:&Num1)
@ Inputs: R0 = & Kseq array,
@ R1 = index of next position to be filled
@ R2 = & Num1 = number of entries equal to 1
@ Output: updated index in R0 = next position to be filled
@ Side effects: update number of entries equal to 1 in R2
@ Expectation: Kseq array has been initialized
@ Pseudo Code:
@ for (n=2;n<MAXRUNS;n++)
@ for (i=1;i<=Kseq[n];i++)
@ Kseq[index] = 1 + (n%2);
@ if (Kseq[index] == 1) (*Num1)++;
@ index++;
@ return(index);

@ This file contains ONLY the function AppendKseq
@ called as an external subroutine from file KolakoskyARM+extern.s
@ Use in ARMSim with Load Multiple
@ ================

.global AppendKseq @exported symbol
.text
AppendKseq:
STMFD sp!,{r1-r9,lr}
mov r9,#2 @n=2
ldr r3,[r2]  @r3=Num1

OuterLoop:
  cmp r9,#MAXRUNS  @i=1
  bgt DoneAppendKseq
  mov r8,#1  @i=1
  ldr r7,[r0,r9,LSL #2]  @r7 = Kseq[n]

InnerLoop:
  cmp r8,r7  @is i = Kseq[n]?
  bgt EndInnerLoop
  and r6,r9,#0x00000001  @r6=r9%2
  add r6,r6,#1
  str r6,[r0,r1,LSL #2]  @kseq[index]=new entry
  add r1,r1,#1  @index++
  cmp r6,#1  @did we add an entry=1?
  bne InnerIncr
  add r3,r3,#1  @else Num1++

InnerIncr:
  add r8,r8,#1  @i++
  bal InnerLoop

EndInnerLoop:
  add r9,r9,#1  @n++
  bal OuterLoop

DoneAppendKseq:
  mov r0,r1  @return new index
  str r3,[r2]  @update Num1
  LDMFD sp!,{r1-r9,pc}
.end
Appendix C

Assignment Case Study: the Catalan Numbers

C.1 The implementation for the Raspberry PI

C.1.1 The C program

/* File: A4Catalan.c - Assignment 4 */
/* Author: Amanpreet dhaliwal */
/* Description: Catalan numbers sequence */

// The Catalan numbers are an infinite list that begins with
// 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440,...
// The formula is stated as:
// Cn = [1/(n+1)] * (2n)
// (n)

// where the above formula implies (2n) choose (n)
// The more efficient way to calculate the numbers is
// to use a recurrence relation instead:
// C[0] = 1
// C[n+1] = { [2*(2n+1)] / (n+2) } * C[n]

// Simple enough but one must be careful about the
// order of computation (!) as it is supposed to be
// all integer arithmetic.

// This program computes the first MAX Catalan numbers
// where:
//   - MAX is initialized at compile time (no user I/O).
//   - A function CatSeq is called to compute the numbers
//     and place them in an array (static).
//   - A procedure PrintSeq is called to print the numbers.

#include <stdio.h>
#define MAX 15     /* MAX number runs for current sequence*/
#define SEQ_PER_LINE 5  /* max numbers printed per line */

/* ==== Declarations of functions */
extern int CatSeq(int CatArray[], int n, int top);
void PrintSeq(int Seq[], int top);
/* ==== MAIN */
int main() {
    int Catalans[MAX];     /* array for sequence */
    int index = 0;
    int UpLimit = MAX;     /* number of entries */
    printf("Jean Luc Picard V00123456\n");
    printf("Catalan Program starts\n\n");
    Catalans[0] = 1;        /* initialize */
    index = CatSeq(Catalans, index, UpLimit);
    printf("The Catalan number sequence of length %d :\n", MAX);
    PrintSeq(Catalans, index + 1);
    printf("Catalans Program ends\n");
    return (0);            /* successful completion */
}

/* ===== PrintSeq(Seq,top) */
// Description: Print a sequence from an array
// Inputs: address of array sequence,
void PrintSeq(int Seq[], int top) {
    int i;
    int j = 0;
    for (i = 0; i < top; i++) {
        printf(" %d\t", Seq[i]);
        j++;
        if (j == SEQ_PER_LINE) {
            printf("\n");
            j = 0;
        }
    }
    printf("\n");
}

