TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SIRIPHAT POMYEN

SIGNAL AND IMAGE PROCESSING WITH MATLAB ON RASPBERRY PI PLATFORM

Master of Science thesis

Examiner: Prof. Irek Defee
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 4 February 2015

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SIRIPHAT POMYEN: Signal and Image Processing with Matlab on Raspberry Pi Platform
Master of Science thesis, 64 pages
May 2015
Major: Multimedia
Examiner: Prof. Irek Defee
Keywords: Raspberry Pi, Signal and Image processing, Matlab and Simulink

Raspberry Pi is a powerful and affordable small computer. It was produced with an aim in education to help young adult learn about computer and programming language. Hence, it consists of high functional features and is sold at low cost for easy access to everybody. Moreover, there are many supports available for Raspberry Pi in both technical and commercial field making it appealing for running various kinds of application.

In this thesis, Raspberry Pi computer is studied and evaluated for its features and capability using Matlab software as a code porting tool due to a free availability of the support package. The implementation is done with examples of signal and image processing based on problem from relevant courses in university level as well as real world applications from Simulink Computer Vision toolbox. The model is constructed using Simulink blocks and Matlab scripts. The input is provided in two ways; one as a parameter added through the software and the other is fed directly from a peripheral device. With an aim to provide the use of Raspberry Pi computer in practical courses, a limitation of model compilation time is taken into account for the evaluation. The results from examples show that time taken to generate the code is slower if the model comprises many Simulink blocks or contains Matlab scripts. The larger size and software input parameter also causes the delay compilation. However, these factors fall into an acceptable range. Hence, Raspberry Pi can be used with Matlab as a learning tool for hands-on experience for students. The possibility of the future work would include the use of graphic processor in Matlab and Simulink application. The research would involve determining Matlab code generation method used as a compiler for a graphic processor modified code.

PREFACE

This thesis has been carried out at the Department of Signal Processing at Tampere University of Technology, Finland. It contains the work studied and evaluated on Raspberry Pi computer. The thesis has been supervised by Prof. Irek Defee and entirely prepared by the author based on available information of the product as provided in references.

I would like to warmly thank my supervisor Prof. Irek Defee for his time and guidance on the topic. Working on it has been interesting and enjoyable. There are new information to learn and update all the time. However, one could not deny the difficulty when no good ideas come up during the writing and implementing process. I feel I have learned a lot, not only about the topic but also about myself. This thesis period will always remain one of my memorable experience in Finland.

Lastly, thank you to my family and friends for your supports and thank "you" for reading.

SIRIPHAT POMYEN
Tampere, May 2015

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

ABI	Application Binary Interface
ALSA	Advanced Linux Sound Architecture
ALU	Arithmetic Logic Unit
API	Application Programming Interface
BIU	Bus Interface Unit
CE	Chip Enable
CLE	Control List Executor
CPU	Central Processing Unit
CRT	Cathode Ray Tube
CSI	Camera Serial Interface
DCU	Data Cache Unit
DPU	Data Processing Unit
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DSI	Display Serial Interface
EAN	European Article Number
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEP	Front End Pipe
FFT	Fast Fourier Transform
FHS	Filesystem Hierarchy Standard
FIFO	First In, First Out
GPGPU	General Purpose computing on Graphic Processing Unit
GPIO	General Purpose Input/Output
GPU	Graphic Processing Unit
HAT	Hardware Attached on Top
HD	High-Definition
HDMI	High-Definition Multimedia Interface
I2C	Inter-Integrated Circuit
I2S	Integrated Interchip Sound
IC	Integrated Circuit
ICU	Instruction Cache Unit
IoT	Internet of Things
L2C	Level 2 Cache
LAN	Local Area Network
LED	Light Emitting Diode
LSB	Linux Standard Base

MISO	Master Input, Slave Output
MOSI	Master Output, Slave Input
MSE	Mean Square Error
NOOBS	New Out Of the Box Software
ntpd	Network Time Protocol daemon
NTSC	National Television System Committee
OS	Operating Systems
PAL	Phase Alternating Line
PC	Personal Computer
PFU	Prefetch Unit
Pi-GEMM	Raspberry Pi General Matrix Multiply
PSE	Primitive Setup Engine
PTB	Primitive Tile Binner
QIC	Instruction cache
QPS	Quad Processing Unit Scheduler
QPU	Quad Processing Unit
QUC	Uniforms cache
RAM	Random Access Memory
RCA	Radio Corporation of America
SCL	Serial Clock Line
SCLK	Serial Clock
SCP	Secure Copy Protocol
SCU	Snoop Control Unit
SD card	Secure Digital Card
SDA	Serial Data Line
SDL	Simple DirectMedia Layer
SDR	Software Defined Radio
SFU	Special Function Unit
SIMD	Single Instruction, Multiple Data
SoC	System on a Chip
SPI	Serial Peripheral Interface
SS	Slave Select
SSH	Secure Shell
STB	Store Buffer
TB	Tile Buffer
TCP	Transmission Control Protocol
TMU	Texture Memory Unit
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol

URL	Uniform Resource Locator
USB	Universal Serial Bus
V4L2	Video for Linux Two
VCD	Vertex Cache Manager Direct Memory Access
VCM	Vertex Cache Manager
VNC	Virtual Network Computing
VPM	Vertex Pipeline Memory
VRI	Varying Interpolation Unit

1. INTRODUCTION

The very first computer had a size as large as a room and consumed high amount of power. However, things had been changed since semiconductor technology was introduced. The size of computer became rapidly shrinking while the computing power was growing. The advent of integrated circuits (ICs) or chips had speeded up the miniaturization of computer technology where smaller and smaller elements are integrated to build a chip [47]. This is described by so called Moore's law which says that the number of transistors on chip is doubling every two years [49]. As a result, personal computers (PCs) and other devices such as laptops, mobile phones, tablets have been developed and using in daily life. Moreover, the process of miniaturization of computers continues leading to an emergence of very small computers which can be used in many applications covering every aspect of life. Ideas for such new applications could be developed by anybody interested if they have knowledge about computer operation, programming and access to a very small experimental platform. In particular, this could be of a great attraction to children, young people and school education. Replying to this potential, a fully operational computer of a credit card size named Raspberry Pi has been developed by a foundation of the same name. Raspberry Pi was originally designed to provide support for school students as a tool for learning about computers, and programming languages. The idea was to make a device which is attractive for hands-on experimenting and cheap enough that everybody could afford [36]. This goal has been achieved and Raspberry Pi became a big success in the market with millions of copies sold [73]. However, Raspberry Pi is not only useful for school students but also for technology enthusiasts, hobbyists, engineers who are using it in commercial applications as well as in universities both in education and research projects. This success of Raspberry Pi is due to the unique combination of proper price, hardware, software and the creation of global community of users and developers sharing information.

The objective of this thesis is to study and evaluate a capability of Raspberry Pi computer through signal and image processing education at university level. The availability of support packages from Matlab, which is a commonly used software in exercises and laboratories, allows the possibility of using Raspberry Pi for an experiment. In addition, the support packages, used for porting software onto Raspberry

Pi, is available for free of charge. As a result, the board can be used as a target device for running software providing an ability to observe the porting operation of the software developed in simulation environment in PC to the application device. The performance of the software in the application device can then be tested. Using Raspberry Pi gives better opportunity than only seeing simulation results. In addition, it encourages creative thinking and experimenting with new applications. However, the application device like Raspberry Pi has limited capabilities and thus software has to be developed with the limitations in mind. The interaction between the PC and Raspberry Pi is handled by Matlab and Simulink software where Simulink makes possible porting of the Matlab software to wide variety of devices and platforms. Matlab in Raspberry Pi can run both in a simulation mode where the board is connected to a PC and in a standalone mode where a software is downloaded onto the board and runs independently from a PC. Experience gained by using Raspberry Pi with Simulink has universal applications.

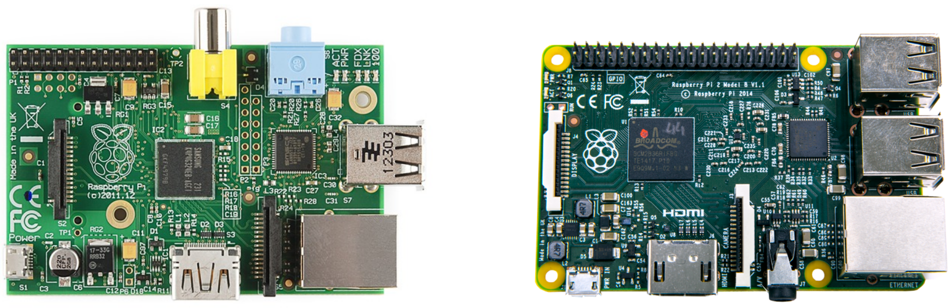
The thesis is organized as follows. Chapter 2 explains background and general information of Raspberry Pi such as released models and components including the details of its architecture: Central Processing Unit (CPU), Graphic Processing Unit (GPU), Random Access Memory (RAM) and general purpose input and output (GPIO) pins with an emphasis on the recently released model of Raspberry Pi, namely Raspberry Pi 2 model B. Basic Operating System (OS) and setup instruction including a secure digital (SD) card preparation, connection and configuration instructions are presented in Chapter 3. Chapter 4 presents the information about Matlab and Simulink support package for Raspberry Pi including setup instructions and detailed functions used to control the computer. The signal and image processing implementation examples based on topics from relevant courses as well as operations of some toolboxes are presented in Chapter 5. The usefulness of Raspberry Pi for the exercise work is evaluated based on the topics of compilation time, computational speed and implementation complexity. Such topics of evaluation concerns the time allocated for class exercises and laboratory work. Hence, it is important to search for the reasonable limits. Finally, Chapter 6 provides the conclusion and the suggestions for future work.

2. RASPBERRY PI PLATFORM

This chapter provides a background information of Raspberry Pi starting with general information such as released models and basic components including GPIO operations. It also provides the information regarding the history of this single-board computer as well as a detail of its system on a chip (SoC) including CPU, GPU and RAM.

2.1 General Information

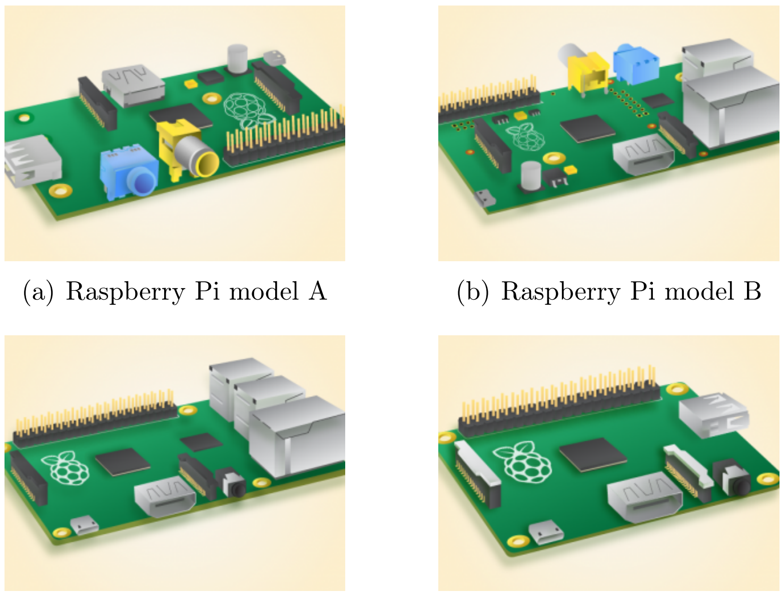
Raspberry Pi, as shown in Figure 2.1, is a single-board computer having a size as small as a credit card from the Raspberry Pi foundation. An idea of producing Raspberry Pi began in 2006 with a realization that young generation is missing knowledges about computer operation. A group of academics and engineers from University of Cambridge, therefore, decided to develop a very small computer which everyone could afford to buy to create learning environment in programming. The Raspberry Pi project became promising with the appearance of cheap and powerful mobile processors with many advanced features allowing a possible development of Raspberry Pi which was continued under specially created Raspberry Pi foundation with the first product launched in 2012 [36].



(a) Raspberry Pi model B   (b) Raspberry Pi 2 model B

***Figure 2.1** Raspberry Pi's generation 1 and generation 2*

Development of Raspberry Pi computer is a continuing process. Until spring of 2015, Raspberry Pi foundation has released five models in two generations of the computer. The first generation consists of four models as shown in Figure 2.2. First released was model B followed by model A, model B+ and model A+ respectively. The two latter models are upgraded versions of their previous releases to make the computer more efficient and convenient to users especially by having lower power consumption and more (Universal Serial Bus) USB ports. The second generation consists of only one model called Raspberry Pi 2 model B whose specification is based on Raspberry Pi model B+ but with faster CPU and more memory [50].



(a) Raspberry Pi model A

(b) Raspberry Pi model B

(c) Raspberry Pi model B+

(d) Raspberry Pi model A+

Figure *2.2 Raspberry Pi's released models*

Raspberry Pi model B, as shown in Figure 2.2(b), was released in the early 2012 with the specification of 256 MB RAM, two USB ports and one Ethernet port. Later in the same year, in a new release, the amount of RAM was increased to 512 MB followed by the release of the lower-spec board, model A as shown in Figure 2.2(a). Model A was released with same amount of RAM of the older model B at 256 MB but with one USB port and no Ethernet port. These first generation computers use Broadcom SoC, BCM2835, which integrates 700 MHz single-core ARM1176JZF-S CPU, VideoCore IV GPU and variety of peripherals. While model B can be used in any applications, the cheaper model A is useful in specific applications that require light-weight and low power consumption such as robotics or any portable media services.

In 2014, an upgraded version of the two previous models were launched, given the name B+ and A+, as shown in Figure 2.2(c) and 2.2(d). Although, the two new models have the same CPU, GPU and RAM as their previous releases, several upgrades and new features have been added such as a micro SD memory socket replacing an SD Card slot, an upgraded version of GPIO from 26 to 40 pins, a low noise audio and a 0.5 to 1 W lower power consumption by having a switching regulator instead of a linear regulator. Additionally, model A+ is approximately 2 cm shorter than its predecessor while model B+ provides four USB ports instead of two in model B [19][20].

In early 2015, the next generation of Raspberry Pi computer called Raspberry Pi 2 model B was released. The model represents significant upgrade of the Raspberry Pi capabilities and was enthusiastically welcomed by the community. Raspberry Pi 2 model B uses a much more powerful BCM2836 which offers 900 MHz quad-core ARM Cortex-A7 CPU, 1 GB RAM and the same specification of graphics processor as previously making it run approximately six times faster comparing to its predecessors at an unchanged price. Moreover, the supports of Windows 10 Internet of Things (IoT) version is available for Raspberry Pi 2 model B for free of charge providing a great potential for future usage [21][71].

Table 2.1 summarizes major features of the Raspberry Pi models. Further details are explained in the next section

Table 2.1 *Major features of the released models of Raspberry Pi*

Raspberry Pi	**Generation 1**				**Generation 2**
Specifications	**Model A**	**Model B**	**Model A+**	**Model B+**	**Model B**
Power	300mA	700mA	200mA	600mA	900mA
Ethernet Port	No	Yes	No	Yes	Yes
USB Port	1	2	1	4	4
GPIO	26	26	40	40	40
SD Card Slot	SD	SD	microSD	microSD	microSD
SoC	BCM2835	BCM2835	BCM2835	BCM2835	BCM2836
CPU	700MHz ARM11 single-core	700MHz ARM11 single-core	700MHz ARM11 single-core	700MHz ARM11 single-core	900MHz ARM Cortex-A7 quad-core
RAM	256MB	512MB	256MB	512MB	1GB

Although a newer Raspberry Pi model provides more features, it is better to know which applications or projects the board is going to be used for in order to select the most efficient one for both power and cost.

