PhD Proposal Writeup

A realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation.

Faculty of Mechanical Engineering,
Universiti Malaysia Pahang (UMP),
26600 Pekan, Pahang Darul Makmur,
Malaysia.

PhD Program Registration Details		
1	Name of Student	**Wan Ruslan bin W Yusoff**
2	Student ID	**PFD18001**
3	National Reg. ID	**560911-03-5067**
4	Faculty	**Faculty of Mechanical Engineering**
5	Program	**Doctor of Philosophy (PhD)**
6	Field of Research	**Mechatronics and System Design**
7	Type of Study	**Research**
8	Mode of Study	**Full Time**
9	Registration Date	**Tue, 03 April 2018**
10	Supervisor	**Dr. Fadhlur Rahman bin Mohd Romlay**
12	External Advisor	**Prof. Yashwant Prasad Singh**
13	Document Date	**March 27, 2021**
14	Research Title	**A realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation**
15	Contact EMail	**wruslandr@gmail.com**
16	Contact Mobile	**6012-3218120**

Reference: **main-phd-proposal-WRY.tex**
Date: **March 27, 2021**
Version: **phd-proposal-update**

Abstract

***TO REMOVE LATER**: The abstract is a brief summary of your Ph.D. Research Proposal, and should be no longer than 200 words. It starts by describing in a few words the knowledge domain where your research takes place and the key issues of that domain that offer opportunities for the scientific or technological innovations you intend to explore. Taking those key issues as a background, you then present briefly your research statement, your proposed research approach, the results you expect to achieve, and the anticipated implications of such results on the advancement of the knowledge domain.*

***TO REMOVE LATER**: To keep your abstract concise and objective, imagine that you were looking for financial support from someone who is very busy. Suppose that you were to meet that person at an official reception and that she would be willing to listen to you for no more than two minutes. What you would say to that person, and the pleasant style you would adopt in those two demanding minutes, is what you should put in your abstract. The guidelines provided in this template are meant to be used creatively and not, by any means, as a cookbook recipe for the production of research proposals.*

Keywords

CNC, interpolation, reference-pulse, look-ahead control, feedrate compensation, realtime, realtime processing, parallel computing, parallel algorithm.

***TO REMOVE LATER**: This section is an alphabetically ordered list of the more appropriate words or expressions (up to twelve) that you would introduce in a search engine to find a research proposal identical to yours. The successive keywords are separated by comas.*

Acknowledgement

Bismillahir-Rahmanir-Rahim. Innal hamdulillah, wa nahmaduhu, wa nasta'ienuhu, wa nastaghfiruhu. Wa na'udzubihi, min syururi anfusina, wa min sayyita a'qmalina. Man yahdhihillahu, fala mudhillalah. Wa man yudhlil, fala ha diyalah. Assalamualaikum Warahmatullah Hiwabarokatuh.

In the name of Allah, the Most Merciful and Most Compassionate. Indeed all praises be unto Allah, and we praise Him, and we seek help from Him, and we seek forgiveness from Him. We seek refuge from Him from the evil of ourselves, and from the evil of our actions. Whomsoever Allah guides, none can misguide. And whomsoever He leaves astray, none can guide. May the Peace, Mercy and Blessings of Allah be upon you.

Foremost, I thank Allah, the Most Glorious and Most High, for granting me the opportunity to tread down the unknown trail on this research journey. I wish to also convey my sincerest gratitude to all people who have directly or indirectly, or will be involved with me on this journey. There are just too many people to mention.

Specially to Prof Yashwant Prasad Singh, perceived as many personalities to me: As my guru, teacher, mentor, advisor, supervisor and a dear friend, I am very grateful to him for his unimaginable faith, persistence and enthusiasm in encouraging, guiding, and sharing with me knowledge throughout the many wonderful years of our academic acquaintance. As my Supervisor, his advice was simple, "Your PhD study should be exciting and fun." We spent long hours and fruitful discussions on almost unlimited topics, from philosophy, politics, religion, and family to the hard sciences, computer science, and engineering. We also made a pact to remain as lifelong friends, Insya Allah, God Willing. With internet facilities today, we are constantly in communication.

As a tribute to Dr. Fadhlur Rahman, my direct supervisor, I am eternally indebted to him for his sincere trust, unbelievable patience, constant guidance and timely assistance with research equipment, resources and many other administrative needs of the university. To my brother, Prof Ir Dr Wan Azhar, I undyingly appreciate his challenge that I should eventually get a doctorate. To my son, Abdulazeez, I thank him adoringly for his unequivocal faith, continuous encouragement and financial assistance in my endeavor.

To my family, especially my wife, my sons and daughter, I thank them affectionately for their unshakable love, utmost patience, undivided support and unwavering understanding during the long hours and sleepless nights I went through. The coffee and snacks were never interrupted. For smooth English writing, my wife is also my bouncer and editor.

To those in my extended family with PhDs, I thank them admiringly for their strange looks and jokes. One senior poked fun with a comment at me, considered as being the most intelligent among them but does not have a doctorate. My cynical response was, "Is it not inspiring that for many years I have been doing work of people with PhDs but without a doctorate myself? I feel, it is certainly humbling but yet assuring being accorded with that

kind of trust and responsibility."

To my friends, I kindly thank them for all support and encouragement rendered to me during my research journey. To Multimedia University (MMU), I thank them for the opportunity provided to me for teaching, conducting research and the unforgettable interactions with Prof Singh and staff at the university. To University Malaysia Pahang (UMP), I thank them for accepting me as a research student and for the generosity of providing research equipment and resources.

Finally, I again praise Allah, for the invaluable gifts of health, time and clear state of mind, without which, I would not have been able to go through this arduous and exhausting journey. There is always a purpose in everything. Thinking of it, I recalled the motto of a local university, "To Allah and Mankind". Without hesitation, I say, this is exactly the one for me. God Willing, may Allah grant me this deserving wish. In closing, I wish to share the following passages from Allah, the All-Knowing and All-Mighty.

> And in His Providence are the keys of the Unseen; none knows them except He. And He knows whatever is in the land and the sea. And in no way does a leaf fall down, except that He knows it, and not a grain in the darkness of the earth, not a thing wet or dry, except that it is in an evident Book.
>
> Whoever submits his whole self to God and is a doer of good, he has grasped indeed the most trustworthy handhold. And with God rests the end and decision of all affairs.
>
> Verily, When He (Allah) intends a thing, his command is "Be ! and it is !".
>
> (*Al-Quran, Al-An'am 6:59, Lukman 31:22 and Ya-Sin 36:82*)

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Introduction

With the availability of increased processing power and memory of personal computers, today Computer Numerical Control (CNC) interpolation is mostly carried out using computer software instead of electronic hardware devices with mathematical logic, for example, the Digital Differential Analyser (DDA) integrator. The basic theory and design of CNC systems, however, have remain the same since its beginning [1].

The two common types of CNC software interpolation techniques are Reference-Pulse Interpolation [2],[3], and Reference-Word (Sampled-Data) Interpolation [4], [5].

Over time, various algorithms have been proposed for CNC software interpolation. For example, techniques for Reference-Pulse interpolation include software version of the Digital Differential Analyser (SDDA), the Stairs Approximation and the Direct Search Interpolation.

Techniques for Reference-Word interpolation include Euler, Improved Euler, Taylor, Tustin and Improved Tustin interpolations.

In CNC machine operation, the function of interpolation is to generate coordinated movements to drive the separate axis-of-motions and/or rotation axes in order to achieve the desired path of the CNC cutting or milling tool relative to the workpiece. Essentially, interpolation generates the final reference commands that moves stepper or servo motor drives to produce the physical part that is to be machined. Reference commands mean commands to the CNC tool to trace/follow (refer to) the desired path or trajectory.

1.2 Types of CNC Machines

There are various types of CNC machines. The main categories are CNC Milling, CNC Lathe, CNC Router and CNC 3D Printing machines. There are many sub-categories, covering variations of CNC machines, and it is too numerous to list.

These machines share one common characteristic, that is, they all are numerically controlled by a computer, thus adopting the name Computer Numerical Control (CNC) machines. Most CNC machines cut or remove materials in order to produce a machined part. The exception is the CNC 3D Printing machine where materials are instead deposited layer by layer to produce a part.

1.2.1 CNC Milling Machine

CNC milling devices are the most widely used type of CNC machine. It is a machine that produces a work part by both drilling and cutting using a rotating cylindrical cutting tool. In a milling machine, the cutter is able to move along multiple axes simultaneously. It can create a part with a variety of shapes, slots and holes. To produce a part, the work-piece in

a milling machine is often moved across the milling tool (drill cutter) in different directions, like linear table movements and circular table rotations. Common milling machines offer from 3 to 5 motion axes. Please see Fig. 1.1, a link to the image of a typical CNC Milling machine.

1.2.2 CNC Lathe Machine

The CNC lathe machine is a machine that rotates a workpiece on a spindle to cut away excess material. The lathe uses cutting tools and drill bits with different diameters to apply on the workpiece and produce a symmetrical object. In a lathe machine, the cutter is not able to move along multiple axes. These machines can produce objects in a variety of shapes, cuts, and details based on cutting a rotating work piece. The two lathe configuration types are the CNC horizontal lathe and the CNC vertical lathe. Please see Fig. 1.2, a link to the image of a typical CNC Lathe machine.

1.2.3 CNC Router Machine

The CNC router machine is very similar in concept to a CNC milling machine. Instead of routing by hand, tool paths for the CNC router are controlled via computer numerical control system. The word rout means to "hollow out" an area in relatively hard material. Routers are normally used for cutting various hard materials, such as engraving wood, composites, aluminum, steel, ceramics, plastics, glass, and foams. The application of routers are mainly in woodworking, especially building cabinets, engraving decorations, doors or panels. Routers are not generally used for drilling or 3D model machining. Please see Fig. 1.3, a link to the image of a typical CNC Routing machine.

1.2.4 CNC 3D Printing

The CNC 3D Printing machine is a machine in which material is deposited, joined or solidified under computer control to create a three-dimensional object. Many different technologies are used in the 3D printing process, the most common being the fused deposit modeling (FDM) method. The machine uses inkjet-technology nozzles to apply a specialized thick liquid or powder to form each new layer. Common materials used in 3D printing are thick waxes and plastic polymers. Please see Fig. 1.4, a link to the image of a typical CNC 3D Printing machine.

1.2.5 Recent Designs in CNC Systems

On designs in CNC machine architectures, recent developments in CNC systems include, for example, designs with the introduction of more motion axes (up to 9 axes), cutting under water or fluid immersion environments, new sensors, smart computing, high-speed machining, energy efficient techniques, artificial intelligence control, wastage reduction in materials removal, process monitoring, machining management operations, modular software construction, and human-machine interface.

On designs in CNC machining control, new developments include design, for example, in software algorithms and methods, for path motion smoothness, for machining speed control, for addressing acceleration and deceleration effects, for 2D path error reduction, for 3D contouring error control, for compensating tool jerks, vibrations, heating and friction effects.

1.3 Basic architecture of a typical CNC system

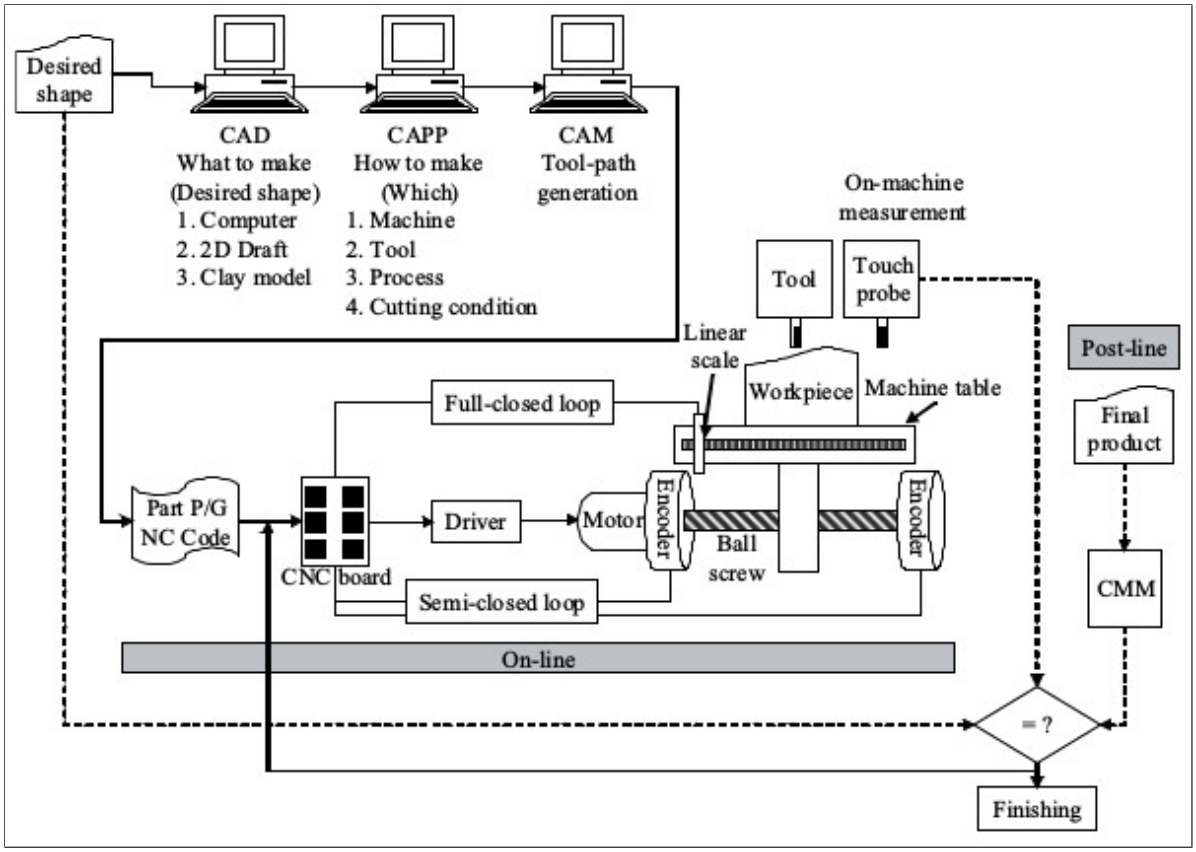


Figure 1.1: Basic architecture of a typical CNC system.

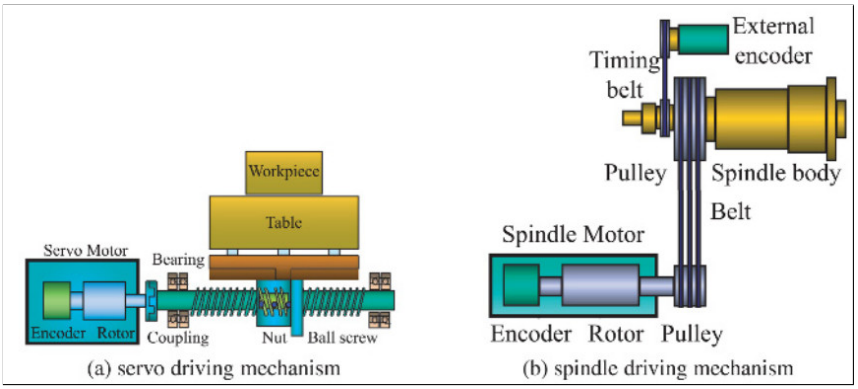


Figure 1.2: CNC Servo and Spindle Driving Mechanism

1.4 Process Flow for a typical CNC system

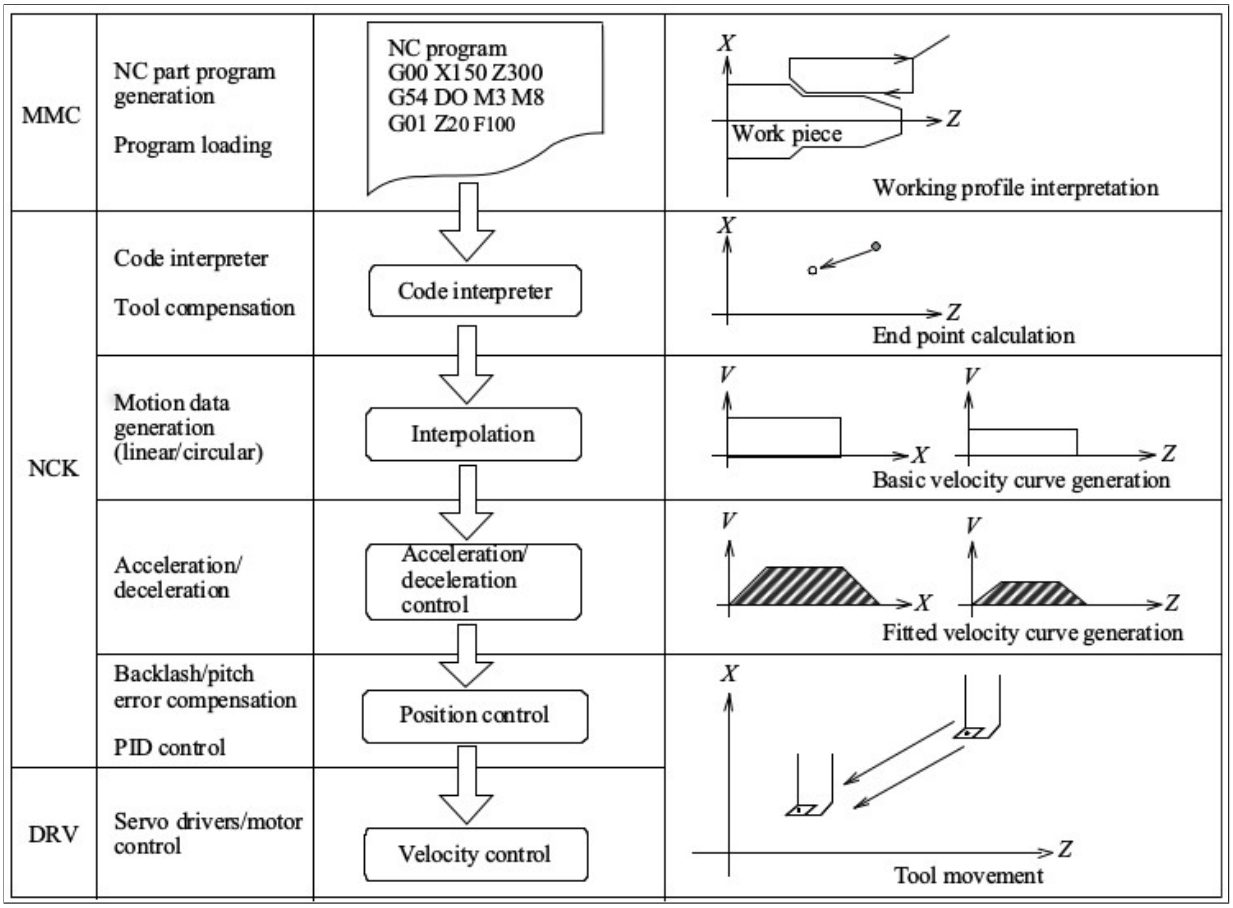


Figure 1.3: Process Flow for a typical CNC system

Legend

1. MMC Man-Machine Control
2. NCK Numerical Control Kernel
3. DRV Driver Module
4. NC Numerical Control
5. PID Proportional Integrative Derivative

1.5 Functional block diagram for a typical CNC system

CNC manufacturing operations involve the generation of reference signals describing the geometrical parts to be machined and the control of the machine. The control is such that it follows those reference signals. In modern CNC machines, the generation of reference signals, construction and execution of control loops are accomplished in software within a computer.

The figure below shows the functional block diagram for a typical CNC system. The meanings of terms used are described in Table [1.1] and Table [1.2].

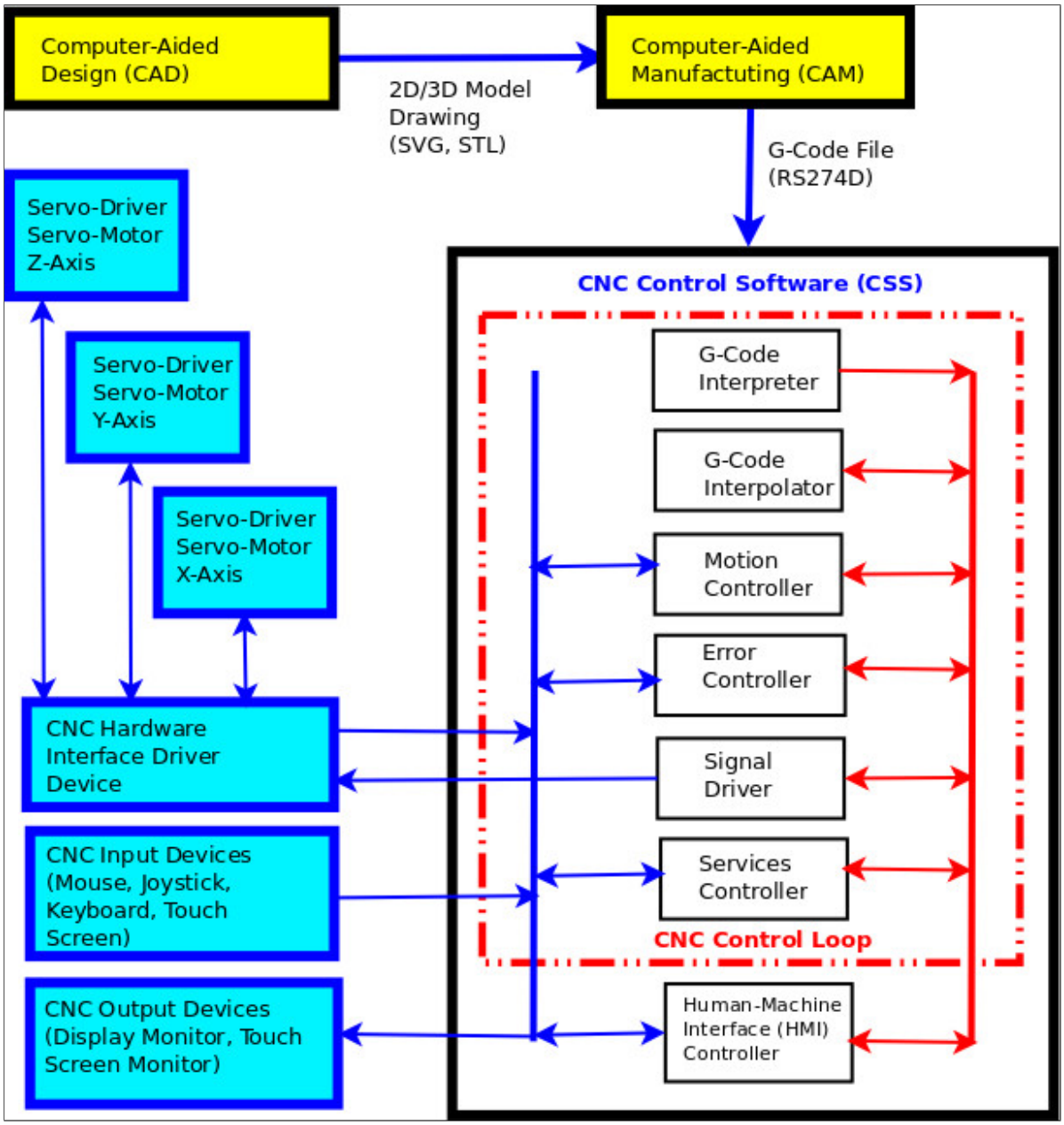


Figure 1.4: Functional Block Diagram for CNC system

1.5.1 CNC operations in CAD/CAM/CNC systems

The overall CNC operations in parts manufacturing using CAD/CAM/CNC systems can be described as follows.

First, the part geometry to be machined is generated by a Computer Aided Design (CAD) system that produces a 2D or 3D geometrical or model drawing.

Next, the Computer Aided Manufacturing (CAM) system converts this geometrical drawing into a part program called its G-Code. This G-Code file consists of a list of motion and machine service commands that provide specific step-by-step instructions for the CNC machine to manufacture the part. The G-Code part program generated by the CAM system is governed by international standards. This standardization ensures that different CAD drawings can be converted to a standardized G-Code by different CAM systems. Thus, different CNC manufacturers can machine the common G-Code and produce identical machine parts.

Next, a CNC interpreter program sequentially reads and interprets this G-Code commands, identifying whether the instruction is about tool movement processing, tool feedrate processing or machine function. For example, the prefix G in the part program instruction (G-Code) is for tool movement like a linear move or a circular arc move. This instruction moves the tool along the translational axes or along the rotary axes for some specified geometric path. The prefix M instruction is for controlling machine service functions like the start and stop of the tool cutting spindle. The prefix F instruction is for setting feedrates or speeds of tool movements along the translational axes or the rotary axes.

The next activity after CNC G-Code interpretation is CNC interpolation processing. CNC software interpolations cover linear, parametric and curve path contour commands provided in the G-Codes. The task of CNC G-Code interpolation is to process the interpreted G-Code and generate the final reference commands (referring to electrical signals) that drive the CNC tool along a pre-determined machining path. This contour path is normally constructed from a combination of linear and circular path segments. For example, interpolation generates reference commands that drive stepper or servo motors such that the required rotary or translational motions occur in a coordinated manner along the different axes simultaneously in tracing the desired contour path. In the process of CNC interpolation, the CNC Error Controller and CNC Motion Controller programs are executed based on to specific computation algorithms to achieve tracing accuracy along the desired contour path. These error-corrected or error-compensated outputs become machine reference commands (referring to the path) that drive the CNC motors.

The machine reference commands basically form a list of software machine instructions, saved in a CNC Signal File. The machine instructions in the signal file are read by the CNC Signal Driver in the CNC control loop and then transmitted to the CNC Hardware Interface Driver, which in turn, converts the software machine instructions that finally generates electrical pulses that drive the CNC electric motors.

The machine part to be manufactured is considered finished when the execution of the full list of reference commands completes and the CNC control loop exits.

1.5.2 Interpreter and interpolator

The interpreter and the interpolator programs are specific to a particular CNC machine. The programs are different across different CNC machines due to differences, for example, in machine hardware configurations, in number of motion axes, in functions of cutting tools, in machine service functions, in G-Code standards adoption, and so on. It is therefore, standard practice that most hardware manufacturers create their own proprietary interpreter and interpolator programs for their CNC machines. These program codes are not public.

Based on the above reason, in this research project, the development of our own interpreter and interpolator programs is our primary goal.

1.5.3 Interpolation Task

In CNC interpolation, it is important to differentiate between CAM model interpolation and CNC G-Code interpolation to avoid confusion. The CAM interpolation process converts 2D/3D model drawings to produce G-Codes files, whereas the CNC interpolation process converts G-Codes files into electrical signals to drive CNC motors.

Most advanced CAM software are commercial, not cost-free like many open-source type software. They are written by experts with well grounded knowledge in the mathematics of geometrical model representations for 2D and 3D objects. We do not want to be CAM designers, thus we leave CAM model interpolation to the experts.

For the above reasons, in this research project, CNC G-Code software interpolation is our only concern. CAM interpolation is not in our scope.

1.5.4 Starting point of CNC operations

Even though the CAD/CAM portion is not considered a component part of the CNC system, it plays an important initial role in the generation of G-Codes files. A high end and full-featured CAD/CAM system can produce G-Codes of different quality and contain different depths of manufacturing information. These sophisticated CAD/CAM applications can sometimes generate G-Codes complying to different G-Code standards.

In this research project, we consider the start of CNC machine operation as beginning from G-Code processing stage, and not from the CAD/CAM stages. In addition, we will only use G-Code files conforming to the RS274-D NGC standard. Other G-Code standards are not in our scope. We will discuss this further in the section on literature review.

1.5.5 Online versus Offline processing

In general, CNC Interpretation and CNC Interpolation can be conducted as online tasks, as offline tasks, or as a combination of both. In online mode, both interpretation and interpolation tasks are computed in memory and then fed into the CNC control loop while

the loop is running. In offline mode, the interpretation and interpolation are pre-computed, stored in a file or memory and later fed to the CNC control loop.

The term realtime processing used synonymously to mean online processing in common CNC literature, is not technically correct. In software engineering, the terms are conceptually unrelated. We clarify this in the next section.

In this research project, we will implement and compare performances of both online and offline options.

1.5.6 Realtime processing

In general usage, people used the word "realtime" as processing at the current time instance. The layman use of the term realtime in communications is very different from its technical use in software. We illustrate the difference as follows:

In software terms, the definition of realtime processing is the execution of a process that meets its time deadline. There are strictly two(2) time deadlines for any event: the start deadline and the end deadline.

As an example, consider two event deadlines for the safety airbag in an car. The event deadline to start opening the airbag is **after** 0.20 seconds from time of impact, and the event deadline for reaching a fully blown airbag is **before** 0.50 seconds from time of impact. It means **both events** must occur within the specific duration.

The two events are considered realtime events if they both strictly meet their deadlines. This is the technical criteria for being realtime. Opening the air bag before 0.20 seconds (too early) means missing the deadline. Also reaching a full blown air bag after 0.50 seconds (too late) also means missing its deadline. In both cases, the car driver may already be dead.

In this case, the safety airbag is designed such that it must open only after 0.20 seconds of impact and must get to full blown state before 0.50 seconds from time of impact.

If the airbag reaches a partially blown state at 0.50 seconds, the effectiveness of the safety airbag may be compromised, meaning, the design cannot guarantee the full protection of the car driver.

In any field of engineering design (including software programming), there is no such thing as an event to occur precisely at an exact point in time. This is unrealistic because there is no such thing as instantaneous in electronic or mechanical systems. Any event can only occur within some time duration, no matter how short the duration (for example, in microsecond or nanosecond range). It could be somewhere in the nanosecond range for timing control of ballistic missiles, whereas it is typically in the order of milliseconds, even for the most advanced CNC systems. Realtime compliance is very important in high performance time processing. Finally we state the technical definition: A realtime event must precisely occur within its specified duration, that is between its start and end deadlines.

In this research project, realtime shall mean the software event or process must comply to the technical definition of realtime above. This will be achieved only when program codes are written using realtime capable programming languages, and the codes are executed on Real Time Operating System (RTOS) platforms. For correct performance, CNC processing events typically runs in the order of 10 milliseconds. We will discuss this further in the section on literature review.

1.5.7 Parallel processing

Similarly, people have used the word "concurrent" as being synonymous with "parallel in time". In software terminology, we have to differentiate the layman use of the word parallel from its actual technical usage. We illustrate the difference between concurrent and parallel processes in software as follows:

Consider an example of a process where a person takes a jog and the jogging process is considered finished when the jogger reaches a specific destination. The jogger starts running and then along the way the jogger stops momentarily, to properly tie his loose shoe laces which took a few moments. During this short duration of tying the shoe lace, we consider the process of "shoe tying" and the process of "jogging" together as occurring concurrently.

Even though there is only a pause in the actual jogging, the jogging process is still considered ongoing because the destination has not yet being reached. When tightening the shoe laces in finished, the jogger resumes running until he reaches the destination, and then only the jogging process is considered finished. This is about concurrency.

All General Purpose Operating Systems (GPOS) platforms run their background processes concurrently. GPOS platforms are considered multi-tasking and time-sharing systems. Note that running in realtime (within start and end timing deadlines) and running in parallel are two separate concepts.

Next we explain about parallel processing. When two running processes or threads are executed on two separate CPUs independently at the same time, we say that the processes are truly running in parallel, meaning, overlapping in time. This is opposite to a sequential execution where one process must finish before another process can start. In parallel execution, processes truly run simultaneously or parallel in time.

In this research project, parallel shall mean simultaneously or overlapping in time. We shall implement program codes that run both in realtime and in parallel. We have successfully executed this before. We will discuss this further in the section on literature review.