C.1.2 The ARM Program

@ File: CatSeq-Extern.s - Assignment 4 */
@ Author: Amanpreet dhaliwal
@ Description: Catalan numbers sequence */
@ The Catalan numbers are an infinite list that begins with
@ 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440,...
@ The formula is stated as:
@ Cn = [1/(n+1)] * ( 2n )
@ ( n )
@ where the above formula implies (2n) choose (n)
@ The more efficient way to calculate the numbers is
to use a recurrence relation instead:
@ C[0] = 1
@ C[n+1] = { [2*(2n+1)] / (n+2) } * C[n]
@ Simple enough but one must be careful about the
order of computation (!) as it is supposed to be
all integer arithmetic.

This program computes the first MAX Catalan numbers
where:
- MAX is initialized at compile time (no user I/O).
- A function CatSeq is called to compute the numbers
  and place them in an array (static).
- A procedure PrintSeq is called the print the numbers.

This file contains ONLY the functions  CatSeq and Div
called as an external subroutines from file CatalanARM+extern.s
Use in ARMSim with Load Multiple

.global CatSeq  @exported symbol
.text

@ ===== int CatSeq(R0:&CatArray;R1:n;R2:top) */
@ Description: Append the next elements in the
  Catalans sequence to the initial array
@ Inputs: address of initial Catalans array,
  n = next position to be filled
  top = max number of elements to be generated
@ Output: index = number of elements in array
@ Side effects: update entries in Catalans array
@ Expectation: CatArray[0] has been initialized
CatSeq:
  STMFD   sp!,{r1-r9,lr}
  mov      r9,#1       @i=1
  mov      r5,r0       @r5=&CatArray

OuterLoop:
  cmp      r9,r2       @all elements generated?
bge     DoneAppendSeq
mov     r8,r1,lsl #1  @r8=2*n
add     r8,r8,#1      @r8=2*n+1
mov     r8,r8,lsl #1  @r8=(2*(2*n+1))
ldr     r7,[r5],#4     @r7=CatArray[n]
mul     r6,r7,r8       @r6=CatArray[n] * (2*(2*n+1))
add     r4,r1,#2       @r4=n+2

mov     r8,r1         @save for later
mov     r7,r2         @save for later
mov     r10,          @save for later
mov     r1,r6         @r1=divident
mov     r2,r4         @r2=divisor
BL Div               @R0:quotient Div(R1:dividend; R2:divisor)
str     r0,[r5]       @Store Cat[n+1]
mov     r1,r8         @restore
mov     r2,r7         @restore
mov     r0,r10        @restore
add     r1,r1,#1      @n++
add     r9,r9,#1      @i++ for loop counter
bal OuterLoop

DoneAppendSeq:
  mov     r0,r2        @return number of elements
  LDMFD sp!,{r1-r9,pc}

=====int Div (R1:m as dividend; R2:n as divisor)
)* Description: Divide m/n using repeated subtractions
)* Inputs: R1 = m as dividend,
)* R2 = n as divisor
)* Output: R0 = quotient
)* Side effects: none
)* Expectation: n != 0

)* Algo: Repeatedly subtract the divisor N from M, as in M = M - N.
)* Count the number of iterations Q until M < 0.
This is one too many iterations and the quotient is Q - 1.
The remainder is M + N, the last positive value of M.
"M MOD N = remainder" is not returned

Pseudo-code 1
integers: m, n, quotient, remainder
quotient = 0
DO
    quotient = quotient + 1
    m = m - n
WHILE m >= 0

quotient = quotient - 1
remainder = m + n

Div:
    stmfd sp!,{r1-r4,lr}
    mov r3,#0 @ r3 = quotient = 0
subloop:
    add r3,r3,#1 @ r3 = quotient + 1
    subs r1,r1,r2 @ compute m - n
    bpl subloop
setresult:
    sub r3,r3,#1 @ r3 = correct quotient
    add r1,r1,r2 @ r1 = remainder
    mov r0,r3 @ return quotient
    ldmfd sp!,{r1-r4,pc}
.end