In late 2012, Raspberry Pi's GPU driver code running on the ARM was released as open source and available for download [57]. Then later, in 2014, the full register-level documentation including a graphics driver stack of the chip were released pub-

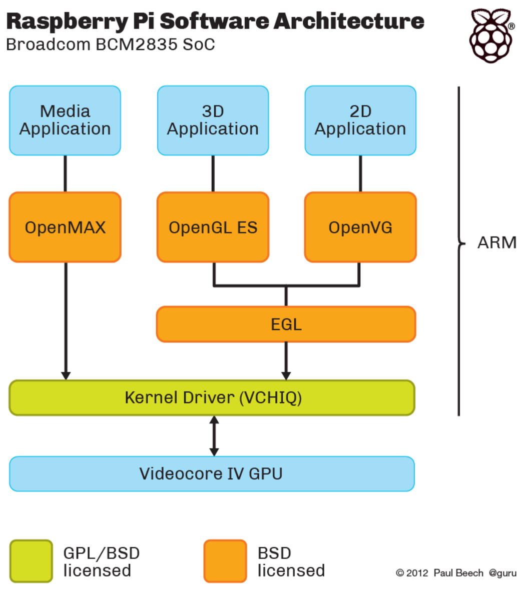


Figure 2.3 *Raspberry Pi's software architecture*

licly [69]. From the diagram shown in Figure 2.3, the part covered with ARM was the first release followed by the part called VideoCore IV GPU. The parts, colored in orange, indicate the closed source such that the driver code running on the ARM was available, however, the graphic libraries provided by outsourced suppliers are closed source [72]. VideoCore IV GPU was a binary blob, which is a closed source binary driver, prior the latter announcement making it difficult for users to fully take advantage of the chip. Although the release from Broadcom was meant for a mobile SoC called BCM21553, it is compatible with BCM2835, a SoC used in Raspberry Pi. This release allows users to fully understand its internal operation as well as develop own open source drivers or write codes for General Purpose computing on Graphic Processing Unit (GPGPU), hence, making it attractive to academics, engineers and hobbyists who are interested in exploiting the capability of the chip.

2.2 Basic Components

The diagram of Raspberry Pi 2 model B containing the basic components is as shown in Figure 2.4. The details of the components are explained as the list below [50][51]. Although the list is based on the second generation of the board, the detail of the components from the first generation is also explained if they are different.

- **Power Connector** is to power the board with a 5 V micro USB port. The need of the power consumption required depends on the peripheral devices

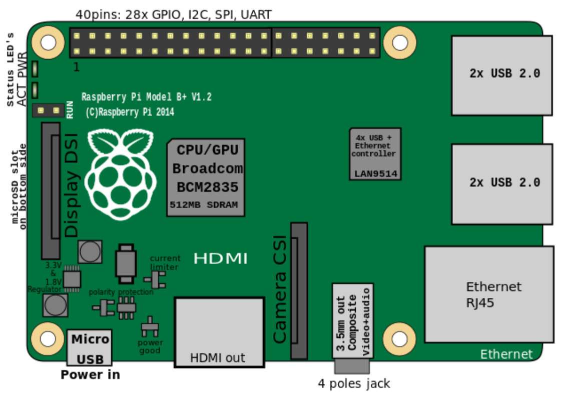


Figure 2.4 *Basic components shown on Raspberry Pi 2 model B diagram*

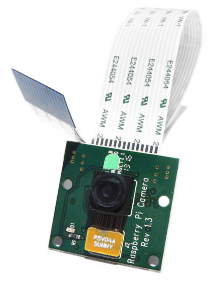
attached to the board, for example, model B requires around 700 mA at the idle state but can consume up to 1000 mA if connects to other devices. In model A+ and B+, the linear regulator was replaced by a switching regulator causing lower power consumption at 200 mA and 600 mA for each model respectively. More power is required for Raspberry Pi 2 model B of at least 900 mA since it has more advanced four core processors. It is recommended to use a power adapter that can produce at least 1200 mA, however, a normal mobile phone charger with a micro USB head can be used as a power supply. Higher power is required for more stable operation [17].

- **High-Definition Multimedia Interface (HDMI) Port** is a connection for High Definition (HD) resolution display device such as a computer screen or a smart television. All models provide a resolution from 640x350 to 1920x1200 including Phase Alternating Line (PAL) and National Television System Committee (NTSC) standards.

- **Camera Serial Interface (CSI) Connector** is a connector for a camera module, as shown in Figure 2.5(a). The camera is specifically designed for Raspberry Pi and can be connected with a ribbon cable providing a support for both photography and HD video [2].

- **Audio Connector** is an audio output through a 3.5 mm headphone jack which also supports an analogue video output for model A+, B+ and Raspberry Pi 2 model B. For earlier models, A and B, the connector is only for an audio output.

- **Ethernet Port** is a network connector providing a speed of 10/100 Mbit/s through an RJ45 Local Area Network (LAN) cable. Ethernet Port is available in model B, B+ and Raspberry Pi 2 model B.


(a) Camera Pi

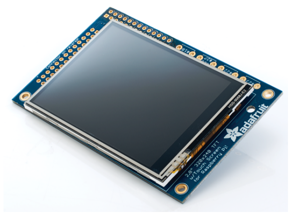

(b) PiTFT

Figure 2.5 *Camera Pi and PiTFT*

- **USB Port** supports any USB devices such as a keyboard, a mouse, a Wi-Fi dongle or a webcam. The USB port provided is a 2.0 version. Both model A and A+ have one USB port while model B has two USB ports. There are four USB ports in model B+ and Raspberry Pi 2 model B.
- **GPIO** is a set of universal input and output connector pins for general purposes such as connecting expansion boards or devices in order to control CPU or checking power consumption. There are 26 pins in model A and B and 40 pins in model A+, B+ and Raspberry Pi 2 model B.
- **Light-Emitting Diodes (LEDs)** indicate the board status, for example, when it is powered, the LED appears red and when it connects to the network, the LED appears yellow. Model A and B provide 5 LEDs in which the first two are for indicating an SD card status and a power status and the other three are for a network status. Model A+, B+ and Raspberry Pi 2 model B, provide only 2 LEDs to indicate SD card and power status. The LEDs for network status are built-in at Ethernet socket.
- **Display Serial Interface (DSI) Connector** is for connecting the board with a display module such as PiTFT, which is a Thin Film Transistor liquid crystal display shown in Figure 2.5(b). The connection is done with a flexible flat cable.
- **SD Card Slot** is for inserting SD memory card to store a desired OS or programs for the board. Raspberry Pi supports up to 32 GB storage and class 10 speed SD card. An SD card slot has been replaced with a micro SD card slot in model A+, B+ and Raspberry Pi 2 model B.
- **Radio Corporation of America (RCA) Video Connector** is for a video output support through a RCA jack allowing a connection with any general televisions or monitors. The connector is only available in model A and B since

in model A+, B+ and Raspberry Pi 2 model B, the video output is shared with a 3.5 mm jack.

- **SoC** is a Broadcom chip containing CPU, GPU and RAM. The first generation of Raspberry Pi, namely model A, B, A+ and B+, uses BCM2835 chip providing 700 MHz ARM1176JZF-S single-core CPU, VideoCore IV GPU and 256 MB or 512 MB RAM depending on the model. For the second generation, Raspberry Pi 2 model B, the chip was upgraded to BCM2836 which provides 900 MHz ARM Cortex-A7 quad-core CPU and 1 GB RAM with the same GPU.

One of Raspberry Pi features which hold special attention is its GPIO pins. GPIO is a generic and reusable pin which can be configured with a logical value, `1` and `0`, to act as an input or output according to a user's purpose in order to indicate the status of the chip and control peripheral devices. Typically, a logical value provided to a pin is a high and low voltage of the chip in which high is a supply voltage and low is 0 V or ground. To prevent a damage caused to the chip from a voltage supplied to GPIO pins, the voltage should be in an acceptable range depending on a specification of each chip. However, it should be considered that supplying a voltage too high than a supply voltage or too low than 0 V is unacceptable. Additionally, a voltage that can not be considered as high or low can cause an unreadable result. GPIO pins are individually configured, however configuring GPIO pins as a group of typically 8 pins results in a GPIO port allowing to perform the same operations of individually configured GPIO pins such as reading input and writing output result [7][8].

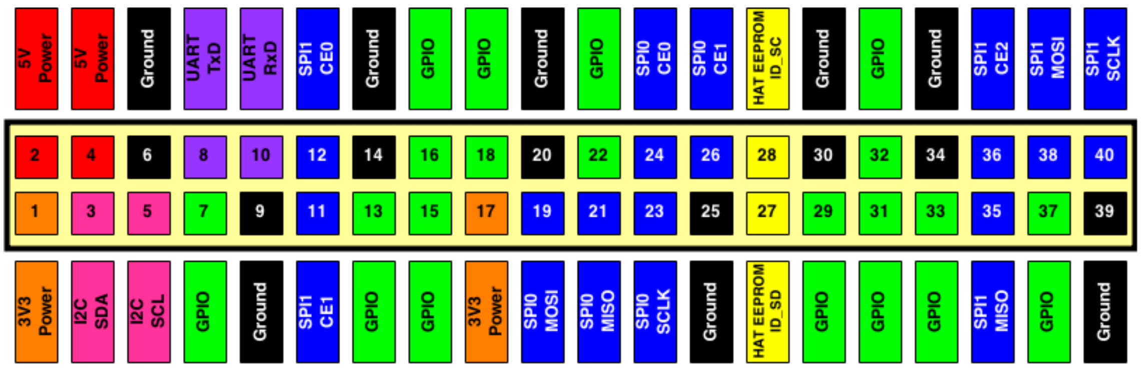


Figure 2.6 *Raspberry Pi's GPIO diagram*

The diagram of Raspberry Pi GPIO pins is shown in Figure 2.6 where the first 26 pins are common for all models and the remaining 14 pins are available only in model A+, B+ and Raspberry Pi 2 model B. GPIO pins allow control over functions such as

Universal Asynchronous Receiver Transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Hardware Attached on Top (HAT) through Electrically Erasable Programmable Read-Only Memory (EEPROM) as well as 3.3 V and 5 V power [24].

The color of the pin shown in a diagram indicates its functionality which can be explained in detail as follows.

- **Power**, colored in orange and red, is provided on 4 pins for 2 levels, namely 3.3 V and 5 V, in which 3.3 V provided on pin number 1 and 17 is considered as a supply voltage of the board while 5 V provided on pin number 2 and 4 is for the purpose of interacting with other devices that require 5 V power supply [52].
- **I2C**, colored in pink, is provided on pin number 3 and 5 for Serial Data Line (SDA) and Serial Clock Line (SCL) respectively. I2C allows a short distance communication between chips on the same board as well as controlling peripheral devices connected to the board via a cable. I2C communication is done through either one or more masters and a number of slaves in which a master generates a clock signal provided on SCL along with initiates a data transmission with a slave via SDA. The available data transfer modes 100 kbp/s, 400 kbp/s, 1 Mbp/s, 3.4 Mbp/s and 5 Mbp/s. I2C on Raspberry Pi can be used to control sensors or other interfaces such as SPI [11].
- **UART**, colored in purple, is provided on pin number 8 and 10 for data transmitting (`TxD`) and receiving (`RxD`). UART is a serial connection allowing two devices to exchange data through their serial interfaces. The connection is done in such a way that a transmitter of the first device is connected to a receiver of the second device and vice versa. In addition, their grounds need to be connected together. UART allows Raspberry Pi to connect to a computer, a microcontroller or any devices with a serial interface [33].
- **SPI**, colored in blue, is a communication interface connection realized in a form of master and slave allowing one master to control one or more slaves through at least four pins, namely Master Output, Slave Input (MOSI), Master Input, Slave Output (MISO), Serial Clock (SCLK) and Slave Select (SS) indicated as Chip Enable (CE) in the diagram. The data is transmitted in a full duplex mode in which a master sends a data to a slave through MOSI and receives a data from a slave through MISO corresponding to a clock signal on SCLK generated by a master. In case of more than one slave available, the slave is selected with a low bit sent from a master through SS. Raspberry Pi offers

two sets of SPI connection, namely SPI0 and SPI1. SPI0 is provided on pin number 19, 21, 23, 24 and 26 allowing the control over two slaves while SPI1 provided on pin number 11, 12, 35, 36, 38 and 40 allows the control over three slaves [34].

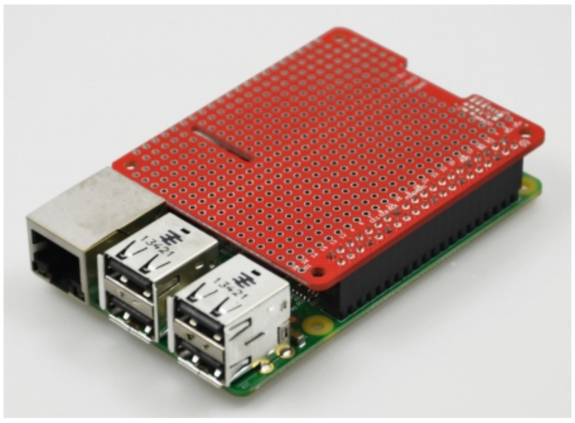


Figure 2.7 *An example of an add-on HAT board on Raspberry Pi 2 model B*

- **HAT**, colored in yellow, is provided on pin number 27 and 28 for the purpose of connecting add-on HAT boards [54] as shown in Figure 2.7, which allows Raspberry Pi to automatically configure GPIO pins and its driver corresponding to an attached device through I2C EEPROM data and clock, namely ID_SD and ID_SC respectively. Raspberry Pi foundation allows a freely design of add-on HAT boards adhering to publicly available specification [26].

By default, all Raspberry Pi GPIO pins act as general purpose input pin except pins assigned as power, ground and UART. However, the UART pins can be configured as a general purpose pin as well. In order to perform any operations using GPIO pins, the pins need to be configured to be either an input or an output, for example, via a programming language such as python or C or via a software support packages such as Matlab.

2.3 System on a Chip

Raspberry Pi's SoC comprises CPU, GPU and RAM. CPU is an ARM based processor which is a CPU used mostly in mobile phones. The first generation of Raspberry Pi uses BCM2835 SoC which offers an ARM version 6 architecture. In the second generation, the board uses BCM2836 ARM version 7 called Cortex-A7 which is supported by much wider range of software.

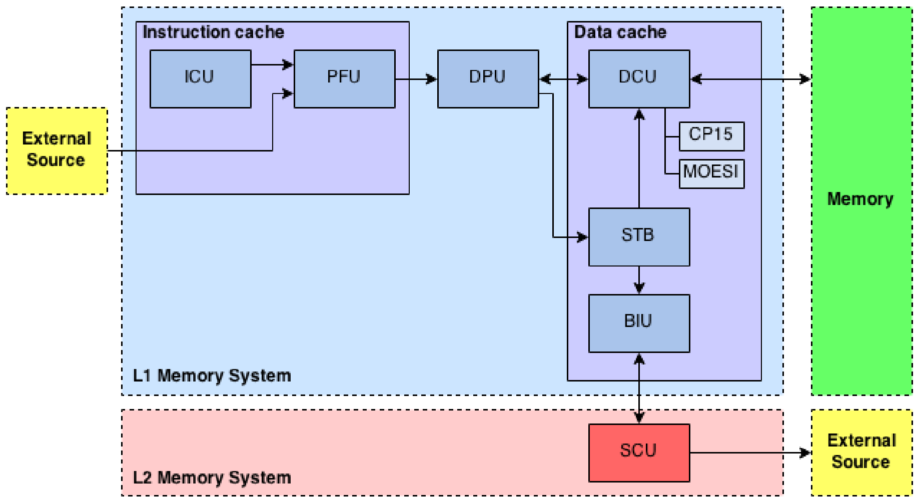


***Figure* 2.8** *ARM Cortex-A7 system block diagram*

ARM Cortex-A7 operates instructions through its Data Processing Unit (DPU) which also holds processor registers such as general-purpose, status and control register. The processor consists of two level memory systems, namely L1 and L2, providing cache subsystem and Snoop Control Unit (SCU) respectively. The cache subsystem in L1 is divided into instruction cache containing Instruction Cache Unit (ICU) and Prefetch Unit (PFU) and data cache containing Data Cache Unit (DCU), Store Buffer (STB) and Bus Interface Unit (BIU). The operation is shown in system block diagram in Figure 2.8 in which instructions from ICU and external sources are fetched to DPU for executing via PFU. In the case that the operation requires data from system memory, DPU operates along with DCU through CP15 which is a system control coprocessor providing a configuration and control over system memory and its associated functionality. DCU also holds MOESI protocol which is a cache coherency protocol stating the shareable data between each processor. After the operations are complete, they are stored in STB waiting to be written out to RAM via DCU or to external source via SCU. SCU, located in L2 memory system, is connected to L1 via BIU which provides SCU interface [56].

Graphic processor provided by Broadcom is a dual-core 3D graphics processing GPU which has performance corresponding to the Xbox 360 or double the performance of iPhone 4S. The GPU is capable of encoding and decoding 1080p30 H.264 Blu-ray quality video and has a speed of 24 GFLOPs for parallelized general purpose computing operations like matrix multiplication [10].