1.5.8 Recent interpolation Methods

Recent techniques for CNC interpolation methods include advanced functions like look-ahead function, feedrate filtering and Non-Uniform Rational B-Splines (NURBS) interpolation. We will describe them in the chapter on literature review.

For this research project, we will study different interpolation methods. We propose the look-ahead control and feedrate compensation methods for realtime and parallel CNC interpolation.

CNC Terminology Part 1 of 2		
1,2	S/W and H/W	Software and Hardware
3	CNC G-Code File	The S/W file that is generated by a CAM application (including CAM interpolation) based on model or drawing files like DXF, STL (generated from CAD) and so on. This G-Code file, like RS-274D or STEP-NC file, contains instructions like various tool motion commands (G-group), machine service commands (M-group), and so on.
4	Command line	A single G-Code file S/W instruction. The contents of a CNC G-Code file is an ordered list of CNC command lines.
5	CNC G-Code Interpreter	The S/W application or program component that interprets G-code command lines and segregates into various groups like motion command G-group, services command M-group, and so on.
6	CNC G-Code Interpolator	The S/W application or program component that converts the G-Code motion command lines into signal commands and save them in a CNC Signals file.
7	Signal command	A single signal file S/W instruction.
8	CNC Signal file	The contents of a CNC Signal file is an ordered list of CNC signal commands. The term reference signal used in most literature is synonymous to this signal command. A reference signal basically refers to the signal after CNC interpretation and interpolation of a single G-Code line command (G-group, M-Group, etc.).
9	CNC Signal Driver	A S/W application or program component that reads the CNC Signal file and interprets a signal command. This application then generates the appropriate H/W electrical pulses using a specific hardware device. The pulses are finally sent to its destination, for example, the servo-driver and servo-motor hardware pair.
10	CNC Conversion from software to hardware electrical pulses	This conversion is the task of the CNC Signal Driver software. It is the boundary or interface point in CNC machine operations, where software codes generate actual physical electrical pulses. The electrical pulses, for example, drive the servo-driver and servo-motor pair, meaning, ultimately driving the CNC machine. The CNC hardware executes only on recognizing electrical pulses. The software codes are not of concern to the CNC machine.

Table 1.1: CNC Terminology Part 1 of 2

CNC Terminology Part 2 of 2		
11	CNC Motion Controller	The S/W component that implements and controls, for example, CNC tool motions, like direction, velocity, position, torque, acceleration, deceleration, feedrate and so on. This is to achieve tool accuracy in following path trajectory, tool motion smoothness, and avoid tool jerks.
12	CNC Error Controller	The S/W component that implements and controls, for example, CNC tool trajectory or path tracking, path error computation, path monitoring and compensation, lookahead control, adaptive control, iterative control, predictive control, AI control, and so on.
13	CNC Services Controller	The S/W component that controls CNC service functions like to start and stop the tool, to start and stop lubrication fluids, to pause and change tool cutters, and so on. It also monitors and controls other auxiliary services like excessive vibrations, over-currents, over speeds, broken tools, over heating and so on.
14	CNC HMI Controller	The Human-Machine Interface (HMI) S/W component that handles inputs/outputs with humans to control the CNC machine. This component also provides display and monitoring services for the entire operation of the CNC machine.
15	CNC Control Loop	The S/W component that coordinates and controls the timely executions of the following software sub-components: 1. CNC G-Code Interpreter 2. CNC G-Code Interpolator 3. CNC Signal Driver 4. CNC Motion Controller 5. CNC Error Controller 6. CNC Services Controller When the CNC Control Loop execution completes and exits, the CNC machine operations is considered completed or execution finished.
15	CNC Controller software (CCS)	The complete S/W application that combines the CNC Human-Machine Interface (HMI) Controller with the CNC Control Loop and its associated sub-components.

Table 1.2: CNC Terminology Part 2 of 2

1.6 CNC System Hardware and Software

1.6.1 CNC Hardware Components

The hardware for the CNC machine varies from low-end machines to very high end commercial systems. We list below some basic hardware for a minimal CNC system.

1. electrical signal generator device
2. motor-controller and its electric-motor pair
3. computer (normal desktop, laptop or dedicated server)
4. display and monitoring device
5. devices for inputs/outputs and machine tools
6. framework assembly for the complete CNC machine
7. other machine accessories (power supply, cooling system, etc)

1.6.2 CNC Software Components

Similarly, the software for CNC machine varies among different machines, from semi-automatic to fully automatic control. We list below some basic software requirements for a minimal CNC system.

1. Computer operating cystem (RTOS or GPOS)
2. CNC Control Software (CCS) comprises
   - CNC Control Loop
   - CNC Human-Machine Interface
3. CNC Control Loop comprises
   - CNC Interpreter
   - CNC Interpolator
   - CNC Motion Controller
   - CNC Error Controller
   - CNC Signal Driver
   - CNC Service Controller
4. Depending on CNC machine usage, software language compilers for the OS platform may be required for development, especially when the user is a systems programmer that wishes to perform customization.
5. Other associated software libraries with API may be required for external interfacing with the CNC Control Software (CCS).

1.6.3 CNC Intepreter

1.6.3.1 Primary memory data storage

In conventional practice, the task of the interpreter is to read a G-Code file, interprets the command line blocks in the file, and stores the interpreted data in memory (we call it primary) for use by the interpolator. When storing activity for one command line block completes, the interpreter reads and interprets the next command line block. In the duration of interpretation of the next command line block, the CNC machine motors may still be executing machine moves based on the previous command or have completed the moves. The expected event is that the moves along the different axis-of-motions for the previous command have completed.

1.6.3.2 Slow interpretation execution

Consider the case of a single processor computer. If the time to interpret the command block is longer than the time to finish the previous command line execution (meaning slow interpretation), the running motors of the CNC machine will have to pause momentarily, because the motors are waiting for completion of interpretation of the next command block. This stop happens because there is only one CPU processor, and that processor is still occupied with interpretation of the next command.

The solution for preventing the machine from regularly stopping (jerking) is to create an internal buffer that temporarily stores the interpreted data. The buffer must always keep a sufficient number of interpreted data for use by the interpolator. This is the usual practice.

For a multi-processor or multi-core computer system, the interpretation of the G-code command line blocks can be executed in parallel using dedicated cores. For example, with a 4-core processor, 3 cores can be used for interpretation while the 4th core is used for CNC loop control. However, it is very important that the organization of interpreted data saved to memory preserves the exact command sequence of the originating G-code file.

One solution is to create a special data structure in memory that accomplishes this ordering. This data organization structure is not necessarily sequential or circular. In addition, with large, fast random access memories, and fast processor speeds available today, for fast access times of interpreted data in memory, we may consider more efficient storage methods, for example, graph or tree data structures.

1.6.3.3 Efficient data I/O methods

The first task of the interpreter is to read the G-Code file, conventionally this happens in sequential order. Reading a file from external storage is very slow compared to reading from computer memory. With parallel I/O for file operations using Message Passing Interface (MPI) library (available since 1997), we may consider executing multiple processes to read and write to a file in parallel.

This means that we can pre-process the G-Code file into a format suitable for parallel file I/O. Reading the G-Code file will be a lot faster. If the G-Code file is very large, using solid state disks (SSDs) to store the G-Code file is another possible option.

1.6.4 CNC Interpolator

1.6.4.1 Secondary memory or file data storage

G-Code interpolation is the next task after G-Code interpretation. Similarly, in conventional practice, the interpolator sequentially reads the interpreted data from data buffer in primary memory (output of interpretation), calculates the position and velocity for each axis, and stores the result in a secondary memory FIFO (first in first out) buffer for later use by the CNC Signal Driver software controller.

The interpolation results are simply coded command signals that drive the CNC motors or other CNC service functions. If interpolation results are saved in a physical file (persistent), the file is named the CNC Signal file. We will show some of our own designs of CNC signal files that we have used in our previous research work,

The main purpose of saving (capturing) the CNC interpolation results into a physical file (CNC Signal file) as secondary memory instead of real memory buffer (non-persistent) on the computer, is for record keeping, inspection and investigation, such that the same CNC run configuration can be re-executed or played-back at another time.

In online conventional mode, interpretation and interpolation tasks are computed, results stored in memory (not in a file) and then directly fed into the CNC control loop while the loop is running. Whereas, in offline mode, the interpretation and interpolation are pre-computed, results stored in a file (CNC Signal file) and later fed to the CNC control loop.

The term online mode here means, as each command line block interpolation is completed, the resulting coded command signal is immediately fed and executed by the CNC control loop. In offline mode, the CNC control loop fetches the coded command signals from the CNC Signal file, line by line for execution.

The advantage of running in offline mode is the ability to analyse all command signals that are to be fed to the CNC control loop in the future. This means by reading the CNC Signal file, we can look-ahead (look forward) to the exact command signals that are going to be fed sequentially to the CNC Control loop. With this information we can calculate ahead ot time, for example, the expected contour error for each move, the expected accumulated error after a few moves, the expected tool velocities for future moves, and so on. This analysis allows us to correct for contour tracking errors, for example, using compensation, adaptation and other error control techniques yet to be considered. This really means we can modify command signal instructions for future moves. This is where the CNC Error Controller and CNC Motion Controller software components comes into play.

After reading one command line "interpreted" data from primary memory, the interpolator will know what kind of instruction to perform. For example, the next position to move to from the current position, the velocity to make the move to the next position, the type of geometric move to execute (linear, circular arc, parabolic or splined segments) for that path segment. The next position to move and the type of move to perform is considered as geometric or trajectory path data. With geometric path data, the distance to be covered by the move will be calculated by the interpolator.

1.6.4.2 Interpolator main task

The main task and responsibility of the CNC interpolator is to generate and send pulses to the respective motors at the corresponding axis-of-motions as commanded by the geometric path data to reach its target position. The interpolator must generate the right number of pulses to cover the distance for the move. It also has to generate the correct number of pulses per unit time (pulse frequency or pulse feedrate) to achieve the commanded velocity for the move.

These calculations are intensive and have to conducted for all of the motors at all of the axis-of-motions. Imagine the computational needs of different CNC machine configurations, like the 3-axis, 5-axis and 9-axis CNC machines. We realize here that having a multi-core computer and implementation of true parallel computing offer their advantages.

Essentially, the job for the interpolator is to make sure that pulses sent to the different axis-of-motions are coordinated, synchronized, and timely, such that the CNC tool accurately traces (tracks) the commanded path data trajectory. In CNC path tracking, a contour or trajectory error in tool motion means the CNC tool movements have deviated from the intended path contour.

1.6.4.3 CNC Accuracy - Basic Length Unit (BLU)

In CNC motion, the distance covered or displacement per pulse determines its accuracy. For example, if an axis can move at the rate of 0.002 mm per pulse, the accuracy of the CNC system is 0.002 mm. Each pulse contributes to a move. This translational displacement per pulse in CNC is called the basic length unit (BLU). In this example, in order to move a linear distance of 20 mm, we need to send 10,000 pulses (0.002 x 10,000). If we want to generate a move with velocity 20 mm per second, we need to generate 10,000 pulses per second or run at 10 KHz pulse frequency.

1.6.4.4 Velocity - Signal Pulse frequency

The pulse frequency limit is a physical constraint in the design of the CNC machine. Consider an example where the maximum pulse frequency of the pulse generator device (hardware) is capable of driving is 50 KHz, and the maximum pulse frequency the servo-driver and servo-motor pair can handle is 30 KHz. In this example, for the combination of pulse generator and servo-motor, the maximum pulse frequency for the CNC machine is only 30 KHz. Even though pulse rates can go up to 50 KHz, the motor can only handle up to 30 KHz. The

bottleneck is the servo-motor.

We know that in software, there is no limit to the value we can set for the driving pulse frequency (number of pulses per second, or feedrate), because it is just a number setting parameter in software, but in hardware there is a real physical limit.

If we use a lead-screw or ball-screw shaft for CNC linear displacement, the finer the screw thread pitch (thread interval distance), the more accurate is our machining. For very fine linear moves, we need very high pulse feedrates to finish the machining job within a reasonable amount of time. To get high feedrates, we need high frequency pulse generators and compatible high frequency servo-servo and servo-motor pairs.

The variable frequency (feedrate) error control method is a common control algorithm in CNC interpolation. This is done by suitably varying the frequency in time so that the CNC tool moves smoothly (smooth velocity profile) as it traces the contour path trajectory. Smooth velocity profiles mean continuous moves at the joints of the path segments. The smooth velocity profile concept is not about contour path displacement error, but it is affected by it (conflicting effects). This conflicting effect is discussed in the next section.

1.6.4.5 Tool Velocity Profile

A simple way to look at conflicting effects of contour path displacement errors against smooth velocity motions is as follows. Consider the analogy of driving a car. In order to quickly reach the end point of a curve on the road (arc segment), we need high velocity moves. With high velocity moves we cannot negotiate the sharp curve quickly because of momentum effects of the car. We will go off track, meaning, we created a contour error. If we drive at a lower velocity (slow), we will trace the contour path very well (good tracking profile). However, it takes a much longer time for us to reach the end point.

Driving a CNC machine is no different from driving a car, in fact, it is similar. Today, we have autonomous and self-driving cars under computer control that can do this job quite (not very) well. More on this will be discussed in the section on literature survey.

The tool velocity profile is concerned with acceleration and deceleration of the CNC cutting tool as it traces the commanded geometric contour path. CAD/CAM systems have to divide curves into a large number of segments while maintaining the set of contour error limits. For example, the segments comprise short linear (G01) segments, circular arc (G02, G03) segments or NURBS (G06 series for surfaces) segments generated by advanced high end CAD/CAM systems. Usually, low end CAD/CAM systems do not generate NURBS G-Code.

Due to the short segments, most of the time spent during interpolation is on stop-start motions (M-Group), typically accelerating, decelerating, or pausing between instructions. The frequent starts and stops for very small linear motions for every segment cause jerks during interpolation, especially at the junctions of the segments.

This results in discontinuous feedrate (velocity) profiles, jerky and generally un-smooth overall machining process.

1.6.4.6 Velocity profile controller

The velocity profile controller is also known as Acceleration/Deceleration (ACC/DEC) controller in older CNC literature.

If position control is executed by using data generated from the interpolator, whenever tool movement starts and stops large mechanical vibrations and shocks occur. In order to prevent mechanical vibration and shock, the filtering for ACC/DEC or velocity control is executed before interpolated data is sent to the position controller. This method is called the ACC/DEC after-interpolation method.

There is also the ACC/DEC before-interpolation method, where velocity control is executed before interpolation. This before-interpolation technique is akin to a look-ahead strategy. We do not consider this strategy as predictive control because the commanded contour path trajectory to be followed is already known for certain. We know for certain the exact position to go as our target. More will be described in the section on literature review.

The data from an velocity controller is sent to a position controller, normally for position adjustment. A position control mechanism typically means a PID controller. This PID controller issues velocity commands to the motor driving system in order to minimize the position difference between the commanded position and the actual position. The actual position is found from the encoder feedback in servo-driven motors.

For example, for our CNC Research machine, the commercial, industry standard and high end Panasonic Minas A4 servo-driver has a built-in configurable electronic circuit PID controller. The servo-driver can be configured to run on any one of three modes: position control, velocity control and torque control. The parameters for the PID controller can be set manually or calibrated automatically against the particular CNC machine characteristics to handle the velocity profiling problems (momentum effects).

Based on using the Minas A4 servo-driver, our experience on PID control at the electric-motor side (receiving end) is very satisfactory. For this reason, in this research project the velocity PID control handling is not our focus and therefore will not be included in our scope. However, we will still consider velocity control but at the CNC Interpolation software level on the trajectory path, and not at the real electric servo-motor end level.

In this project, our focus shall be more on contouring control, that is, making sure that our CNC tool contour tracking is accurate against the G-Code instructions. We are not even concerned with prior errors generated in CAM processing conversion of 2D/3D model drawings to produce G-Codes because we consider the start of our research project as beginning from the provision of G-Codes itself, not earlier.

1.6.5 CNC Signal Driver software

The CNC Signal Driver is the software component that comprises an important part of the CNC Control Loop. This software program reads the CNC Signal file and interprets a signal command. It then generates the appropriate electrical pulses (hardware) using a specific hardware device. The pulses are finally sent to its destination, for example, the servo-driver and servo-motor hardware pair. The CNC Signals file allows for offline CNC execution.

The Signal Driver software is simply tasked with converting software commands into electrical pulses. This conversion is the boundary or interface point in CNC machine operations, where software codes generate actual electrical pulses. The electrical pulses, for example, drive the servo-driver and servo-motor pair, meaning, ultimately driving the CNC machine. The CNC hardware executes only on electrical pulses. The software commands (instructions) are not of concern to the CNC machine.

Note that the CNC Signal Driver software itself cannot directly generate electrical pulses. It has to fetch from the CNC Signals file, the signal codes and send the software codes to a real physical device (hardware), called the CNC Device Driver. It is this actual hardware device that generates electrical pulses. We explain this further in the next section.

Even though anyone can buy the signal/pulse generator hardware device from the market, the generation of software signals and its contents are proprietary. This is another strong motivation, why in this research project we want to develop our own CNC Interpreter and CNC Interpolation software.

In addition, despite having the same signal/pulse generator hardware device, everyone will have their own way of creating and defining the contents of CNC Signal file. We have experienced the creation and definition of our own signals file and have executed them successfully. The signals file is machine dependent because it is specific for each pulse generator hardware. We will show this in the section on previous project experiences.

The CNC Signal file is useful in the following sense. In offline CNC operation mode, both interpretation and interpolation are pre-computed, results stored in a file (CNC Signal file)

and later the contents of the file fed to the CNC control loop.

Whereas in online CNC operation mode, both interpretation and interpolation tasks are computed in memory and then fed immediately and directly as it arrives into the CNC control loop while the loop is running. The results of computations for the software signals are held in memory and will be lost when the computer shuts down.

However, with multi-core computers like 8-cores, we can have 1 of the cores dedicated to just saving the results as it arrives. that is, by reading from memory and writing to a file like a "memory dump" for persistent storage.

This provides us with another opportunity in our research project to implement parallel computations. One processor core assigned solely for signal dump task while the other cores are assigned for normal tasks like driving the many axes of the CNC machine, processing of interrupts from the CNC Services controller, and so on.

1.6.6 CNC Device Driver hardware

The CNC Device Driver is the hardware device that interfaces between the CNC software and the CNC hardware. This is the boundary point where software commands get converted to electrical signals. This device receives software commands and generates electrical signals (either digital or analog signals) and transmits the signals to the motor control driver.

This hardware device could be an internal device like the computer parallel port, USB port, or serial port, or a specialized embedded board (ISA, PCI, PCI Express) like motion control boards installed inside the computer or an industry external interface board interfaced with the computer. There are many motion control board offerings available in the market for this specialized category of hardware.

Depending on configuration and type of signal generator device, communications can be unidirectional or bidirectional between the CNC control computer and CNC hardware. The actual signal generator device does not matter. The CNC machine operations will work as long as the device can generate appropriate electrical signal pulses to drive the CNC motors.

1.6.6.1 Research pulse generator devices

On our CNC research machine, the signal generator devices that we have used and tested successfully (from 2010-2018) comprise the following:

1. Personal Computer (PC) parallel port, ref[6], links[**??**], [**??**], [**??**].
2. Arduino Due board, ref[7], link[**??**].
3. Heber X10i USB board, ref[8], link[**??**].
4. Raspberry Pi 2 and Raspberry Pi 3 SBC boards, ref[9], links[**??**], [**??**].
5. Banana Pi SBC board, ref[10], link[**??**].
6. Velleman K8000 Parallel and K8055 USB interface boards, ref[6], links[**??**], [**??**].
7. BeagleBoard xM SBC board, ref[10], link[**??**]
8. Prolific PL2303 USB and PL2305 Parallel ports, ref[6], link[**??**].
9. Digilent Nexys-3 Spartan-6 FPGA development board, ref[11], link[**??**].

As we mentioned earlier, the creation and definition of the contents of CNC Signal file for the different pulse generating hardware devices listed above are different from each other. The signals file is machine dependent because, for example, the pin assignments, the registers and access APIs are different.

Details of CNC device driver implementations in our research using the above devices as signal generators are provided in the section on related research work.

1.6.7 CNC Motor Control Driver hardware

In industrial practice, most software controlled electric motors comes as a pair, consisting of the motor-control driver and the actual electrical motor itself. For example, for a CNC servo system, it consists of a separate servo-driver and a servo-motor hardware pair. In some systems, the motor-control driver is built as a part attached to the electric motor.

The motor-control driver is a hardware device with built-in electronics that commands and controls the actual running of the electric motor. This hardware, on one side is wired to the CNC driver device like the parallel port on the computer, and on the other side is wired to the CNC electric motor. In our CNC Research machine, this motor-control driver is the Panasonic Minas A4 AC servo-driver hardware.

The motor-control driver hardware usually contains embedded software that performs control functions like PI, PID, and encoder feedback. The control parameters can be configured manually or by external software. The motor-control configuration parameters depend on the choice of driving schemes, for example, position control, velocity control, torque control, open-loop and closed-loop controls. Some motor-control hardware operate in open-loop control, for example, stepper-motor control without feedback, while others are used for close-loop control, like servo-motor control with feedback.

1.6.8 CNC Electric Motor hardware

There are many different types of electric motors used in industry, some are manual hardware controlled while others are computer software controlled. For software controlled motors, they come in a pair, the motor control-driver and the electric-motor pair.

In the CNC industry, an electric motor can be direct current (DC) or alternating current(AC) driven. Each type has its own advantages and disadvantages. Some motors are used for open-loop control while others are used for close-loop control.

1.6.9 CNC Control Loop design

The actual control loop design varies between different kinds of CNC machines. However, they share the same concepts and principles, that is, to run the system correctly obeying their designed parameters and limits.

The following are some control methods: open-loop control, closed-loop control, position control, velocity control, torque control, iterative control, adaptive control, compensation control, predictive control, artificial intelligence control, optimal control, security control, safety control, and so on. These control methods or strategies will be addressed in the section on literature review.

1.6.9.1 Control objectives

In general, the first objective of control is to ensure that the system performs its specified function. The second control objective is to ensure that any variations in performance must

stay within its prescribed and designed limits. A successful control design must satisfy these two objectives.

A control system consisting of interconnected components is designed to achieve a desired purpose. The standard software, engineering practice today, is to build systems with modular component architecture. In modular designs, new components can be added and existing components can be modified or removed without affecting the overall behaviour of the system.

1.6.9.2 Control loop operations

For example, the CNC software control loop operates by invoking and executing separate functional components according to some control algorithm. Depending on the design of the control algorithm, the invoked components may run sequentially, concurrently or in a parallel manner. (parallelly is acceptable in old English usage ... ha ha ha).

The control loop is considered the sole director that is tasked to orchestrate the interaction and interfacing among the different component functions within the system, and with the external environment. Thus, the construction of the control loop algorithm is the most important software component in the system. It is called a loop because it is supposed to run in-resident, stay continuously active, waiting to service any command request as it comes, and act to those events accordingly. The control loop is a service or daemon type of running program. If the control loop program dies, the entire system dies.

1.6.9.3 Control concepts

Control algorithms are designed using to the following concepts (PRMCC).

1. a plan - the step-by-step sequence of actions to follow
2. a review - the action of monitoring some parameters, act accordingly if the values are out of range
3. a measure - the action of taking measurement of some parameters, used for progress checking and decision making
4. a coordination - the action of timely communicating with other components inside and outside the system
5. a control action - the decision on what action to take given the current state of the system.

1.6.9.4 Programming principles in control

In software design for control algorithms, the following are a few programming principles that apply:

1. process-driven programming - this is a step-by-step execution of functions, pre-planned and laid out to achieve some defined objectives
2. interrupt-driven programming - this is a request or notification event to the control director that an event occurred which required action

3. target-driven programming - this is a specification contract that the software must achieve the target by whatever means necessary

4. optimal-control programming - this is a specification contract such that software parameters stay within certain agreed limits by whatever means necessary

1.6.9.5 Issues in control loop algorithm

For the CNC control loop algorithm, all of the above mentioned concepts and principles apply. The design of the algorithm is proprietary. No sensible manufacturer will reveal this secret. In this research, we will build our own CNC control loop algorithm.

For the CNC control loop in particular, for process-control the control algorithm must handle machine services, like start and stop the motors, monitor tool locations, monitor path tracking errors, monitor velocity limits, etc, in order to achieve process targets and act accordingly.

For safety-control, the control algorithm must monitor temperature limits, friction limits, heating limits, vibrations, limits, etc, in order to service interrupt events and act accordingly.

For optimal control, the control algorithm must act by invoking and executing appropriate functions accordingly in order to achieve the optimal value (target maximum or minimum) of some dependent variable based on current values of independent variables or parameters.

Essentially, the control actions in the CNC control loop program work in a similar manner to a human being who has to make simultaneous control decisions and acts accordingly when faced with many things to do at one time. The control sensors for a human being is akin to the monitored parameters in the CNC machine.

The CNC control loop simultaneous multiple objectives are, as examples: to minimize contour error in tool path tracking, to ensure tool motion smoothness, to ensure the machining job is completed within reasonable time, to ensure minimum vibrations, to ensure heating is within tolerance, and so on, but not to forget that the machining job must be executed safely.

As a consequence, the design of the CNC control loop is not an easy task. It is one of the challenges that we will undertake in this research project. We will discuss more on this in the section on literature review.

1.6.10 Commercial CNC Control Software

Based on the best CNC control software survey in 2017 published on the internet, [12], the following are some results for large industrial CNC machines, and small and medium shop-hobbyist type CNC systems.

1.6.10.1 High end CNC machines

For large and commercial CNC machines, the CNC control software are built special-purpose for the respective machines. Some famous brand names are as follows:

1. Fanuc, [13].
2. Haas, [14].
3. Mazak, [15].
4. Siemens, [16].
5. Centroid, [17].
6. Heidenhain, [18].
7. Mitsubishi, [19].
8. Allen-Bradley [20].

1.6.10.2 Low end CNC Machines

For small and medium size commercial CNC machines, the CNC control software are similarly built special-purpose for the respective machines. Some famous brand names are as follows:

1. Mach CNC, [21].
2. PathPilot/LinuxCNC, [22].
3. Probotix CNC, [23].
4. Planet CNC, [24].
5. Eding CNC, [25]
6. UCCNC Motion Control. [26].

1.6.10.3 Survey results

The survey concluded that FANUC and HAAS are way ahead of everyone else at the high end market, while Mach3 and PathPilot-LinuxCNC rule the low end market. In this proposed research, we will be dealing only with LinuxCNC because it is cost-free but PathPilot is not.

1.7 Research Motivations

Commercial interest

Recall that CNC interpolation is the task that generates the actual reference commands that drives the CNC tool along the different axis-of-motions in the machine, such that, it accurately follows the desired machining path, in a timely and coordinated manner.

On this fact, it can be said that interpolation is both the "brain and heart" of the CNC machine. The brain refers to the logic (program or algorithm) that gets the CNC running correctly and accurately, while the heart refers to the continuous, timely and coordinated supply of just the right power (pumping blood, oxygen and nutrients) to the CNC machine. Many people can copy, duplicate or reverse-engineer the physical aspects of the CNC machine because they are visible and physical, whereas the CNC interpolation software is intangible, and not easy to copy if properly protected.

It was said that you can buy the machine, and the completed compiled software, but you cannot buy the source code that shows the internals on how the software was developed, so that you can learn from it. You need to develop one on your own and then learn from it.

Ownership and accomplishment

We do not have our own CNC interpolator because existing interpolators are proprietary, meaning, owned by the CNC manufacturers and developed for their own unique commercial machines. It is plain common sense that manufacturers will not share or reveal their commercial secrets, except possibly marketing and generally well-known information.

We experienced real situations where suppliers interrogate us, the seller is suspicious of us, for example, regarding the purpose of our purchase when comes to high intellectual property products. Suppliers want to ensure that their products do not land into the wrong hands, particularly potential future competitors. They want to sell only to users but not knowledgeable developers. That is the game. The only way out is to always stay ahead, or always be the faster man than the fastest man.

Therefore, it is of utmost interest for us to develop our own CNC interpolator. We know that in Malaysia, statistics show that more than 80 % of commercial CNC machines and the like in use are imported.

Enterpreneurship

We are interested and excited to manufacture our own commercial CNC machine in the near future, getting into the competitive CNC machining market and initially targeting at specific segments in our local market. In a borderless world today, we can even go to internet marketing, as many small and medium enterprises do today. We are excited means happy in the heart, for example, to study, explore, learn, undertake, execute and discover new things.

Our CNC product, interpolator plus machine, must be run effectively and efficiently. We must be effective means it must take effect, for example, to do exactly and correctly what it is supposed to do, that is according to its specifications. We must be efficient means be minimal in computation time and resources, for example, executes fast, speedy, compact and does not consume large resources.

Technical challenges

It is challenging technically, for example, to study the look-ahead control and feedrate filtering issues in CNC machining and come up with our way of addressing those issues. Our success in providing alternative solutions to problems will arouse a personal sense of accomplishment, satisfaction combined with a feeling of worthiness, one who is useful and can contribute in some small way for the benefit of mankind.

Future possibilities

With the exception of *non-computable functions* in computing, software programming and control opens unlimited possibilities in what software technologies can do and accomplish.

The prospects extend from directing sequential step-by-step commands, parallel executions, sorting, search, scheduling, exotic software concepts like futures and asynchronous executions, communication channels, and so on, are all recent ideas coming from the human mind to automated logical reasoning, software self-healing, fuzzy reasoning, self-learning, and so on, conducted within and by the software itself.

It is not easy to think of doing things in parallel, executing many things in overlapping time, not necessarily concurrent, but that is reality of nature. It will be extremely satisfying, not only being able to run things in parallel, but doing so in true realtime elegantly, while meeting both the start and end design deadlines. The thoughts to come up with software solutions addressing those issues are truly exhilarating.

1.8 Scope of Research

The scope of work proposed for this research study are as follows:

1. Start from the provision of G-Codes, specifically RS274D NGC standard G-Code.
2. Implement reference-pulse CNC interpolation for computational efficiency.
3. Conduct look-ahead and feedrate error compensation in G-Code interpolation.
4. Execute realtime, parallel, online/offline computations for the CNC control loop.
5. Compare appropriate parallel execution methods for 2D/3D G-Code interpolation
6. Address designs for extension to 3-axis and 5-axis interpolation implementations.

1.9 Proposed Research Title

The proposed research title shall be "A realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation."

1.10 Expected knowledge contributions

The expected knowledge contributions for this research study are as follows:

1. an implementation of a simple, practical and achievable strategy in CNC interpolation
2. a technique that utilizes both realtime and parallel execution in CNC machining
3. a contribution in innovative ideas for an efficient design of the CNC interpolator

1.11 Summary on Introduction

In this introductory chapter on our proposed research project, we began with a short overview of CNC, followed by the extensive adoption of software techniques in the control of CNC machines, particularly, contour tracking control executed in the CNC interpolation loop.

We briefly described recent designs in CNC machining environments, covering new cutting tools and technologies, and innovative software algorithms that address issues of path motion smoothness, machining speed control, and path error reduction.

The typical hardware and software requirements for a CNC machine were covered. A typical functional process flow for CNC machine operation was described. We illustrated what realtime, parallel and concurrent execution concepts mean to the common person compared to the correct understanding in software terminology through clear examples.

We briefly explained the functions of various software components in a typical CNC system, the G-code interpreter, the G-Code interpolator, the motion controller, the error controller, the signal driver, service controller, the human-machine controller and the overall CNC controller loop.

The interface and boundary point between software and hardware in CNC machine was specially defined, by differentiating the terms signal driver software against signal driver hardware. The signal driver hardware is generally referred to as electrical pulse generating devices.

Some signal driver hardware interface boards that have been used in our previous research projects were listed. In addition, the issues that need to be addressed by each of the software components to work with those signal driver devices were also discussed.

On the hardware side of the CNC machine, we described the industry practice of a compatible set of motor-driver and electric-motor pair, such that there is no mismatch of performance, for example, pulse frequency handling. In standard practice, the software control loop of the CNC machine only interacts with the motor-driver hardware, not directly to the electric-motor. We shared survey results on commercial companies with leading products in the CNC machining market, and identified the major brands that offer high end and low end CNC machines.

Finally, we described our motivations for conducting research on CNC machines, our proposed research topic, our proposed the scope of research and the expected contributions of the work.

In summary, the focus of our research is CNC interpolation. We restate, that CNC interpolation is about the task of generating and sending signal pulses to the respective electric motors at the machine's axis-of-motions, in a coordinated and timely manner, such that the machine tool accurately tracks the desired contour path in the move from the current position to the next position. To achieve this task, the interpolator must generate the right number of signal pulses that achieves the desired distance for the move, and the correct pulse feedrate (frequency) that achieves the desired velocity for the move.

2 Literature Survey

Semi-systematic review with a table of comparisons. In the appendix. Only summary here.

2.1 Reviews - CNC State of the Art

[27] CNC Algorithms for Precision Machining: State of the Art Review

As geometry of machined parts becomes complex the demands for more precise and faster machining using advanced computerized numerical control (CNC) are increased. Especially, recently improved computing power of CNC enables the implementation of the complicated control algorithms. Consequently a variety of intelligent control algorithms have been studied and implemented in CNC. This paper reviews the recent progress of control technologies for precision machining using CNC in the area of interpolation, contour control and compensation.

In terms of interpolation several corner blending methods and parametric curves are introduced and the characteristics of each method are discussed. Regarding contour control algorithms recently developed multi-axis contour control methods are reviewed.

[28] Recent development in CNC machining of freeform surfaces: A state-of-the-art review

[29] Interpolator for a CNC System
[30] Design of Computer Control for Manufacturing Systems

[28] - Freeform surfaces, also called sculptured surfaces, have been widely used in various engineering applications. Freeform surfaces are primarily manufactured by CNC machining, especially 5-axis CNC machining. Various methodologies and computer tools have been developed in the past to improve efficiency and quality of freeform surface machining. This paper aims at providing a state-of-the-art review on recent research development in CNC machining of freeform surfaces. This review primarily focuses on three aspects in freeform surface machining: tool path generation, tool orientation identification, and tool geometry selection. For each aspect, first concepts, requirements and fundamental research methods are briefly introduced. The major research methodologies developed in the past decade in each aspect are presented with details. Problems and future research directions are also discussed.

2.2 Overview of current issues in CNC

TO DO

2.3 G-Codes Standards

2.3.1 NGC RS274D G-Codes

TO DO -
G-code (also RS-274 NGC), which has many variants, is the common name for the most widely used numerical control (NC) programming language. It is used mainly in computer-aided manufacturing to control automated machine tools.
ISO 6983-1:2009 specifies requirements and makes recommendations for a data format for positioning, line motion and contouring control systems used in the numerical control of machines. ISO 6983-1:2009 helps the co-ordination of system design in order to minimize the variety of program manuscripts required, to promote uniformity of programming techniques, and to foster interchangeability of input programs between numerically controlled machines of the same classification by type, process, function, size and accuracy. It is intended that simple numerically controlled machines be programmed using a simple format, which is systematically extensible for more complex machines.

2.3.2 NURBS G-Codes

Low end CAD/CAM systems do not generate NURBS G-Code.

Non-Uniform Rational B-Splines NURBS have been used by CAD systems for some time. That is why it seems so natural that CNCs should be able to employ tool paths that are also defined in terms of NURBS. However, most CNCs today instead require contoured tool paths to be defined using straight lines, or chords.

And this long-practiced approach can lead to inefficiencies familiar to almost any die or mold shop. Using chords to define complex geometries accurately results in large, data-dense program files that historically have been difficult to manage and slow to execute. The development of NURBS-interpolating CNCs promised programs that could define the same complex geometries with fewer blocks of code, and thus could provide some relief for the data-flow bottlenecks.

But something happened along the way. CNCs became more powerful, and powerful CNCs became less costly. Compared to the best controls available only a short time ago, many new controls today offer superior networking capability, cheap memory and faster processing speed. These improvements mean enormous program files for complex workpieces are easier to deliver to the CNC, where they can be stored entirely instead of drip-fed. And once there, the programs can be executed more efficiently. Processing rates on the order of 1,000 blocks per second, combined with CNC look-ahead features to smooth out inertial effects, can let a new control today execute even a long series of very tiny chords fast enough to keep a machine moving accurately at a high feed rate.

Using nurbs interpolation requires not just a CNC capable of it, but also a CAM system able to output nurbs tool paths. And these CAM systems can use a variety of approximations to get from one set of nurbs to the other. Here are just two reasons why:

Surfaces are not tool paths. Mr. Arnone states it this way: "When a plane is intersected with a nurbs surface to create a tool path, a nurbs curve does not result."

In other words, the software is still approximating to get to the tool path. And like any approximation, this one is governed by an accuracy band, comparable to chordal tolerance.

A straight line may be the route between two curves. Some CAM systems translate the CAD geometry into nurbs toolpaths by generating a series of chords first, then translating the chord tool paths into nurbs. For CAM systems that use this approach, there literally is a chordal tolerance at work.

Finally, it's worth noting that at the level where the cutting tool hits the workpiece, the movement is still in straight lines. The CAD model may be represented by nurbs, the CNC may read tool paths in terms of nurbs, but when the CNC communicates movement commands to the processor controlling a given axis, it does this by specifying a target point. That is, a target toward which the axis advances in a straight line.

========== Optimized NURBS Based G-Code Part Program for High-Speed CNC Machining

August 2014 DOI: 10.1115/DETC2014-34884

Conference: ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference

2.3.3 STEP-NC G-Codes

[31] STEP-Compliant CAD/CNC Systems for Feature-Oriented Machining

[32] - In the projection toward the development of next generation CNC system, the major problem of current International Standards Organization (ISO) data interface model (ISO 6983) limitations and commercial CNC unit vendor specifications dependency were found in the CNC machines and systems. Later on, an ISO standard known as Standard for The Exchange of Product Data (STEP) or ISO 10303 was introduced to provide remedy for the problems of current data interface model limitations in Computer Aided Design (CAD)/Computer Aided Manufacturing (CAM) systems. After that successful implementation, the standard further extended to implement the STEP features on the CNC, for that a new standard known as STEP-Numeric Control (NC) or ISO 14649 was introduced. This standard has the abilities to achieve the aims of modern CNC systems. However, for enabling these facilities into the CNC machine, the machines need to be independent from any of the vendor specifications. Open Architecture Control (OAC) technology provides a more open environment to the CNC machine. In the race towards the development of next generation CNC, the combination of the STEP-NC and OAC technology became the hot topic of the research. However, in this work both ISO data interface model interpretation, its verification and execution has been highlighted with the introduction of the new virtual component technology based techniques. The system was composed of ISO data interpretation, 3D simulation and machine motion control modules. The system was also tested experimentally and found to be very satisfactory.
The development of STEP started in 1984 as a successor of IGES, but due to the complexity of the project, the initial standard was only published in 1994. Initial Graphics Exchange Specification (IGES) is a neutral file format designed to transfer 2D and 3D drawing data between dissimilar CAD systems. The IGES standard defines two file formats: fixed-length ASCII, which stores information in 80-character records, and compressed ASCII.

2.4 Reference-Pulse interpolation

In the Reference-Pulse interpolation method, reference signals from the computer are transmitted as a sequence of reference pulses, basically a sequence of external interrupts sent to the control loop of the CNC machine drive. In this interpolation method, the computer

transmits a sequence of reference pulses to each axis-of-motion, where each pulse produces one BLU (Basic Length Unit) of movement.

Each axis-of motion is controlled by two pulsed lines, one for clockwise rotation (CW) and the other for counter-clockwise (CCW) rotation. The accumulated number of pulses for each axis represents the location of the tool on that axis. The pulse frequency, generated by varying the time spacing between pulses, is proportional to the velocity of the tool along the axis. This frequency represents the axis feedrate.

The reference-pulse method can be used for actuating stepper motors in an open-loop system or sent as reference signals to servo motors in a closed-loop control system. For example, a pulse train of varying frequency is output to the servo control module. For each pulse, the servo system for an axis causes an incremental movement along the axis.

All reference-pulse interpolations are iterative and is usually controlled by an adjustable interrupt clock. A single iteration of the routine is executed at each interrupt, and this produces the output pulse. For the computation, the maximum feedrate (feed pulses) generated is limited by the maximum attainable computer interrupt rate. The interrupt rate in turn depends on the computation time of the algorithm.

The faster the computation, the higher will be the feedrate. The number of iterations required to move the CNC machine to a certain distance on a contour path segment for each axis-of-motion is also calculated by the interpolation algorithm.

[33] CNC Machining Technology [34] CNC Controllers and Programming Techniques
[3] Reference-Pulse circular interpolators for CNC Systems
[2] - A Reference-Pulse Generator for Motion Control System

2.5 Reference-Word interpolation

In the Reference-Word (Sampled-Data) interpolation method, the interpolator runs in an online iterative mode and generate words (sampled data) which are supplied as references to the running CNC control loops. It is a point-to-point control method used to move to the desired position between two successive interpolated points along each axis. The coordinate points to reach from the present position are computed for each iteration and the calculated data is transmitted to each axis in the form of sampled data.

For example in 2D motion, this calculated sample data consists of the next x and y coordinates and their velocities along each of the x and y axes. In a 2D contouring system the tool is cutting while the machine axes are moving. The contour path of the part is determined by the ratio between the velocities along the two axes. During a closed loop execution, the interpolation subroutines continuously provide destinations and velocity set points to the stepper or servo drive systems.

In contrast, in reference-word interpolation the maximum feedrate (velocity) is not limited by the execution speed of the processor. It is the stepper or servo system dynamics that limits the speed rather than the interpolator loop execution time.

[5] Design Parameters for Sampled-Data Drives for CNC Machine Tools
[4] Reference-Word circular interpolators for CNC systems
[35] Research and Development of Sampled-data Interpolation Algorithm Software in CNC System Based on the Visual C++

2.6 NURBS CNC Interpolations

Some of the newer CNCs have NURBs interpolation available, but most controls won't have it unless you order it as an option. NURBs interpolation would let you write a G-code program with the NURBs data, which can make a single pass on a complex surface with just one block of G-code data. It reduces the size of the G-code file, and it also increases the maximum feedrate you can achieve.

FANUC USA uses G06.1 G-Code.

========== G05 P10000 High-precision contour control (HPCC) M Uses a deep look-ahead buffer and simulation processing to provide better axis movement acceleration and deceleration during contour milling

G05.1 Q1. AI Advanced Preview Control M Uses a deep look-ahead buffer and simulation processing to provide better axis movement acceleration and deceleration during contour milling

G06.1 Non-uniform rational B-spline (NURBS) Machining M Activates Non-Uniform Rational B Spline for complex curve and waveform machining (this code is confirmed in Mazatrol 640M ISO Programming)

========== ed (I just got the DXF), but you might be able to apply that reasoning to surfacing. And if you can create a CAD model, CAM it from there.

Real-time NURBS curve interpolator for 5-Axis CNC machining
February 2018
DOI: 10.7736/KSPE.2018.35.2.129
Sungchul Jee
[36] - The first appearance of NURBS is in K. Vesprille's PhD thesis [151], where he makes heavy use of homogeneous coordinates, an approach that goes back to S. Coons and R. Forrest. Riesenfeld [130] realized early that using homogenous coordinates mandates working with projective geometry. This book follows these ideas and attempts to base the theory of NURBS firmly in projective geometry. This way, we gain more insight into theoretical issues as well as practical ones. After a general outline of projective geometry, we introduce conics through a classical projective definition. Using the concept of cross ratios, we arrive at the de Casteljau algorithm for conics. Then we cover areas such as rational Bezier curves curves, NURBS curves and surfaces, triangular patches, Gregory patches, and more. The book closes with some practical examples, including a discussion of the IGES NURBS data specifications.
[37] - Parametric interpolator versus linear interpolator for precision surface machining
[38] - Development of a NURBS Curve Interpolator with Look-ahead Control and Feedrate Filtering for CNC System
[39] A real-time scheme of cubic parametric curve interpolations for CNC systems
[40] Feedrate fluctuation compensating NURBS interpolator for CNC machining
[41] - Accuracy and Error Compensation of CNC Machining Systems
[42] - Computer Numerical Controlled System with NURBS Interpolator
[43] - Development of Real-time Look-Ahead Algorithm for NURBS Interpolator with Consideration of Servo Dynamics
[44] Development of a dynamics-based NURBS interpolator with real-time look-ahead algorithm
The International Journal of Advanced Manufacturing Technology
April 2008, Volume 36, Issue 9 –10, pp 927 – 935 Cite as The design of a NURBS pre-interpolator for five-axis machining
Authors Authors and affiliations
Wei LiEmail authorYadong LiuKazuo YamazakiMakoto FujisimaMasahiko Mori
==========
https://www.wittystore.com/g-code-commands
========== G06.1
Non-uniform rational B-spline(NURBS) Machining M
Activates Non-Uniform Rational B Spline for complex curve and waveform machining (this code is confirmed in Mazatrol 640M ISO Programming)
========== List of G-Codes commonly found on FANUC and similarly designed controls Published In: Tech Explained Hits: 800
G-codes, also called preparatory codes, are any word in a CNC program that begins with the letter G. Generally it is a code telling the machine tool what type of action to perform, such as:
Rapid movement (transport the tool as quickly as possible in between cuts) Controlled feed in a straight line or arc Series of controlled feed movements that would result in a hole being bored, a workpiece cut (routed) to a specific dimension, or a profile (contour) shape added to the edge of a workpiece Set tool information such as offset Switch coordinate systems
========== Warning:
G5.2, G5.3 is experimental and not fully tested.
G5.2 is for opening the data block defining a NURBS and G5.3 for closing the data block. In the lines between these two codes the curve control points are defined with both their related weights (P) and the parameter (L) which determines the order of the

2.7 Contouring control and Error Compensation

[45] The Error Analysis of Surface Machining on the CNC Machine Tools
[46] Accuracy improvement of three-axis CNC machining centers by quasi-static error compensation
[47] Adaptive Error Control Algorithm for High Speed Two-Axis CNC Contouring
[48] Reduction of the Contouring Error in High-Feed-Speed Machining by Real-Time Tracking-Error Compensation
[49] Integrated Geometric Error Compensation of Machining Processes on CNC Machine Tool
[50] Embedded Iterative Learning Contouring Controller Based on Precise Estimation of Contour Error for CNC Machine Tools
[51] Tracking error reduction in CNC machining by reshaping the kinematic trajectory
[52] Research on Error Compensation Technology for CNC Machining
[53] Contour error and control algorithm in CNC machining tool
[54] Dynamic evaluation of spatial CNC contouring accuracy
[41] Accuracy and Error Compensation of CNC Machining Systems
[51] Tracking error reduction in CNC machining by reshaping the kinematic trajectory
[49] Integrated Geometric Error Compensation of Machining Processes on CNC Machine Tool [55] Develop on feed-forward real time compensation control system for movement error in CNC machining
[56] CNC machining accuracy enhancement by tool path compensation method

2.8 Look-ahead control

Conventional CNC systems only provide line (G01) and circular (G02, G03) interpolations, so the CAD/CAM systems have to divide the curves into a large number of small linear and circular arc segments while maintaining the set of contour error limits.

Due to the short segments, most of the time spent in interpolation is on stop-start motions, typically accelerating, decelerating, or pausing between instructions. The frequent starts and stops for very small linear motions for every segment causes jerks during interpolation, especially at the junctions of the segments. This results in discontinuous feedrate (velocity) profiles, jerky and unsmooth overall machining process.

The feedrate look-ahead control algorithm is introduced to deal with sudden changes of feedrate [13-15,23-25]. Look-ahead control is a pre-processing task of the path contour before the real machining. The main goal is to achieve a smooth feedrate profile by looking at deceleration regions in the path.

The feedrate look-ahead control algorithm has three tasks: (1) to detect the deceleration point timely before reaching a linking corner or terminal point, (2) to smoothen and re-plan the feedrate in the deceleration region, and (3) to make the deceleration (braking) meet the capabilities of the physical machine like electrical motors. Braking is very important especially for high speed machining because exceeding the braking capability of the machine can cause unwanted machine vibrations.

The look-ahead interpolation algorithm is executed such that the processing error lies

within a limited range. The look-ahead control strategy can be performed as either an off-line prediction, or as an on-line computing operation [24].

Many methods have been proposed for look-ahead control strategies. Some examples are:

1. Generate a recursive trajectory to estimate and determine the deceleration stage according to the distance left to travel on the segment.
2. Apply a hybrid digital convolution technique through a look-ahead scheme that smoothed the feedrate between the joint of two curve segments.
3. Implement an integrated look-ahead dynamics-based algorithm by considering contour and servo errors simultaneously.
4. Introduce a look-ahead trajectory generation to determine the acceleration stage according to the fast estimated arc length and the reverse interpolation of each curve segment.
5. Create a real-time look-ahead scheme that combines of path-smoothing, bi-directional scanning and feedrate scheduling.

[43] - Development of Real-time Look-Ahead Algorithm for NURBS Interpolator with Consideration of Servo Dynamics
[44] Development of a dynamics-based NURBS interpolator with real-time look-ahead algorithm
[38] - Development of a NURBS Curve Interpolator with Look-ahead Control and Feedrate Filtering for CNC System
[57] - This paper presents a perfect tracking method by combining iterative learning control (ILC) with disturbance observer (DOB) for CNC machine tools that perform the same tasks repeatedly. Although CNC systems have many nonlinear factors, they can be easily solved by ILC instead of complicated modeling. Also, for repeated disturbance, ILC performs very well and for non-repeated disturbance, DOB works much better. Besides, in order to facilitate the application, this paper proposes a variable gain algorithm with conditional compensation. Simulative results demonstrate the proposed learning scheme can improve machining accuracy when the CNC machine tools perform repetitive machining tasks.

2.9 Feedrate Filtering

The function of feedrate filtering is to obtain a continuous acceleration profile and avoid sudden changes in acceleration/deceleration during machining. The function of the adaptive feedrate adjustment is to adjust the feedrate, confine chord error and make acceleration meet the capabilities of the machine.

To obtain a continuous feedrate and acceleration during the whole interpolation process, one method is feedrate filtering [13]. This method adaptively adjusts the feedrate according to the different curvatures to meet the demand of the CNC machining accuracy.

Another popular method to eliminate jerky motion is spline interpolation [13-18]. In spline interpolation, the tool moves smoothly from one point to another. For example,

in Non-Uniform Rational B-Spline (NURBS) interpolation, the curve is parametrized and approximated by a set of interpolated data points, consisting of control points, knot points and weights.

NURBS parametric interpolation is often preferred over conventional linear and circular interpolator for its merits in terms of the model representation, feedrate smoothness and application range. The parametric curve is smooth and continuous so feedrate continuity is achieved effectively since the sharp junctions between curve segments are avoided.

Other methods include constant feedrate interpolation algorithms based on first-order, second-order Taylor expansions, and adaptive interpolation algorithm with confined chord error [10,12,22]. A feedrate fluctuation compensating algorithm that predicts and compensates feedrate fluctuation in real-time, has also been considered [18].

[55] Develop on feed-forward real time compensation control system for movement error in CNC machining
[40] Feedrate fluctuation compensating NURBS interpolator for CNC machining
[38] - Development of a NURBS Curve Interpolator with Look-ahead Control and Feedrate Filtering for CNC System

2.10 Realtime and Parallel Computing

Parallel
[58] A parallel CNC system architecture based on Symmetric Multi-processor
Realtime
[55] Develop on feed-forward real time compensation control system for movement error in CNC machining
[39] A real-time scheme of cubic parametric curve interpolations for CNC systems
[43] - Development of Real-time Look-Ahead Algorithm for NURBS Interpolator with Consideration of Servo Dynamics
[6] C/C++ and Python for Linux Realtime Parallel Port Software Driver
[44] Development of a dynamics-based NURBS interpolator with real-time look-ahead algorithm
[59] Real-time curve interpolators
[39] A real-time scheme of cubic parametric curve interpolations for CNC systems
[48] Reduction of the Contouring Error in High-Feed-Speed Machining by Real-Time Tracking-Error Compensation

Various researchers have implemented realtime CNC software systems on Windows platform running its RTX realtime operating system [26]. Free and open source CNC software are also available for the Linux platform through RT-Linux [27] realtime operating system (RTOS) or Real-Time Application Interface (RTAI) extensions.

The LinuxCNC [31] system is an open source RTAI-based CNC machine controller. It can drive milling machines, lathes, 3d printers, laser cutters, plasma cutters, robot arms, hexapods, and more. The controller for LinuxCNC is named EMC2 (Enhanced Machine

Controller version 2).

To take advantage of multi-cores and multi-processors available on personal computers, several implementations for high-speed CNC interpolation scheme using parallel computing have also been conducted.

In one method, the CNC interpolation method was divided into two tasks, the rough task executing in the personal computer and the fine task in the I/O card. During the interpolation procedure, double data buffers were constructed to exchange interpolation data between the two tasks in parallel [28].

In another method, parallelization of the CNC algorithm was implemented using multi-threading based on the Pthread software programming library [29]. Another variation is the method where a parametric curve realtime CNC interpolator processing was executed in parallel using the C++ Open Multi-Processing (OpenMP) Application Program Interface (API) library [30].

2.11 AI Approaches in CNC Machining

[50] Embedded Iterative Learning Contouring Controller Based on Precise Estimation of Contour Error for CNC Machine Tools
[57] The design of iterative learning control scheme for CNC machine tools
[60] CNC Interpolators: Algorithms and Analysis
[29] Interpolator for a CNC System
[45] The Error Analysis of Surface Machining on the CNC Machine Tools
[47] Adaptive Error Control Algorithm for High Speed Two-Axis CNC Contouring

2.12 Commercial versus Open CNC systems

[32] Frame Work of LV-UTHM: AN ISO 14649 Based Open Control System for CNC Milling Machine

2.13 Embedded devices in CNC systems

[9] Using Raspberry Pi 3 Model B to drive a CNC System in Real Time
[6] C/C++ and Python for Linux Realtime Parallel Port Software Driver
[7] Using Arduino Due to drive a CNC System
[11] Using the Nexys-3 Spartan-6 FPGA board to develop a closed-loop feedback CNC system
[61] A CNC machining system for education
[62] On the approach to CNC machining simulation improving

[50] Embedded Iterative Learning Contouring Controller Based on Precise Estimation of Contour Error for CNC Machine Tools

2.14 Summary on Literature Survey

3 Research Methodology

3.1 Research Objectives

The proposed research is to implement a realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation.

Interpolation is the task that generates the actual reference commands that drives the CNC tool along the different axis-of-motions in the machine, such that, it accurately follows the desired machining path, in a timely and coordinated manner.

3.1.1 Research scope

The scope of work proposed for this research study are as follows:

1. Start from the provision of G-Codes, specifically RS274D NGC standard G-Code.
2. Implement reference-pulse CNC interpolation for computational efficiency.
3. Conduct look-ahead and feedrate error compensation in G-Code interpolation.
4. Execute realtime, parallel, online/offline computations for the CNC control loop.
5. Compare appropriate parallel execution methods for 2D/3D G-Code interpolation
6. Address designs for extension to 3-axis and 5-axis interpolation implementations.

3.1.2 G-Codes Coverage

We will start from RS274D NGC G-Code files because the format is the base standard for G-Codes supported by all machines used in the CNC industry. Our research does not include SVG or STL files generated by CAD applications. Our research also excludes the generation of G-Codes from CAM processing of SVG or STL input files. Using NURBS interpolation requires a CNC machine capable of handling NURBS G-Code generated tool paths. Since the NURBS G-Code format is proprietary and only used in the high end FANUC CNC machines, NURBS G-Code will not be in the scope of this research. It should be noted that NURBS G-Code files are not the same as NURBS interpolation method.

3.1.3 Reference-Pulse Interpolation

We will implement CNC Reference-Pulse Interpolation instead of Reference-Word interpolation because of computational efficiency. All reference-pulse interpolations are iterative and controlled by an adjustable interrupt clock. Various free and open source software libraries can take advantage of this fact and that makes integration practical. This integration method is also more streamlined with the reference-pulse method. Various implementation methods using these libraries can be explored for computational efficiency.

3.1.4 Look-ahead control

The feedrate look-ahead control algorithm is introduced to deal with sudden changes of feedrate pulses, essentially the move velocity. Look-ahead control is a pre-processing task of the path contour before real machining takes place. The main goal is to achieve a smooth feedrate (velocity) profile by looking ahead at deceleration regions in the given contour path. The idea of look ahead is to consider velocities to move toward future target points from the current machine position using the geometric path data already provided in the G-Code. That could be one-step forward, or two-step forward look-ahead methods. Using look-ahead control, the movement velocity can then be adjusted continuously to ensure smooth contouring movements, not jerky.

3.1.5 Feedrate compensation

Feedrate compensation is also about adjusting move velocities but with a focus on reducing contouring path errors, instead of maintaining movement smoothness. The two goals are conflicting, meaning, to reduce path errors we need to slow down the machine move velocity. Whereas, to speed up machine movements, we may increase path tracking errors. In this research, we will look at methods that strike a balance in minimizing contour errors, meaning, accurately following the desired machining path, and at the same time achieving a reasonable machining velocity. The term feedrate compensation or velocity compensation is used here to refer to velocity adjustments that have to be made. The target priority is placed on contour tracking accuracy compared to fast machining completion and velocity profile smoothness. Ultimately, there is no point to quickly finish cutting a part when the output machined part itself is not accurately machined.

3.1.6 Realtime and parallel execution

As discussed in Chapter 1, Introduction, our technical concept of realtime execution means every software task must be completed within its start and end timing deadlines, unlike the layman notion of realtime which means running in the current instance of time. In terms of parallel tasks execution, we will consider both software multi-threading and multi-processing methods. Our research strategy is to implement, wherever possible, both task-realtime and task-parallel executions simultaneously.

3.1.7 Parallel execution strategies

There are various parallel execution strategies that will be considered in CNC interpolation. The signals-and-slots (sigslot) mechanism is one execution strategy that will be used to allow different CNC software modules to inter-operate by calling each other's built-in functions. The sigslot method implements a type-safe, thread-safe signal/slot mechanism in software execution. The parallel executions for the signals-and-slots (signal/slot) method will be discussed further in the section on research implementation plan.

For example, in the CNC control software (CCS), the different software controller classes must be aware about each other in some detail. The centerpiece or crux of our methodology in the implementation of the CNC Control software is to design and execute the signal/slot algorithm that handles the different software classes through the CNC control loop.

The invocation of these classes must happen in a coordinated and timely manner, such that many tasks will be seen running in either serial, parallel or concurrent modes. This inter-operation, signal/slot, and control-loop strategy is the key to the success of our research on CNC interpolation. As discussed in Chapter 1, Introduction, the controller classes handled by the control loop comprise the following:

1. G-Code Interpreter class
2. G-Code Interpolator class
3. Signal Driver class
4. Motion Controller class
5. Error Controller class
6. Services Controller class
7. Human-Interface Controller class

3.1.8 Extension to 5-axis interpolation

The proposed research covers the 3-axis CNC Research machine, that we have available for development and experimentation on CNC control implementations. The CNC Research Machine and its details will be described in the next chapter, Chapter 4 on Related Research Work, Section 4.1 with a link here [4.1]. We will design the software control infrastructure to be ready for future inclusion and extension that cover 5-axis CNC machines.

3.2 Research Methodology

3.2.1 Software engineering perspective

Most of the work in CNC interpolation in published literature are conducted by control and instrument engineers, in which, the perspective is on the control and operational aspects of the CNC machine. We undertake this research project from the perspective of a software engineer, with a focus on exploring different software engineering technologies that can be applied to efficient CNC control operations.

3.2.2 Software engineering methods

Most of the work in CNC interpolation in published literature do not mention specific software programming techniques, data structures, and various tools, that are currently available for use by the software engineer. For CNC interpolation generally, the focus is direct to the mathematical aspects on geometry, path contours, and machine dynamics. Despite having common mention of realtime in CNC literature, the meaning is not in the true sense the correct interpretation of software defined realtime. This subject was discussed earlier in Chapter 1, on Introduction. For CNC machines in particular, with the readily available multi-cores, multi-processors, and networked computers today, there are less that a handful of CNC specific publications that even mention the word parallel for CNC interpolation. Thus, the two areas of realtime and parallel, are opportunities we will explore in this research.

For realtime software executions, we will implement task time recordings for true realtime using the RTAI (RealTime Application Interface) C/C++ library. For true parallel software executions, we will implement tasks for multi-threading, multi-processing, and parallel file HDF5 I/O operations using the OpenMPI (Open Message Passing Interface), OpenMP (Open Multi-Processing) and HDF5 (Heirarchical Data Format) C/C++ libraries. For offline CNC interpreter and CNC interpolator executions, we will implement the HDF5 high speed file I/O system. For CNC control loop operations, we will implement the Signal/Slot mechanism that allows programmed invocations and inter-operations of the various controller classes handled by the control loop.

3.2.3 Linux, open source and free software

Most of the work in CNC interpolation in published literature do not use Linux, open source and free software. As expected, Microsoft operating system (OS) seemed to be popular among control and instrument engineers. Except for embedded systems which is micro-controller (MCU) based, most motion-control hardware boards with micro-processors (MCU or CPU) for CNC use are Microsoft OS based. As we discussed in Chapter 1, Introduction, we avoided the Microsoft (MS) Windows platform because we need full and open access to the operating system and to the software components that control the devices attached to the operating system. With the Linux OS, we have full and open access to all software codes.

Our implementation for the CNC control loop and all its component controller classes shall be based primarily on the C/C++ software programming language. We will consider also the Rust programming language as the low-level systems language on par with C/C++. For scripting languages, we will consider Python and Julia to execute non-critical codes in the control loop. For complex mathematical computations, we will consider Octave-NURBS and Scilab-NURBS backend components that will be interfaced to the C/C++ primary control loop codes.

The software components, Rust, Python, Julia, Octave and Scilab are all open source and cost-free. In addition, all of the mentioned components can run in true parallel mode. For this research project, we have practically opted for total open access to all software components and total cost-free software.

3.2.4 Motion control devices

We will implement the Pico Universal PWM (Pulse Width Modulation) Servo Controller hardware board as our motion controller interface board to the 3-axis servo-controlled CNC Research machine. This is a LinuxCNC based interface board, that can handle the control of a 2-axis, 3-axis or 4-axis machine tool with PWM-driven servo amplifiers. It contains 4 PWM generators with variable PWM drive frequency (variable feedrate), and 4 digital encoder counters as feedback to follow the machine position. The image and specifications of the board can be seen at the link [C 3.1] and Fig. [1.5] in the appendix.

As comparison, we will implement the 28-Pin LIN (Local Interconnect Network) Demo Microchip Board for PIC16F/PIC18 MCUs as our own-developed interface board to drive the 3-axis servo-controlled CNC Research machine. The PIC Micro-controllers can be programmed from the Linux based MPLAB IDE software application provided by Microchip

Corporation. We have sufficient familiarity through previous successful experiences in programming PIC Microcontroller chips. The image and specifications of the board can be seen at the link [C 3.3] and Fig. [1.6] in the appendix.

In another variation, we will implement the Microchip Curiosity Development Board Demo for micro-controllers (MCU) as another own-developed interface board to drive the 3-axis servo-controlled CNC Research machine. Similarly, the MCUs can be programmed from the Linux based MPLAB IDE software application provided by Microchip Corporation. This Curiosity Demo Board allows for some user prototyping but not the full prototyping as provided in the 28-Pin-LIN Demo Board. The image and specifications of the board can be seen at the link [C 3.5] and Fig. [1.9] in the appendix.