VideoCore IV GPU [58], as shown in Figure 2.9, consists of twelve special purpose floating-point shader processors known as a Quad Processing Unit (QPU). Each QPU is a 4-way single instruction, multiple data (SIMD) processor but it can be

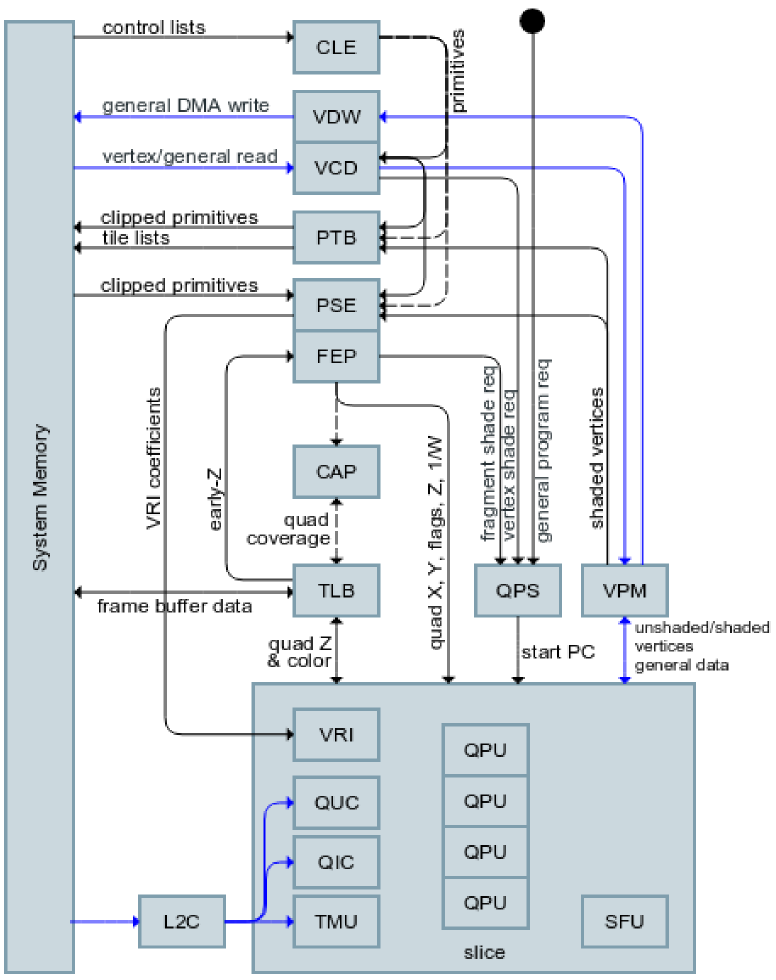


Figure 2.9 *Raspberry Pi's GPU system block diagram*

considered as a 16-way SIMD processor since its four actual vectors of four values are executed simultaneously under an instruction for 4 clock cycle times in which each time the results are multiplexed, giving a vector of 16 values after completing all 4 cycles. In other words, a new instruction can be issued to QPU every 4 clock cycles, hence, the operation appears to be run under a single cycle which gives a 16-value output.

A slice is a group of up to four QPUs in which they share common resources, namely Special Function Unit (SFU), Texture Memory Unit (TMU), Varying Interpolation Unit (VRI) and instruction caches. Common resources are to support QPUs based on their functions such that SFU provides special mathematical operations such as reciprocal, reciprocal square root, logarithm and exponential, TMU fetches texture information as well as general purpose data, VRI stores varying interpolations co-efficients and instruction caches, known as Uniforms cache (QUC) and Instruction cache (QIC), stores instructions from system memory.

Each QPU provides two Arithmetic Logic Units (ALUs) for add and multiply operation which are capable to work in parallel and support both integer and floating-point data.

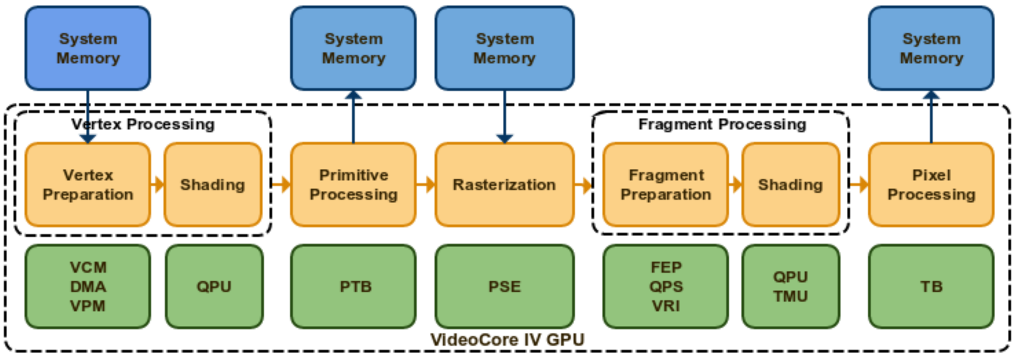


Figure 2.10 *Raspberry Pi's GPU graphics pipeline*

A graphics pipeline of VideoCore IV GPU [60], as shown in Figure 2.10, is controlled by control lists from system memory in which the sequences of primitives and system state data are fetched to Control List Executor (CLE), situated inside GPU, to control other units. Since the operations are done mostly through hardware units inside GPU with less software drivers involved, it can be counted as a hardware operation. The explanation of each operation is as follows.

- **Vertex Processing** comprises two individual operations, namely vertex preparation and shading, where vertex attributes from system memory are collected into groups at Vertex Cache Manager (VCM) using Direct Memory Access (DMA), together known as VCD, before being fetched to store at Vertex Pipeline Memory (VPM) to prepare for shading.

  Shading is controlled by QPU scheduler (QPS) which gets a vertex shade request from VCD. Vertex shading is the calculation process done within a slice by one of the QPUs on groups of vertex attributes which are arranged into a matrix of 16 columns. The calculation results are given in a form of shaded data and coordinates data before being fetched back to store at VPM.

- **Primitive Processing** is performed on Primitive Tile Binner (PTB) after VPM fetches the coordinates data of shaded vertices to the unit. PTB is responsible for building primitives from the received coordinates and finding which pixels or tiles are covered within those primitives. After that, the list containing primitives information relative to the tiles including the state change as well as the clipped primitives are transferred to system memory.

- **Rasterization** is the parameters calculation process done in Primitive Setup Engine (PSE) responsible for calculating rasterizer setup parameters and interpolation coefficients from the shaded data being fetched from VPM and the clipped primitives information sent from system memory. Most of the results are stored at Front End Pipeline (FEP) while varying interpolation coefficient is sent to store at VRI memory on the slice.

- **Fragment Processing**, similar to vertex processing, executes two individual operations which are fragment preparation and shading. Fragment preparation involves FEP sending a fragment shade request as well as a vector of initial parameters to QPS to command QPU to perform fragment shading operation. At the same time, VRI starts calculating for varying interpolation using coefficients stored in its memory.

  Fragment shading is the process to determine the characteristic of each pixel such as texture and color. The process is done in such a way that TMU passes texture information to QPU regarding a request and receive First In, First Out (FIFO) in which QPU sends a request FIFO to TMU, then TMU passes textured pixels, stored in system memory via L2 cache (L2C), to a receive FIFO which will be fetched back to process at QPU. Since shading operation is performed on both groups of vertex attributes and fragment, the operation is done in turn such that QPU performs shading on groups of vertex attributes until there are no groups available to process, it then switches to perform shading on fragments.

- **Pixel Processing** is a rendering process performed in TB in which the unit updates color, stencil and depth value to the pixel output of a shaded fragment using pixel coordinates taken from QPU internal registers. Scoreboard, situated in TB, is to ensure the correct order of pixel rendering and that the pixel can be accessed by one QPU at a time. After the rendering process is complete, the color data is written to system memory frame buffer and all data is cleared from TB in order to prepare for the next rendering.

For general purpose data operation, QPU can access data from system memory either through TMU or VPM. Although TMU is meant to process texture information, it can be used for general purpose data lookups. The data, which is stored in system memory, can be brought to TMU via L2C before being forwarded to QPU. Moreover, TMU can be configured to use up to 2 units per slice. Although Raspberry Pi's GPU is primarily for graphics purpose, general purpose data can be processed under specific command parameters in which the features of QPU, TMU and VPM are exploited. Some examples of such possibility are Fast Fourier transform (FFT) [68] and Raspberry Pi general matrix multiply (Pi-GEMM) implementations [70].

RAM offered with the SoC depends on the models. The first generation offers two types of RAM, namely 256 MB for model A and A+ and 512 MB for model B and B+. In Raspberry Pi 2 model B, the memory has been upgraded to 1 GB and the memory chip was relocated to the other side of the board instead of stacking on top of the SoC as in the previous models. RAM is shared between CPU and GPU where users can configure the amount of RAM provided to GPU in software, the rest will be given to CPU. The minimum value to be set is 16 MB and the default value is 64 MB. It is advisable to split half of the memory to GPU if the board works heavily for media tasks such as 3D rendering or if it is going to be connected with a camera module [3].

3. OPERATING SYSTEM AND SETUP INSTRUCTIONS

In this chapter, the basic OS information and setup of Raspberry Pi computer are described. The setup steps for Raspberry Pi will go through the preparation of a SD card in order to install the desired OS, the information of devices and headless connection followed by a brief information of booting process and a configuration after the board is booted. Useful information, for example, a remote desktop connection via Virtual Network Computing (VNC), is also presented.

3.1 Operating System

Raspberry Pi runs on special derivatives of Linux OS. These derivatives make possible selection of the OS best suiting specialized needs. Basic distribution of Raspberry Pi software is called New Out Of the Box Software (NOOBS) which contains six OS variants called Raspbian, Pidora, OpenELEC, RaspBMC, RISC OS and Arch Linux. User can select and quickly install any of them [15].

Raspbian is the OS which is specially and most developed for Raspberry Pi. It was developed by a group of developers who are interested in the board with the aim to use Raspberry Pi's hardware to its best advantage. The OS is based on Debian Wheezy armhf which is a hard float application binary interface (ABI) port. It uses floating point registers available in hardware for floating point parameters operation. This operation is faster than using soft float ABI, armel, where floating points parameters are operated through software. Since armhf supports ARM version 7 and higher, Raspbian packages for ARM version 6 in the first generation needs to be recompiled with the different compiler for hard float compatible. There is no issue in the second generation with ARM version 7. Raspbian contains Debian packages, additional packages built for Raspberry Pi, for example, camera module software, Scratch and Wolfram Mathematica and application examples such as GPU FFT [28][46].

Raspbian directory structure follows Filesystem Hierarchy Standard (FHS) and can be explained as in Table 3.1 [6].

Table 3.1 *Raspbian's directory structure*

Directory	**Description**
`/`	The root directory containing all directories and files of the system.
`/bin`	Contains executable files to operate the system.
`/boot`	Contains files necessary for booting process such as Linux kernel and boot loader.
`/dev`	Contains peripheral device files available to the system.
`/etc`	Contains configuration files for system as well as scripts for startup.
`/home`	The home directory for the user. By default, user's files are collected in `/home/pi` directory.
`/lib`	Contains shared libraries for the system.
`/opt`	Contains installed software packages.
`/proc`	A virtual directory containing kernel and process information.
`/root`	The home directory for the superuser.
`/sbin`	Contains executable files used in booting process or rescuing the system.
`/sys`	Contains system files.
`/tmp`	Contains temporary files. The files will be cleared after reboot.
`/usr`	Contains files that support user's installed programs.
`/var`	Contains files that change especially in size during the operation of the system.

Apart from Raspbian, the second generation of Raspberry Pi also supports developer versions of Ubuntu and Windows, namely Snappy Ubuntu Core and Window 10 IoT but many other OSs supporting ARM version 7 could be compiled for the board.

3.2 SD Card Preparation

Raspberry Pi boots from a SD card of 4 GB minimum size. The available OS images such as Raspbian and NOOBS can be downloaded from Raspberry Pi's official website [5]. Preparation of the SD card is done as follows [31]:

1. Download a desired OS image from Raspberry Pi's official website. The command `unzip imageName.zip` is used to extract the file. The *.img* file will be found after extracting *.zip* file.
2. Run command `df -h` before and after inserting the SD card into a computer or a SD card reader connected to the computer to check the name of the SD card. The name of the SD card can be shown more than one if it has more than one partition.

3. To prevent any interruptions while copying the image, unmount all the partitions of the SD card with the command `umount /dev/sdName`.

4. Run command `dd bs=4M if=/path/to/imageName.img of=/dev/sdName` to copy the image onto the SD card. Check the name of the SD card to ensure that the image is copied onto the whole SD card, not just one partition. The `bs` indicates the speed of copying. The `if` indicates the read input file and `of` indicates the written output file.

5. Remove the SD card from the computer after the operating is done. The SD card is now ready to be used on Raspberry Pi.

Additional information for running the command are shown as the list below.

- In case of using an old SD card, ensure that the card is completely formatted to FAT32 file system type. Other type of file system will not work.

- The `dd` command will not show any progress while copying the image. To see the operating progress, the command `pkill -USR1 -n -x dd` can be run in the new terminal. A prefix `sudo` is needed if a user is not `root`. The operating progress is then shown on the terminal which runs the `dd` command.

- It is advisable to have a backup image of the SD card so that when things go wrong, there is no need to start from scratch. The `dd` command can be used to backup by running `dd bs=1M if=/dev/sdName of=/path/to/backupFile.img` where the input file is the SD card name and the output file is path to a directory on the computer where the backup image file will be saved.

- A micro SD card is used instead of an SD card for model A+, B+ and Raspberry Pi 2 model B.

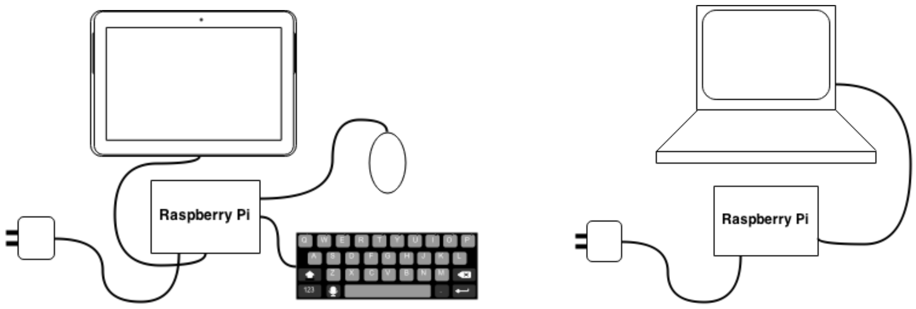
After the SD card preparation is complete, the next step is to connect Raspberry Pi either to the peripheral devices or using headless connection to be able to operate the board as needed.

3.3 Connection Methods

The connection methods for Raspberry Pi can be divided into two methods. In the first one, there are devices such as a monitor, a keyboard and a mouse which are connected through their available ports, namely HDMI, RCA connector and a USB port. The second method is called a headless connection where the connection is done by connecting Raspberry Pi to a computer via a LAN cable or a Wi-Fi dongle.

Device and headless connection

The requirements for the devices connection are a monitor, a HDMI and a USB keyboard and mouse as shown in Figure 3.1(a). After the SD card with a desired OS is inserted into Raspberry Pi's card slot and all the required devices are connected to the board, the board will be automatically booted to the desktop once powering up with a micro USB power supply. Raspberry Pi is ready to use once every required devices are connected. Then, the configuration can be done via terminal. If there are not enough USB ports available to connect to the required devices, for example, when using model A or A+, a USB hub is needed [18].



(a) Device connection (b) Headless connection

Figure 3.1 *Raspberry Pi's connection methods*

The headless connection, shown in Figure 3.1(b), is an alternative connection when there are no any other devices except Raspberry Pi, a computer and a LAN cable or a Wi-Fi dongle. The LAN cable connection is suitable for model B, B+ and Raspberry Pi 2 model B where the Ethernet port is available. The Wi-Fi dongle connection is suitable for all models. The most important part for the headless connection is to find the right Internet Protocol (IP) address to establish the connection between Raspberry Pi and the computer. The connection for either a LAN cable or a Wi-Fi dongle is done as follows [61].

1. If the connection is done with a LAN cable, plug in one end of a LAN cable to the Ethernet port of Raspberry Pi and another end to a port in a computer. If the connection is done with a Wi-Fi dongle, insert a Wi-Fi dongle to a USB port on Raspberry Pi.

2. After inserting a SD card into Raspberry Pi and powering the board with a micro USB power supply, the power LED appears in red and the other LEDs

start blinking in yellow meaning the card has been read and the connection between Raspberry Pi and a computer has already established. For model A+, B+ and Raspberry Pi 2 model B, the LEDs at Ethernet socket appear in yellow and green if a connection is done with a LAN cable.

3. To be able to control Raspberry Pi through a computer, an IP address used for connection is required. To find an IP address for a LAN cable connection, a command such as `arp-scan -l` can be run in terminal. The command will then scan the local network for an active IP address and show in the terminal.