Using our own developed CNC Control software, we will compare performance as well as limitations of all three board interface devices for driving the 3-axis servo-controlled CNC Research machine.

3.2.5 Research Validation

It is generally quoted that, *"The proof of the pudding is in the eating."* People say this to mean that something can only be judged to be good or bad after it has been tried or used.

For contour tracking error, we will compute the deviation from G-Code reference commands by the encoder feedback mechanism from our CNC Research machine servo-motor. This can be accomplished using our Pico Universal PWM Controller hardware board which comes equipped with 4-nos of digital encoder counters as feedback to follow the machine position (X, Y, Z).

For validation of our machining control technique, we will take manual micrometer measurements of our machine part based on 2D models of common objects, like a circular disk, a semi-circular disk, a square plate, a rectangular plate, a pentagon plate or a hexagon plate, a star and so on. These measurements will be compared with the G-Code exact coordinates drawn for those simple models. We will test our CNC Control Software on a real CNC Laser Cutting machine available at UMP for precision cutting.

For realtime and parallel tasks execution, we will capture (dump) in nanoseconds, timing records for the tasks, and save the data in a file, like HDF5 binary files. Parallel I/O HDF5 data formats are used for fast serialization and parallelization of large datasets. The files will later be analyzed to validate actual realtime and/or parallel tasks performance against its design.

3.3 Summary on Research Methodology

We will start our research from the point of being provided with the RS274D NGC formatted G-codes. Next, we implement CNC Reference-Pulse interpolation based on the provided G-codes. This is followed by either the one-step forward, or two-step forward look-ahead methods to determine machine move velocity adjustments for smooth overall machining.

The next step is to perform feedrate or velocity compensation, an activity that strikes a balance between achieving contour tracking accuracy against machine movement smoothness. In implementing our CNC control loop execution, our strategy is to implement, wherever possible, both task-realtime and task-parallel executions simultaneously. In the CNC control loop, the parallel executions using the signals-and-slots (signal/slot) method will allow different CNC software modules to inter-operate by calling each other's built-in functions, in both synchronous and asynchronous modes.

It must be noted that the centerpiece or crux of our methodology in the implementation of the CNC control software is to correctly design and execute the signal/slot algorithm that handles the different software classes through the CNC control loop. We shall also design the software control infrastructure to be ready for future extension to cover 5-axis CNC machines.

We undertake this research project from the perspective of a software engineer, with a focus on exploring different software engineering technologies that can be applied to efficient CNC control operations.

Realtime and parallel executions are two opportunities we will explore in this research on CNC interpolation. In addition, for this research project, we opt for total open access to all software components and total cost-free software by including Linux based, open source and only cost-free software resources. For the delivery of the CNC control software product, we will consider inclusion of software components, like Rust, Python, Julia, Octave and Scilab, in addition to the base C/C++ codes.

In this research project, we will implement three(3) hardware interface devices to drive our CNC Research machine, comprising of the Universal PWM (Pulse Width Modulation) Servo Controller board from Pico systems, the 28-Pin LIN (Local Interconnect Network) Demo Board for micro-controllers (MCU) from Microchip Corporation, and the Curiosity Development Board Demo for micro-controllers (MCU) also from Microchip Corporation. Our goal is to develop our own CNC Control software, and use this software to compare performance as well as limitations of all three board interface devices in driving the 3-axis servo-controlled CNC Research machine.

For research validation on our developed CNC Control software, it will be judged as being good or bad after it has been tried or used. We will perform computational assessments, as well as manual measurements of real parts produced to compare them against G-Code exact coordinates and dimensions drawn for the source of some simple 2D models.

In overview, the proposed research is to implement a realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation. The descriptions above summarizes the research methodology for this project.

4 Related Research Work

4.1 UMP CNC Research machine

The images of the UMP 3-axis CNC research machine for our previous work are provided in next three figures. It is an experimental CNC router-type, that instead of a tool cutter, uses a pen to create drawings on paper in the X-Y plane. The Z-axis motion is used to raise and lower the pen. As a consequence, circular arc (G02, G03 G-Code) moves are applicable to the X and Y axes only, while linear (G01 G-Code) moves are applicable to all three X, Y and Z axes.

The CNC Research machine is driven on each axis by the Panasonic Minas A4, alternating current (AC) servo system. The configuration for each axis consists of a servo-driver and a servo-motor pair. Each servo-motor is installed on a ball-screw shaft. Electrical TTL-type, 0 to 5 volts, digital pulses are used as input signals to drive the servo-driver. Since each motor is installed on a ball-screw shaft, the motor rotational motion gets converted to linear translational motion in the direction of each axis.

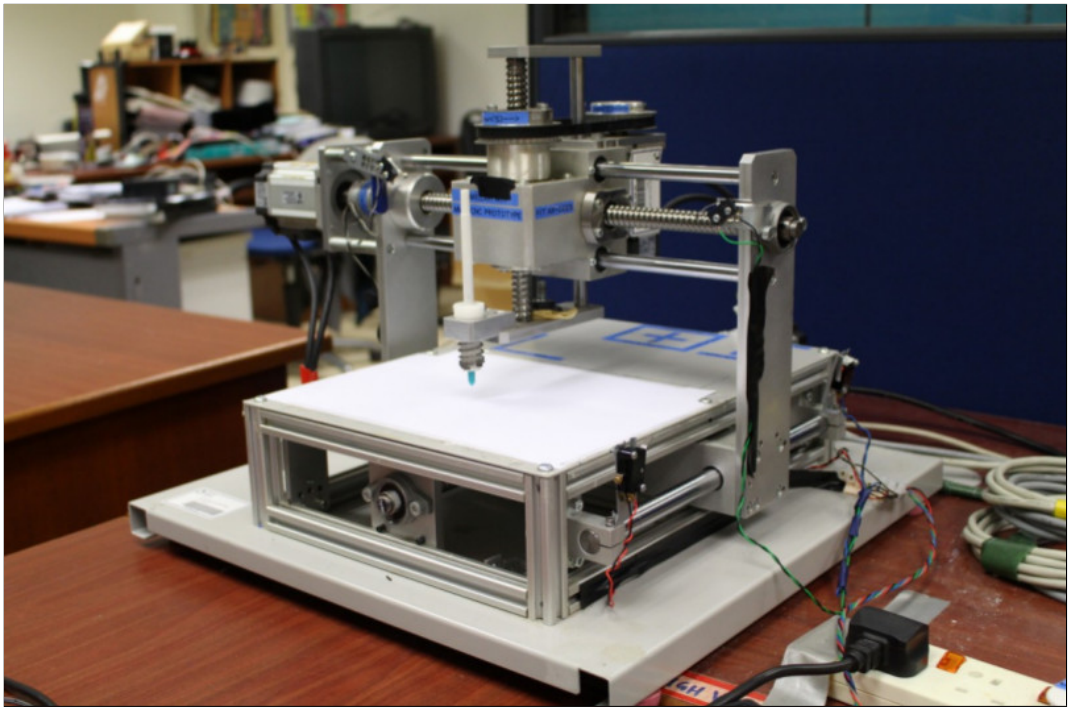


Figure 4.1: The UMP 3-axis CNC Research Machine

Electrical signal pulses sent to the servo-driver provide information like rotate clockwise (CW), rotate counter-clockwise(CCW), travel distance to rotate, speed to rotate, and so on. The actuation using electrical pulses makes the physical CNC machine instantaneously active.

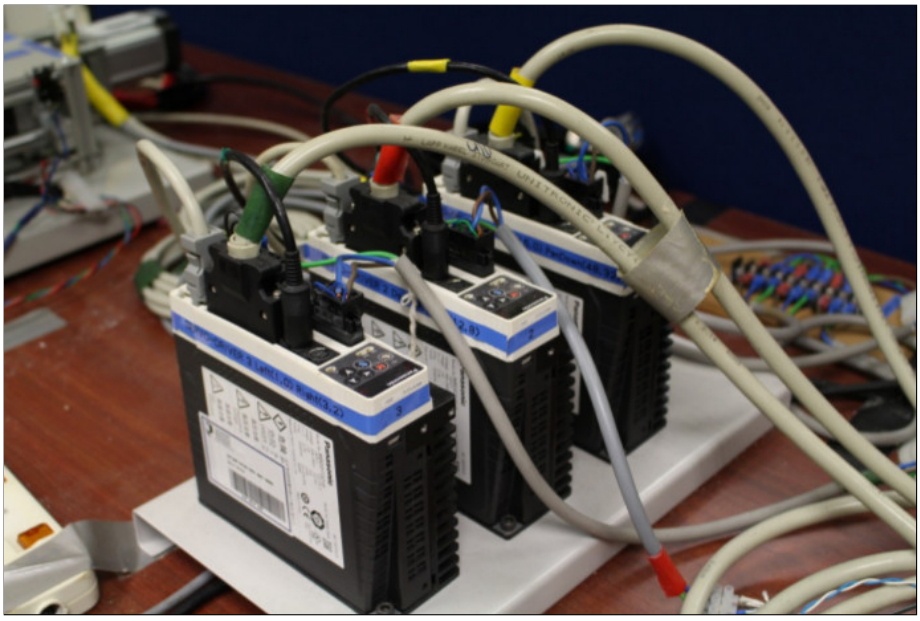


Figure 4.2: Servo-Drives for the 3-Axis CNC Research Machine

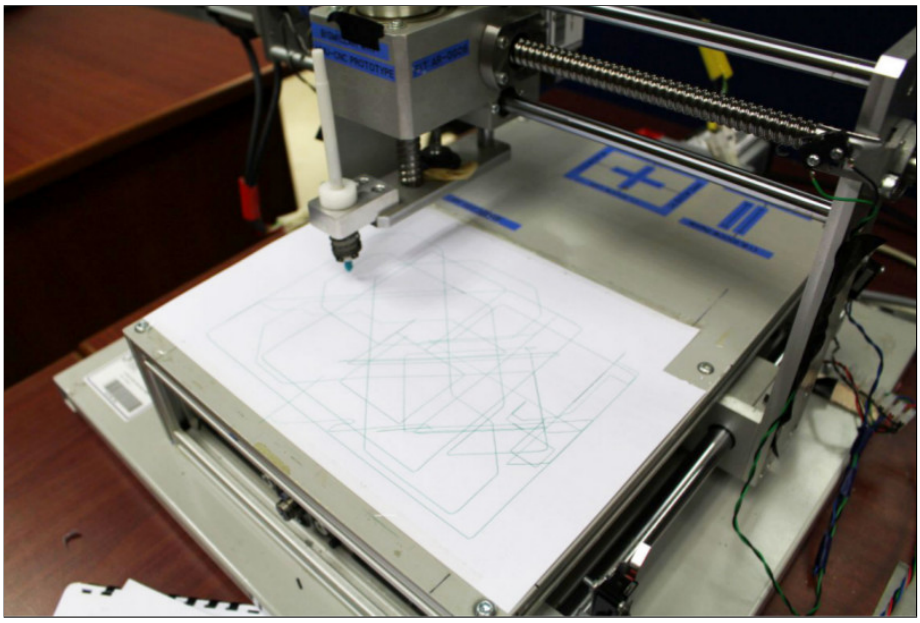


Figure 4.3: CNC Research Machine X-Y Movements

For our CNC research machine, we used a commercial, industry standard, Panasonic MINAS A4 AC servo-driver and servo-motor pair. We have three sets of this pair because the research machine is a 3-axis CNC servo driven machine.

Each Panasonic MINAS A4 servo-motor is a 24-volts, 750-watts, high torque, alternating current (AC) driven motor. It can be configured to operate in a single-phase or a three-phase, AC 60 Hz input current mode.

We can configure our CNC research machine to run in three control modes: position control, velocity control, and torque control. It can also be configured either in an open-loop or a closed-loop control scheme.

The CNC research machine was procured on a joint collaboration project between University Malaysia Pahang (UMP) and Multimedia University, Cyberjaya (MMU), in the year 2008, under the E-Science grant, Ministry of Science, Technology and Environment (MOSTE), Government of Malaysia.

4.2 Previous CNC research projects

We have successfully developed and executed C/C++ software codes using various devices to drive our CNC research machine. We have written codes that combine the following programming techniques: standard serial-running codes, realtime-running codes (RTAI) and parallel-running codes.

All of the pulse-generator devices that we have used on all of our research projects run C/C++ software codes on the open source Linux platform. On Linux, every software that we need is cost-free and open source. The C/C++ software compiler is cost-free. We even customize the operating system kernel to meet our needs. All of the interpolation computations we accomplished in our projects are of reference-pulse CNC interpolations.

We avoided the Microsoft (MS) Windows platform because we need full and open access to the operating system and to the software components that control the devices attached to the operating system. In addition, Microsoft Windows operating system is not free. Even, the C/C++ software compiler is not cost-free. In many cases, we cannot find suitable software applications on MS Windows that meets our project needs and that are cost-free. For low-level systems programming (our type of projects), the MS Windows is not a suitable platform for a cost-free development environment.

4.3 Previous software programming experiences

On programming experience, we covered realtime execution using RTAI kernel modules on the LinuxCNC system. In the Raspberry Pi 3 project, we managed to drive the CNC system using C++ multi-threading. The concept of RTAI interrupt-style programming architecture can be found in the appendix at [1.11].

Examples of realtime codes that we have used in our projects can be found in the appendix at [1.12], [1.13], and [1.14].

Examples of parallel C/C++-2011 codes using multi-threading can be found in the appendix at [1.15] and [1.16], while codes for parallel C/C++ MPI (Message Passing Interface) using multi-processing can be found in the appendix at [1.17] and [1.18].

For our research project, we will combine both realtime and parallel codes suitably to meet our CNC machine needs. We will also consider integrating codes or taking ideas from modern and efficient computer programming languages like Rust programming language, with an example multithreading code at [1.19] placed in the appendix. Rust is an upcoming low level systems language, a direct competitor to C/C++. Rust was designed from the

ground up with Safety, Security and Concurrency (SSC) built-into the Rust engine. This is unlike the C/C++ language or most languages, where these functionality come as separate linked software libraries. More will be discussed in the section on literature review.

Internally, inside the CNC control software we may use Python programming language or Julia programming language scripts, that have excellent processing and data handling capabilities. These languages support parallel programming internally not through external libraries. This researcher is quite well versed in the use of these computer programming languages. Example codes for Python multithreading and multiprocessing can be found in the appendix at [1.21] and [1.21]. And similarly, parallel codes for the Julia language are provided in the appendix at [1.20].

The CNC machine only needs electrical pulses of the right characteristics to run. The kind of devices that generate these electrical pulses does not matter. However, the interface software codes varies between different hardware devices.

As an example, the full C/C++ software codes that actually generate electrical pulses at the serial and parallel ports of the computer concurrently, is provided in the appendix at this link, [D 4.39]. The named, Concurrent-Writes-to-Parallel-and-Serial-Ports program, is the boundary point (or transition point) where "software codes write bits that finally generate hardware electrical pulses". This bit-writing software program is essentially the CNC Signal Driver software, that in turn takes inputs from the CNC Signal File. The CNC Signal File is, in turn, the output of the complicated work executed by the CNC Interpreter and CNC Interpolator combination, that also in turn, takes its inputs from the G-Code file, which is the starting point of the entire CNC operations.

Essentially, what we have described above is the "software backward trail" to its starting point in CNC machine operations. From the clarification above, we realize that in order to accomplish this project successfully, from start to finish, software design knowledge, software programming methods and software implementation skills are evidently important.

4.4 Pulse generator devices

4.4.1 Computer extension boards

The following are projects we have undertaken using pulse generator devices that successfully drive the CNC research machine. For computer extension boards as pulse generator devices, we worked on the following six(6) devices for our research:

1. PC parallel port, [1.26], [1.27], [1.28].
2. Velleman K8000 Parallel extension board, [1.30].
3. Velleman K8055 USB extension board, [1.31].
4. Heber X10i USB extension board, [1.32].
5. Arduino Due USB extension board, [1.33].
6. Digilent Nexys-3 Spartan-6 FPGA board. [1.34].

4.4.2 Single Board Computers

For single board computers (SBC) as pulse generator devices, we worked on the following three(3) systems for our research:

1. Raspberry Pi SBC, General purpose. [1.35], [1.36].
2. Banana Pi SBC, Graphic processing focus. [1.37].
3. Beagle-Board xM SBC. Graphic processing plus DSP focus. [1.38].

The above three(3) SBCs all have different brands of processors (CPUs). The Raspberry Pi SBC hardware is suited for general purpose use, while the Banana Pi SBC and BeagleBoard xM SBC are graphic processing focused.

In addition, we have chosen to study Beagle-Board xM because it has a digital signal processor (DSP) chip, which is not available on both the Raspberry Pi SBC or the Banana Pi SBC.

The primary purpose of all nine(9) devices mentioned above, is to generate electrical pulses through the associated pins on each board. Writing software that generates pulses (high/low for 1/0) at the pins means binary representation in software terms. This is another critical software programming activity in CNC development.

We need to write software that converts codes into binary formatted strings, for example, generating 8-bit binary strings to drive eight(8) individual parallel lines simultaneously, parallel in time. Similarly, with 16-bit binary strings, we can drive 16 parallel lines simultaneously. Fot that purpose, we provided the full C/C++ program code that converts outputs to 8-bit, 16-bit, and 32-bit binary strings in the appendix at [1.24].

Even though the binary string generation is common, the actual software driving implementation varies between boards. This arises due to the different pin configurations on each hardware device. We succeeded in our previous projects by manipulating software codes for each board, accordingly.

4.5 Computer Extension Boards as devices

4.5.1 Project 1 - PC parallel port

The parallel port hardware in a computer can be a built-in device located on the mother-board or an add-in PCI card attached to the PCI bus of the mother-board.

There are also converter devices, that converts USB to parallel port, like the Profilic USB to IEEE1284 Bridge Controller Bi-Direction Parallel Interface that uses the PL2305 software driver and is supported on both Windows and Linux. The parallel port software driver must be installed on the Linux computer for software to communicate with it.

A parallel port control program code written in C/C++ provides commands for the parallel port device (hardware) to generate electrical pulses to drive the CNC machine. The

parallel port pins are appropriately connected (wired electrically) to the different servo-drivers of the CNC machine.

The report for this work is at reference [6]. In this scheme, the various devices connected to the Linux computer are the devices responsible for generating electrical pulses that drives the CNC machine. Images of the parallel port devices that we have used are provided at [1.26] for parallel port built-in the computer motherboard, [1.27] for the parallel port PCI adapter card and [1.28] for the USB-to-Parallel port PL2305 converter cable.

4.5.2 Project 2 - Velleman K8000 Parallel extension board

The Velleman K8000 is a computer interface board, a hardware card to control external electrical devices with our computer through the standard parallel cable. The card offers some analog/digital connections to do this.

The card is provided with software drivers to run on both Windows and Linux computers. For the K8000 board, software programs on the computer can directly communicate via the parallel protocol IEEE1284 with the board. Software codes can be written in different programming languages like C/C++, Python, VB, Qbasic and Turbo Pascal.

Similarly, a parallel port control program code written in C/C++ provides commands for the parallel port device (hardware) to generate electrical pulses to drive the CNC machine.

The K8000 board digital output pins are appropriately connected (wired electrically) to the different servo-drivers of the CNC machine. On the Velleman K8000 board, the electrical pulses can be seen during CNC execution because LEDs (blinking on and off) are provided for selected pins.

The report for this work is at reference [6]. In this scheme, the Velleman K8000 Parallel interface board hardware is the device responsible for generating electrical pulses that drives the CNC machine. The image of the K8000 parallel interface board that we have used is provided at [1.30].

4.5.3 Project 3 - Velleman K8055 USB extension board

The Velleman K8055 board is a USB extension board that has 5 digital input channels and 8 digital output channels. There are two analogue inputs and two analogue outputs with 8 bit resolution. The number of inputs and outputs can be further expanded by connecting up to a maximum of four K8055 cards.

The board is provided with software drivers to run on both Windows and Linux computers. For the K8055 board, software programs on the computer can directly communicate via the USB protocol with the board. It is like driving a typical USB printer. Software codes can be written in different programming languages like C/C++, Delphi, Python, Visual Basic, Qbasic and Turbo Pascal.

Similarly, a USB port control program code written in C/C++ provides commands for the USB port device (hardware) to generate electrical pulses to drive the CNC machine. The K8055 board digital output pins are appropriately connected (wired electrically) to the different servo-drivers of the CNC machine. Also on the Velleman K8055 board, the electrical

pulses can be seen during CNC execution because LEDs (blinking on and off) are provided for selected pins.

The report for this work is at reference [10]. In this scheme, the Velleman K8055 USB interface board hardware is the device responsible for generating electrical pulses that drives the CNC machine. The image of the K8055 USB interface board that we have used is provided at [1.31].

4.5.4 Project 4 - Heber X10i USB extension board

The Heber X10i USB Board is also an extension board connected to the Linux computer via the USB port. The Heber X10i software driver must be installed on the Linux computer for software to communicate with it. The Heber board digital output pins are connected (wired electrically) to appropriate servo-drivers for the 3-axis CNC machine.

For the Herber board, software programs written in C/C++ on the Linux machine can directly communicate via USB with the board. It is like driving a typical USB printer.

As an extension board, there is no need for a compiled firmware to be downloaded onto the Heber board. Software instructions to generate electrical pulses come directly from the C/C++software programs on the Linux computer. Software codes can be written in different programming languages like C/C++ and Python.

The report for this work is at reference [8]. In this scheme, the Heber X10i USB Board is the device responsible for generating electrical pulses. The image of the Heber X10i USB board that we have used is provided at [1.32].

4.5.5 Project 5 - Arduino Due USB extension board

The Arduino Due board is an extension board connected to the Linux computer via the USB port. The Arduino Due software driver must be installed on the Linux computer for software to communicate with it.

A control program written in C/C++ on the Linux machine is compiled to produce machine codes (basically firmware) that is downloaded and installed into the memory of the Arduino Due board.

For the Arduino Due board, software programs on the computer cannot directly communicate online with the board. When any changes made in the software code to drive the CNC machine, it has to be recompiled and re-downloaded to the Arduino Due memory. This is different from the driving schemes in both Velleman K8000, Velleman K8055 and Heber X10i boards, where control programs on the Linux machine can directly drive electrical pulses through the boards to the CNC machine.

The relevant digital output pins on the Arduino Due are appropriately connected (wired electrically) to the CNC machine.

The report for this work is at reference [7]. In this scheme, the Arduino Due USB Board is the device responsible for generating electrical pulses. The image of the Arduino Due USB board that we have used is provided at [1.33].

4.5.6 Project 6 - Digilent Nexys-3 Spartan-6 FPGA USB board

The FPGA (Field Programmable Gate Array) USB board is also an extension board connected to the Linux computer via the USB port. The FPGA board software driver must be installed on the Linux computer for software to communicate with it.

FPGAs are semiconductor devices comprising programmable logic blocks and interconnection circuits. It can be programmed or re-programmed to the required functionality after manufacturing. Programming FPGA generates firmware. It was said that FPGA is about using software to program hardware.

The FPGA firmware for the Nexys-3 Spartan-6 FPGA, is a compiled VHSIC Hardware Description Language (VHDL) software on the Linux computer. After compilation it is downloaded onto the FPGA board to control the board.

The FPGA board digital output pins are connected with wires to appropriate servo-drivers for the 3-axis CNC machine.

After installation of the firmware on the FPGA board, software program instructions written in C/C++ on the Linux machine can directly communicate via USB with the board. With these received instructions, the FPGA pins appropriately generate the required electrical pulses to the CNC machine.

The Nexys-3 FPGA hardware development board manufactured by Digilent Inc. This board was used to control the CNC machine using its programmable input/output (I/O) pins. This hardware is a USB extension board to the Linux computer. The board was installed with a set of software drivers for the Linux platform.

A linux based personal computer running Ubuntu 10.04 LTS (Long Term Support) with RTAI (Real Time Application Interface) was deployed. The linux kernel version was linux-2.6.32-122-rtai, and was installed as part of the LinuxCNC system.

The FPGA board is an external control board connected to the linux computer via a USB port. The FPGA board pins were then connected with wires to appropriate servo-drivers for the 3-axis CNC machine.

The Xilinx ISE ver. 14.7 software application was used to generate the FPGA firmware. The FPGA firmware program was written using VHSIC Hardware Description Language (VHDL) language. VHDL is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays (FPGA) and integrated circuits (ICs).

VHDL is hardware programming using software, that results in firmware. Firmware is loaded into hardware to make the hardware work, basically controls the hardware. VHDL is inherently a general purpose parallel programming language. A VDHL program source code was created on the personal computer. The Xilinx ISE application compiles this VDHL code into a firmware code we called (FPGA-NEXYS3). This generated firmware was then uploaded from the computer into the FPGA board. Now the board is ready for use with our

CNC machine.

Essentially, the VHDL programming that was carried out to produce the FPGA-NEXYS3 firmware code is to map FPGA parameters and hardware pins for connections to servo-drivers for the 3-axis CNC machine. The G-Code File Interpreter (GFI) software was developed on the Linux platform. The Signals File Sender (SFS) software was developed on the Linux platform using C/C++ code.

It is important to note that in this FPGA research project, it is the combination of three(3) components: the GFI, the SFS and the FPGA-NESYS3 firmware that drive the CNC machine. First, a RS274D NGC G-Code file is processed by the GFI program to produce a signals file. Next, the SFS program transmits the contents of this signals file to the FPGA board.

The FPGA board with ready loaded-to-use FPGS-NEXYS3 firmware automatically sends separate electronic signals (pulses) in parallel to the individual servo-drives of the 3-axis CNC machine.

The report for this work is at reference [11]. In this scheme, the Nexys-3 Spartan-6 FPGA USB Board is the device responsible for generating electrical pulses to drive the CNC machine. The image of the FPGA USB board that we have used is provided at [1.34].

4.6 Single Board Computers as devices

4.6.1 Project 7 - Raspberry Pi 2 and Raspberry Pi 3 SBC boards

The Raspberry Pi 3 Model B, is a cheap, tiny credit card size, full-fledged computer system that can run software applications. It is called a Single Board Computer (SBC). The Raspberry Pi 2 is similar to Raspberry Pi 3, but is has less memory, less CPU speed (GHz), less USB ports, and less in everything. However, it works the same. This SBC is based on the ARM CPU processor and not the Intel CPU processor. This SBC includes built LAN and wireless networking.

The SBC (single board computer) is a true full-fledged computer in itself. It is not an extension board to some computer. The SBC is like a "desktop" computer that requires external peripherals. To use this Raspberry Pi SBC, we need just connect a keyboard, mouse, display monitor, power supply, and a micro SD card with an installed Linux Distribution, in a manner similar to connecting the same peripherals to the desktop computer.

With an operating system like Raspbian (Ubuntu-based) installed, we have full fledged Linux capabilities on the Raspberry Pi, including all software applications and language compilers available for Linux Ubuntu. The required software drivers, for example, BCM2835 must be installed to access specified components and peripherals on the Raspberry Pi.

The Raspberry Pi digital output pins are connected with wires directly to appropriate servo-drivers for the 3-axis CNC machine. Software programs written in C/C++, residing on the Raspberry Pi can directly drive electrical pulses through the digital output pins to the CNC machine. There is no need for any other computer.

The report for this work is at reference [9]. In this scheme, the Raspberry Pi-3 Model-B

SBC is the device responsible for generating electrical pulses to drive the CNC machine.

The image of the Raspberry Pi-3 Model-B that we have used is provided at [1.35] and for the Raspberry Pi-2 Model-B is provided at [1.36]. The image comparison between Raspberry Pi-3 and Raspberry Pi-2 is provided at [1.36].

4.6.2 Project 8 - Banana Pi BPI-M2 SBC board

Banana Pi BPI-M2 Ultra is a quad-core mini single board computer built with Allwinner R40 SoC. It features 2GB of RAM and 8GB eMMC. It also has onboard WiFi and BT. On the ports side, the BPI-M2 Ultra has 2 USB A 2.0 ports, 1 USB OTG port, 1 HDMI port, 1 audio jack, a DC power port, and last but not least, a SATA port.

Also being a member of the Banana Pi family, the M2 Ultra is a direct upgrade from the Banana Pi M1/M1+ that support SATA from the SoC. The SATA performance on the R40 is fitting for media related projects such as storage servers. Backed by our community, starting a project and building servers is fun and rewarding.

The Banana Pi operating system is similar to the Raspberry Pi, but a modified version to suit its hardware. It uses the same Raspbian system based on the Ubuntu software for desktops and notebooks. Software C/C++ programming is the same as for the Raspberry Pi. The only difference is the software driver for devices on the board. Raspberry Pi uses different hardware chips on its board compared to the Banana Pi, basically their product differentiations and marketing strategy.

The report for this work is at reference [10]. In this scheme, the Banana Pi BPI-M2 SBC is the device responsible for generating electrical pulses to drive the CNC machine. The image of the Banana Pi BPI-M2 SBC that we have used is provided at [1.37].

4.6.3 Project 9 - Beagle-Board xM SBC board

The BeagleBoard is a pocket-sized reference board containing a Texas Instruments Open Multimedia Application Platform (OMAP) 3 system-on-a-chip (SoC) processor, which includes an ARM Cortex-A8 core, Texas Instruments C64x+ digital signal processor (DSP), and onboard graphics engine, as well as integrated dual data rate (DDR) random-access memory (RAM). The BeagleBoard is an inexpensive platform for hobbyists, academics, and professionals who are learning Linux and small systems.

We chose this BeagleBoard xM for research because it has a digital signal processor (DSP), which is not available on the Raspberry Pi SBC or the Banana Pi SBC.

Similarly, the Beagle-Board xM operating system is similar to the Raspberry Pi, but a modified version to suit its hardware. It uses the same Raspbian system based on the Ubuntu software for desktops and notebooks. Software C/C++ programming is the same as for the Raspberry Pi. The only difference is the software driver for devices on the board. Raspberry Pi uses different hardware chips on its board compared to the Beagle-Board xM, basically their product differentiations and marketing strategy. The report for this work is at reference [10]. In this scheme, the Beagle-Board xM SBC is the device responsible for generating electrical pulses to drive the CNC machine. The image of the Beagle-Board xM SBC that we have used is provided at [1.38].

4.7 Summary on Related Research Work

We started by providing a brief description of our UMP CNC Research machine, a 3-axis experimental CNC router-type machine, that instead of using a tool cutter, uses a pen to create drawings on paper in the X-Y plane. We also showed images of the CNC Research machine. Each axis of the machine is driven by an alternating current servo-driver and servo-motor pair. The servo-driver can be driven using 3 different control strategies: position control, velocity control and torque control.

Next, we mentioned that only C/C++ software codes were used on various hardware devices as pulse generators to drive the CNC Research machine. The written codes include combinations of standard serial-running codes, realtime-running codes (RTAI) and parallel multi-threaded running codes.

We ran the C/C++ software codes on the open source Linux platform only. We avoided the Microsoft (MS) Windows platform because it is not possible to get full and open access to the operating system environment, that is, to the hardware devices attached to the operating system. We concluded that the MS Windows environment is not a suitable platform for our specific research project.

All of the interpolation computations we accomplished in our projects are of reference-pulse type CNC interpolations. We also implemented both non-realtime and realtime executions on the CNC Research machine.

Since the CNC machine only needs electrical pulses of the right characteristics to run, the kind of devices that generate these electrical pulses does not matter. On this matter, we described the hardware pulse generator devices comprising of six(6) computer extension board devices and three(3) single board computers (SBC) that we have successfully implemented in driving the CNC Research machine.