4. For a Wi-Fi dongle connection, a network information, namely a network name and a password, is required in the script `interfaces` in `/etc/network/` directory to establish the connection. The information can be added either after booting Raspberry Pi with an Ethernet cable and edit the script in the directory or add the information directly onto a SD card from a computer before using it on the board. Both methods can be done by a command `sudo nano /path/to/etc/network/interfaces` and editing the information as in the example below:

   ```
   auto lo

   iface lo inet loopback
   iface eth0 inet dhcp

   auto wlan0
   allow-hotplug wlan0
   iface wlan0 inet dhcp
   wpa-ssid "myNetworkName"
   wpa-psk "myNetworkPassword"
   ```

   The `auto wlan0` is to automatically start the network interface at boot. A network name and password are added to `wpa-ssid` and `wpa-psk` respectively.

5. After an IP address is acquired, the Secure Shell (SSH) communication, which is enable by default in Raspberry Pi, is used to login to the board via command `ssh pi@acquiredIPAddress` in which `pi` is a default username. Raspberry Pi will then ask for a password to allow the connection. The default password is `raspberry` [35]. After that, a configuration can be performed.

The headless connection is suitable for those who wants to work with Raspberry Pi without any additional devices. It allows the remote control over the board especially if the connection is done with a Wi-Fi dongle, making it useful for portable applications.

For a headless connection, there are several useful informations such as sharing an internet connection, correcting date and time for a Wi-Fi dongle connection, setting a connection with a static IP address and using a remote desktop via VNC. The details can be found as follows.

Sharing internet connection

For Raspberry Pi to access the internet in a headless connection, the network needs to be shared from a computer. There are two cases, in the first case, a computer connects to a wireless internet and connects to Raspberry Pi with a LAN cable. The second case is the other way around, where a computer connects to a wired network and connects to the board with a Wi-Fi dongle.

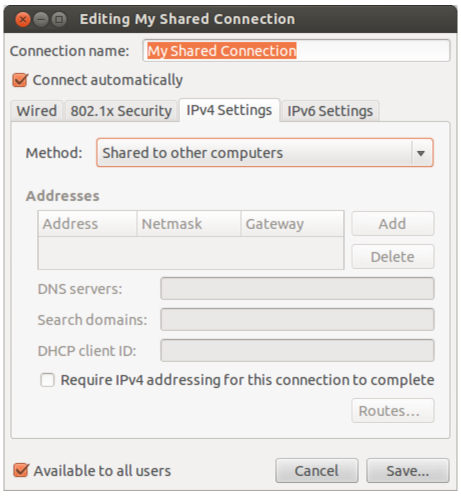

Editing My Shared Connection
Connection name: My Shared Connection
☑ Connect automatically
Wired | 802.1x Security | IPv4 Settings | IPv6 Settings
Method: Shared to other computers
Addresses
Address | Netmask | Gateway
DNS servers:
Search domains:
DHCP client ID:
☐ Require IPv4 addressing for this connection to complete
☑ Available to all users

***Figure* 3.2** *A method to share internet using Network Connections in a Linux computer*

For a LAN cable connection, sharing an internet can be done via *Network Connections* on Ubuntu OS. Under *Edit Connections*, add a new connection under a *Wired* tab. Choose *Shared to other computers* from a drop-down menu for method under *IPv4 Settings* tab as shown in Figure 3.2. The internet is then shared to the computer's Ethernet port allowing Raspbery Pi to access the internet once the LAN cable is plugged.

For a Wi-Fi dongle connection, a computer needs to be made as a hotspot, in order to share an internet, which can be done in many ways such as using a script called *ap-hotspot* [55]. After downloading and installing the script, a command `ap-hotspot start` can be run in terminal to start sharing a wireless connection allowing Raspberry Pi with a Wi-Fi dongle to connect to the network and access the internet.

Correcting date and time

The problem of incorrect date and time occurs when Raspberry Pi is connected to a computer with a Wi-Fi dongle. Raspberry Pi has no real-time clock and uses Network Time Protocol daemon (ntpd) to synchronize the system time with the time server when it can access the internet. When Raspberry Pi is off for several hours and reconnects, the slow connection of the Wi-Fi causes it fail to synchronize, therefore, the date and time shown on the desktop or terminal is counted from the previous connection. To set the correct date and time, a command `dpkg-reconfigure ntp` [41] can be run in terminal followed by `shutdown -r now` or `reboot`. The commands need to be run with `sudo` as a prefix if a user is not `root`. The date and time can be checked using command `date`.

In order to correct date and time at boot up process, a script, based on an Init Script Linux Standard Base (LSB) [43] format, needs to be created and saved in `/etc/init.d` directory with a command `sudo nano /etc/init.d/myDate` where `myDate` is the name of the created script. Required information for an Init Script LSB is add followed by `sudo dpkg-reconfigure ntp` as shown in an example below [62]:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides: myDate
# Required-Start: $all
# Required-Stop: $all
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: correct date and time for Wi-Fi
connection
### END INIT INFO

sudo dpkg-reconfigure ntp
```

To make the script executable, the commands below should be run in terminal:

```
sudo chmod 755 /etc/init.d/myDate
sudo insserv /etc/init.d/myDate
```

The first command is to make it executable while the second one is to make it run automatically at startup. It is advisable to reboot Raspberry Pi after finishing all the steps.

Setting static IP

An IP address is the most important part for a headless connection since it works as a route between a computer and Raspberry Pi. If there is no IP address or an IP address is wrong, Raspberry Pi and a computer can not connect to each other. The IP address acquisition can be divided into two types in which one is a Dynamic Host Configuration Protocol (DHCP) IP and another one is a Static IP. The DHCP IP is allocated dynamically and can be changed any time Raspberry Pi reconnects to a computer. This is not suitable for running an application that requires a stable IP address. In addition, it is not favourable to search for the right IP address every time Raspberry Pi reconnects to a computer. Therefore, a static IP address is needed.

The script `interfaces` in `/etc/network/` directory provides Raspberry Pi's network interfaces, namely `eth0` and `wlan0`, for a wired and wireless connection respectively and can be viewed with a command `cat /etc/network/interfaces` as shown below [42]:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

In the script, a wired connection interface, `eth0`, is set to use a DHCP IP address as `iface eth0 inet dhcp` by default. To set a static IP address, `dhcp` needs to be changed to `static`. The script can be edited using a command `nano /etc/network/interfaces`. A prefix `sudo` is needed if a user is not `root`. The required information, namely `address`, `netmask` and `gateway` needs to be added as shown in an example below:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 10.42.0.45
netmask 255.255.255.0
gateway 10.42.0.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

The commands `ifconfig` and `netstat -nr` can be used to get the information for `address`, `netmask` and `gateway` of the board. The same method can be applied to set a static IP address for a wireless connection interface, `wlan0`.

Another useful script for a wireless connection interface is `wpa_supplicant.conf` which is a configuration file in `/etc/wpa_supplicant` directory containing the information of the network. In the case the network information, namely `wpa-ssid` and `wpa-psk`, is not directly added to `/etc/network/interfaces` directory, it can be added using a command `nano /etc/wpa_supplicant/wpa_supplicant.conf`. A prefix `sudo` is needed if a user is not `root`. The two most important fields to be added are `ssid` and `psk` which are the name and password of the network respectively, other fields can be left out. It is possible to add more than one network to the script as shown in an example below:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="myNetworkName_1"
        psk="myNetworkPassword_1"
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        auth_alg=OPEN
}

network={
        ssid="myNetworkName_2"
        psk="myNetworkPassword_2"
        key_mgmt=WPA-PSK
}
```

Setting a static IP address for Raspberry Pi ensures that the IP address will remain the same in every connections.

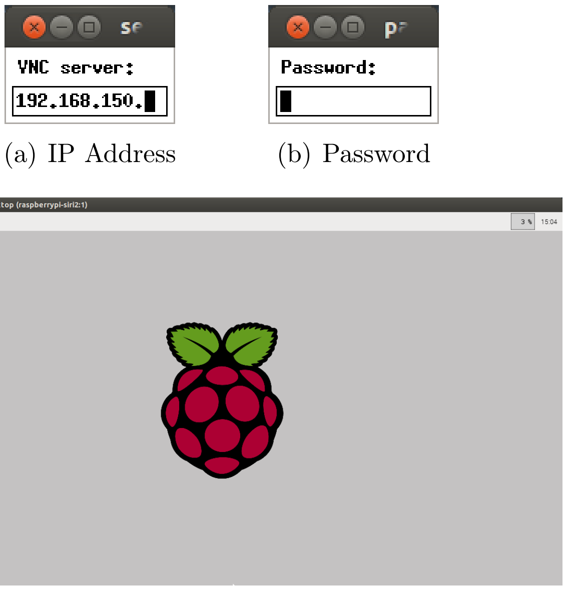
Setting VNC connection

VNC allows users to control Raspberry Pi remotely in a graphical way by showing a Raspberry Pi desktop on top of computer monitor. To be able to use VNC, the host server and client software need to be installed in Raspberry Pi and a computer respectively. There are several software packages to choose for the VNC, for example, *tightvnc*, where *tightvncserver* is chosen to be installed as a host server on Raspberry Pi and *vncviewer* as a client on a computer [75].

After login to Raspberry Pi via `ssh` and installing the software, running a command `tightvncserver` in terminal will ask for a password to be set. The password is an 8-character long and will be used to establish a connection from a client computer. The software will also tell the port for a virtual desktop as:

```
New 'X' desktop is raspberrypi-siri:1
```

In this case, the port is 1. To start running a virtual desktop, a command `vncserver :1 -geometry 1024x720 -depth 24` is run in the terminal. An option `-geometry` and `-depth` are to set the width and height and the depth color of the virtual desktop.



VNC server:
192.168.150.

Password:

(a) IP Address

(b) Password

(c) Raspberry Pi's Desktop

***Figure* 3.3** *A window of IP address, password and Raspberry Pi desktop under vncviewer*

On a computer, after a client software is installed, a command `vncviewer` is run. A window, as in Figure 3.3(a), will appear asking for an IP address and a port to establish the connection. The IP address is the address which Raspberry Pi uses to connect to the computer and the port is as specified by the server. By adding `IPAddress:Port` into the box and enter, a new window, shown in Figure 3.3(b), will appear asking for a password. If everything is correct, a new window of Raspberry Pi desktop will be shown as in Figure 3.3(c).

To start Raspberry Pi desktop automatically without having to log in via `ssh`, an additional script provided by *penguintutor* [38] needs to be created and called at start up. The script is based on an Init Script LSB format and saved in `/etc/init.d` directory. A command `nano /etc/init.d/newScriptName` is run to create a new script. After the content is added and saved to the script, the commands below should be run in terminal to make the script executable.

```
sudo chown root:root /etc/init.d/newScriptName
sudo chmod 755 /etc/init.d/newScriptName
sudo chmod +x /etc/init.d/newScriptName
sudo update-rc.d newScriptName defaults
```

The first command is to change the owner of the script to `root`, the second command is to make it executable and the last command is to make it run automatically at startup. It is advisable to reboot the board after finishing all the steps.

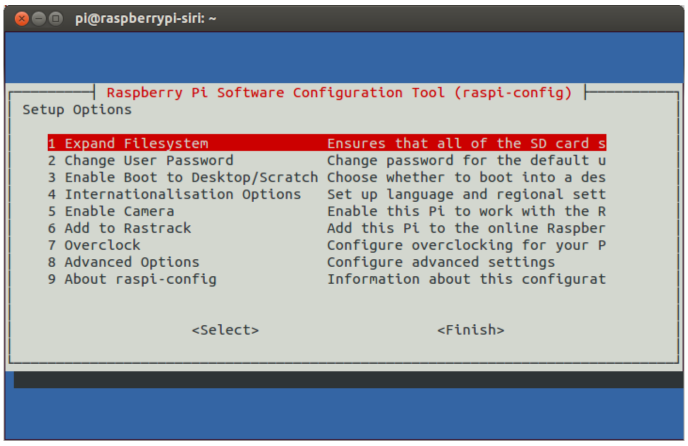
3.4 Configuration Instructions

Raspberry Pi's boot up process involves its SoC and a software on a SD card. It boots from GPU. The process can be explained such that a FAT32 boot partition on a SD card is mounted to the processor through a code programmed onto a SoC which then calls *bootcode.bin* to start GPU. Once the GPU starts, RAM is enabled as well as GPU firmware, *start.elf*, is loaded to start the settings for CPU such as setting a shared memory between GPU and CPU and loading a configuration file, *config.txt*, as well as a kernel image, *kernel.img*. After the kernel image is loaded, an OS starts and the execution is transferred from GPU to CPU [44].

In the case that Raspberry Pi does not boot, it is most likely that *start.elf* is not found. The yellow LED will blink for awhile, then stop and remain lid stating that the SD card is corrupted. Performing a fresh installation of the SD card can solve this problem. In the normal case where the board boots properly, the yellow LED will blink in an irregular pattern for about 20 seconds [40].

Once Raspberry Pi boots under a Raspbian OS, a Raspi-config screen, as shown in Figure 3.4, will automatically appear on the monitor if the connection is done via a devices connection. For a headless connection, after logging in with a username and a password through `ssh`, a command `raspi-config` can be run in terminal to perform the first configuration. The same Raspi-config screen will appear.

The raspi-config tool has several options. To select the options, the up and down arrow keys are used to move the highlight for an option up or down. The right arrow key is used to choose between `Select` or `Finish` option and the left arrow key is used to go back to the option lists. The enter keys is used for selecting. The options in Raspi-Config are explained as below [30]:



```
pi@raspberrypi-siri: ~

        Raspberry Pi Software Configuration Tool (raspi-config)
 Setup Options

    1 Expand Filesystem              Ensures that all of the SD card s
    2 Change User Password           Change password for the default u
    3 Enable Boot to Desktop/Scratch Choose whether to boot into a des
    4 Internationalisation Options   Set up language and regional sett
    5 Enable Camera                  Enable this Pi to work with the R
    6 Add to Rastrack                Add this Pi to the online Raspber
    7 Overclock                      Configure overclocking for your P
    8 Advanced Options               Configure advanced settings
    9 About raspi-config             Information about this configurat


                 <Select>                         <Finish>
```

***Figure* 3.4** *Raspi-config screen for Raspberry Pi configuration*

- **Expand Filesystem** is to ensure that the OS installation takes place for the whole of an SD card, not only at some partition, making more space available for any programs or files which will be installed afterwards. When the option is selected, the system will be expanded and the change will take place after reboot.

- **Change User Password** is to change the password of Raspberry Pi. The default password is `raspberry`. In terminal, the option will ask to enter a new password and a confirmation. If they do not match, the password will not be changed.

- **Enable Boot to Desktop/Scratch** is to select the boot options for Raspberry Pi. There are three options available which are boot to terminal where a password is needed for logging in, boot to a Raspberry Pi's Desktop and boot to Scratch programming tool which is a tool designed to help young people to learn programming without having to write programs. A default option is boot to terminal.

- **Internationalization Options** has three options available which are change locale, change timezone and change keyboard layout. Change locale is to set the region and language Raspberry Pi will be used. The UTF-8 is chosen by default. Change timezone is to set the time zone used on the board by selecting the region and city. Change keyboard layout is to match the keyboard layout with the user's keyboard layout.

- **Enable Camera** is to allow Raspberry Pi to work with the camera module by providing at least 128 MB of RAM for the GPU usage.

- **Add to Rastrack** is an option to add Raspberry Pi to a Google map based website named *Rastrack* which shows the location of Raspberry Pi's users around the world. Users can add Raspberry Pi directly to the map through the website if prefer [74].

- **Overclock** is to choose the operating speed of the CPU. There are five options available for Raspberry Pi 2 model B which are 700 MHz, 800 MHz, 900 MHz, 950 MHz and 1000 MHz. The default setting is 700 MHz.

- **Advanced Options** has ten options available as explained in the list below:

    1. **Overscan** is an option to adjust the displaying area in a monitor to either show a full screen image or with a border in case a monitor connected to Raspberry Pi is an old cathode ray tube (CRT) screen. The CRT screen has a problem in showing images whose edges will be cropped to avoid displaying any visible edge problem after image processing prior displaying on the screen. New monitors do not have this problem. Disable overscan allows showing a full screen in the monitor.
    2. **Hostname** is to set a name for Raspberry Pi to be visible in the network. It allows only ASCII letters from a to z with no case sensitive, a number from 0 to 9 and a hyphen to form a name.
    3. **Memory Split** is to adjust the memory used between CPU and GPU depending on user's preference. If Raspberry Pi will be used mostly for media related work, it is advisable to split half of the memory for GPU usage, for example, GPU can get at most 500 MB of RAM on Raspberry Pi 2 model B.
    4. **SSH** is an option to enable or disable a remote access to Raspberry Pi from any computers in the same network. The default setting is enable.
    5. **Device Tree**, associated with HAT, is an option to enable or disable a kernel for device tree which is a data structure describing the hardware such as names and properties. The hardware information is passed to Raspberry Pi OS during boot allowing the automatically configuration over the devices attached to Raspberry Pi GPIO pins. Device tree is OS-neutral meaning it is compatible with any OSs. The option is enable by default [4].
    6. **SPI** allows Raspberry Pi to communicate in a full-duplex mode with any peripheral devices, such as a PiFace, over a short distance. This option is to enable or disable an automatic loading of its kernel module. It is disable by default.