For driving the CNC Research machine using the parallel port, the four(4) devices used comprised, the built-in parallel port on the computer motherboard, the parallel port PCI adapter card, the USB-to-Parallel port PL2305 converter cable and the Velleman K8000 Parallel extension board.

For driving the CNC Research machine using the USB port, the four(4) devices used comprised, the Velleman K8055 USB extension board, the Heber X10i USB extension board, the Arduino Due USB extension board, and the Digilent Nexys-3 Spartan-6 FPGA USB development board.

For driving the CNC Research machine using standalone Single Board Computers (SBC), the four(4) devices used comprised, the Raspberry Pi 2 and Raspberry Pi 3 SBC boards, the Banana Pi BPI-M2 SBC board, and the Beagle-Board xM SBC board.

In the Appendix for Chapter-4 at this link [D 4], we included some results on our previous CNC research. Following that, we included images and specifications of the various hardware pulse generator devices we have successfully implemented on our CNC Research machine. And we included full contents of realtime and parallel running software codes, as examples.

5 Research Implementation Plan

5.1 Main Tasks in Research Implementation Plan

The total estimated duration for this research project is 356 days or about 1 calendar year. As shown in the figure below, the project tasks have been separated into four main categories as follows:

1. Project Preliminaries and Setup (Task No. 2 until Task No. 117)
2. Software Design and Implementation (Task No. 146 until Task No. 178)
3. Software Testing and Reporting (Task No. 190 until Task No. 219)
4. Project Publication and Closing (Task No. 224 until Task No. 233)

File Print Project

	Name	Duration	Start
1	⊟Research Implementation Plan	356 days	11/20/18 8:00 AM
2	⊞PhD Proposal Preparation	20 days	11/20/18 8:00 AM
18	⊞Issue Drafts PhD Proposal	26 days	11/20/18 8:00 AM
29	⊞Procurement Long Lead Items	30 days	11/20/18 8:00 AM
46	⊞Setup CNC Research Machine	5 days	11/20/18 8:00 AM
55	⊞Other Hardware Resources	5 days	11/26/18 8:00 AM
60	⊞Computer Notebooks Setup	5 days	11/20/18 8:00 AM
117	⊞Computer Desktops Setup	7 days	12/25/18 8:00 AM
146	⊞UseCase Design - CNC Control Software	30 days	1/1/19 8:00 AM
157	⊞Architecture Design - CNC Control Software	15 days	2/1/19 8:00 AM
168	⊞Process and Data Flow Design	15 days	2/15/19 8:00 AM
178	⊞CNC Control Loop Development	20 days	3/1/19 8:00 AM
190	⊞CNC Control Loop Integration Testing	20 days	3/15/19 8:00 AM
199	⊞Create Test Models G-Codes	3 days	4/12/19 8:00 AM
202	⊞Execute Test Models without Error Compensation	3 days	4/15/19 8:00 AM
205	⊞Contour Error Module Development	40 days	4/1/19 8:00 AM
216	⊞Execute Test Models with Error Compensation	4 days	5/25/19 8:00 AM
219	⊞Test with signal generator devices	20 days	6/3/19 8:00 AM
224	⊞Publications Plan	140 days	6/30/19 8:00 AM
228	⊟Project Closing	110 days	10/30/19 8:00 AM
229	Write PhD Thesis	20 days	10/30/19 8:00 AM
230	Submit PhD Thesis WriteUp	2 days	11/30/19 8:00 AM
231	Perform Thesis Corrections	10 days	1/31/20 8:00 AM
232	Resubmit Thesis	2 days	2/27/20 8:00 AM
233	Complete Viva Voce	2 days	3/30/20 8:00 AM

Figure 5.1: Main Tasks in Research Implementation Plan

The Work Breakdown Structure (WBS) or subtasks with details for the Research Implementation Plan are provided in the appendix at the link Fig [E 5].

5.2 Overview of Research Implementation Schedule

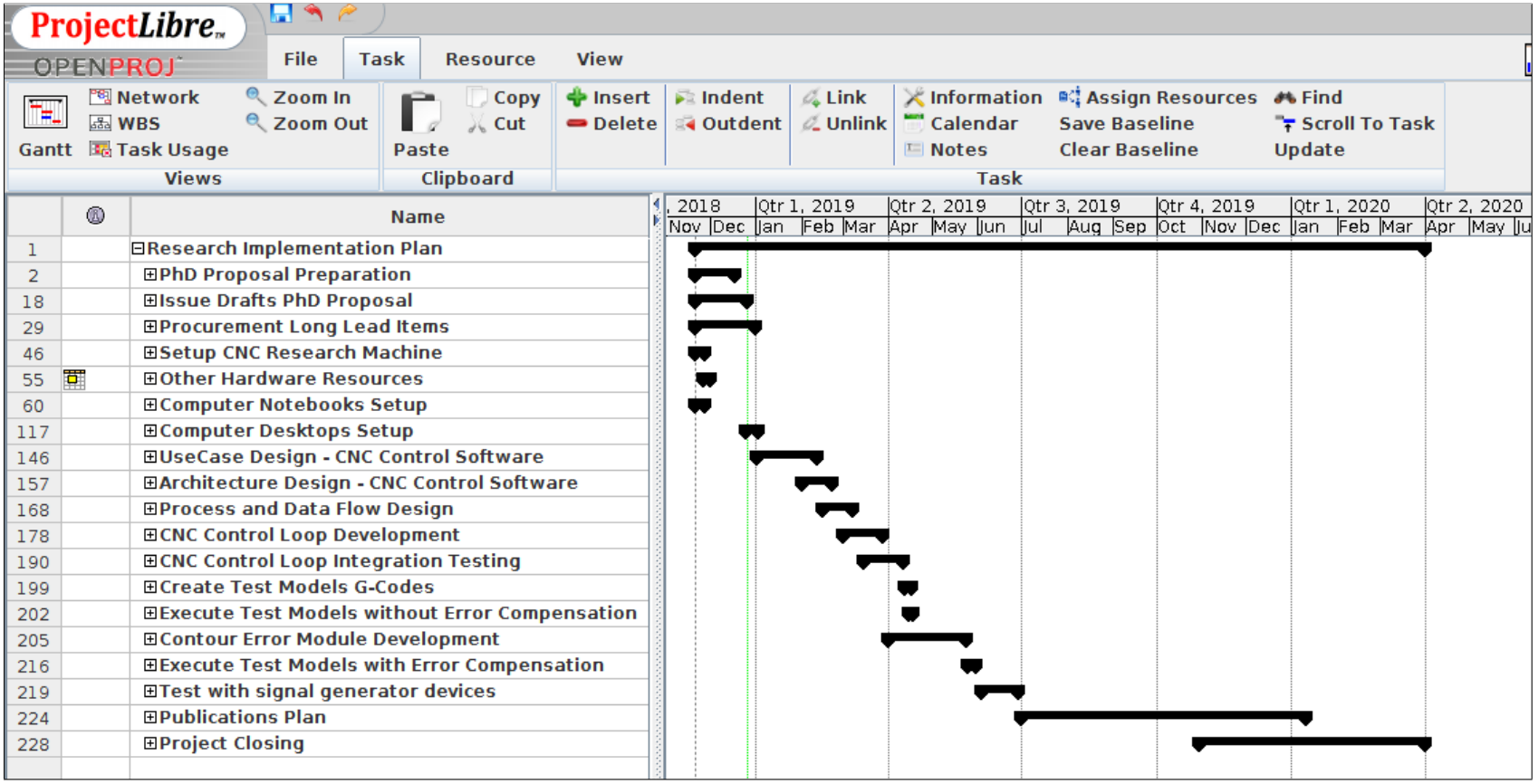


Figure 5.2: Overview of Research Implementation Schedule

5.3 Critical Project Tasks

The identified critical tasks are the UseCase Design, Architecture Design, Process and Data Flow Design, and the Control Loop Design.

The UseCase design must be correct and complete covering all possibilities of system usage, link at [E 5.5]. UseCases are the blueprint for software design. Incomplete UseCases may cause missing features and functions in the software. The software architecture, process flow, data flow and control loop designs, link at [E 5.6], must ensure that the system will be built with flexible and upgradeable features. A design that is incomplete and rigid may require a complete design rework if certain new features need to be added or existing features need to be modified.

Since software design is the blueprint for software construction, good design facilitates good software construction. In this project, we anticipate a few rounds of iterative designs since software construction will be conducted following the incremental and prototyping model. This software construction method saves time and eliminates complex logical design errors. In the incremental and prototyping model, the system is developed by starting from having small features, then incrementally built, tested, and then reworked as necessary until an acceptable complete system is achieved. This model works best where not all of the project requirements are known in detail ahead of time. This process development model is iterative, sort of, a trial-and-error process that repeats until the final design goals have been achieved.

5.4 Research Milestones

The project milestones follow closely the project critical tasks. Successful completion of usecases, software architecture, process flow, data flow and control loop designs are the five primary milestones in this research project. Successful completion of control loop testing, integrated testing, hardware system testing and publication submissions are another four secondary milestones in this project.

5.5 Publications Plan

We anticipate a total of three publications spread three months apart in this research project. The target is two conference publications and one journal publication. The first conference publication will be for the Pico Universal PWM hardware with LinuxCNC, and the second publication will be for the Microchip MCU Curiosity and 28-Pin LIN development boards and the CNC Control Software. The target journal publication will be for CNC Interpolation, which is the main thesis for this research.

5.6 Implementation approach

The implementation approach is to focus on the utilization of modern software engineering technologies like efficient data structures, fast algorithms, realtime and parallel computations, tractable program executions, integrations of different programming language libraries and many more that will be explored during the software design stage. The final goal is to achieve an efficient implementation for a realtime and parallel look-ahead control and feedrate compensation strategy for CNC reference-pulse interpolation.

6 Conclusion

The focus of this research is CNC interpolation. Technically, CNC interpolation is about the task of generating and sending signal pulses to the respective electric motors at the machine's axis-of-motions, in a coordinated and timely manner, such that the machine tool accurately tracks the desired contour path in the move from the current position to the next position.

To achieve the task, the interpolator must generate the right number of signal pulses that achieves the desired distance for the move, and drive the correct pulse feedrate (frequency) that achieves the desired velocity for the move.

This research emphasizes the adoption of effective and efficient software engineering technologies in the implementation details of CNC interpolation. The adoption includes sophisticated data structures, efficient process and data flow algorithms, high resolution realtime monitoring, true parallel executions, thread-safe process designs, high speed data dumps (captures), signal/slot programming mechanism and many more deployment techniques that can be creatively developed with modern software compilers available today.

Software program control paradigms considered for the CNC Control Loop include process-driven, interrupt-driven, target-driven and optimal-driven methods. Software integration strategies considered include the utilization of back end packages and libraries like Scilab-NURBS, Octave-NURBS, Rust low-level thread safe constructs, Julia and Python parallel libraries over and above the C/C++ base program codes. This program control is the core activity for this research.

All software implemented in this research are open source, cost-free, and actively supported by the software community. This facilitates sharing of knowledge, software solutions, programming tricks and passionate lifelong learning.

For performance comparisons, hardware architectures comprising both 64bit and 32bit systems with different configurations will be used. In addition, three different pulse generator devices will be deployed in this research project.

The ultimate goal is to suitably implement selected software technologies that achieves our primary research target, that is, a realtime and parallel look-ahead control and feedrate compensation for the CNC reference-pulse interpolation task. For this research, the other target is to produce two conference papers and a journal paper.

This proposal is an exciting and exhilarating research project, especially for a software engineer. However, success can be elusive. On that note, it is our firm belief that the eventual success of this research rests on the help and inspiration from Allah, the All Mighty, the Most Gracious and Most Merciful. In this endeavor, we pray to Allah for his continuous blessings.

Bibliography

[1] S. H. Suh, S. K. Kang, D. H. Chung, and I. Stroud, *Theory and Design of CNC Systems, Springer Series in Advanced Manufacturing*. Springer-Verlag London Limited, 2008.

[2] N H Giap, J H Shin, W H Kim, "A Reference-Pulse Generator for Motion Control System," *Intelligent Control and Automation*, vol. Vol 5, pp 111-119, 2014.

[3] Y Koren, O Masory, "Reference-Pulse circular interpolators for CNC Systems," *Transactions ASME, Journal of Engineering for Industry*, vol. Vol 103, pp 131-136, 1981.

[4] Y Koren, O Masory, "Reference-Word circular interpolators for CNC systems," *Transactions ASME, Journal of Engineering for Industry*, vol. Vol 104, pp 400-405, 1982.

[5] Y Koren, J G Bollinger, "Design Parameters for Sampled-Data Drives for CNC Machine Tools," *IEEE Transactions on Industry Applications*, vol. Vol 1A-14, No 3, pp 255-264, 1978.

[6] A. Kalimbetov and W. R. Yusoff, "C/C++ and Python for Linux Realtime Parallel Port Software Driver," Jan 2012. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[7] H. I. Z. Ariffin and W. R. Yusoff, "Using Arduino Due to drive a CNC System," Jan 2014. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[8] M. S. A. Abdelgader and W. R. Yusoff, "Driving Computer Numerical Controlled (CNC) Machine Using PC, Heber X10i, and Raspberry Pi 2," Jun 2014. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[9] A. H. Ahmad and W. R. Yusoff, "Using Raspberry Pi 3 Model B to drive a CNC System in Real Time," Feb 2017. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[10] R. Selvarajah and W. R. Yusoff, "Using the Beagle Board xM SBC to drive a CNC System," Feb 2015. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[11] C. Teh and W. R. Yusoff, "Using the Nexys-3 Spartan-6 FPGA board to develop a closed-loop feedback CNC system," Feb 2016. Final Year Project, Multimedia University, Cyberjaya, Unpublished.

[12] B. Warfield, "Choose the Best CNC Control: 2017 CNC Control Survey Results," Dec 2017. https://www.cnccookbook.com/choose-best-cnc-control-2017-cnc-control-survey-results/ [Accessed: 06-Dec-2018].

[13] Fanuc, "FANUC CNC Control Series." https://www.fanuc.eu/uk/en/cnc/controls [Accessed: 06-Dec-2018].

[14] Haas, "HAAS CNC Automation Machines." https://www.haascnc.com/index.html [Accessed: 06-Dec-2018].

[15] Mazak, "MAZAK CNC Machines." https://www.mazakusa.com/machines/ [Accessed: 06-Dec-2018].

[16] Siemens, "SIEMENS CNC Sinumerik: Intelligent solutions for machine tools." https://www.siemens.com/global/en/home/products/automation/systems/cnc-sinumerik.html [Accessed: 06-Dec-2018].

[17] Centroid, "CENTROID CNC controls." http://www.centroidcnc.com/ [Accessed: 06-Dec-2018].

[18] Heidenhain, "HEIDENHAIN CNC controls." https://www.heidenhain.com/ [Accessed: 06-Dec-2018].

[19] Mitsubishi-Electric, "MITSUBISHI Computerized Numerical Controllers(CNCs)." http://www.mitsubishielectric.com/fa/products/cnt/cnc/index.html [Accessed: 06-Dec-2018].

[20] Allen-Bradley, "ALLEN-BRADLEY Intelligent solutions for CNC machine tools ." https://ab.rockwellautomation.com/Motion-Control/Software [Accessed: 06-Dec-2018].

[21] Mach, "The Next Level of CNC Control." https://www.machsupport.com/ [Accessed: 06-Dec-2018].

[22] PathPilot, "America's most loved CNC Controller." https://www.tormach.com/pathpilot/ [Accessed: 06-Dec-2018].

[23] Probitix, "Probotix CNC Control Systems." https://www.probotix.com/CNC-CONTROL-SYSTEMS [Accessed: 06-Dec-2018].

[24] Planet, "CNC USB Controllers: Complete, hardware and software solutions." https://planet-cnc.com/ [Accessed: 06-Dec-2018].

[25] EdingCNC, "Welcome at Eding CNC Machine Controls." http://www.edingcnc.com/ [Accessed: 06-Dec-2018].

[26] UCCNC, "CNC Drive Motion Control." https://www.cncdrive.com/UCCNC.html [Accessed: 06-Dec-2018].

[27] C.-Y. Lee, S. H. Kim, T. I. Ha, J. Min, S.-H. Hwang, and B.-K. Min, "CNC Algorithms for Precision Machining: State of the Art Review," *Journal of the Korean Society for Precision Engineering*, vol. 35, pp. 279–291, Mar 2018.

[28] A. Lasemi, D. Xue, and P. Gu, "Recent development in CNC machining of freeform surfaces: A state-of-the-art review," *Computer-Aided Design*, vol. 42, pp. 641–654, Jul 2010.

[29] Y. Koren, "Interpolator for a CNC System," *IEEE Transactions on Computers*, vol. C-25, pp. 32–37, Jan 1976.

[30] Y Koren, "Design of Computer Control for Manufacturing Systems," *Transactions ASME, Journal of Engineering for Industry*, vol. Vol 101, pp 326-332, 1979.

[31] C. S. (LLC), "STEP-Compliant CAD/CNC Systems for Feature-Oriented Machining," in *CAD 18*, CAD Solutions (LLC), Jul 2018.

[32] Y. Yusof and K. Latif, "Frame Work of LV-UTHM: AN ISO 14649 Based Open Control System for CNC Milling Machine ," *Applied Mechanics and Materials*, vol. 330, pp. 619–623, 2013.

[33] G. T. Smith, *CNC Machining Technology*. Springer London, 1993.

[34] G. T. Smith, "CNC Controllers and Programming Techniques," in *CNC Machining Technology*, pp. 217–316, Springer London, 1993.

[35] L Yang, G Zhang, "Research and Development of Sampled-data Interpolation Algorithm Software in CNC System Based on the Visual C++," *IOP Conference Series: Materials Science and Engineering*, vol. Vol 382, 042019, 2018.

[36] G. E. Farin, *NURBS From Projective Geometry to Practical Use*. Boca Raton, FL: CRC Press: Taylor and Francis Group, 2nd. ed., 1999.

[37] D C H Yang, T Kong, "Parametric interpolator versus linear interpolator for precision surface machining," *Computer-Aided Design*, vol. Vol 26, pp 225-234, 1994.

[38] X Zhang, Dong Yu, Yi Hu, H Hong, W Sun, "Development of a NURBS Curve Interpolator with Look-ahead Control and Feedrate Filtering for CNC System," *IEEE International Conference on Industrial Engineering and Applications*, vol. Vol 47, pp 2755-2759, 2009.

[39] B Bahr, X Xiao, K Krishnan, "A real-time scheme of cubic parametric curve interpolations for CNC systems," *Computers in Industry*, vol. Vol 45, pp 309-317, 2001.

[40] C. N. Cho and H. J. Kim, "Feedrate fluctuation compensating NURBS interpolator for CNC machining," in *2017 20th International Conference on Electrical Machines and Systems (ICEMS)*, IEEE, Aug 2017.

[41] Editors, "Accuracy and Error Compensation of CNC Machining Systems," in *Metal Cutting Theory and Practice, Third Edition*, pp. 827–896, CRC Press, Mar 2016.

[42] C. Liangji, "A Computer Numerical Controlled System with NURBS Interpolator," *IEEE World Congress on Computer Science and Information Engineering*, vol. pp 216-219, 2009.

[43] M T Lin, M S Tsai, H T Yau, "Development of Real-time Look-Ahead Algorithm for NURBS Interpolator with Consideration of Servo Dynamics," *46th IEEE Conference on Decision and Control*, vol. pp 1862-1867, 2007.

[44] M T Lin, M S Tsai, H T Yau, "Development of a dynamics-based NURBS interpolator with real-time look-ahead algorithm," *International Journal of Machine Tools and Manufacture*, vol. Vol 47, No 3, pp 2246-2262, 2007.

[45] D. X. Zheng, "The Error Analysis of Surface Machining on the CNC Machine Tools," *Applied Mechanics and Materials*, vol. 494-495, pp. 681–684, Feb 2014.

[46] X. Chen, A. Geddam, and Z. Yuan, "Accuracy improvement of three-axis CNC machining centers by quasi-static error compensation," *Journal of Manufacturing Systems*, vol. 16, pp. 323–336, Jan 1997.

[47] J. C. Dong, T. Y. Wang, Y. Y. Ding, and Y. L. Cui, "Adaptive Error Control Algorithm for High Speed Two-Axis CNC Contouring," *Key Engineering Materials*, vol. 584, pp. 149–153, Sep 2013.

[48] J.-W. Ma, D.-N. Song, Y.-Y. Gao, and Z.-Y. Jia, "Reduction of the Contouring Error in High-Feed-Speed Machining by Real-Time Tracking-Error Compensation," in *Mechanics and Materials Science*, WORLD SCIENTIFIC, Oct 2017.

[49] X. Zuo, B. Li, J. Yang, and X. Jiang, "Integrated Geometric Error Compensation of Machining Processes on CNC Machine Tool," *Procedia CIRP*, vol. 8, pp. 135–140, 2013.

[50] Y. M. Hendrawan and N. Uchiyama, "Embedded Iterative Learning Contouring Controller Based on Precise Estimation of Contour Error for CNC Machine Tools," in *2018 26th Mediterranean Conference on Control and Automation (MED)*, IEEE, Jun 2018.

[51] J. Guo, Q. Zhang, and X.-S. Gao, "Tracking error reduction in CNC machining by reshaping the kinematic trajectory," *Journal of Systems Science and Complexity*, vol. 26, pp. 817–835, Oct 2013.

[52] X. Chen and W. Zhang, "Research on Error Compensation Technology for CNC Machining," *MATEC Web of Conferences*, vol. 100, p. 03028, 2017.

[53] J. Wang, H. Liang, D. gong Guan, X. Wang, H. Shen, and M. Luo, "Contour error and control algorithm in CNC machining tool," in *2012 IEEE International Conference on Mechatronics and Automation*, IEEE, Aug 2012.

[54] T. Schmitz and J. Ziegert, "Dynamic evaluation of spatial CNC contouring accuracy," *Precision Engineering*, vol. 24, pp. 99–118, Apr 2000.

[55] W. Li and Q. Yang, "Develop on feed-forward real time compensation control system for movement error in CNC machining," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, IEEE, Oct 2010.

[56] P. F. Lau, *CNC machining accuracy enhancement by tool path compensation method*. PhD thesis, The Hong Kong University of Science and Technology Library, 2005.

[57] M. Tingting, J. Li, D. Zheng, and Z. Li, "The design of iterative learning control scheme for CNC machine tools," in *2016 IEEE International Conference on Information and Automation (ICIA)*, Aug 2016. DOI: 10.1109/ICInfA.2016.7831903.

[58] H. Fu, C. Li, and Y. Fu, "A parallel CNC system architecture based on Symmetric Multi-processor," in *2016 Sixth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, pp. 634–637, 2016. DOI 10.1109/IMCCC.2016.74.

[59] M Shpitalni, Y Koren, C C Lo, "Real-time curve interpolators," *Computer-Aided Design*, vol. Vol 26, pp 832-838, 1994.

[60] Y. Koren, M. Shpitalni, and C. C. Lo, "CNC Interpolators: Algorithms and Analysis," *ASME Manufacturing Science and Engineering.*, vol. Vol 64, pp 83-92, 1993.

[61] Elsevier, "A CNC machining system for education," *Journal of Manufacturing Systems*, vol. 12, no. 1, p. 71, 1993.

[62] E. I. Kats and A. M. Kitaev, "On the approach to CNC machining simulation improving," in *2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, IEEE, May 2017.

1 Appendices

The contents of the appendices are organized according to chapters presented in the main text. The appendices are as listed as follows: The link to directly jump to each appendix is also provided.

(1): Chapter-1 Appendices (Introduction) [A 1]

(2): Chapter-2 Appendices (Literature Survey)[B 2]

(3): Chapter-3 Appendices (Research Methodology)[C 3]

(4): Chapter-4 Appendices (Related Research Work)[D 4]

(5): Chapter-5 Appendices (Research Implementation Plan)[E 5]

A 1 Appendix-A1 Introduction

A 1.1 App1-CNC Milling Machine

Figure 1.1: App1-CNC Milling Machine

A 1.2 App1-CNC Lathe Machine

Figure 1.2: App1-CNC Lathe Machine

A 1.3 App1-CNC Routing Machine

Figure 1.3: App1-CNC Routing Machine

A 1.4 App1-CNC 3D Printing Machine

Figure 1.4: App1-CNC 3D Printing Machine

B 2 Appendix-B2 Literature Survey

B 2.1 App2-Ha Scilab NURBS versus Octave NURBS

Table 1.1: App2-Scilab NURBS versus Octave NURBS

Scilab NURBS versus Octave NURBS			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

B 2.2 App2-Scilab NURBS versus Octave NURBS

Table 1.2: App2-Computer Notebook Specifications

Computer Notebook Specifications			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

B 2.3 App2-Bla bla bla

Table 1.3: App2-Computer Notebook Specifications

Computer Notebook Specifications			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

Page 84 of 230

C 3 Appendix-C3 Research Methodology

C 3.1 App3-Pico Universal PWM Servo Controller

Figure 1.5: App3-Universal PWM Servo Controller Parallel Port Interface Board

C 3.2 App3-Specifications Pico Universal PWM Servo Controller

Reference: http://www.pico-systems.com/motion.html

The Universal PWM Servo Controller is a small board with everything needed to control a 2-axis, 3-axis or 4-axis machine tool with PWM-driven servo amplifiers. It contains 4 PWM generators with variable PWM drive frequency, 4 digital encoder counters to follow the machine position, 16 channels of opto-isolated digital inputs, and 8 positions for Solid State Relays of your choice to be mounted. It is connected to a computer by the parallel port. The parallel port must be at least capable of bi-directional exchange of data, but EPP or ECP modes give the best data transfer rate. The digital I/O section also implements emergency stop logic. There is an on board watchdog timer, which can be set to cause an emergency stop in case the computer fails to update the PWM generators in a timely fashion. This could shut off the servo amps, spindle motor, coolant, etc. For machines with more than 4 axes, 2 or more boards can be used together.

The computer reads the position from the encoder counters and computes a new PWM duty cycle to send to the PWM generators. This takes only about 50 uS on a 333 MHz Pentium II, so that reasonable servo update rates of 10 KHz could be made on such a machine. I usually use 1 KHz, because that is all that seems needed for machine-tool type applications.

The PWM generators divide a 10 MHz crystal clock by a minimum of 2 up to 216-1, which comes out to 5 MHz down to 153 Hz. A PWM frequency of 1 to 100 KHz is practical, and can be selected to suit the servo amplifiers. The duty cycle of the PWM waveform can be programmed in 100 nS steps, which is 1 percent at 100 KHz, but 0.2 percent at 20 KHz.

The PWM and direction outputs can source or sink 12 mA to drive opto-coupled amplifier inputs.

The encoder counters keep a continuous watch over the encoder signals and can count up to 300,000 encoder counts/second, per channel. (The rev 5.x and later boards have an adjustable digital filter so that counting above 5 MHz can be performed.) They can also sense the index pulse from an encoder which has this feature. This can be used to more precisely locate the home position. If the encoder has no index channel, connect the index input (Z) to A.

The digital input section has 16 opto-isolators which can sense the condition of switches, relays, pressure switches, float sensors, etc. to allow the machine to be stopped if a fault condition occurs, sense when an axis is close to the travel limit or home position, etc. The board provides isolated 5V power to power the switches and/or sensors.

The digital output section provides sockets for up to 8 Opto-22 compatible Solid State Relays to be mounted directly onto the board. These sites are left unpopulated to allow the user to select SSRs with the output configuration and current capacity needed. A terminal strip is provided for connection to the outputs of the SSRs. LEDs monitor the command signal to each of the SSRs.

The last digital input is configured to monitor an emergency stop chain. A series circuit of normally-closed switches breaks the continuity of the circuit when an unsafe condition or problem develops (ie. spindle motor stall, servo amp overheat, lube level low, manual E-stop switch activated, etc.) An analog timer circuit can also monitor the flow of commands from the computer, and if the computer ceases updating the rate generators, then an E-stop can be caused. The E-stop condition turns off all signals to the solid state relays, as well as stopping the PWM generators, to bring the machine to a safe stop.

This boards contains a power regulator that produces all power needed by it from a provided 'wall-wart' type plug-in power supply.

The Universal PWM Controller takes advantage of the IEEE-1284 hardware signalling protocol, allowing multiple register addresses to be selected and transfers accomplished with minimum CPU overhead, and maximum data transfer rate. This requires a parallel port that can operate in the ECP or EPP mode. A male-female DB-25 cable specifically designed for IEEE-1284 compatability is required. Note: You MUST use a cable marked "IEEE-1284 compliant" for this system to work reliably.

C 3.3 App3-MCU Microchip 28-Pin LIN Development Board

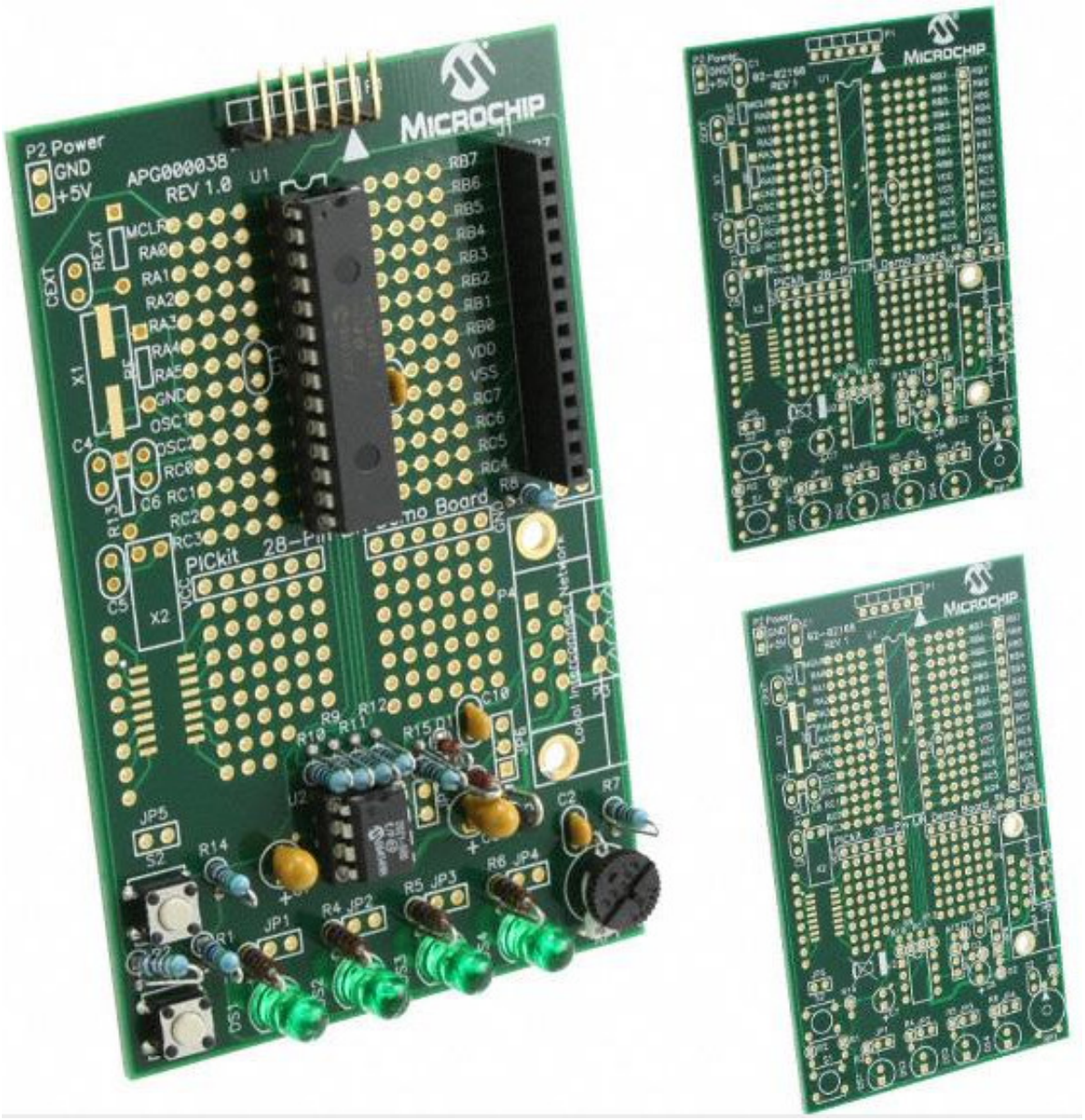


Figure 1.6: App3-MCU Microchip 28-Pin LIN Development Demo Interface Board

C 3.4 App3-Specifications MCU Microchip 28-Pin LIN Development Board

Reference: https://www.digikey.my/products/en?keywords=DM164130-3-ND%20%20
28-Pin LIN DEMO BOARD USER'S GUIDE
2009 Microchip Technology Inc.DSxxxxx

The 28-Pin LIN (Local Interconnect Network) Demo Board is a small and simple demonstration PCB for Microchip's 28-pin Dual Inline Package (DIP) PIC Microcontroller Units (MCU). It is populated with a PIC16F886 MCU, a MCP2021 LIN Transceiver with voltage

regulator, four LEDs, 2 push buttons and a potentiometer. The demo board has several test points to access the I/O pins of the MCU and a generous prototyping area. The MCU can be programmed with the PICkit 2 Microcontroller Programmer or the MPLAB ICD 2 using the RJ-11 to 6-pin inline adapter (AC164110).

LIN (Local Interconnect Network) is an automotive networking technology, a sub-bus system based on a serial communications protocol. The bus is a single master/multiple slave bus that uses a single wire to transmit data.

The 28-Pin LIN Demo Board can be used with virtually any 28-pin Dual Inline Package (DIP) PIC MCU. The assembled 28-Pin LIN Demo Board is populated with a PIC16F886-I/P microcontroller. Additional 28-Pin LIN Demo Boards can be ordered from Microchip technology and distributors. Part number, DM164120-3, comes with one assembled and two blank 28-Pin LIN Demo Boards.

The blank demo board can be used for evaluating or prototyping circuits using any of the 28-pin devices listed below.

PIC16CR63	PIC16F913	PIC18F2510
PIC16CR76	PIC16F916	PIC18F2520
PIC16C63A		PIC18F2515
PIC16C745		PIC18F2523
PIC16C773	PIC18F2220	PIC18F2525
	PIC18F2221	PIC18F2550
PIC16F737	PIC18F2320	PIC18F2580
PIC16F767	PIC18F2321	PIC18F2585
PIC16F870	PIC18F2331	PIC18F2610
PIC16F872	PIC18F2410	PIC18F2620
PIC16F873A	PIC18F2420	PIC18F2680
PIC16F876A	PIC18F2423	PIC18F2682
PIC16F882	PIC18F2431	PIC18F2685
PIC16F883	PIC18F2450	PIC18F24J10
PIC16F886	PIC18F2455	PIC18F25J10
	PIC18F2480	

Figure 1.7: App3-MCU Chips Supported-for Dev Demo Interface Board

The 28-Pin LIN Demo Board is populated with a PIC16F886 MCU (U1), a MCP2021 LIN Transceiver with Voltage Regulator (U2), four LEDs (DS1-DS4), Two push buttons (SW1 and SW2), 32 KHz crystal (X2) and potentiometer (RP1). The board layout is shown in the figure below. The demo board has several test points to access the I/O pins of the MCU and a generous prototyping area. The MCU can be programmed with the PICkit 2 Microcontroller Programmer from header P1.

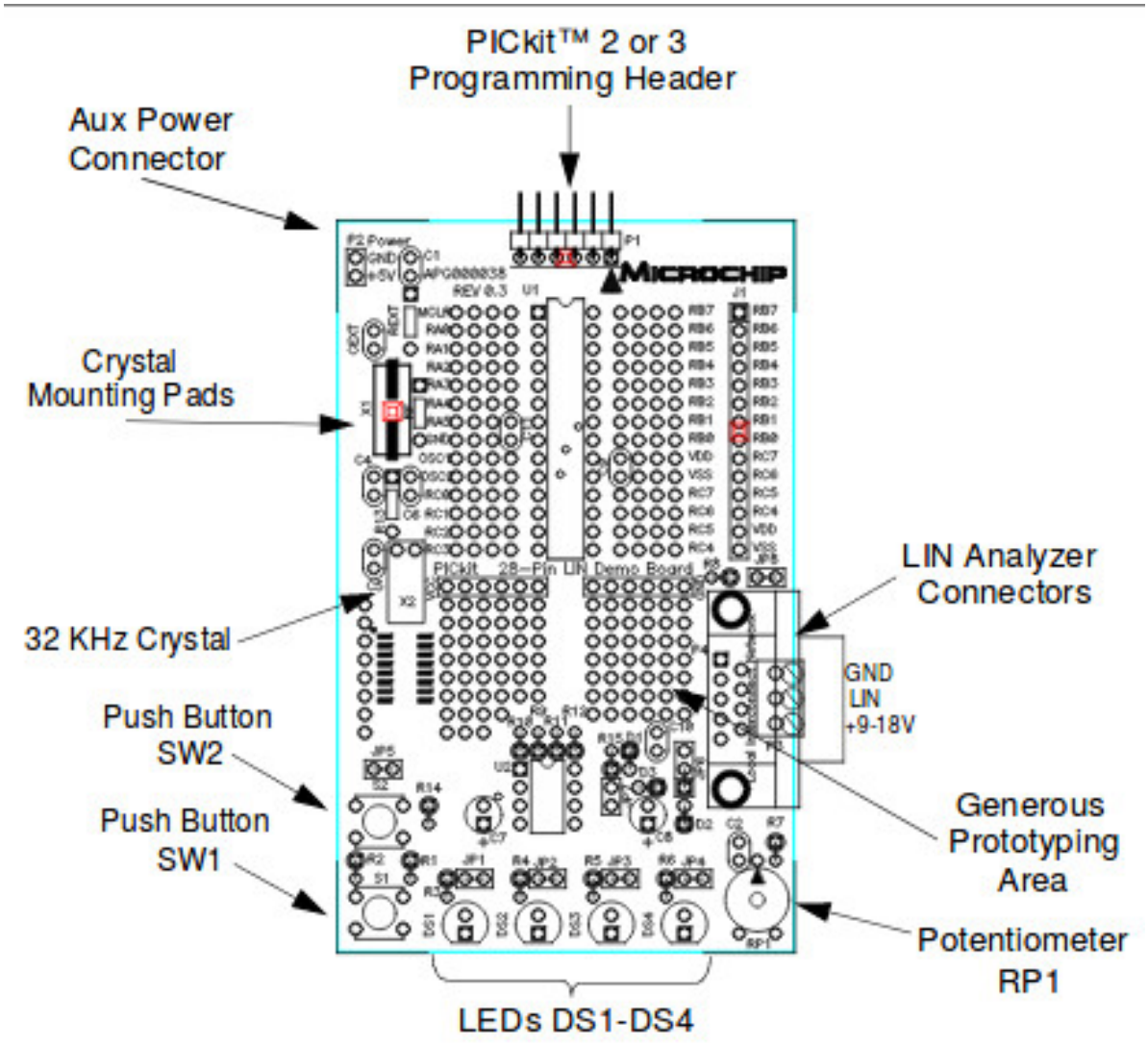


Figure 1.8: App3-MCU Microchip 28-Pin LIN Dev Demo Board Layout Diagram

C 3.5 App3-MCU Microchip Curiosity Development Board

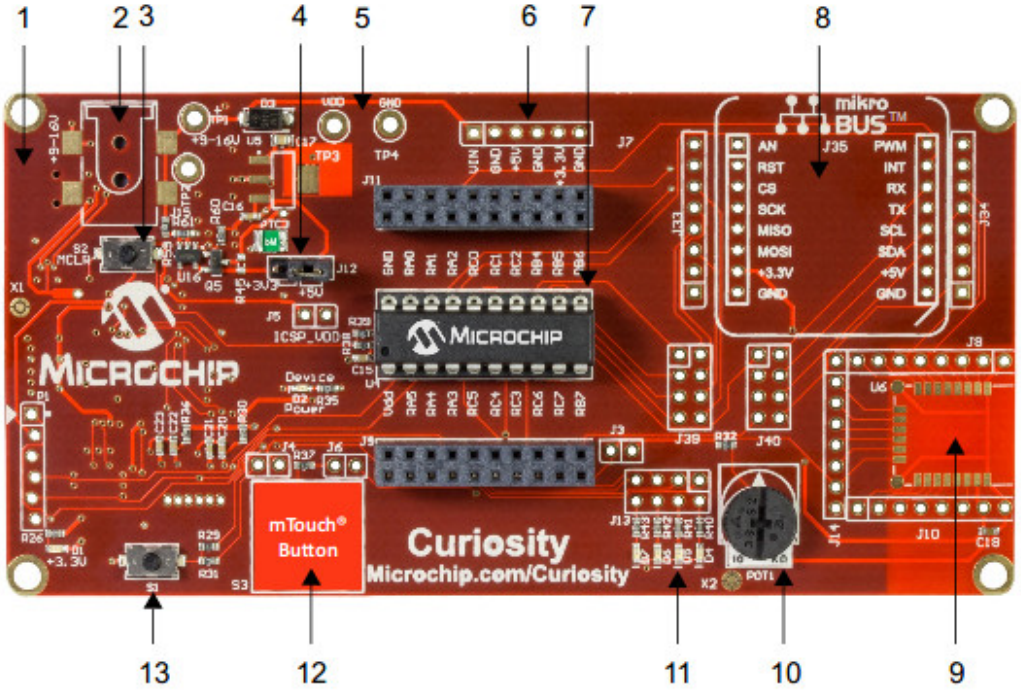


Figure 1.9: App3-MCU Microchip Curiosity Demo Board Layout Diagram

1. USB mini-B connector (on back)
2. Footprint for 9V connector
3. Master Clear Reset button
4. 3.3/5V power jumper (J12)
5. Posts for external variable power supply
6. Expansion board connector
7. PIC® MCU socket for 8, 14, and 20-pin microcontrollers
8. mikroBUS™ Click Board footprint for application development
9. RN4020 Bluetooth® Module Footprint
10. Potentiometer
11. LEDs
12. mTouch® button
13. Push button

Figure 1.10: App3-MCU Microchip Curiosity Demo Board Layout Legend

C 3.6 App3-Specifications MCU Microchip Curiosity Development Board

Reference: Curiosity Development Board User's Guide
2015-2016 Microchip Technology Inc. DS40001804B

The Curiosity Development Board supports Microchip's 8-pin, 14-pin and 20-pin 8-bit PIC MCUs. Dual-row expansion headers on either side of the socket offer flexibility of connectivity to all pins on the PIC MCUs. This board provides flexibility for experimentation through an application header with ground (GND) and supply voltage (VDD) connections. It also includes a set of indication LEDs, mTouch button and push-button switches, and a variable potentiometer. Additionally, it features a Bluetooth low-energy footprint and a mikroBUS footprint to accommodate a variety of plug-in Click Board sensors that can be used in application development.

The Curiosity Development Board can be powered in one of three ways, depending on its usage.

Power1 via USB Connector (J2)

The USB connector (J2) will power the entire Curiosity Development Board. A shunt jumper must be placed onto jumper J12. The right two pins of J12 will connect +5V from the USB connector J2. The left two pins of J12 will connect +3.3V from the USB voltage regulator on the back side of the development board. With USB power connected to J2, power LED D1 will always be ON to indicate that +3.3V is available on the board.

Power2 via 9V External Power Supply (J15)

The 9V external power supply (J15) will also power the entire Curiosity Development Board. A shunt jumper must be placed onto jumper J12. The right two pins of J12 will connect +5V from the on-board voltage regulator circuitry connected to connector J15. The left two pins of J12 will connect +3.3V from the on-board voltage regulator circuitry. With 9V external power connected to J15, power LED D1 will always be ON to indicate that +3.3V is available on the board. Power LED D2 will only be ON when power (+3.3V or +5V) is applied to VDD via a shunt jumper placed on J12.

Power3 via Variable External Power Supply (TP3, TP4)

A variable external power supply connected to TP3 and TP4 will power the entire Curiosity Development Board. A shunt jumper is not needed on J12, thus either +3.3V or +5V can be directly applied via a variable external power supply to VDD.

Getting Started on Curiosity Board

The Curiosity Development Board must be used with MPLAB X Integrated Development Environment (IDE), available free on Microchip's web site, www.microchip.com. Use version v3.05 or later. The Curiosity Development Board, through MPLAB X, is a low-voltage in-circuit debugger, as well as a low-voltage programmer, for all supported devices. In-circuit debugging allows the user to run, examine and modify programs for the supported

device embedded in the Curiosity hardware. This facilitates the debugging of firmware and hardware concurrently. Use the Curiosity Development Board with MPLAB X IDE to run, stop and single-step through programs breakpoints can be set and the processor can be reset. When the processor stops, the contents of the register are available for examination and modification.

Programming the Curiosity Board

After connecting the Curiosity Development Board to the computer using the on-board USB connector (J2 on the back of the board), open the MPLAB X IDE. Then create a new project or open an existing project. Click on the Project Properties icon located in the project's Dashboard window.

Alternatively, the Project Properties window can be opened by clicking on File, Project Properties, or by right-clicking on the project name in the Projects window and clicking Properties.

And it goes on.

D 4 Appendix-D4 Related Research Work

D 4.1 App4-Real Time Application Interface (RTAI) architecture

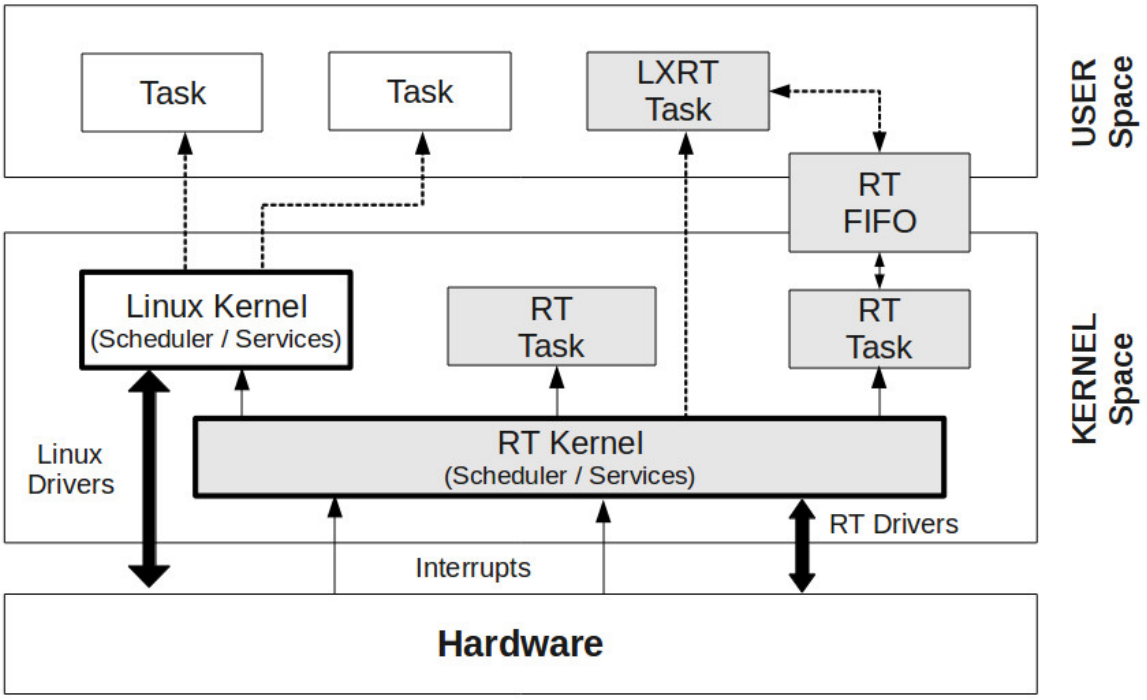


Figure 1.11: App4-Real Time Application Interface (RTAI) architecture on Linux

There are two(2) aspects of writing software codes for CNC machines we undertook in our past projects.

1. writing C/C++ **realtime codes** using the Linux environment RTAI. Realtime for a process here means strictly meeting both its start and end deadlines.

2. writing C/C++ **parallel codes** using the multi-threading library in C++11 onwards (version 2011 and up). Parallel for two processes here means both processes run overlapping in time.

In the next section, we provide a glimpse, or an extract of the realtime codes we wrote using RTAI libraries (written in C/C++). The RTAI library uses the prefixes (RT_ and rt_) exclusively, to denote constants, variables and functions that belong to its modules. The RTAI codes can intermingle with standard C/C++ codes. A few examples of RTAI codes are: rt_task_init(...), rt_make_hard_real_time(), rt_set_oneshot_mode(), rt_set_periodic_mode(), rt_task_delete(), start_rt_timer(), stop_rt_timer(), and so on. The RTAI library is huge.

RTAI programming is basically interrupt-based programming, as can be seen in the figure above. We will discuss this further in the section on literature review.

In a later section, we will provide some sample parallel running C/C++ multi-threading codes that we have used in our past projects. Since, the codes are also standard C/C++ codes, they can also intermingle with RTAI codes. Similarly, we will discuss this further in the section on literature review.

D 4.2 App4-C/C++ Code excerpt for Real Time (RTAI)

Listing 1.1: C/C++ Code excerpt for Real Time (RTAI)

```
1 // File: rtai-CNC14.c
2 // Date: May 05, 2018 10:04
3 // Author: WRY
4 //*
5 For full RTAI Ver 5.1 LOCAL DOCUMENTATION
6         file:///usr/realtime/share/doc/rtai-5.1/html/api/files.html
7         For <rtai.h>
8         file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai_8h_source.
            html
9         For <rtai_lxrt.h>
10        file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai__lxrt_8h.
            html
11 */
12 // ================================================================
13 // EXAMPLE USAGE OF RTAI FUNCTIONS (INTERRUPT-BASED)
14 // ================================================================
15 /*      rt_set_oneshot_mode();
16        rt_set_periodic_mode();
17        start_rt_timer();
18        rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1)
19        rt_task_init();
20        rt_task_make_periodic();
21        rt_make_hard_real_time();
22        rt_make_soft_real_time();
23        rt_allow_nonroot_hrt();
24        rt_task_delete(WRY_RT_task);
25        stop_rt_timer();
26 */
27 // Later on in program we make task periodic (change value to 0)
28        int             WRY_RT_task_status = 1;
29        static RTIME    WRYexpected;
30        static RTIME    WRYsampling_interval;
31
32 // SELECT REAL TIME TIMER PERIOD
33 // NANO SECONDS MEANS (10)^(-9) SEC PERIOD.
34        static RTIME    CNCtimer_period_ns = 1000*1000*1000;
35
36 // SET REAL TIME RUN MODE (BOTH CANNOT BE TRUE)
37        int      WRY_set_periodic_mode = 1; // TRUE  = 1
38        int             WRY_set_one_shot_mode = 0; // FALSE = 0
39
40 // GET NANOSECOND TIME
41        CNCunixtime2 = rt_get_time_ns();
42        gettimeofday(&TASKfinish,NULL)
43 ... and so on ;
```

1. For the summary of functions we created for program entry, the standard functions invoked in main(), please refer to the link [1.12] to the appendix.

2. For the full code listing, please refer to the link [1.13] to the appendix.

3. For RTAI compilation and terminal display of results on code execution, please refer to the link [1.14] to the appendix.

D 4.3 App4-CNC machine - First successful run (26 June, 2010)

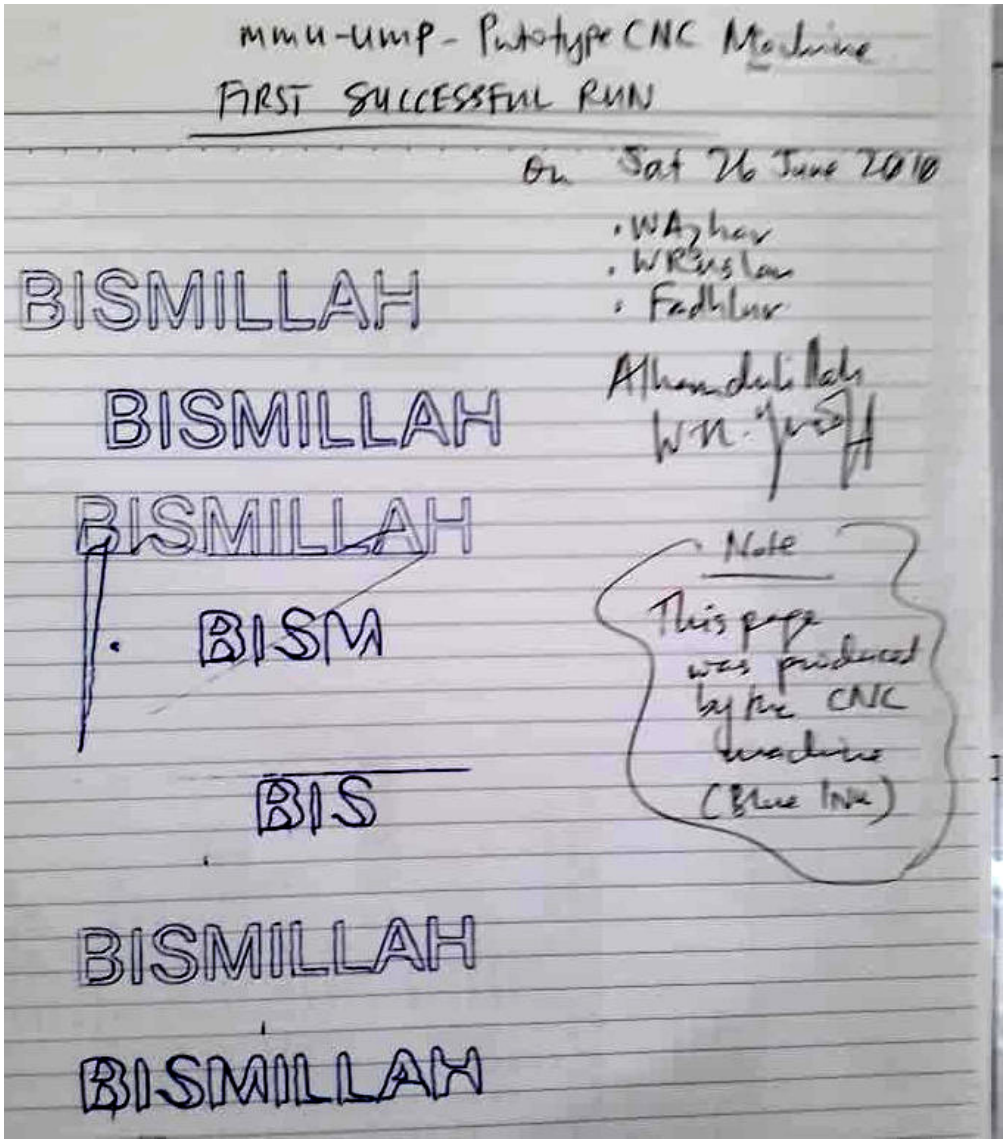


Figure 1.12: App4-Witnessed first successful run CNC machine on Sat, 26 June 2010

The University Malaysia Pahang (UMP) CNC Research Machine was delivered on Sat 04 December, 2009. The first successful run was conducted on Sat 26 June, 2010, less than 6 months later. The auspicious inauguration event was witnessed by three(3) team members from both UMP and MMU, as handwritten in the figure above.

The CNC drawing of BISMILLAH (meaning In The Name of Allah) took about 10 minutes to complete. The pulse generator device in use was the parallel port built-in on the motherboard of the computer. The operating system was Ubuntu 10.04 LTS 32-bit linux kernel 2.6.122 with RTAI version 2.7 modules installed. Today (as of 2018), we are using RTAI version 5.1 on our own compiled linux kernel ver 4.9.80 rtai, 64-bit configuration.

The CNC research machine was procured on a joint collaboration project between University Malaysia Pahang (UMP) and Multimedia University, Cyberjaya (MMU), in the year 2008, under the E-Science grant, Ministry of Science, Technology and Environment (MOSTE), Government of Malaysia.

D 4.4 App4-First succesful run CNC Research Machine

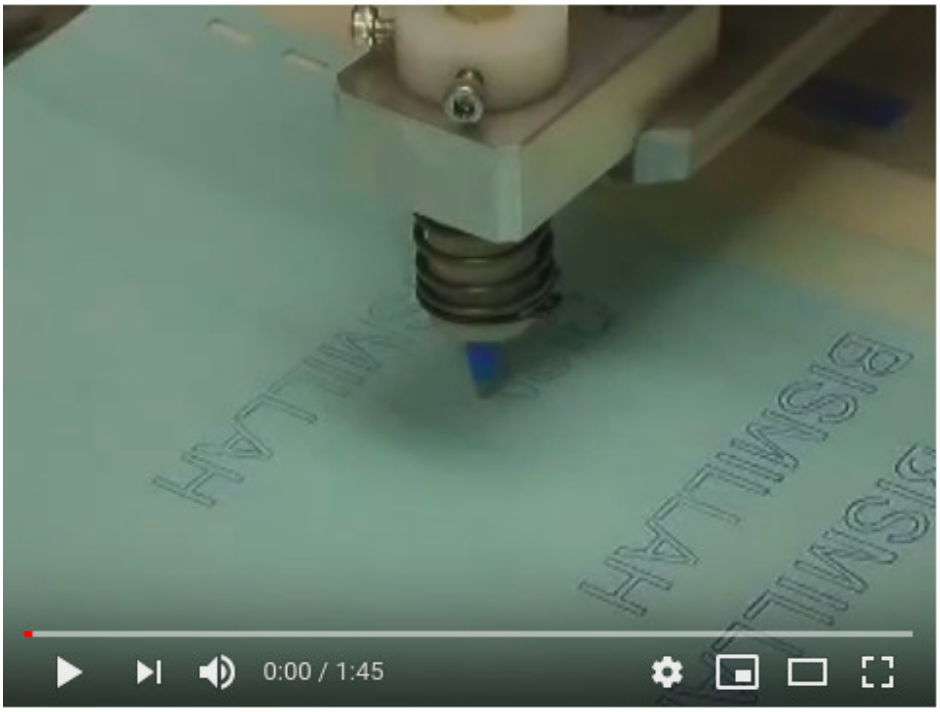


Figure 1.13: App4-BISMILLAH as first succesful run CNC Research Machine

Youtube video CNC Writing BISMILLAH (1:45 mins), link at https://www.youtube.com/watch?v=0CHSr0F0bjM, authored by wruslanhahaha, and published on Jun 5, 2011.

D 4.5 App4-Pulsing both X-Y axes simultaneously

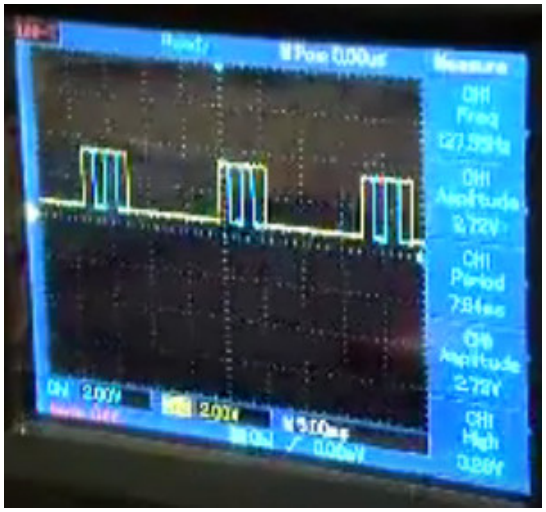


Figure 1.14: App4-Pulsing both X(blue) and Y(yellow) axes simultaneously

Both X-axis motor and Y-axis motor are receiving electrical pulses simultaneously tracing a path on the X-Y plane. The youtube link is at https://www.youtube.com/watch?v=I8IGaXkV7Qs, (0:46 mins) by WRY, published on Feb 16, 2015.

D 4.6 App4-Result 1 - Parallel port driving CNC Machine

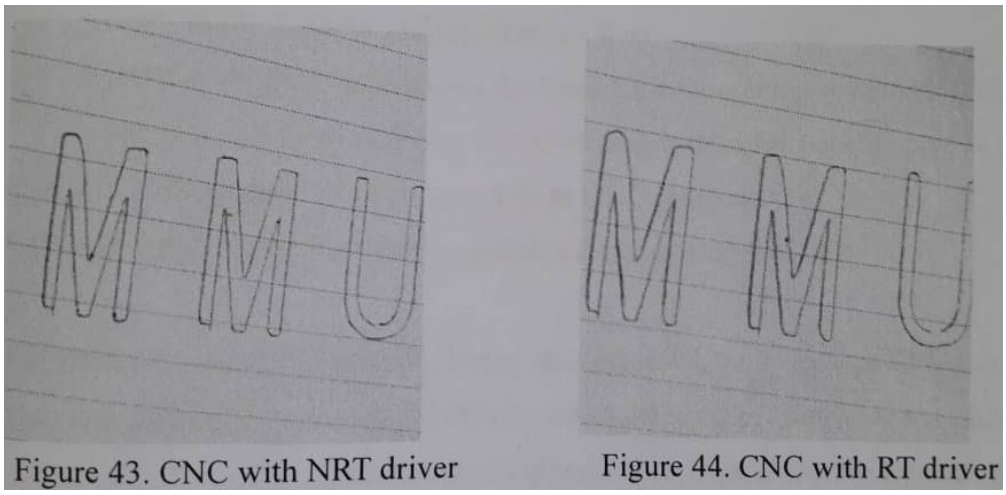


Figure 1.15: App4-Comparison Non-Realtime (NRT) versus Realtime (RT) pulse driving.

The two(2) figures above were drawn by the CNC Research Machine using the standard desktop parallel port as the pulse generator device for the MMU Final Year Project (Abzal-2012). The comparison was made by writing non-realtime (NRT) and realtime (RT), C/C++ CNC pulse driver program codes.

D 4.7 App4-Result 2 - Arduino Due USB driving CNC Machine

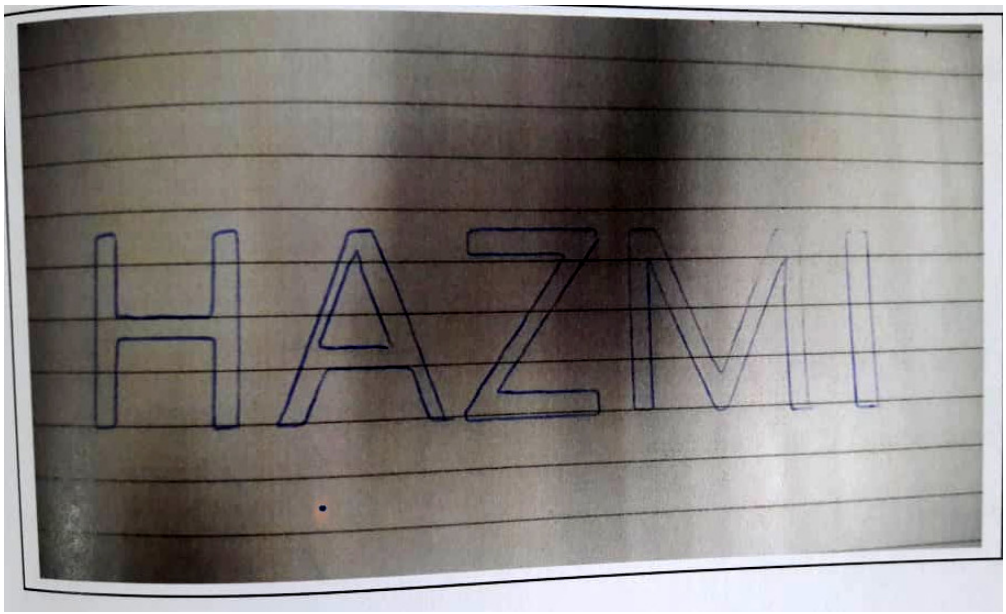


Figure 1.16: App4-Result 2 - Arduino Due USB driving CNC Research Machine

The figure above was drawn by the CNC Research Machine using the Arduino Due USB board as the pulse generator device for the MMU Final Year Project (Hazmi-2014).