7. **I2C** is an option to enable or disable an automatic loading of I2C kernel module which is a module that allows Raspberry Pi to connect to any peripheral devices, such as sensors, through GPIO pins. The option is disable by default.
8. **Serial** is an option to enable or disable a log in through a serial connection between Raspberry Pi and a computer it connects to. The option is enable by default. However, if Raspberry Pi needs to connect to any microcontrollers such as Arduino, the serial connection needs to be disable.
9. **Audio** has three options available which are auto, force 3.5 mm headphone jack and force HDMI. It is an option to set an audio output port to either a 3.5 mm headphone jack or a HDMI port which are two available audio output ports for Raspberry Pi. If a user has no preferences in choosing any options, Raspberry Pi will automatically select an output port by itself.
10. **Update** is to update Raspi-config tool to the latest version.

The advanced option is mostly to enable and disable kernels or to adjust the user's preferred options for Raspberry Pi's features.

- **About raspi-config** shows a short description of `raspi-config`.

The raspi-config tool can be run any time with the command `raspi-config` in terminal if there is need of any reconfiguration. Command `sudo` is needed if a user is not `root`. In addition, the commands that should be run during the first boot are `apt-get update` followed by `apt-get upgrade` to ensure that Raspberry Pi has the latest version of software [39]. It is advisable to run the commands once in a while to get the latest updates or run before installing any new packages or scripts.

Raspberry Pi can be properly turned off by running a command `shutdown -h now` where `-h` is an option for halt and `now` tells the system to turn off immediately.

In summary, setting up Raspberry pi requires a Raspberry Pi board, a working SD card and a micro USB power supply that provides enough power. Other peripheral devices needed depend on a connection type such that a devices connection requires a monitor, a keyboard and a mouse while a headless connection requires a computer and a LAN cable or a Wi-Fi dongle. Setting an IP address can be useful not only for a headless connection but also for the board to access the internet. Raspberry Pi under Raspbian OS has a default username and password as `pi` and `raspberry`, respectively, which can be used for a remote login through ssh, however, if a graphical login is more preferable, VNC can be an option.

Raspberry Pi offers a configuration tool under a command `raspi-config`, which can be run any time a configuration is needed. To get the latest version of the software, two commands `apt-get update` followed by `apt-get upgrade` can be run to update and upgrade the board. Raspberry Pi has no switch to turn off, thus, a command `shutdown -h now` should be run to properly shutdown the board in order to prevent a corruption of an SD card. A command `reboot`, as well as `shutdown -r now` where `-r` is an option for reboot is for rebooting the board after serious changes, such as expanding file system, have been made. All commands need a root privilege, otherwise a prefix `sudo` is needed.

4. MATLAB AND SIMULINK SUPPORT PACKAGE FOR RASPBERRY PI

Matlab, integrated with Simulink, is a programming language software developed by MathWorks. It provides numerical computation tools and models used by academics and engineers for various systems design and development [48]. It is also widely used in university education.

Matlab and Simulink support package for Raspberry Pi is composed of two parts of which one is for Matlab and another one is for Simulink. Matlab support enables development software for algorithms which can run in Raspberry Pi. It also allows to control peripheral devices connected to the board through its GPIO interfaces, namely serial, I2C and SPI as well as a camera module via command functions in Matlab command window [22]. Simulink support package allows users to create Simulink models in various fields such as audio, image or embedded system using general and support package toolboxes. Additionally, users can create standalone applications by deploying Simulink model onto Raspberry Pi [23].

This chapter explains the installation and connection instructions for Matlab and Simulink support package on Raspberry Pi 2 model B followed by the list and description of command functions and toolboxes.

4.1 Installation Instructions

Setting Raspberry Pi to work with Matlab and Simulink software requires a support package to be installed onto both a computer and the board. Command functions and special toolboxes are installed in the connected computer and a modified Raspbian OS with Matlab and Simulink compatibility is installed in Raspberry Pi.

To enable Matlab and Simulink support package for Raspberry Pi, Matlab has to be started in the administrator mode to allow complete control of all operations. The installation instruction is explained as the steps below [9].

1. Download Matlab and Simulink support package either from Matlab's official

website or from *Get Hardware Support Packages* in Add-Ons tab of Matlab software, as shown in Figure 4.1. This can also be used to check for a new version and to install an update of the support package.

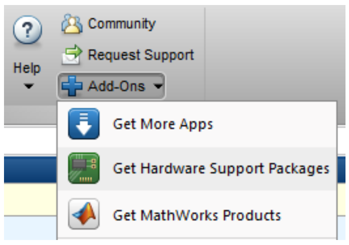
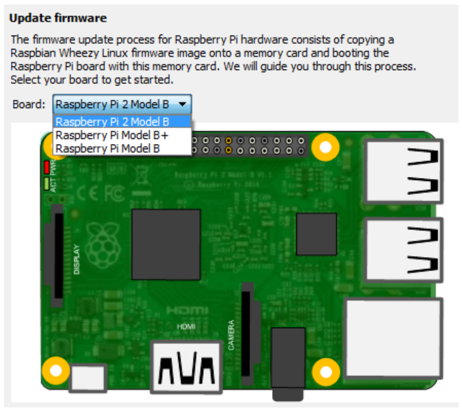


Figure 4.1 *Download support packages in Matlab*

If the support package is downloaded from the website, an installation window starts right away after clicking the support package installation software. In *Get Hardware Support Packages* option, select Raspberry Pi to install the support package.

2. A MathWorks account is needed to begin the installation process. After log in, read and agree to the license agreement, the installation process starts with downloading the relevant software including Matlab and Simulink support packages as well as Raspbian OS, then, installing the packages onto the selected folder.



Update firmware

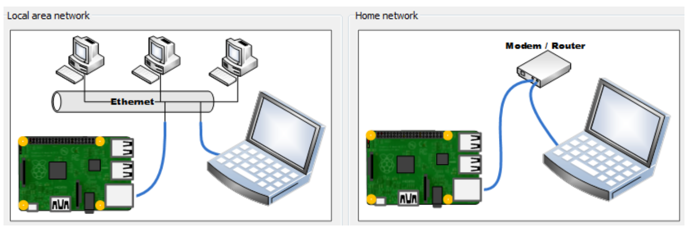
The firmware update process for Raspberry Pi hardware consists of copying a Raspbian Wheezy Linux firmware image onto a memory card and booting the Raspberry Pi board with this memory card. We will guide you through this process. Select your board to get started.

Board: Raspberry Pi 2 Model B

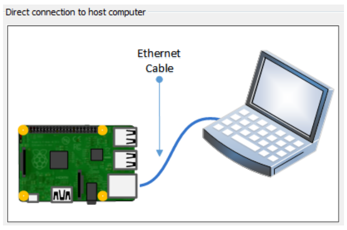
Figure 4.2 *Select Raspberry Pi board*

3. After the installation process is complete, the set up process for Raspberry Pi starts. A desired Raspberry Pi board either Raspberry Pi model B, B+ or Raspberry Pi 2 model B can be selected from a drop-down menu as shown in Figure 4.2. The process will prepare a suitable support package image for the board to be installed onto a memory card.

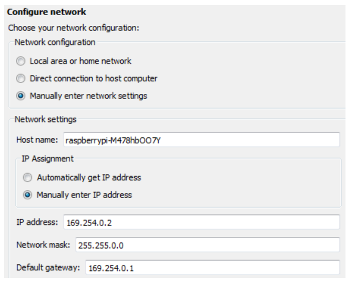
4. To use Matlab and Simulink support package with Raspberry Pi, the board needs to be connected to a host computer with a correct IP address which can be set via a network configuration. There are three options available, namely local area or home network, direct connection to host computer and manually enter network settings as shown in Figure 4.3. The first two options will automatically set a suitable IP address for Raspberry Pi while in the last option, users can select to obtain an IP address automatically or manually set it.



(a) Local area network and home network



(b) Direct connection



Configure network

Choose your network configuration:

Network configuration

- Local area or home network
- Direct connection to host computer
- [✓ Manually enter network settings]

Network settings

Host name: raspberrypi-M478hbOO7Y

IP Assignment

- Automatically get IP address
- [✓ Manually enter IP address]

IP address: 169.254.0.2

Network mask: 255.255.0.0

Default gateway: 169.254.0.1

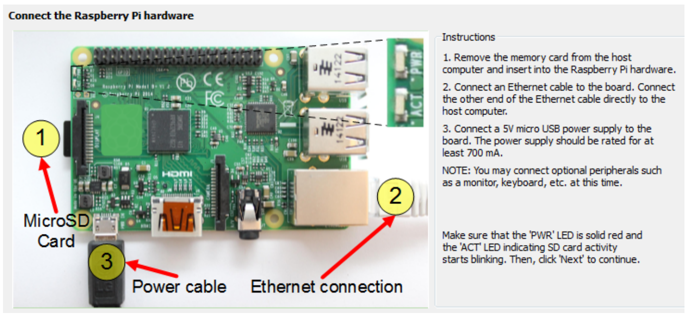
(c) Network settings

***Figure** 4.3 Network configuration for Matlab and Simulink support package*

For manually setting an IP address, a host name is a specific name for Raspberry Pi. Users can decide to keep a program generated name or change it if they prefer. IP address, network mask and default gateway can be set as Raspberry Pi's static IP address. The IP settings can be changed later after Raspberry Pi boots in `/etc/network/interfaces`.

5. After setting a correct IP address for Raspberry Pi, a support package image is ready to be written onto a memory card. Insert an SD card or a micro SD card with an adaptor into an SD card slot on a computer then start writing an image.

6. After completing the image writing process, a screen as shown in Figure 4.4 appears. Follow an instruction to confirm the connection between Raspberry Pi and a computer.



Instructions

1. Remove the memory card from the host computer and insert into the Raspberry Pi hardware.
2. Connect an Ethernet cable to the board. Connect the other end of the Ethernet cable directly to the host computer.
3. Connect a 5V micro USB power supply to the board. The power supply should be rated for at least 700 mA.

NOTE: You may connect optional peripherals such as a monitor, keyboard, etc. at this time.

Make sure that the 'PWR' LED is solid red and the 'ACT' LED indicating SD card activity starts blinking. Then, click 'Next' to continue.

Figure 4.4 *A connection instruction for Matlab and Simulink support package*

7. An installation process is complete with a confirm board configuration screen informing a host name, a username and a password where a host name is set during connection configuration step and a username and password is `pi` and `raspberry` respectively. Raspberry Pi with Matlab and Simulink support package is ready to start.

Additional information for Matlab and Simulink support package installation on Raspberry Pi can be found below.

- It is important that Matlab is started in the administrator mode, otherwise, the installation program will not be able to find a memory card drive. To start Matlab in the administrator mode, right click on Matlab icon on Windows desktop and select run as administrator.
- An SD or a micro SD card used in an installation should be a newly formatted FAT32 file system type SD card. The installation will also install Raspbian OS onto the card.
- For the first time installation, both Matlab and Simulink support packages are available to be selected for installation at the same time. However, for an update of a new version, only one of the support packages can be selected to be installed. Users can also choose a preferred folder to install the package.
- In case the support packages are already downloaded and saved into a folder in a computer, a command `targetupdater` can be run in Matlab command window to start installation process.
- If choosing to obtain an IP address automatically, the IP address can be edited later once Raspberry Pi boots at `/etc/network/interfaces`.

The installation steps are based on Matlab and Simulink release 2015a version in which the support package are available for a 32 and 64 bit Windows version and a 64 bit MAC OS X.

4.2 Connection Instructions

The connection and configuration for Raspberry Pi with a modified Raspbian OS to support Matlab and Simulink can be done in the same way as a normal setup step such that the board is connected to a computer via either a LAN cable or a Wi-Fi dongle through an assigned IP address. The configuration, for example, expanding the file system through an option in `raspi-config` or setting a static IP address in `/etc/network/interfaces` can be done by logging in to the board through `ssh` in terminal prior the connection with Matlab and Simulink software. Upon completion of the configuration, the connection between Raspberry Pi and Matlab software can be explained in two ways in which the first one is for Matlab and the second one is for Simulink. Both methods require a correct IP address to establish the connection, however, the connection manners are different. To connect Raspberry Pi with Matlab, the connection is established with a command function `raspi` provided by the support package. As for Simulink, the connection is established through *Run on Target Hardware* option in a Simulink model. Once the connection is established, Raspberry Pi can be operated through command functions and toolboxes. Matlab and Simulink connection instructions can be explained as follows.

Raspberry Pi and Matlab software is connected via a command function `raspi` provided by a support package [29]. The connection is done in such a way that the command function creates Raspberry Pi object which will be used as an input parameter for other command functions to access and control the board as well as its connected devices. The usage examples are shown below.

```
mypi = raspi
mypi = raspi('ipAddress','userName','password')
mypi = raspi('hostname')
```

Parameters such as an IP address, a user name and password or a host name should be provided to the function if there are any changes in connection, for example, a changed password or a connection to a different board. In addition, Raspberry Pi object holds the board's properties which can be used as a reference parameter for the configuration. An example of the properties are shown below.

```
mypi =
raspi with properties:
DeviceAddress: '192.168.150.5'
Port: 18726
BoardName: 'Raspberry Pi 2 Model B'
AvailableLEDs: {'led0'}
AvailableDigitalPins: [4 5 6 12 13 14 15 16 17 18 19 20
                        21 22 23 24 25 26 27]
AvailableSPIChannels: {'CE0'  'CE1'}
AvailableI2CBuses: {'i2c-1'}
I2CBusSpeed: 100000
```

Apart from the information of an IP address, Transmission Control Protocol (TCP) port and model version, the properties display the information of a controllable LED and GPIO pins as well as additional information for SPI and I2C interfaces. A command function `clear` is used to disconnect Raspberry Pi from the software.

The connection between Raspberry Pi and Simulink is established in two ways. The first is for running a model in external mode on the board for a simulation purpose [25] and the latter one is for deploying the model onto the board for a standalone application [32]. The connection steps can be explained as follows:

1. After Raspberry Pi is powered and connects to the same network as a computer, the command function `raspberrypi` can be used to check the status of the board. The command will display a host name, a user name, password and the board's default folder on Matlab command window. In addition, users can run a command as shown in an example below in Matlab command window to ensure the connection:

   ```
   >> h = raspberrypi
   h =
   LinuxServices with properties:
   HostName: '192.168.150.5'
   UserName: 'pi'
   Password: 'raspberry'
   BuildDir: '/home/pi'

   >> h.connect
   ans =
   Connection successful
   ```

   The command `raspberrypi` creates Raspberry Pi object, `h`, with properties in which its status can be checked using a parameter `connect`.

2. Upon completion of a Simulink model, users can make an interaction between

Raspberry Pi and the model through an option by selecting *Tools* followed by *Run on Target Hardware* and *options* as shown in Figure 4.5.

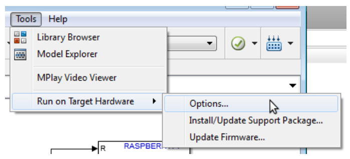
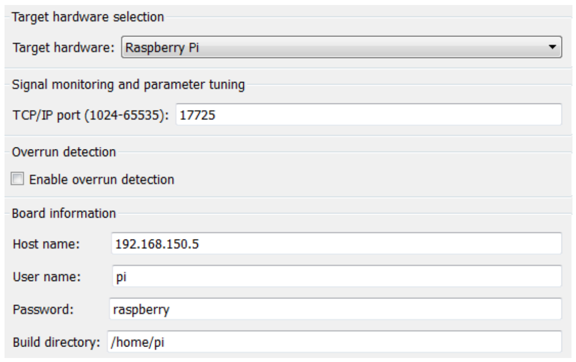


Figure 4.5 *Prepare a connection between Raspberry Pi and Simulink model*

3. Confirm the information of a target hardware, in this case, Raspberry Pi, in a configuration window shown in Figure 4.6. Ensure that an IP address, a user name and password are correct since these parameters are used to establish the connection.