D 4.8 App4-Result 3 - FPGA USB board driving CNC Machine

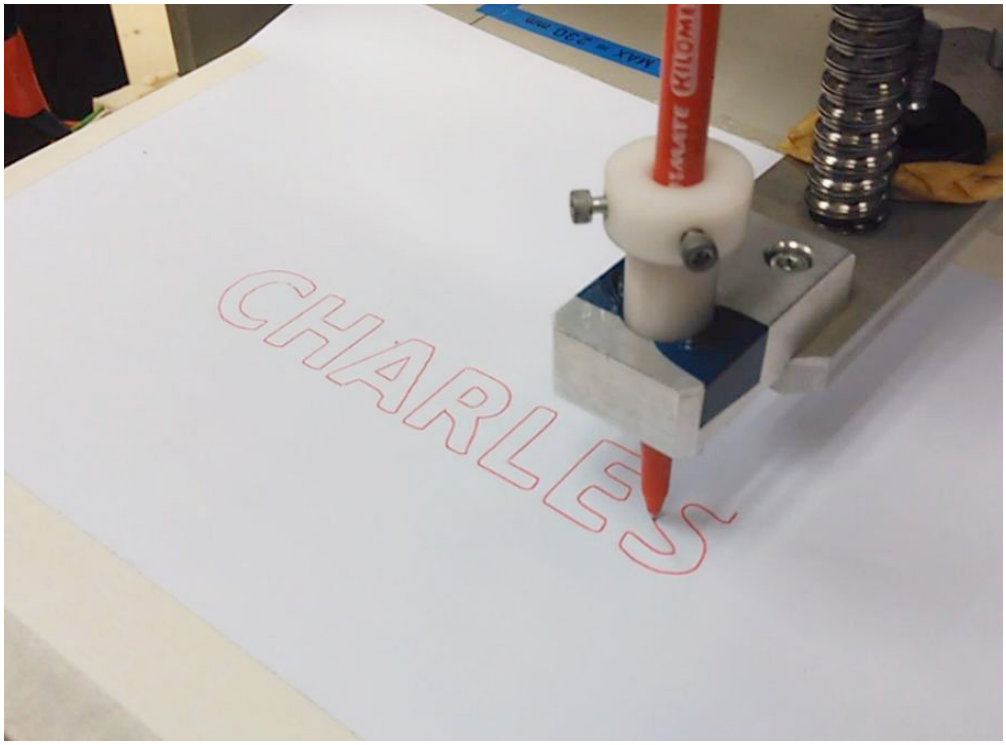


Figure 1.17: App4-Result 3 - FPGA USB driving CNC Research Machine

The figure above was drawn by the CNC Research Machine using the Digilent Nexsys3 Spartan6 Xilinc USB Extension Board as the pulse generator device for the MMU Final Year Project (Charles-2015). Youtube link at https://www.youtube.com/watch?v=3GuhE5dlcYk, Charles Teh, published on Feb 15, 2016.

D 4.9 App4-Youtube video LINKS made for CNC Machine

Listed below are some of the Youtube video links for the CNC Research Machine in operation. The videos are still available as at time of this writing.

1. CNC Jogging Test (4:00 mins)
   https://www.youtube.com/watch?v=gvlKlqfEXro
   Charles Teh, published on Oct 1, 2015.

2. FPGA USB Communication Test (1:33 mins)
   https://www.youtube.com/watch?v=9kf__vntpq4
   Charles Teh, published on Oct 22, 2015.

3. USB Data Streaming to FPGA Test (0:58 mins)
   https://www.youtube.com/watch?v=VvAyLqt_wLQ
   Charles Teh, published on Jan 28, 2016.

4. Tracing Bismillah in Arabic using LinuxCNC (4:28 mins)
   https://www.youtube.com/watch?v=cSgsf9p9wWs&feature=youtu.be
   Rajeef Selvaraja, published on Apr 10, 2015.

D 4.10 App4-Result 4 - Raspberry Pi-3 SBC on CNC machine

Figure 1.18: App4-Raspberry Pi 3 SBC driving CNC Research machine

The figure above was drawn by the CNC Research Machine using the Raspberry Pi 3 Single Board Computer (SBC) board as the pulse generator device for the MMU Final Year Project (Hafetzy-2017).

D 4.11 App4-Display pulses Counter-Clock-Wise (CCW) rotation

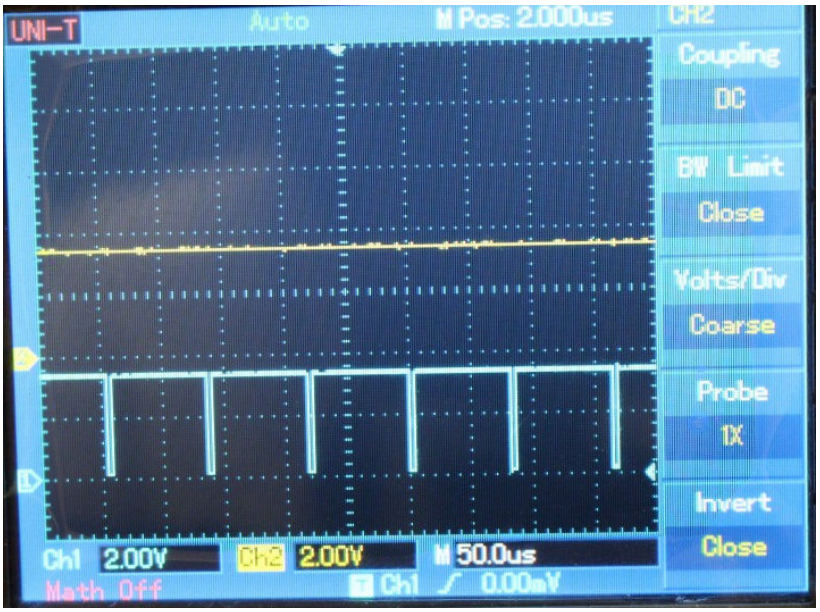


Figure 1.19: App4-Display pulses Counter-Clock-Wise (CCW) motor rotation

For motor counter-clock-wise rotation direction, yellow signal line is +5 volts above.

D 4.12 App4-Display pulses Clock-Wise (CW) motor rotation

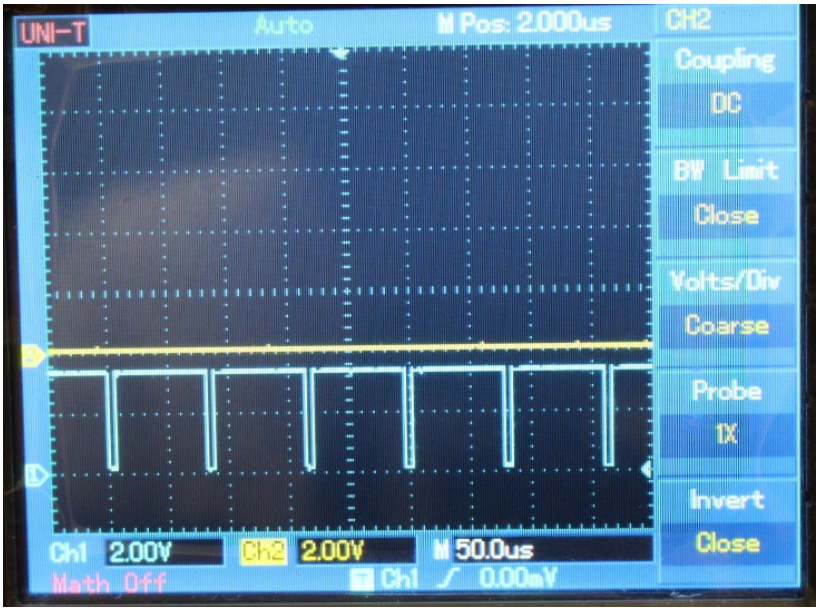


Figure 1.20: App4-Display pulses Clock-Wise (CW) motor rotation

For motor clock-wise rotation direction, yellow signal line is 0 volts above.

D 4.13 App4-Connection for Nexsys3 Spartan6 Xilinc USB Board

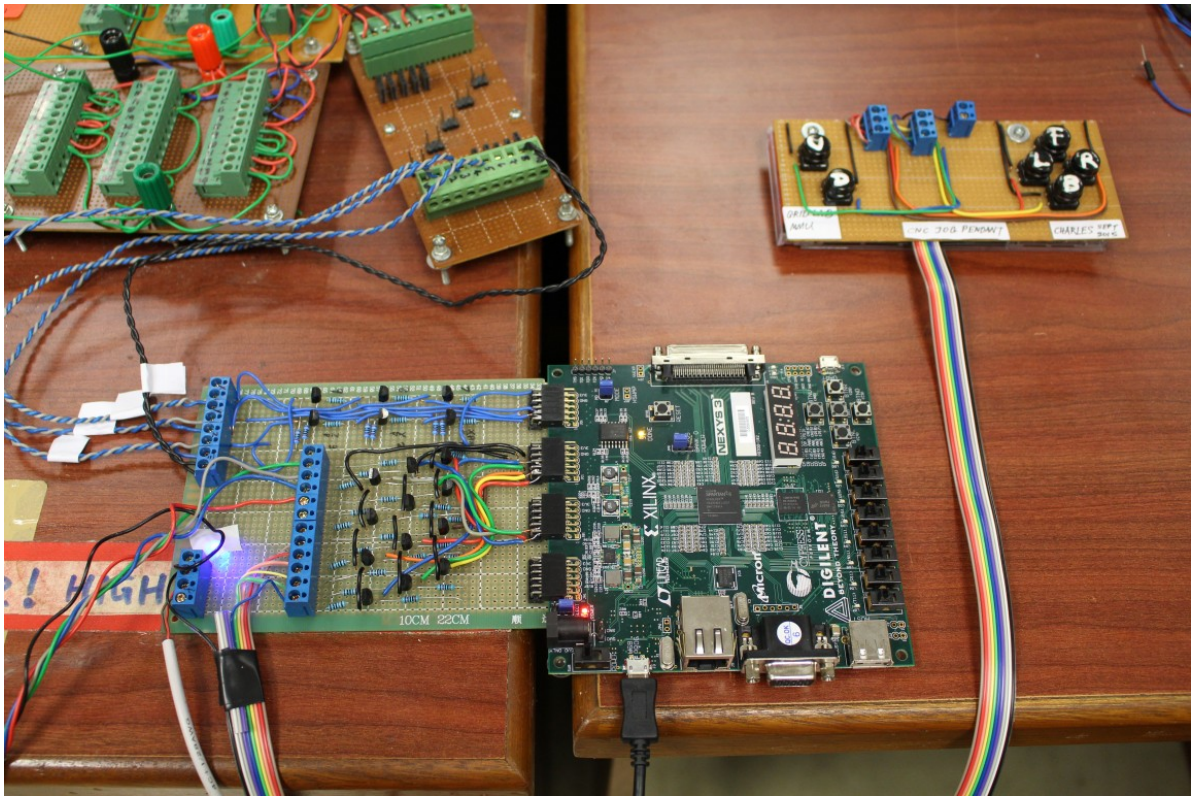


Figure 1.21: App4-Connection setup for Digilent Nexsys3 Spartan6 Xilinc board

The Digilent Nexys3 Spartan6 Xilinc board (bottom right) is connected via USB to a computer Notebook (bottom middle cable). The Digilent board is connected to the interface board (bottom left), which is then connected to the CNC Research Machine through a mini break-out board (top middle). The mini break-out board then connects to the servo-driver via the board (top left). A manual control switch board, called the pendant (top right), can manually drive the pen in the X-Y directions simultaneously on the CNC Research Machine, for homing or initialization tasks.

D 4.14 App4-Oscilloscope 3.3 volts pulses for CW motor rotation

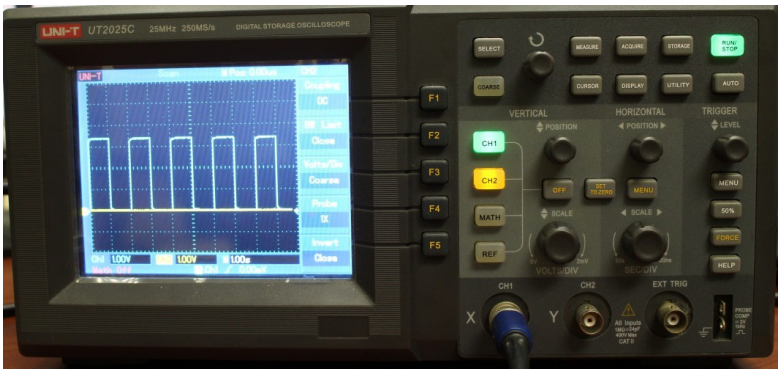


Figure 1.22: App4-Pulsing 3.3 volts display on a 2-channel digital capture oscilloscope

D 4.15 App4-G-Code versus CNC Signal file

charles.tap

```
N0010 G21
N0020 M6 T1
N0030 G00 Z5.0000
N0040 G00 X10.6141 Y18.0505
N0050 G00 Z0.0000
N0060 G01 X9.5718 Y17.9527 F600
N0070 G01 X8.5676 Y17.6571
N0080 G01 X7.6387 Y17.1746
N0090 G01 X6.8193 Y16.5230
N0100 G01 X6.1400 Y15.7265
N0110 G01 X5.6258 Y14.8147
N0120 G01 X5.2958 Y13.8213
N0130 G01 X5.1622 Y12.7830
N0140 G01 X4.9832 Y11.3445
N0150 G01 X5.0343 Y9.8958
N0160 G01 X5.3144 Y8.4735
N0170 G01 X5.8164 Y7.1136
N0180 G01 X6.5275 Y5.8504
N0190 G01 X7.3527 Y5.0297
N0200 G01 X8.3463 Y4.4237
N0210 G01 X9.4538 Y4.0658
N0220 G01 X10.6141 Y3.9756
N0230 G01 X12.0168 Y4.0935
N0240 G01 X13.3763 Y4.4585
N0250 G01 X16.0376 Y5.9782
N0260 G01 Y1.5896
N0270 G01 X14.9566 Y1.0261
N0280 G01 X13.8213 Y0.5823
```

temp.sig

```
00100000000000000000000000001004E200038401
00100A5E00064151011A10003ADB00000000000000
00100000000000000000000000001004E200038400
00100105000C29000001800843E210000000000000
001000FB000BAC000004A00279721000000000000
001000E8000B9350000790016310000000000000
001000CD000BE8C0000A3000EFA510000000000000
001000AA000CECD0000C7000B0AA10000000000000
00100081001 4D170000E4000BC7510000000000000
0010005300234C20000F8000BD0310000000000000
0010002100602D3000104000C3501000000000000
0010002D0061A80000168000C3501000000000000
0010000D01520A710016A000C23B10000000000000
00100046003B4A9100164000BA8810000000000000
0010007D00213401001540 00C3501000000000000
001000B2001492D10013C000B96C10000000000000
001000CE000BD9F1000CD000BE8C10000000000000
001000F8000BD03100097001366E10000000000000
00100115000B7531000590023A9310000000000000
00100122000BC9310001700949B710000000000000
0010015F000BD3110001D008F1DE00000000000000
00100154000C35010005B002D9BD00000000000000
00100299000C1D810017C001533A00000000000000
00100000000000000000449000C0491000000000000
0010010E000BC1400008D00168261000000000000
0010011C000C08F00006F001ECAE10000000000000
00100126000BA03000050002AB9810000000000000
0010012D000C2A9000030004C4B410000000000000
```

Figure 1.23: App4-G-Code (left) versus CNC Signal file (right)

D 4.16 App4-Two inputs lines for motor CW and CCW rotations

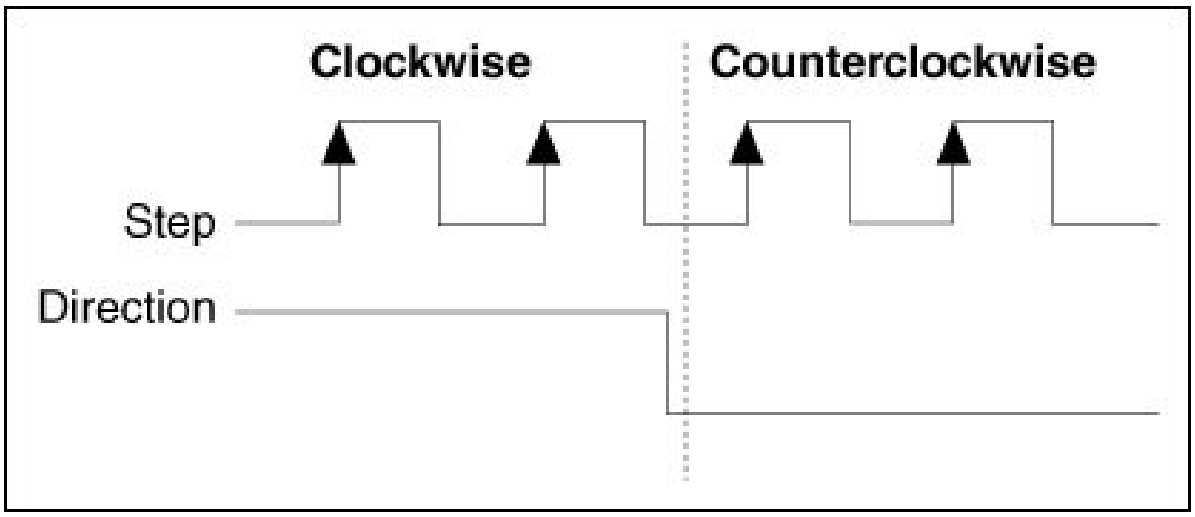
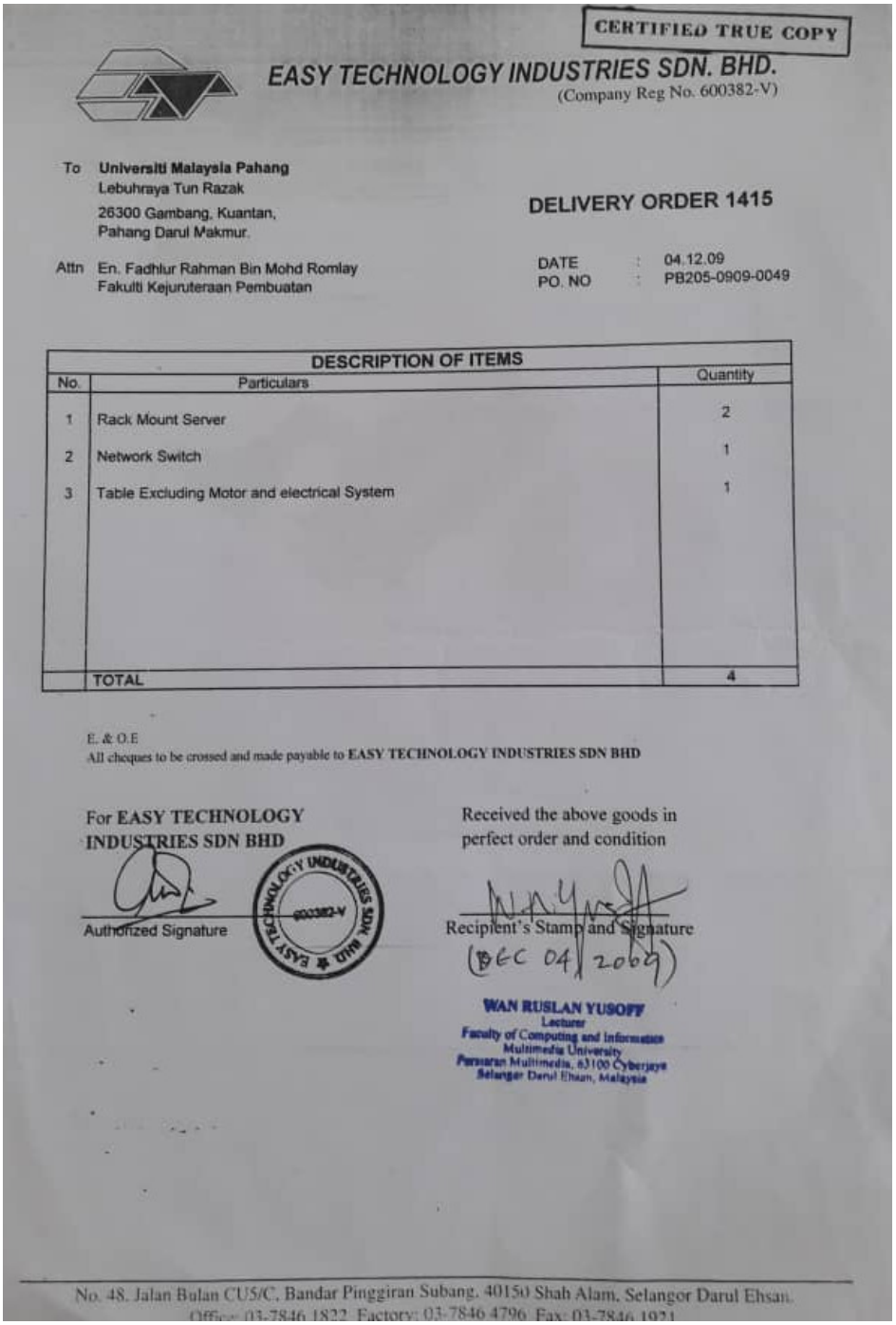


Figure 1.24: App4-Two inputs signal lines to servo-motor (CW and CCW) rotations

To drive each one of the CNC Research machine servo-driver and servo-motor pair, two(2) signal cables are required. One cable called STEP, provide a train of pulses at a rate corresponding to the pulse frequency (feedrate) to drive (rotation speed) of the electrical motor. Another cable, called DIRECTION, provides only one of two signal (high/low) states, and that state determines the direction of motor rotation (CW/CCW) or (clockwise/counterclockwise) For example high means CCW, low means CW. This is hardware dependent.

D 4.17 App4-Acceptance delivery of UMP CNC Research Machine



CERTIFIED TRUE COPY

EASY TECHNOLOGY INDUSTRIES SDN. BHD.
(Company Reg No. 600382-V)

To Universiti Malaysia Pahang
Lebuhraya Tun Razak
26300 Gambang, Kuantan,
Pahang Darul Makmur.

Attn En. Fadhlur Rahman Bin Mohd Romlay
Fakulti Kejuruteraan Pembuatan

DELIVERY ORDER 1415

DATE : 04.12.09
PO. NO : PB205-0909-0049

DESCRIPTION OF ITEMS		
No.	Particulars	Quantity
1	Rack Mount Server	2
2	Network Switch	1
3	Table Excluding Motor and electrical System	1
	TOTAL	4

E. & O.E
All cheques to be crossed and made payable to EASY TECHNOLOGY INDUSTRIES SDN BHD

For EASY TECHNOLOGY INDUSTRIES SDN BHD

Authorized Signature

Received the above goods in perfect order and condition

Recipient's Stamp and Signature
(DEC 04 2009)

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia

No. 48, Jalan Bulan CU5/C, Bandar Pinggiran Subang, 40150 Shah Alam, Selangor Darul Ehsan.
Office: 03-7846 1822 Factory: 03-7846 4796 Fax: 03-7846 1921

Figure 1.25: App4-Acceptance of UMP CNC Research Machine on Sat, 04 Dec 2009

D 4.18 App4-Parallel Port Devices

.

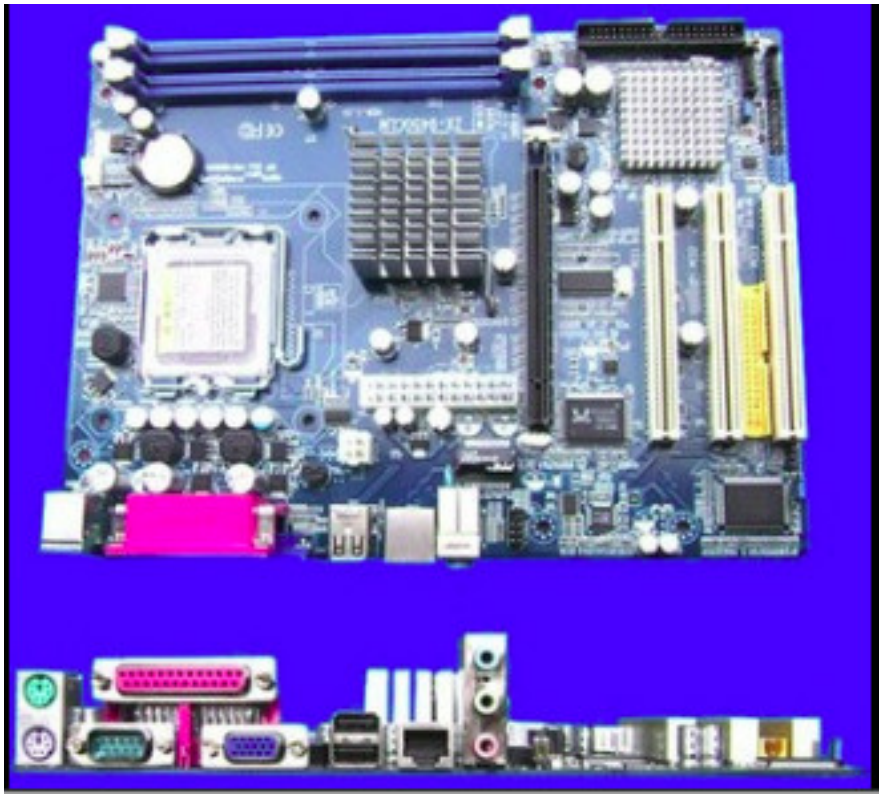


Figure 1.26: App4-Parallel Port device built-in on Motherboard (purple)

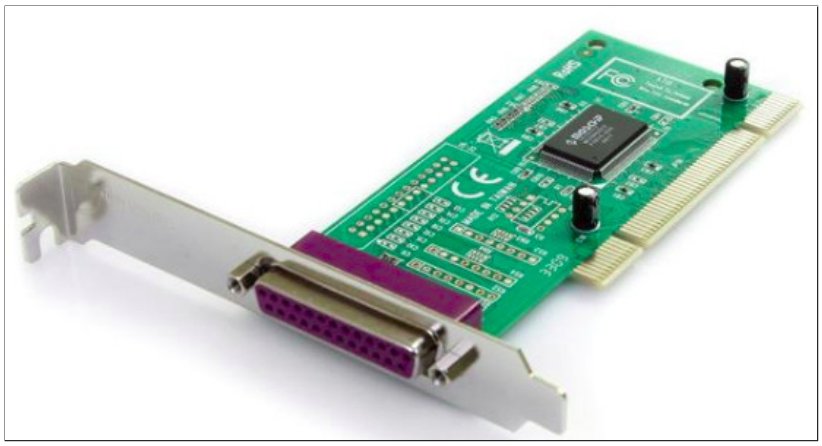


Figure 1.27: App4-Parallel Port PCI Adapter Card

The Netmos Nm9805 is a IEEE 1284 parallel port controller with PCI bus interface. Nm9805 fully supports the existing Centronics printer interface as well as PS/2, EPP, and ECP modes. Single 5V operation. Low power. PCI compatible 1284 printer port. Multi-mode compatible controller (SPP, PS2, EPP, ECP). Fast data rates up to 1.5 Mbytes/s (parallel port). Microsoft and Linux compatible.

Figure 1.28: App4-USB to Parallel Port PL2305 Converter Cable

Figure 1.29: App4-USB to-Serial and to-Parallel Idle(Left) and Writing(Right) Modes

LEFT PICTURE: The blue color LED status is for cable device idle and ready.
RIGHT PICTURE: The red color LED status is for cable device busy during active reading and writing modes.

Listing 1.2: App4-Detection of USB-to-Parallel Cable

```
1 Bus 001 Device 021: ID 067b:2305 Prolific Technology, Inc.
2 PL2305 Parallel Port  <=== FOUND
3 [ 3978.848284] usb 1-1.6: Manufacturer: Prolific Technology Inc
    .
4 [ 3978.855271] usblp 1-1.6:1.0: usblp0: USB Bidirectional
5 printer dev 21 if 0 alt 1 proto 2 vid 0x067B pid 0x2305
```

D 4.19 App4-Velleman K8000 Parallel Interface Board

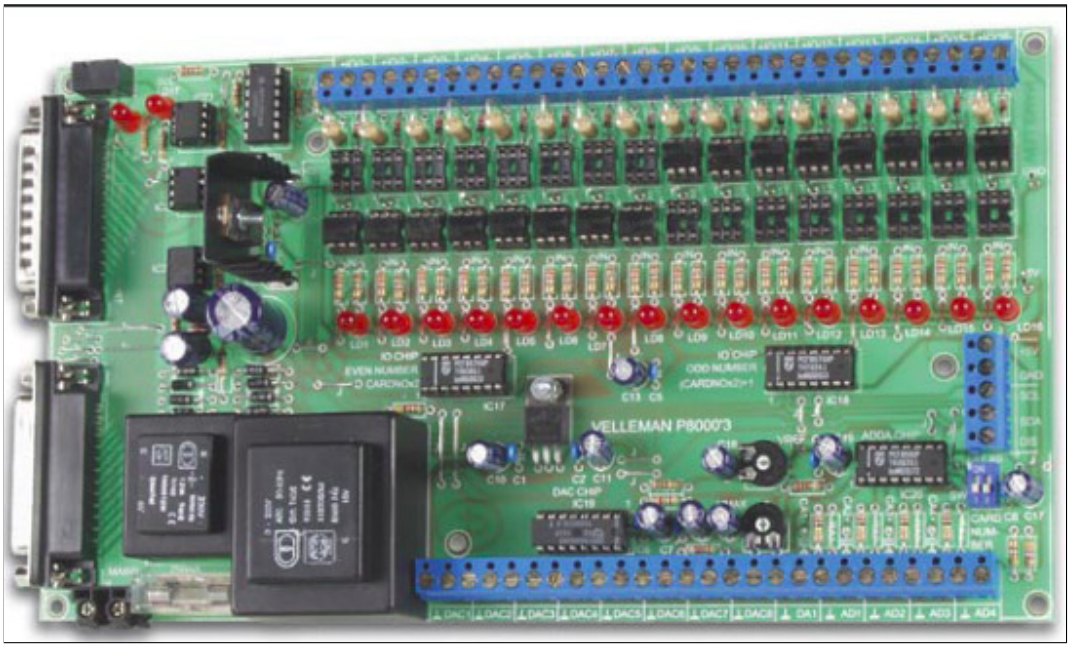


Figure 1.30: App4-Velleman K8000 Parallel Port Extension Board

Listing 1.3: App4-Specifications of Velleman K8000 Parallel Interface Board

```
DIGITAL OUTPUTS:
    optocoupler, open collector output: 50mA - max. 30VDC
DIGITAL INPUTS:
    optocoupler input: 5V/5mA, max. 20V/40mA
ANALOG OUTPUTS:
    8 outputs DAC1 to DAC8, resolution: 64 steps
    minimum output voltage: 0.1V at 2mA
    maximum output voltage: 11.5V adjustable at 2mA
    resolution per step from 0.1 to 11.5V: 160mV +/- 90mV
    1 precision output DA1, resolution: 256 steps
    minimum output voltage: 0V
    maximum output voltage: 4.5V adjustable at 0.5mA
    resolution per step from 0 to 4.5V: 17.5mV
ANALOG INPUTS:
    4 analogue inputs AD1 to AD4, resolution: 256 steps
    minimum input voltage: 0V
    maximum input voltage: 5V
    input impedance: 50Mohm
    resolution: 19.5mV
    communication protocol: I2Cbus
    LED indication for each I/O
BOARD:
    25 pin D series connector for computer
    25 pin D series connector for printer
    supply voltage: 230Vac
    PCB dimensions: 237 x 133mm (9.3" x 5.2")
```

D 4.20 App4-Velleman K8055 USB Interface Board

Figure 1.31: App4-Velleman K8055 USB Port Extension Board

Listing 1.4: App4-Specifications of Velleman K8055 USB Interface Board

```
DIGITAL INPUTS:
    5 digital inputs (0 = ground, 1 = open)
    (on board test buttons provided)
ANALOG INPUTS:
    2 analogue inputs with attenuation and amplification
    option (internal test +5V provided)
DIGITAL OUTPUTS:
    8 digital open collector output switches
    (max. 50V/100mA) (on board LED indication)
ANALOG OUTPUTS:
    2 analogue outputs:
    0 to 5V, output resistance 1K5
    PWM 0 to 100% open collector outputs
    max 100mA / 40V (on board LED indication)
    general conversion time: 20ms per command
BOARD:
    power supply through USB: approx. 70mA
    dimensions: 145 x 88 x 20mm / 5.7 x 3 x 0.8"
```

D 4.21 App4-Heber X10i USB Board

.