Target hardware selection

Target hardware: Raspberry Pi

Signal monitoring and parameter tuning

TCP/IP port (1024-65535): 17725

Overrun detection

☐ Enable overrun detection

Board information

Host name: 192.168.150.5

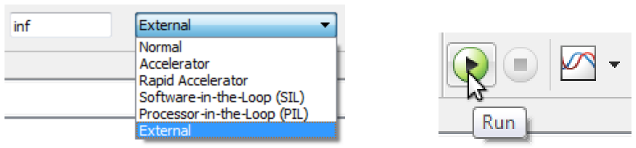
User name: pi

Password: raspberry

Build directory: /home/pi

Figure 4.6 *Run on target hardware configuration window*

4. For running a Simulink model in external mode, *External* mode is selected from a drop-down menu as shown in Figure 4.7(a). Input a desired running time onto a space next to the mode option. The default is 10. In this case, it is set as *inf* to indicate the infinity running time. The simulation process starts by clicking a run button shown in Figure 4.7(b). The software will prepare a model to run and display a result on a computer's screen if it is connected.



(a) External mode option (b) Run button

Figure 4.7 *Run a Simulink model in external mode*

For deploying a Simulink model onto Raspberry Pi, after having confirm the information on a configuration window, a model can be deployed onto the board by clicking a deploy to hardware button shown in Figure 4.8. The Simulink model will be uploaded onto the default folder of the board.

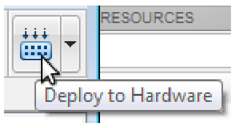


Figure 4.8 *Deploy a Simulink model onto Raspberry Pi*

To disconnect Raspberry Pi with a Simulink model for an external mode, click a stop button next to a run button on a model window or apply a parameter `stop` with Raspberry Pi object, `h`, along with the name of the model as `h.stop('modelName')`. While the first option is only for an external mode, the second option can be used to stop a standalone application running on Raspberry Pi.

4.3 Command Functions

Matlab support package allows users to operate Raspberry Pi and its attached devices through command functions [12]. The board's interfaces such as serial, I2C and SPI as well as LED and a camera module can be controlled by command functions through Matlab command window. This section provides the list of command functions and their descriptions separated by their categories in the following tables. Table 4.1 provides information including existing examples for Matlab and Simulink support package and connection command functions for Raspberry Pi.

Table 4.1 *Information and connection command function*

Command Function	**Description**
Information	
`rasberrypi`	Display host name, user name, password and build directory of the successfully connected board.
`raspi_examples`	Display Matlab and Simulink support package examples for Raspberry Pi in a pop-up window.
Connection	
`raspi`	Create a connection between Raspberry Pi and Matlab software with or without specifying an IP address, user name and password or a host name. Return board's properties (address, port, model and interfaces information) in Matlab command window.

The command function `raspi` creates Raspberry Pi object which allows Matlab software to control an on-board or attached device such as LEDs or a camera module with command functions shown in Table 4.2. Raspberry Pi allows only one controllable LED in which users can turn it on or off to indicate the operation's condition as needed. LED is back to its original state after restarting the board. For a camera operation, a camera module is connected to the board through the CSI connector via its ribbon cable in which the connection is established with the command `cameraboard`. The camera's properties can be shown in Matlab command window after the connection is complete. Additionally, users can configure the properties by providing *Name-Value Pair Arguments* [14] parameters to the `cameraboard` command. The new setting will take effect after 5 frames. The camera command functions allows users to capture a still image as well as to record and stop recording a video. An image output is directly saved onto a Matlab current folder but a video output is saved into Raspberry Pi default folder, namely `/home/pi` which can be accessed using a Linux command function.

Table 4.2 *LEDs and camera operation command function*

Command Function	**Description**
LEDs	
`writeLED`	Turn an available LED on or off by setting a value `1` or `true` for on and `0` or `false` for off.
`showLEDs`	Display an information (location, name and color) of available LEDs through Matlab figure.
Camera Board	
`cameraboard`	Create a connection between a camera module and Raspberry Pi. Arguments such as name and value are optionally added.
`snapshot`	Return an RGB format still image.
`record`	Return a video file with user's specified name and record duration. The video file is saved onto Raspberry Pi.
`stop`	Stop recording a video.

Apart from LED and a camera module, Raspberry Pi GPIO interfaces are also supported by the support package's command functions listed in Table 4.3.

Raspberry Pi GPIO pins can be used to operate serial, I2C and SPI interfaces. The list of available pins can be found in the Raspberry Pi object properties while the information and location of the pins are shown in a Matlab figure using the command `showPins`. The pins are allowed to be configured as an input or an output by the user. To change the state of the configured pin, the command `configureDigitalPin` is required to prevent unintentional damage to the board, however, for an unconfigured pin, reading and writing commands configure it to be an input and an output respectively. For serial, I2C and SPI interface, the command function to create the interface object is needed to established the connection in order to operate the peripheral devices. The command functions, `read` and `write`, are available for serial

Table 4.3 *Interfaces operation command function*

Command Function	**Description**
GPIO Pins	
`configureDigitalPin`	Configure a GPIO pin to act as an input or output using a pin number provided by the board's properties.
`readDigitalPin`	Return a logical value (`0` or `1`) from an input GPIO pin using a pin number. If a pin is an output or is in used by another interfaces (serial, I2C or SPI), the function returns an error message.
`writeDigitalPin`	Set a logical value (`0` or `1`) to an output GPIO pin using a pin number. If a pin is an input or is in used by another interfaces (serial, I2C or SPI), the function returns an error message.
`showPins`	Display a diagram of GPIO pins through Matlab figure.
Serial Port	
`serialdev`	Create a serial device object through GPIO pins available for serial connection using parameters (port, baud rate, data bits, parity bit and stop bits).
I2C Interface	
`scanI2CBus`	Return addresses of I2C available bus.
`i2cdev`	Create an I2C device object through GPIO pins available for I2C connection using available bus and address.
`readRegister`	Return a scalar value of I2C device's register number. Data precision is optionally added.
`writeRegister`	Write a hexadecimal value to I2C device's register number. Data precision is optionally added.
`enableI2C`	Enable GPIO pins for I2C connection with or without specifying bus speed. The default bus speed is `100000`.
`disableI2C`	Disable GPIO pins for I2C connection.
SPI Interface	
`spidev`	Create an SPI device through GPIO pins available for SPI connection using its channel. Mode and speed are optionally added. The default mode is `0`.
`writeRead`	Return a written data of an SPI device as a row vector. The written data is specified as a vector of hexadecimal values. Data precision is optionally added.
`enableSPI`	Enable GPIO pins for SPI connection.
`disableSPI`	Disable GPIO pins for SPI connection.
Common Function	
`read`	Return data from a serial device or an I2C device.
`write`	Write data to a serial devcie or an I2C device. The written data is specified as a vector or a vector of hexadecimal values for serial and I2C respectively.

and I2C interfaces to read and write data while in SPI, due to its full duplex communication mode, the command `writeRead` provides the same result. Since GPIO pins are shared among three other interfaces, namely serial, I2C and SPI, the command function will provide an error message if the pins are already occupied. To free the pins for usage, enable and disable command functions concerning the specific interfaces are used. By default, I2C is enabled while serial and SPI are disabled.

Table 4.4 *Linux command function*

Command Function	Description
Linux	
`system`	Allow to run Linux command. Return the result in Matlab command window. A prefix `sudo` is needed for a superuser privilege command.
`openShell`	Open an SSH terminal. The default user name and password for login are `pi` and `raspberry` respectively.
`getFile`	Download a file from Raspberry Pi to a computer with or without specifying a destination folder. The default destination folder is a Matlab current folder.
`putFile`	Upload a file from a computer to Raspberry Pi with or without specifying a destination folder. The default destination folder is `/home/pi` folder.
`deleteFile`	Delete a specified file from Raspberry Pi.

Regarding the access to Raspberry Pi, the support package provides command functions to manage files saved in Raspberry Pi's folders by specifying the name and folder's path of the file as parameters of the command functions. The available operations, based on Secure copy (SCP) and SSH protocols, are `getFile`, `putFile` and `deleteFile` for download, upload and delete the file respectively. The default folder in a computer is a Matlab current folder while in Raspberry Pi, the default folder is specified as `/home/pi`. The command operations based on a Linux command line manner can also be done in the Matlab command window and through an SSH terminal window with a command function `system` and `openShell` respectively. The available command functions for Linux are as shown in Table 4.4. The command functions provided by Matlab and Simulink support package allows a convenient way to operate Raspberry Pi interfaces and peripheral devices through Matlab command window as well as access the board using Linux command line making it easier for users to create any desired projects on the board.

4.4 Block Library

Simulink support package for Raspberry Pi offers a block library to operate the board in various fields such as audio, image and network as well as blocks to control

LED and GPIO input and output port [13]. The block library can be shown by running a command function `raspberrypilib` or `simulink` then search for *Simulink Support Package for Raspberry Pi*. Window containing Raspberry Pi block library will be shown as in Figure 4.9. The detail of each block separated by its category is explained below.

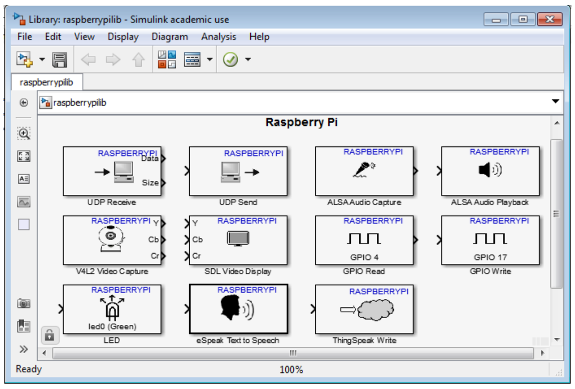


***Figure 4.9** Simulink block library for Raspberry Pi*

Audio blocks are supported by Advanced Linux Sound Architecture (ALSA) driver framework which provides a kernel driver to operate sound under Linux OS. Raspberry Pi audio input can be accessed through GPIO Integrated Interchip Sound (I2S) pins [59] or a USB sound card attached to one of the available USB ports while audio output are provided through a HDMI port and a 3.5 mm headphone jack.

- **ALSA Audio Capture** captures a 2-channel audio input given an output as an N-by-2 frame of int16 values where N is a frame size and 2 determines left and right channels. The device name, sampling frequency and frame size can be configured in a block parameter window. The default device name is set to `'hw:1,0'` in which `1` and `0` are card and device number of the hardware used for capturing audio input respectively. The sample rate can be calculated by dividing frame size with a sampling frequency.

- **ALSA Audio Playback** plays a 2-channel audio output in which an input to the block is an N-by-2 frame of int16 values corresponding to the output from *ALSA Audio Capture.* The device name and sampling frequency can be configured in a block parameter window. The default device name is set to `'hw:0,0'`.

- **eSpeak Text to Speech** converts a uint8 array containing ASCII text to speech then plays an output speech through Raspberry Pi's audio output port.

Raspberry Pi provides an image and video input through a CSI connector for a camera module but a compatible USB camera [53] can also be connected. Image and video blocks are supported by Video for Linux Two (V4L2) driver framework and Simple DirectMedia Layer (SDL) library for capturing and displaying video respectively.

- **V4L2 Video Capture** captures a video input from a camera module or a compatible USB camera. A block parameter window allows a device name, image size, pixel format and sample time to be configured. The default device name is set to `'/dev/video0'` in which `video0` is automatically created by a Linux kernel driver once a compatible USB camera is connected to the board, however, the driver needs to be installed if using a camera module.
- **SDL Video Display** displays a video output through SDL video display in which a display window will be shown on a computer screen for an external mode and on a monitor for a standalone mode. Display window is not shown if running a standalone mode through VNC connection. Block parameter allows only one parameter to be configured, namely a pixel format which can be configured between YCbCr 4:2:2 and RGB. The default format is YCbCr 4:2:2. Only one SDL Video Display block can be added to a Simulink model.

For a camera module to work with Simulink image and video blocks, a kernel driver *bcm2835-v4l2* [27] needs to be installed onto Raspberry Pi. The installation steps are as follows [45].

1. Log in to Raspberry Pi via `ssh` and prepare the board to its latest version then reboot the board by running the commands below.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo rpi-update
sudo shutdown -r now
```

2. After reboot, load a kernel driver *bcm2835-v4l2* onto Raspberry Pi with the command.

```
sudo modprobe bcm2835-v4l2
```

   The driver will add `video0` onto `/dev` directory of Raspberry Pi, however, it will be deleted every time the board reboots. To automatically load the driver when the board starts, the command can be added to the script `/etc/rc.local` by running `sudo nano /etc/rc.local` and add the command before `exit 0` line.

Network blocks for Raspberry Pi supports sending and receiving data using User Datagram Protocol (UDP) as well as publishing data from the board to a web service. UDP allows a connectionless communication between a transmitter and a receiver where a guarantee of the transmitted data is unnecessary and fast communication is preferable.

- **UDP Send** gets an input data from source and sends the data using UDP packets to a remote device by specifying its IP address and port in a block parameter. The default values are `'255.255.255.255'` and `25000` for an IP address and port respectively.
- **UDP Receive** receives UDP packets from a network and produces two outputs, namely data in a one-dimensional vector format and its size. A block parameter allows IP port, data type and its size as well as sample time to be configured in which data type and size need to match those of the input data.
- **ThingSpeak Write** allows to publish data to *ThingSpeak* web service for IoT applications. The block establishes a communication between Raspberry Pi and the web service though Uniform Resource Locator (URL) and Application Programming Interface (API) key which can be obtained from ThingSpeak website [37].

Raspberry Pi's LED and GPIO pins can be controlled via a Simulink block in which the block allows to turn a user-controllable LED on or off as well as read and write a logical value of GPIO pin. GPIO pins need to be operated with care when connecting to a voltage source since an exceeding voltage can cause a damage to the board. The maximum input voltage for a pin is 1.8 V.

- **LED** receives a boolean value as an input to turn a user-controllable LED on or off where `1` is on and `0` is off.
- **GPIO Read** reads a logical value of an input GPIO pin and produces the value as an output. A pin number to be read and sample time can be configured in a block parameter.
- **GPIO Write** receives a boolean value as an input to write to a GPIO pin configured as an output by specifying a pin number to be written in a block parameter.

Apart from a block library for Raspberry Pi, a command function `raspberrypi` along with other command functions listed in Table 4.5 can be run on Matlab command window to control a Simulink model for an external and a standalone mode.

Table 4.5 *Simulink command functions for Raspberry Pi*

Command Function	**Description**
Simulink	
`raspberrypi`	Display host name, user name, password and build directory of the successfully connected board.
`connect`	Display a message on Matlab command window to ensure the connection between Raspberry Pi and Simulink is established.
`run`	Run a Simulink model by specifying a model name.
`stop`	Stop a running Simulink model by specifying a model name.
`execute`	Allows executing a Linux command line on Matlab command window.

The command function `raspberrypi` for Simulink model operates in the same way as Matlab command function `raspi` by creating Raspberry Pi object to be used as an input for other command functions. The difference between the two command functions is that `raspberrypi` does not allows input parameters such as an IP address or a host name to establish the connection between the board and the software. Instead, it passes the connection information, namely an IP address or a host name, a user name, password and a default folder, of a successfully connected board to Raspberry Pi object which can be used as an input parameter for other command functions or link with other command functions to perform an action. Usage examples are shown in Table 4.6.

Table 4.6 *Usage Examples of Simulink command functions for Raspberry Pi*

Use as Input Parameter	**Link with Command Function**
`h = raspberrypi`	`h = raspberrypi`
`connect(h)`	`h.connect`
`run(h,'modelName')`	`h.run('modelName')`
`stop(h,'modelName')`	`h.stop('modelName')`
`execute(h,'linuxCommand')`	`h.execute('linuxCommand')`

The left side of the table shows Raspberry Pi object as an input parameter for other command functions while the right side shows the object linking with other command functions. Despite the different manner in issuing the command functions, they operate in the same way and accept the same input parameter such as Simulink model name or Linux command.

Simulink block library for Raspberry Pi consists of eleven blocks which can be categorized into four groups, namely audio, image and video, network and on-board device and interface. The blocks allow an operation over Raspberry Pi's input and output connectors such as a USB port or a 3.5 mm headphone jack to create Simulink models for simulation as well as building standalone applications. Matlab command functions allow an operation over GPIO pins. The combination of block library and command functions allows total control of Raspberry Pi making Matlab and Simulink support package a convenient tool to create various projects for the board.

5. SIGNAL AND IMAGE PROCESSING EXAMPLES

This chapter shows some examples from signal and image processing running on Raspberry Pi. The examples are taken from exercise classes held in Tampere University of Technology and the available toolbox examples from Matlab. The Matlab scripts are implemented in Simulink model using appropriate toolboxes then run on Raspberry Pi in external mode. The parameters which are collected are code generation time and, in some cases, the simulation time as well as code running time.

5.1 Signal Processing Examples

In signal processing examples, Raspberry Pi performs calculation of FFT, inverse FFT, FFT convolution and digital filter. The Simulink models are implemented using appropriate toolboxes from Simulink to generate signal and perform the operation. The collected parameters are simulation time, clock time and code generation time. The settings for the functions can be explained as follows.

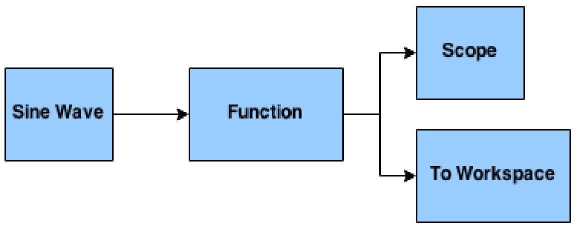


Figure 5.1 *Simulink model general structure for signal input*

The functions use a `Sine Wave` block to generate 8 N-length samples of length being a power of 2, starting from 32 to 4096. The block is set to use sample based with amplitude 1 and sample time 0.0001. The FFT and inverse FFT are calculated using `FFT` and `IFFT` block respectively while the FFT convolution, each N-length FFT sequences are multiplied with a sample of length 16 in frequency domain and then inverse FFT is performed. In the digital filter example, a sequence of length

4096 with a uniform noise is filtered by 8 N-length samples using `CONV` block. The output results for all models are observed using `Scope` block to view the output signal and `To Workspace` block to save the results in *structure with time* format providing simulation time and N-length sample results. The Simulink model general structure is as shown in Figure 5.1 in which each function is set to run for 0.5 Simulink time unit. The timing results, shown in Table 5.1 and 5.2, are simulation time, clock time and code generation time indicated as `t_sim`, `t_clock` and `t_code` respectively where clock time and generation time can be observed by *Diagnostic Viewer* window.

Table 5.1 *FFT and inverse FFT timing results*

Sample	**FFT**			**Inverse FFT**		
N-length	**t_sim**	**t_clock**	**t_code**	**t_sim**	**t_clock**	**t_code**
32	0.0031	54	37	0.0031	46	32
64	0.0063	49	32	0.0063	49	32
128	0.0127	49	32	0.0127	46	30
256	0.0256	48	31	0.0256	47	30
512	0.0511	49	32	0.0511	48	31
1024	0.1023	45	31	0.1023	51	32
2048	0.2045	48	32	0.2045	49	31
4096	0.4095	48	30	0.4095	48	33

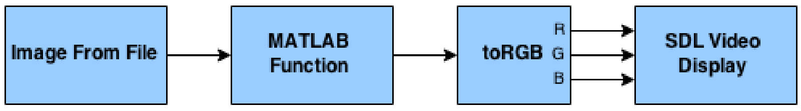
Table 5.2 *FFT convolution and digital filter timing results*

Sample	**FFT Convolution**			**Digital Filter**		
N-length	**t_sim**	**t_clock**	**t_code**	**t_sim**	**t_clock**	**t_code**
32	0.0031	55	36	0.0031	44	29
64	0.0063	47	31	0.0063	44	29
128	0.0127	47	32	0.0127	41	28
256	0.0256	49	33	0.0256	44	30
512	0.0511	44	29	0.0511	44	29
1024	0.1023	43	28	0.1023	43	29
2048	0.2045	49	30	0.2045	43	29
4096	0.4095	45	30	0.4095	44	29

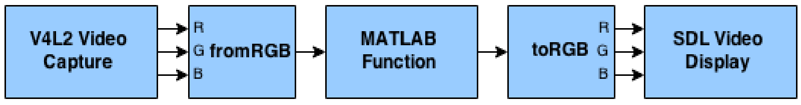
The simulation time, `t_sim`, is obtained from `To Workspace` block output file. The result is according to a number of sample and sample time parameter set in `Sine Wave` block such that the model produces one sample per 0.0001 second. Hence, the output signal for 32 samples are produced within 0.0031 seconds and so on for any sequence of length N. Clock time and code generation time, `t_clock` and `t_code`, are obtained from *Diagnostic Viewer* window in which clock time is the total running time of the function in Raspberry Pi and code generation time is the total time Matlab used to generate and deploy the code onto the board.

5.2 Image Processing Examples

The image processing examples are performed on both an image and a video input fed to Raspberry Pi through a specific toolbox and a camera module respectively. Both options can provide the result through a SDL video display. A user-defined code is needed as an aid to adjust the input from a camera module as well as when showing the result.



(a) Simulink model for image input



(b) Simulink model for video input

***Figure* 5.2** *Simulink model general structure for image and video input*

Figure 5.2 shows a general structure of two Simulink models for the image and video input. For the image input, the image is fed to Raspberry Pi using `Image From File` block. Scripts for the operation and preparing the result are written to two `MATLAB Function` blocks, in which one is referred as `toRGB`, then the result is displayed through `SDL Video Display` block. The structure for a video input is generally the same as for an image input with an additional `MATLAB Function` block, referred as `fromRGB`, added before performing an operation in order to prepare an input to a suitable type.

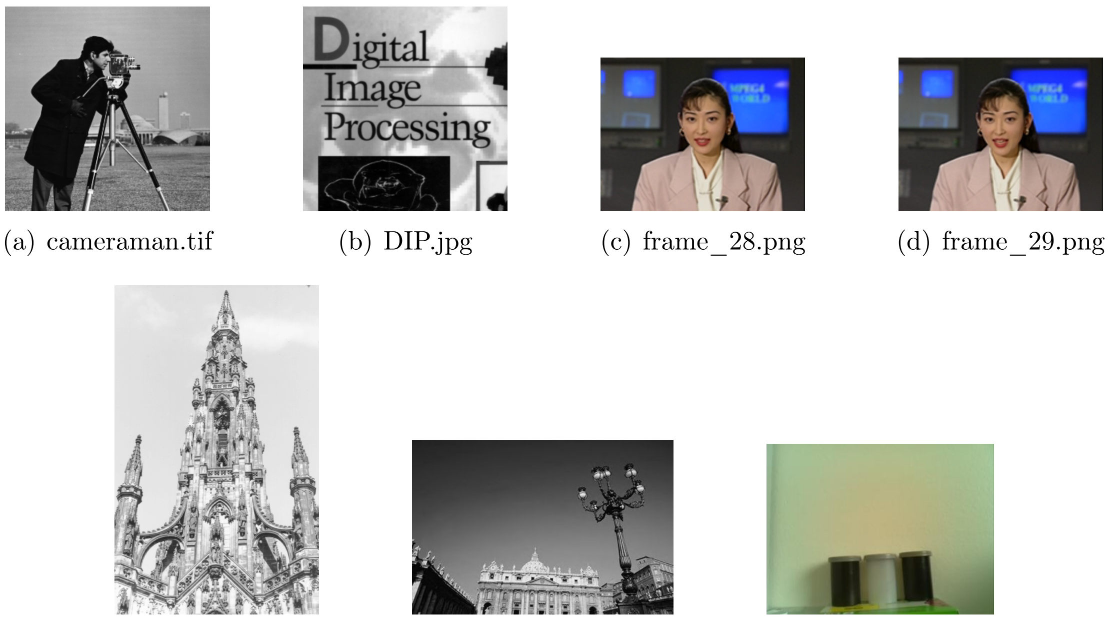
The `V4L2 Video Capture` block provides a sequence of RGB images in contrast to the `SDL Video Display` block which accepts separated R, G and B input. Hence, the codes on `fromRGB` and `toRGB` are written in such a way that they concatenate and separate the object respectively. The examples of the codes are as shown below.

```
function img = fromRGB(Rin, Gin, Bin)
% prepare image
in = cat(3, Rin, Gin, Bin);
img = double(rgb2gray(in));

function [R, G, B] = toRGB(img)
% convert to RGB
img = img.*255;
imgRGB = cat(3, img, img, img);
r = uint8(imgRGB(:, :, 1));
```

```
g = uint8(imgRGB(:, :, 2));
b = uint8(imgRGB(:, :, 3));
% rotate 90 then flip
R = flipud(rot90(r));
G = flipud(rot90(g));
B = flipud(rot90(b));
```

For `fromRGB`, the code connects each R, G and B input together then convert into grayscale and `double` type in order to prepare for an operation. For `toRGB`, the input is multiplied with 255 to prepare for a `uint8` type before being separated for R, G and B as an input for a `SDL Video Display` block. In case, the output displayed on a `SDL Video Display` block is an image or a graph, it needs to be rotate and flip to provide the correct position of the result. This, however, is not needed if the display output is video.



(a) cameraman.tif (b) DIP.jpg (c) frame_28.png (d) frame_29.png

(e) church.png (f) corel.png (g) pattern.png

***Figure* 5.3** *Images used in Image Processing Functions*

The image and video examples are performed for histogram, histogram equalization, edge detection, images difference, Butterworth filter and motion blurred filter. Each function is set to run for infinity, *inf*, of Simulink time unit in which the model keeps producing the result until users decide to stop by pressing stop button on a Simulink model or running a command `h.stop('modelName')` on Matlab command window. Hence, the collected parameter is code generation time only. The images used for the operation is as shown in Figure 5.3.

The specific settings for each function are explained as follows.

- **Histogram** calculates histogram of an input through its pixel value. The value is rounded to be maximum as 256 in order to fit the size of the display window at 256 by 256 for width and height created with a user-defined code [63].
- **Histogram equalization** performs histograms equalization on an image using another image histogram. This function is performed only on an image input [65].
- **Edge detection** uses `Edge Detection` block to detect the edge of an image and in video input. The results are shown in two types for only edge detection and edge detection masked on a grayscale and colored input.
- **Images difference** calculates the difference between 2 images. This function also performs only on an image input as same as histogram equalization [64].
- **Butterworth filter** performs Butterworth lowpass and highpass filter on an image and a video input. The code provided through `MATLAB Function` block uses `fft2`, `ifft2` and `fftshift` for Butterworth calculation which requires the input to be the size of a power of 2. Hence, `Image Pad` block is used to get the correct input size [66].
- **Motion blurred filter** performs motion blurred and its inverse filter on an image and a video input. The motion blurred filter code is written on `MATLAB Function` block using `fft2` and `fftshift`. This function is similar to Butterworth filter in which it requires `Image Pad` block to get the input size in a power of 2. The result is displayed in an image or a video and a mean square error (mse) value through a `Display` block [67].

Table 5.3 shows code generation time for each function on an image and a video input as well as image size and name. For the video input, a camera module is connected to Raspberry Pi and captures the same scene for all functions. According to the information shown in the table, the code generation time is affected by two factors in which one is the size of the input, especially for an image, and another one is the complexity of the code. The size of an image shows that the total greater size requires longer code generating time than the smaller one while the complexity of the code depends on the Matlab functions used to perform the result which can be explained in such a way that Matlab needs more time to generate the input object and the code then deploy onto Raspberry Pi, in contrast to a video input, where the input is fed directly from a camera module attached to the board.

Table 5.3 *Image processing examples timing result*

Code Generation Time	Image			Video	
Function	**Name**	**Size**	**Time**	**Size**	**Time**
Histogram	DIP.jpg	256x256	56 sec	256x256	37 sec
Histogram equalization	corel.png	384x256	1.07 min	N/A	N/A
	church.png	412x663			
Edge detection	pattern.png	320x240	44 sec	320x240	54 sec
Edge detection: grayscale	pattern.png	320x240	1.01 min	320x240	44 sec
Edge detection: color	pattern.png	320x240	52 sec	320x240	56 sec
Images difference	frame_28.png	640x480	2.05 min	N/A	N/A
	frame_29.png	640x480			
Butterworth: lowpass	cameraman.tif	256x256	42 sec	256x256	46 sec
Butterworth: highpass	cameraman.tif	256x256	43 sec	256x256	58 sec
Motion blurred	DIP.jpg	256x256	1.33 min	256x256	49 sec
Inverse motion blurred	DIP.jpg	256x256	1.12 min	256x256	45 sec

5.3 Toolbox Examples

Examples in this section are taken from Computer Vision system toolbox examples provided by Matlab. Two examples include barcode recognition and pattern matching. Input provided to Raspberry Pi is taken from a camera module and result is displayed as a video output. Similar to image processing examples, a user-defined code is needed as an aid to adjust the input from a camera module as well as when showing the result. The model is set to run as infinity, *inf*, of Simulink time unit, therefore, the only collected parameter is code generation time. Simulink model general structure for toolbox examples is as shown in Figure 5.4.

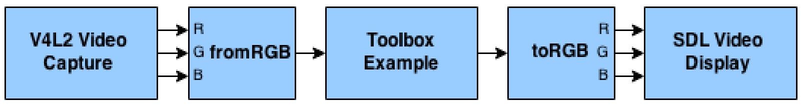
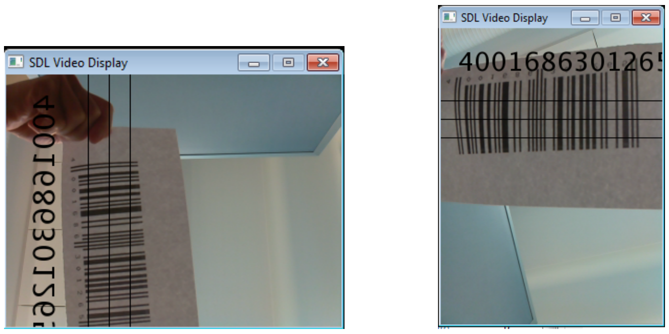


Figure 5.4 *Simulink model general structure for toolbox examples*

Generally, toolbox examples take a video file as an input through `From Multimedia File` block and display a result through `Video Viewer` or `To Video Display` block. To port toolbox examples to Raspberry Pi, a camera module is used to provide an input instead since the board does not accept an input from `From Multimedia File` block and result is displayed through `SDL Video Display` block. The user-defined codes, `fromRGB` and `toRGB`, are added in the model to help preparing the image. The specific settings and result for barcode recognition and pattern matching example are explained as follows.

Barcode recognition [1] detects a European Article Number (EAN)-13 barcode. The input is provided through a `V4L2 Video Capture` block and the result is displayed using a `SDL Video Display` block. The algorithm can be explained such that a barcode's scanline is interpreted into a vector sequence containing values between 1 and -1 where 1 is black and -1 is white. Unclassified color is valued in between. After feature extraction and transformation, the vector sequence is then compared with a codebook to provide the result as shown in Figure 5.5.



(a) Original Display (b) Adjusted Display

Figure 5.5 *Barcode recognition result*

The output display has a dimension as 240x320 and code generation result takes 1.31 minutes. The display output needs to be rotated then flipped in order to provide a correct position of the result. The script for rotate and flip is similar to that provided in `toRGB` function shown in image processing example. This model example requires the barcode used for recognition to be in a clear and large size figure due to the limitation of a camera module's focal length. In this case, a barcode is printed on a A4 paper size.

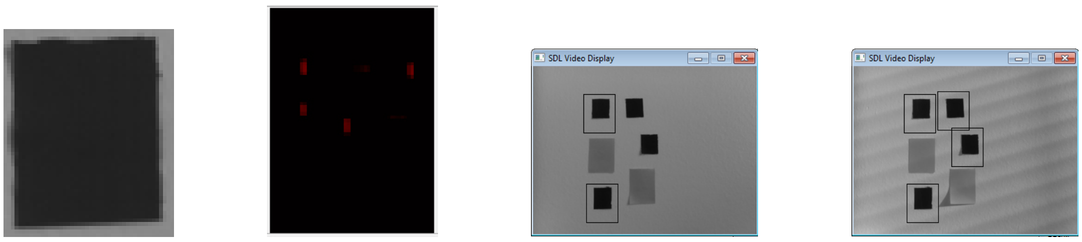
The second toolbox example is pattern matching [16]. Pattern matching matches a target image with an input from a camera module using 2-D normalized cross-correlation. The model allows adjusting a target image, number of target to be detected, Gaussian pyramid, threshold value and correlation method. The `InitFcn` in `Model Properties` initializes parameters for the model. In order to port a model to Raspberry Pi, those parameters need to be adjusted. The adjusted parameters are explained in Table 5.4.

The target image used in the model is a `uint8` type image which can be taken by capturing the scene using a camera module. User can add the image to the model by initializing it as `target_img` in Matlab Workspace. Gaussian pyramid needs to be set to a proper value for a target image to match with an input. In this case, value `a` for `Coefficient source` parameter in `Gaussian Pyramid` block is set as `0` instead

Table 5.4 *Pattern Matching Parameters in InitFcn*

Parameter	**Description**
`param_path = 'modelName/Parameters';`	Set path to the model.
`fmmf_path = 'modelName/V4L2 Video Capture';`	Set source block as `V4L2 Video Capture` for a video input from a camera module.
`iut_dim = ceil([W H]/2^(pfactor));`	Set dimension for Gaussian pyramid where `W` and `H` are width and height of a video input.

of a default value at `0.375`. The higher value `a`, the more intensity cross-correlation image. If cross-correlation image is too dark, the algorithm can not differentiate the object from background resulting in unable to match a target image with an input. Target image, cross-correlation image and matching results are as shown in Figure 5.6.



(a) Target image (b) Cross-correlation image (c) Target image = 2 (d) Target image = 4

Figure 5.6 *Pattern matching results and cross-correlation image*

The captured scene is a scene of four black squares and two gray squares on a white background. The target image is a black square as shown in Figure 5.6(a). The cross-correlation image, shown in Figure 5.6(b), shows four red squares corresponding to the position of the target images. The number of target image to be matched can be added up to 4. The output display has a dimension as 320x240 and code generation result takes 1.07 and 1.08 minutes for number of target image 2 and 4 respectively.

Table 5.5 shows a display output size corresponding to code generation time of barcode recognition and pattern matching. The results show that the display output size does not affect the code generation time for both models. It also shows that code generation times for barcode recognition take longer than those of pattern matching. This is due to the difference in the organization of the models. Barcode recognition comprises many scripts from `MATLAB Function` blocks, in contrast to

Table 5.5 *Output Size and Code Generation Time Result*

Toolbox Example	**Barcode Recognition**	**Pattern Matching**
Output Size	**Time**	**Time**
160x120	1.22 mins	1.05 mins
320x240	1.31 mins	1.08 mins
640x480	1.13 mins	1.06 mins
800x600	1.13 mins	1.10 mins

pattern matching, which is constructed purely on Simulink blocks.

Matlab's signal and image processing examples provide an insight to Raspberry Pi's operation under a specific condition. The examples show that time taken to generate the code to run on Raspberry Pi depends on the complexity and model construction. This can be explained in such a way that code generation takes more time if the model is more complex and constructed with mostly Matlab scripts. Since Matlab software provides code porting tool for target hardware through Simulink model, the Simulink blocks are more convenient to transform into codes than Matlab scripts which need to be adjusted to a proper format. Input's size also affects code generation time but not its source. Matlab software takes more time to generate codes for larger image input than smaller one. But the size does not hold important characteristic if the input is fed directly as a video.

6. CONCLUSIONS

In this thesis, Raspberry Pi 2 model B is studied and evaluated using Matlab software. In the beginning, the background information including setup steps for the board and Matlab software support package are presented. The background information shows that the board is capable of computer substitution with its powerful SoC, various connectors and GPIO pins to support peripheral devices. The availability of its graphic processor driver documentation also draws attention for parallel computing. This provides the relevant knowledge to start any implementations on the board.

Matlab software provides support package for Matlab and Simulink to generate and port codes to Raspberry Pi to run as a standalone application as well as running the code in an external mode. The implementation is done with Simulink model on signal and image processing examples taken from courses held in Tampere University of Technology and Computer Vision toolbox provided in Matlab. The model is constructed using general and Raspberry Pi special Simulink blocks as well as Matlab scripts. The evaluation is emphasised on code generation time and Simulink model complexity. The result shows the correlation between time and model complexity in which the more complex a model, the longer time Matlab use to generate the code. In this case, the complexity includes the need to create Matlab scripts for Simulink block. The other affected factor is the independence of an input such that Matlab software requires less time to compile code from an independent input source such as a camera pi. Concerning the results from signal and image processing examples, the time used to generate and port the code to run on Raspberry Pi is in an acceptable range. Therefore, the board can be used as a learning tool providing hands-on experience for students in practical Matlab education.

There are many possibilities for future work on Raspberry Pi platform. One interesting topic would be implementing Matlab and Simulink applications which exploits its graphic processor capability. The implementation could take advantage of an availability of Raspberry Pi's GPU driver documentation from Broadcom and various GPU tutorial examples such as GPU FFT which is provided in Raspbian OS. To be able to run Matlab and Simulink applications using the graphic processor instead of its central processor, the Matlab scripts need to call the processor with parameters provided in the driver documentation. Then, it is compiled with available Matlab code generation methods before porting to Raspberry Pi. However, there is a limitation to determine the code generation method that successfully compiles the adjusted scripts. More research and work can be done in such topic.

REFERENCES

[1] “Barcode recognition,” [online], Available: http://se.mathworks.com/help/vision/examples/barcode-recognition.html [accessed on May 3, 2015].

[2] “Camera module,” [online], Available: https://www.raspberrypi.org/products/camera-module/ [accessed on Feb 17, 2015].

[3] “Config.txt,” [online], Available: https://www.raspberrypi.org/documentation/configuration/config-txt.md [accessed on Apr 9, 2015].

[4] “Device trees, overlays and parameters,” [online], Available: https://github.com/raspberrypi/documentation/blob/master/configuration/device-tree.md [accessed on Feb 19, 2015].

[5] “Downloads,” [online], Available: http://www.raspberrypi.org/downloads/ [accessed on Feb 17, 2015].

[6] “Filesystem,” [online], Available: https://www.raspberrypi.org/forums/viewtopic.php?f=27&t=6170&p=80639#p80639 [accessed on May 8, 2015].

[7] “Gpio (general purpose input/output) inputs,” [online], Available: http://www.scriptoriumdesigns.com/embedded/gpio_in.php [accessed on Mar 26, 2015].

[8] “Gpio interfaces,” [online], Available: https://www.kernel.org/doc/Documentation/gpio/gpio.txt [accessed on Mar 19, 2015].

[9] “Guide for updating firmware and troubleshooting connection issues,” [online], Available: http://www.mathworks.com/matlabcentral/answers/uploaded_files/5422/Raspberry%20Pi%20Firmware%20Upgrade%20Guide.pdf [accessed on Feb 14, 2015].

[10] “How powerful is it?” [online], Available: https://www.raspberrypi.org/help/faqs/#performanceSpeed [accessed on Feb 22, 2015].

[11] “I2c,” [online], Available: https://learn.sparkfun.com/tutorials/i2c [accessed on Apr 5, 2015].

[12] “Matlab support package for raspberry pi hardware,” [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypiio/ [accessed on Mar 15, 2015].

[13] “Modeling,” [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypi/model-preparation.html [accessed on Mar 18, 2015].

[14] “Name-value pair arguments,” [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypiio/ref/cameraboard.html#namevaluepairs [accessed on Mar 15, 2015].

[15] “Noobs,” [online], Available: https://www.raspberrypi.org/documentation/installation/noobs.md [accessed on May 8, 2015].

[16] “Pattern matching,” [online], Available: http://se.mathworks.com/help/vision/examples/pattern-matching-1.html [accessed on Apr 20, 2015].

[17] “Power,” [online], Available: https://www.raspberrypi.org/help/faqs/#power [accessed on Feb 17, 2015].

[18] “Quick start guide,” [online], Available: https://www.raspberrypi.org/help/quick-start-guide/ [accessed on Feb 17, 2015].

[19] “Raspberry pi 1 model a+,” [online], Available: https://www.raspberrypi.org/products/model-a-plus/ [accessed on Feb 16, 2015].

[20] “Raspberry pi 1 model b+,” [online], Available: https://www.raspberrypi.org/products/model-b-plus/ [accessed on Feb 16, 2015].

[21] “Raspberry pi 2 model b,” [online], Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ [accessed on Feb 16, 2015].

[22] “Raspberry pi support from matlab,” [online], Available: http://se.mathworks.com/hardware-support/raspberry-pi-matlab.html [accessed on Feb 14, 2015].

[23] “Raspberry pi support from simulink,” [online], Available: http://se.mathworks.com/hardware-support/raspberry-pi-simulink.html [accessed on Feb 14, 2015].

[24] “Raspberry pinout,” [online], Available: http://pi.gadgetoid.com/pinout [accessed on Mar 26, 2015].

[25] “raspberrypi,” [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypi/ref/raspberrypi.html [accessed on Mar 11, 2015].

[26] “raspberrypi/hats,” [online], Available: https://github.com/raspberrypi/hats [accessed on Apr 25, 2015].

[27] “raspberrypi/linux,” [online], Available: https://github.com/raspberrypi/linux/blob/rpi-3.10.y/Documentation/video4linux/bcm2835-v4l2.txt [accessed on Mar 15, 2015].

[28] “Raspbian faq,” [online], Available: http://www.raspbian.org/RaspbianFAQ [accessed on May 8, 2015].

[29] "raspi," [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypiio/ref/raspi.html [accessed on Mar 11, 2015].

[30] "raspi-config," [online], Available: https://github.com/raspberrypi/documentation/blob/master/configuration/raspi-config.md [accessed on Feb 19, 2015].

[31] "Rpi easy sd card setup," [online], Available: http://elinux.org/RPi_Easy_SD_Card_Setup [accessed on Feb 17, 2015].

[32] "Run model on raspberry pi hardware," [online], Available: http://se.mathworks.com/help/supportpkg/raspberrypi/ug/create-and-run-an-application-on-raspberry_pi-hardware.html [accessed on Mar 11, 2015].

[33] "Serial communication," [online], Available: https://learn.sparkfun.com/tutorials/serial-communication [accessed on Apr 6, 2015].

[34] "Serial peripheral interface (spi)," [online], Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi [accessed on Apr 6, 2015].

[35] "Ssh using linux or mac os," [online], Available: https://www.raspberrypi.org/documentation/remote-access/ssh/unix.md [accessed on Feb 18, 2015].

[36] "The making of pi," [online], Available: https://www.raspberrypi.org/about/ [accessed on Jan 31, 2015].

[37] "Thingspeak," [online], Available: https://thingspeak.com/ [accessed on Mar 15, 2015].

[38] "tightvncserver," [online], Available: http://www.penguintutor.com/otherfiles/tightvncserver-init.txt [accessed on Feb 17, 2015].

[39] "Updating and upgrading raspbian," [online], Available: https://www.raspberrypi.org/documentation/raspbian/updating.md [accessed on Feb 19, 2015].

[40] "Not booting?! read this boot problem sticky (also for pi2b)," [online], 2013, Available: https://www.raspberrypi.org/forums/viewtopic.php?t=58151 [accessed on Apr 9, 2015].

[41] "Time sync with wifi," [online], 2013, Available: http://raspberrypi.stackexchange.com/questions/8732/time-sync-with-wifi [accessed on Feb 7, 2015].

[42] "Tutorial - how to give your raspberry pi a static ip address," [online], Jul 19 2013, Available: https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address [accessed on Feb 18, 2015].

[43] "Lsbinitscripts," [online], Jul 15 2014, Available: https://wiki.debian.org/LSBInitScripts [accessed on Feb 17, 2015].

[44] "Rpi software," [online], Mar 18 2014, Available: http://elinux.org/RPi_Software [accessed on Apr 9, 2015].

[45] "Simulink with raspberry pi camera capture," [online], Mar 19 2014, Available: http://www.mathworks.com/matlabcentral/answers/122199-simulink-with-raspberry-pi-camera-capture [accessed on Mar 18, 2015].

[46] "Can i put debian on my raspberry pi?" [online], Mar 20 (modified) 2015, Available: https://wiki.debian.org/RaspberryPi [accessed on May 8, 2015].

[47] "Computer," [online], 2015, Available: http://en.wikipedia.org/wiki/Computer [accessed on Jan 28, 2015].

[48] "Matlab," [online], May 13 (modified) 2015, Available: http://en.wikipedia.org/wiki/MATLAB [accessed on Feb 12, 2015].

[49] "Moore's law," [online], 2015, Available: http://en.wikipedia.org/wiki/Moore%27s_law [accessed on Jan 26, 2015].

[50] "Raspberry pi," [online], May 14 (modified) 2015, Available: http://en.wikipedia.org/wiki/Raspberry_Pi [accessed on Feb 16, 2015].

[51] "Rpi hardware," [online], Mar 30 (modified) 2015, Available: http://elinux.org/RPi_Hardware [accessed on Feb 17, 2015].

[52] "Rpi low-level peripherals," [online], May 8 (modified) 2015, Available: http://elinux.org/RPi_Low-level_peripherals [accessed on Mar 19, 2015].

[53] "Rpi usb webcams," [online], Apr 2 (modified) 2015, Available: http://elinux.org/RPi_USB_Webcams [accessed on Mar 15, 2015].

[54] J. Adams, "Introducing raspberry pi hats," [online], Jul 31 2014, Available: https://www.raspberrypi.org/introducing-raspberry-pi-hats/ [accessed on Apr 25, 2015].

[55] Andrew, "How to set up a wireless hotspot (access point mode) that supports android in ubuntu," [online], June 17 2013, Available: http://www.webupd8.org/2013/06/how-to-set-up-wireless-hotspot-access.html [accessed on Feb 18, 2015].

[56] *Cortex-A7 MPCore Technical Reference Manual*, ARM, Apr 11 (revision: r0p5) 2013, Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0464f/DDI0464F_cortex_a7_mpcore_r0p5_trm.pdf [accessed on Apr 8, 2015].

[57] A. Bradbury, "Open source arm userland," [online], Oct 24 2012, Available: https://www.raspberrypi.org/open-source-arm-userspace/ [accessed on Feb 22, 2015].

[58] *VideoCore® IV 3D Architecture Reference Guide*, BROADCOM CORPORATION, 5300 California Avenue, Irvine, CA 92617, Sep 16 2013, Available: http://www.broadcom.com/docs/support/videocore/VideoCoreIV-AG100-R.pdf [accessed on Apr 8, 2015].

[59] *BCM2835 ARM Peripherals*, Broadcom Europe Ltd., 406 Science Park Milton Road Cambridge CB4 0WW, Feb 6 2012, Available: https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf [accessed on May 18, 2015].

[60] K. Fatahalian and M. Houston, "Gpus: A closer look," *Queue*, vol. 6, no. 2, pp. 18–28, March/April 2008, Available: http://queue.acm.org/detail.cfm?id=1365498 [accessed on Feb 28, 2015].

[61] S. Monk, "Adafruit's raspberry pi lesson 3. network setup," [online], Nov 16 2013, Available: https://learn.adafruit.com/downloads/pdf/adafruits-raspberry-pi-lesson-3-network-setup.pdf [accessed on Feb 18, 2015].

[62] M. O'Hanlon, "Raspberry pi - run program at start-up," [online], Jun 10 2012, Available: http://www.stuffaboutcode.com/2012/06/raspberry-pi-run-program-at-start-up.html [accessed on Feb 7, 2015].

[63] *SGN-3016 Digital Image Processing I Exercise 5 Image Histogram*, Tampere University of Technology, 2011.

[64] *SGN-3106 Digital Video Processing Exercise 6 Compute Mean Squared Error (MSE) values*, Tampere University of Technology, 2012.

[65] *SGN-12006 Basic Course in Image and Video Processing Exercise 4 Histogram Equalization*, Tampere University of Technology, 2013.

[66] *SGN-12006 Basic Course in Image and Video Processing Exercise 6 Filtering in the Frequency Domain*, Tampere University of Technology, 2013.

[67] *SGN-12006 Basic Course in Image and Video Processing Exercise 9 Image Restoration*, Tampere University of Technology, 2013.

[68] E. Upton, "Accelerating fourier transforms using the gpu," [online], Jan 30 2014, Available: https://www.raspberrypi.org/accelerating-fourier-transforms-using-the-gpu/ [accessed on Feb 23, 2015].

[69] ——, "Android for all: Broadcom gives developers keys to the videocore® kingdom," [online], Feb 28 2014, Available: http://blog.broadcom.com/chip-design/android-for-all-broadcom-gives-developers-keys-to-the-videocore-kingdom/ [accessed on Feb 22, 2015].

[70] ——, "More qpu magic from pete warden," [online], Aug 8 2014, Available: https://www.raspberrypi.org/more-qpu-magic-from-pete-warden/ [accessed on Feb 24, 2015].

[71] ——, "Raspberry pi 2 on sale now at 35," [online], Feb 2 2015, Available: https://www.raspberrypi.org/raspberry-pi-2-on-sale/ [accessed on Feb 2, 2015].

[72] L. Upton, "Libraries, codecs, oss," [online], Jan 31 2012, Available: https://www.raspberrypi.org/libraries-codecs-oss/ [accessed on Feb 22, 2015].

[73] ——, "Five million sold!" [online], Feb 18 2015, Available: https://www.raspberrypi.org/five-million-sold/ [accessed on Apr 26, 2015].

[74] R. Walmsley, "Rastrack," [online], Available: http://rastrack.co.uk/t [accessed on Feb 19, 2015].

[75] S. Watkiss, "Remote gui access to a linux computer using tightvnc," [online], Available: http://penguintutor.com/linux/tightvnc [accessed on Feb 18, 2015].