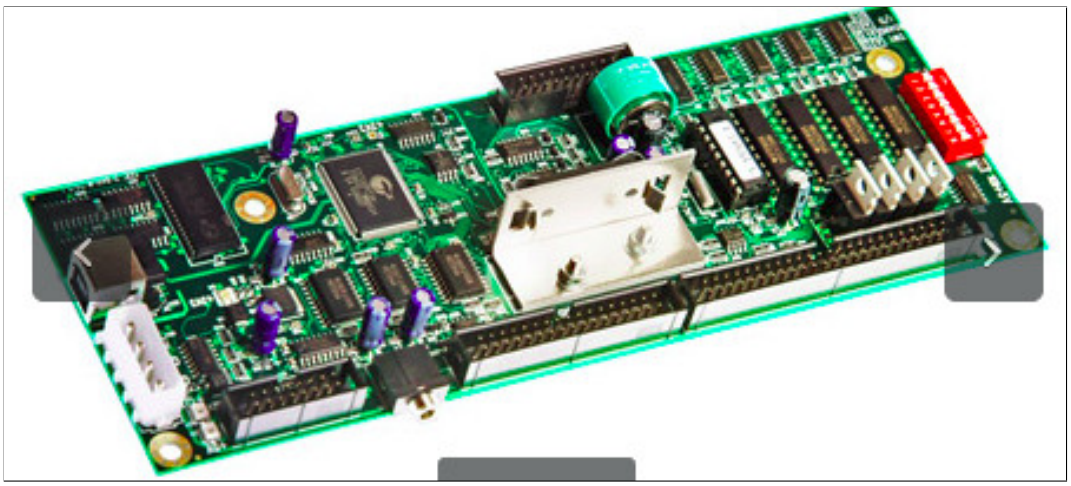


Figure 1.32: App4-Heber X10i USB Port Extension Board

Listing 1.5: App4-Specifications of Heber-X10i USB Interface Board

```
Heber X10i = USB real-time PC I/O controller
HARDWARE:
        64 Switched Inputs / Outputs
        Real-time I/O processor
        Battery backed SRAM
        Secure data retention
        DALLAS unique identifier
        Audio amp 5W RMS, Stereo audio amplifier
        Serial I/O
        SEC meter
        ccTalk
        LED control
        EEPROM 32KB
        Real time clock
        Current sensing 12v supply
        Open drain outputs for lamps and meters
        High current outputs
SOFTWARE:
        Serial I/O: RS232, TTL, ccTalk
        On-board security device
        Program in C, C+, C# or BlitzMax
        Microsoft Windows XPe / Linux / Raspberry Pi drivers
```

D 4.22 App4-Arduino Due USB Board

.

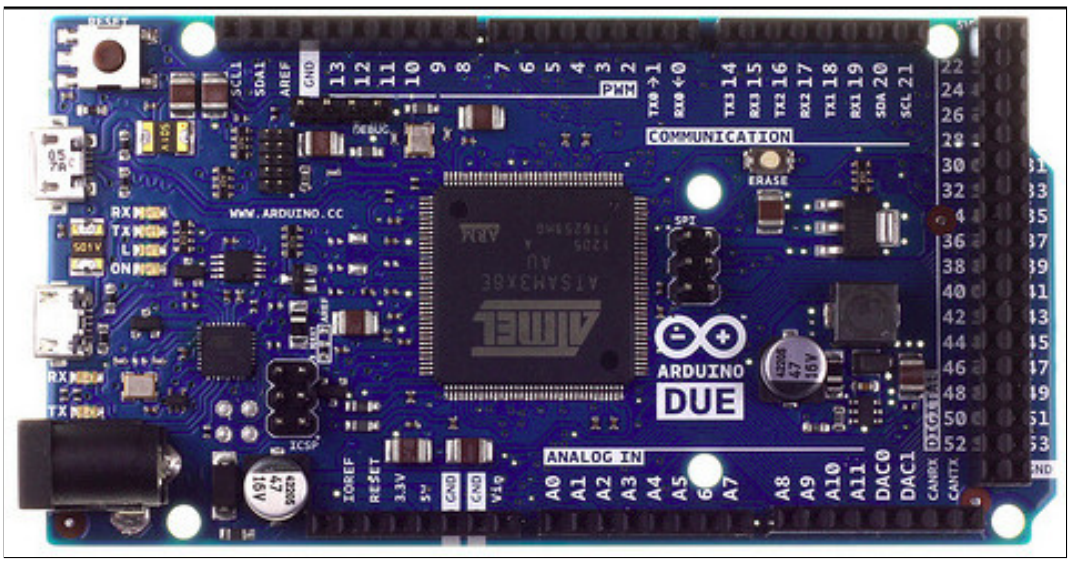


Figure 1.33: App4-Arduino Due USB Port Extension Board

Unlike most Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board.

The SAM3X provides one hardware UART and three hardware USARTs for TTL (3.3V) serial communication.

Listing 1.6: App4-Specifications of Arduino Due USB Interface Board

```
Microcontroller          AT91SAM3X8E
Operating Voltage        3.3V
Input Voltage (recommended) 7-12V
Input Voltage (limits)   6-16V
Digital I/O Pins         54 (of which 12 provide PWM output)
Analog Input Pins        12
Analog Output Pins       2 (DAC)
Total DC Output Current on all I/O lines        130 mA
DC Current for 3.3V Pin          800 mA
DC Current for 5V Pin    800 mA
Flash Memory     512 KB all available for the user applications
SRAM     96 KB (two banks: 64KB and 32KB)
Clock Speed      84 MHz
Length   101.52 mm
Width    53.3 mm
Weight   36 g
```

D 4.23 App4-Nexys-3 Spartan-6 FPGA USB Board

.

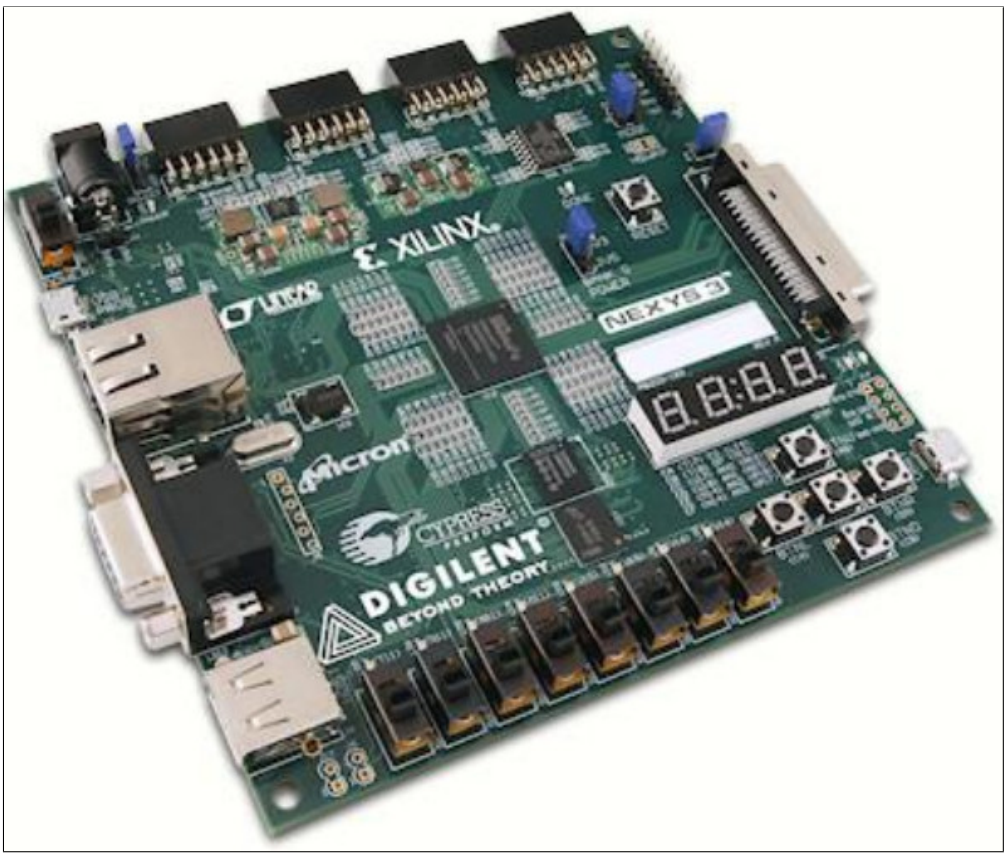


Figure 1.34: App4-Nexys-3 Spartan-6 FPGA USB Port Extension Board

Nexys 3 is compatible with all Xilinx CAD tools, including ChipScope, EDK, and the free WebPack. The Nexys 3 uses Digilent's newest Adept USB2 system that offers FPGA and ROM programming, automated board tests, virtual I/O, and simplified user data transfer facilities. In our project we used the Adept USB2 system.

Listing 1.7: App4-Specifications of Nexys-3 Spartan-6 FPGA USB Interface Board

```
1 Xilinx Spartan-6 LX16 FPGA in a 324 pin BGA package
2 16 Mbyte Cellular RAM (x16)
3 16Mbytes SPI (quad mode) PCM non volatile memory
4 16Mbytes parallel PCM non volatile memory
5 10/100 Ethernet PHY
6 On board USB2 port for programming and data xfer
7 USB UART and USB
8 HID port (for mouse/keyboard)
9 8 bit VGA port
10 100 MHz CMOS oscillator
11 72 I/Os routed to expansion connectors
12 GPIO includes 8 LEDs, 5 buttons, 8 slide switches and
13 4-digit seven segment display
14 USB v2 - programming cable included
```

D 4.24 App4-Raspberry Pi-3 Model-B Single Board Computer

.

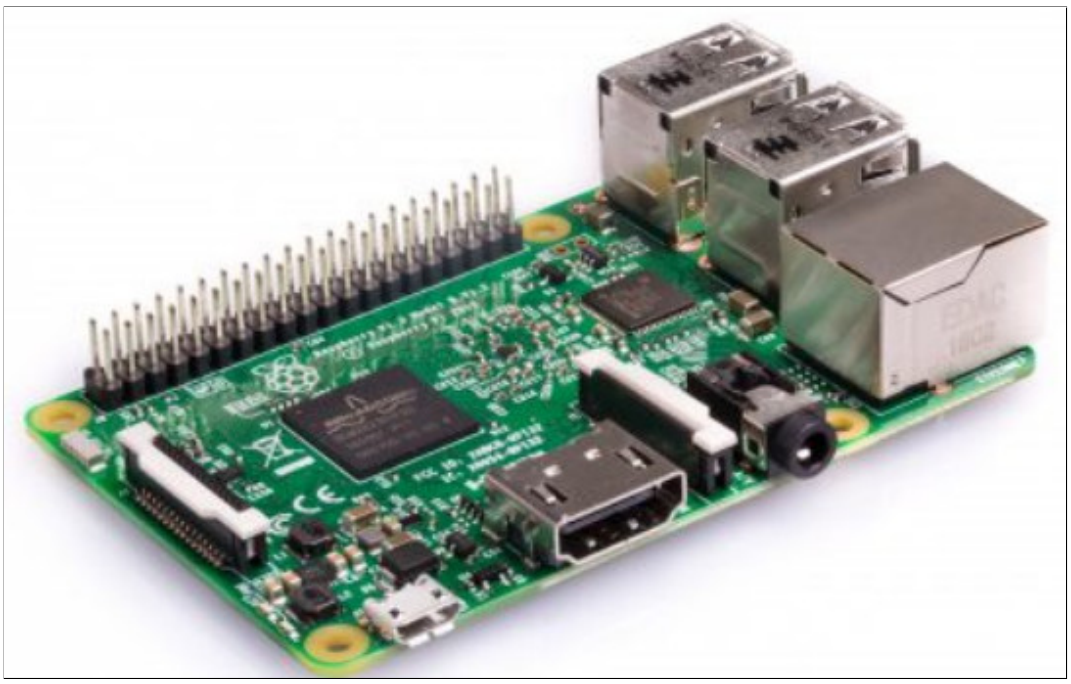


Figure 1.35: App4-Raspberry Pi-3 Model-B Single Board Computer

D 4.25 App4-Raspberry Pi-2 Model-B Single Board Computer

.

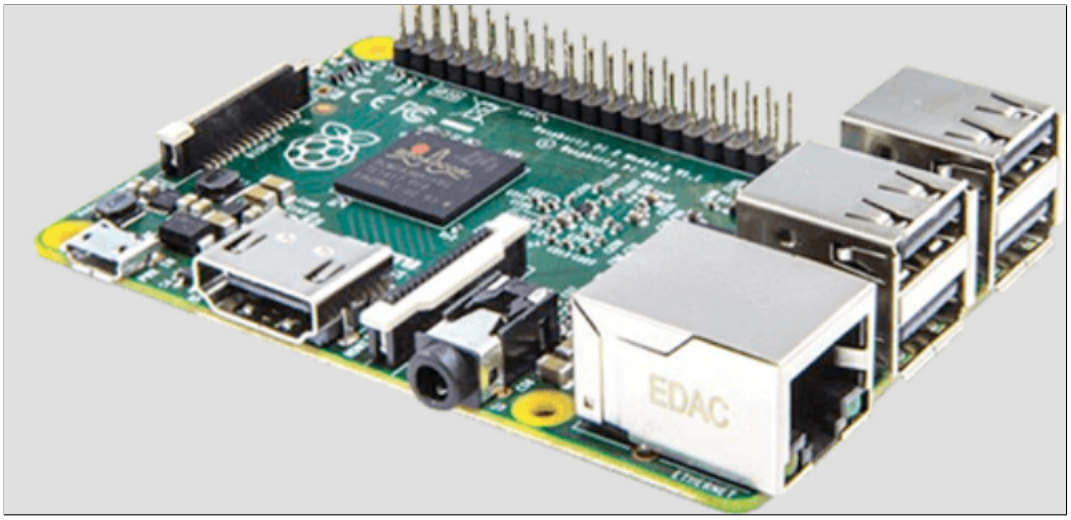


Figure 1.36: App4-Raspberry Pi-2 Model-B Single Board Computer

Listing 1.8: App4-Specifications of Raspberry Pi 3 SBC Board

```
SoC: Broadcom BCM2837
CPU: 4 x ARM Cortex-A53, 1.2GHz
GPU: Broadcom VideoCore IV
RAM: 1GB LPDDR2 (900 MHz)
Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage: microSD
GPIO: 40-pin header, populated
Ports: HDMI, 3.5mm analogue audio-video jack,
4 x USB 2.0, Ethernet, Camera Serial Interface (CSI),
Display Serial Interface (DSI)
And more ....
```

Listing 1.9: App4-Specifications of Raspberry Pi 2 SBC Board

```
SoC: Broadcom BCM2836 (CPU, GPU, DSP, SDRAM)
CPU: 900 MHz quad-core ARM Cortex A7 (ARMv7 instruction set)
GPU: Broadcom VideoCore IV @ 250 MHz
More GPU info: OpenGL ES 2.0 (24 GFLOPS); 1080p30 MPEG-2
and VC-1 decoder (with license), h.264/MPEG-4 AVC
high-profile decoder and encoder
Memory: 1 GB (shared with GPU)
USB ports: 4
Video input: 15-pin MIPI camera interface (CSI) connector
Video outputs: HDMI, composite video (PAL and NTSC)
via 3.5 mm jack
Audio input: I2S
Audio outputs: Analog via 3.5 mm jack; digital
via HDMI and I2S
Storage: MicroSD
Network: 10/100Mbps Ethernet
Peripherals: 17 GPIO plus specific functions, and HAT ID bus
Power rating: 800 mA (4.0 W)
Power source: 5 V via MicroUSB or GPIO header
Size: 85.60mm x 56.5mm
Weight: 45g (1.6 oz)
```

D 4.26 App4-Banana Pi M2U Single Board Computer

.

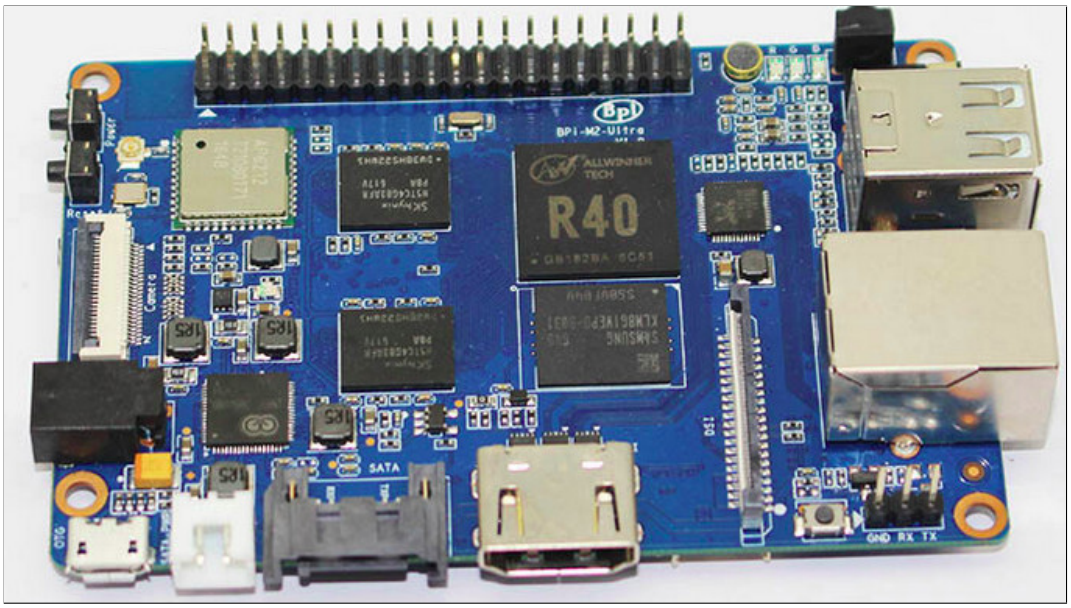


Figure 1.37: App4-Banana Pi M2U Single Board Computer

Listing 1.10: App4-Specifications of Banana Pi M2U SBC Board

```
Soc      Allwinner R40/V40
CPU      quad-core cortex-A7,the most power efficient CPU core
GPU      dual-core MALI-400 MP2 and runs at 500MHz,
GPU provides OpenGL ES 2.0, hardware-accelerated OpenVG,
1080p45 H.264 high-profile encode and decode.
SDRAM    2 GB DDR3 with 733MHz\(shared with GPU\)
SATA     suppoort SATA interface
GPIO 40 Pins Header, 28 x GPIO, for UART, I2C, SPI, PWM, I2S.
On board Network 10/100/1000Mbps Ethernet\(Realtek RTL8211E)
Wifi Module      WiFi 802.11 b/g/n \(AP 6212 module on board\)
Bluetooth        BT4.0, Storage MicroSD\(TF\) card, 8GB eMMC
Display 4-lane MIPI DSI display,or RGB panel, LVDS panel,
Video    Multi-format FHD video decoding, including Mpeg1/2,
Mpeg4, H.263, H.264, etc H.264 decode up to 1080P60,support
Audio outputs    HDMI, analog audio \(via 3.5 mm TRRS jack\)
Camera  A CSI input connector Camera:Supports 8-bit YUV422
CMOS sensor interface,Supports CCIR656 protocol for NTSC, PAL,
Supports 5M pixel camera sensor,Supports video capture
solution up to 1080p at 30fps
Audio    input On board microphone
USB      2 USB 2.0 host, 1 USB 2.0 OTG
Buttons Reset button, Power button, U-boot button
Leds     Power status Led and RJ45 Led
IR       onboard IR receiver
DC Power 5V/2A with micro USB port
battery 3.7V lithium battery power support
Sizes    85mmX56mm,same size as raspberry pi 3, Weight 40g
```

D 4.27 App4-Beagle-Board xM Single Board Computer

.

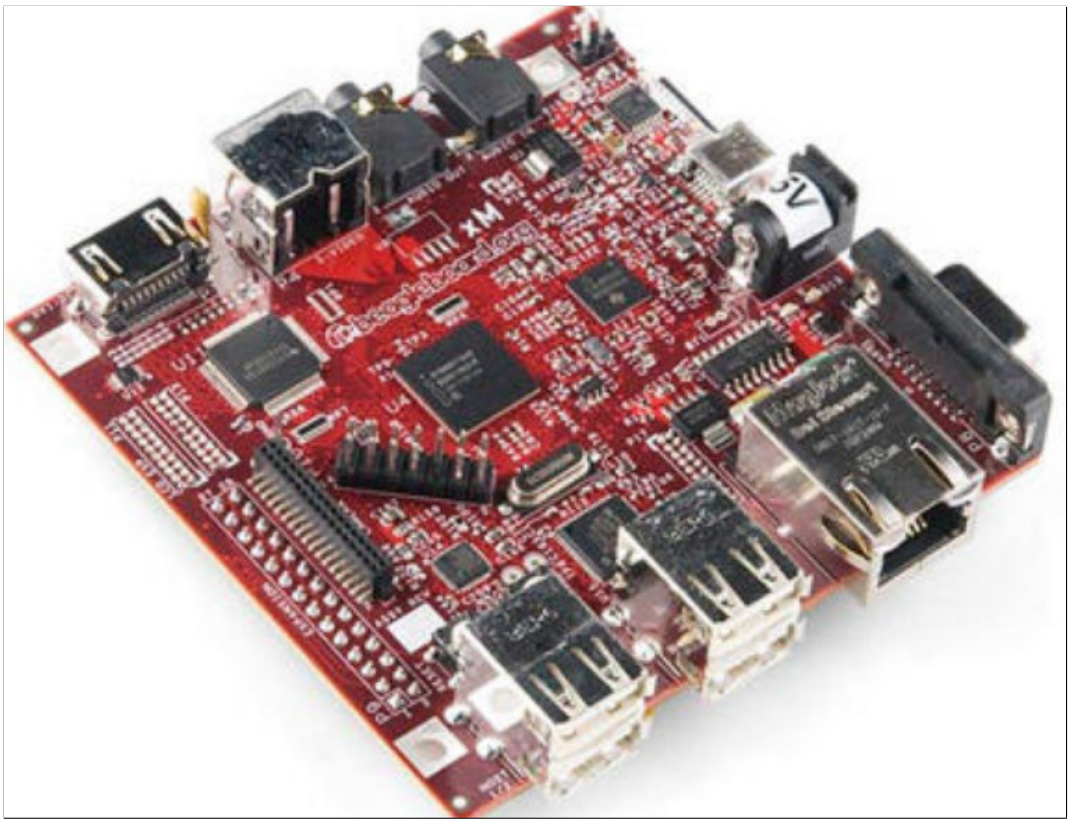


Figure 1.38: App4-Beagle-Board xM Single Board Computer

Listing 1.11: App4-Specifications of Beagle Board xM SBC Board

```
1   Processor TI DM3730 Processor 1 GHz ARM Cortex-A8 core
2   HD capable TMS320C64x+ core (800 MHz - 720p 30 fps)
3   Imagination Technologies PowerVR SGX 2D/3D
4   graphics processor supporting dual independent displays
5   512 MB LPDDR RAM, MicroSD/MMC card
6   4 GB microSD card supplied with the BeagleBoard-xM
7   and loaded with The Angstrom Distribution
8   DVI-D (HDMI connector chosen for size - maximum
9   resolution is 1400x1050)  S-Video
10  USB OTG (mini AB), 4 USB ports, Ethernet
11  Stereo in and out jacks
12  RS-232 port  JTAG connector
13  Power socket (5 V barrel connector type)
14  Camera port, Expansion port
15  Boot code stored on the uSD card, Boot from uSD/MMC only
16  Alternative Boot source button.
17  Has been demonstrated using Android,[18]
18  Angstrom Linux,[19] Fedora, Ubuntu, Gentoo,
19  Arch Linux ARM and Maemo Linux distributions,
20  FreeBSD, the Windows CE operating system, and RISC OS.
```

D 4.28 App4-Summary main() function C-Code listing for RTAI

Listing 1.12: App4-Summary main() function C-Code listing for RTAI

```
1
2// File: rtai-CNC14.c
3// Date: May 05, 2018 10:04
4// Author: WRY
5// ================================================================
6int main(int argc, char* argv[]) {
7// ================================================================
8WRY00_Print_DateTime_Usec();
9printf("Bismillah from WRY executed in main().\n\n");
10
11// FOR KEYBOARD INTERRUPT TO STOP (e.g. <Ctrl-C>) EXECUTION
12        signal(SIGTERM, WRYcleanup);
13        signal(SIGINT,  WRYcleanup);
14        signal(SIGKILL, WRYcleanup);
15// INVOKE FUNCTIONS SEQUENTIALLY TO RUN RTAI
16        system("./load-rtai-modules.sh"); // LOAD RTAI KERNEL MODULES
17        WRY01_Create_and_Initialize_Real_Time_Task_LXRT();
18        WRY02_Get_Address_and_Name_of_Real_Time_Task_LXRT();
19        WRY31_Make_Hard_Real_Timer_Run();
20        WRY32_Check_Selected_Hard_Real_Time_Run_Mode();
21        WRY33_Execute_Selected_Hard_Real_Time_Run_Mode();
22        WRY04_Allow_NonRoot_Users_For_Hard_Real_Time();
23        WRY05_Lock_Memory_Swapping_For_Hard_Real_Time();
24        WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ();
25        WRY07_ParallelPort_Check_RT_Task_Running_Mode();
26        WRY08_Start_Run_CNC_Machine();
27        WRY09_Stop_Run_CNC_Machine();
28        system("./unload-rtai-modules.sh"); // UNLOAD RTAI KERNEL MODULES
29
30WRY00_Print_DateTime_Usec();
31printf("Alhamdulillah from WRY executed in main().\n");
32return (0);
33}
```

D 4.29 App4-Full C-Code listing for Real Time (RTAI)

Listing 1.13: App4-Full C-Code listing for Real Time (RTAI)

```
1 // File: rtai-CNC14.c
2 // Date: May 05, 2018 10:04
3 // Author: WRY
4 //
5 // ================================================================
6 // (1) The prefixes WRY and TASK were used extensively in
7 // order to differentiate our own defined variables and functions
8 // from standard C/C++ and RTAI libraries defined variables and functions.
9 //
10 // (2) We must either link our C/C++ program against liblxrt or we
11 // have to configure RTAI with "CONFIG_RTAI_LXRT_STATIC_INLINE=y".
12 // during linux kernel compilation.
13 //
14 // ================================================================
15 // STANDARD C/C++ HEADER FILES USED
16 // ================================================================
17
18       #include <sched.h>      // Scheduler
19       #include <stdio.h>      // Standard Input/Output
20       #include <stdlib.h>     // Standard Library
21       #include <time.h>       // For local date-time with usec
22       #include <signal.h>     // Signal for user interrupts, <Ctrl>-C to stop
23       #include <fcntl.h>      // For file control
24       #include <math.h>       // Mathematical functions
25       #include <unistd.h>     // Low level constants, types, function declarations
26       #include <errno.h>      // Printing C errors
27       #include <string.h>     // Handling strlen IN gcode . h
28       #include <curses.h>     // Handling getch () , wgetch ()
29       #include <sys/ioctl.h>  // System Input/Output control
30       #include <sys/io.h>     // System Input/Output
31       #include <sys/mman.h>   // System Memory Manager
32       #include <sys/time.h>   // For system local date-time with usec
33
34
35
```

```
36
37 // ============================================================
38 // REALTIME HEADER FILES (RTAI)
39 // ============================================================
40         #include <rtai.h>                // RTAI configuration switches (User space and Kernel space)
41         #include <rtai_lxrt.h>           // RTAI User Space libraries
42
43 // ============================================================
44 // RTAI VER. 5.1 DOCUMENTATION REFERENCES
45 // ============================================================
46 /*
47         For full RTAI Ver 5.1 LOCAL DOCUMENTATION
48                 file:///usr/realtime/share/doc/rtai-5.1/html/api/files.html
49         For <rtai.h>
50                 file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai_8h_source.html
51         For <rtai_lxrt.h>
52                 file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai__lxrt_8h.html
53         For <rtai_sem.h>
54                 file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai__sem_8h.html
55         For <rtai_sched.h>
56                 file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai__sched_8h_source.html
57         For <rtai_usi.h>
58                 file:///usr/realtime/share/doc/rtai-5.1/html/api/rtai__usi_8h_source.html
59 */
60 // ============================================================
61 // EXAMPLE USAGE OF RTAI FUNCTIONS (INTERRUPT-BASED)
62 // ============================================================
63 /*       rt_set_oneshot_mode();
64         rt_set_periodic_mode();
65         start_rt_timer();
66         rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1)
67         rt_task_init();
68         rt_task_make_periodic();
69         rt_make_hard_real_time();
70         rt_make_soft_real_time();
71         rt_allow_nonroot_hrt();
72         rt_task_delete(WRY_RT_task);
73         stop_rt_timer();
74 */
```

```
75
76 // ============================================================
77 // RTAI DEFINED FUNCTIONS
78 // ============================================================
79 /*
80         static void *           rt_get_adr (unsigned long name);
81                 Get an object address by its name.
82                 Returns the address associated to name on success, 0 on failure
83
84         static unsigned long    rt_get_name (void *adr);
85                 Get an object name by its address.
86                 Returns the address associated to name on success, 0 on failure
87
88         static RT_TASK *        rt_task_init (unsigned long name, int priority, int stack_size, int max_msg_size);
89                 Create an RTAI task extension for a Linux process/task in user space.
90
91         static void             rt_make_soft_real_time (void);
92                 Return a hard real time Linux process, or pthread to the standard Linux behavior.
93
94         static void             rt_make_hard_real_time (void);
95                 Give a Linux process, or pthread, hard real time execution capabilities allowing full kernel
                    preemption
96
97         static void     rt_allow_nonroot_hrt (void);
98                 Allows a non root user to use the Linux POSIX soft real time process management and memory lock
                    functions, and allows it to do any input-output operation from user space.
99 */
100 // ============================================================
101 // GLOBAL DEFINITIONS
102 // ============================================================
103         #define PERIOD          500000          // nanoseconds
104         #define TICK_TIME       1000000         // nanoseconds
105         // #define CPUMAP       0xF             // 16-cpus
106         #define CPUMAP          0x4             // 4-cpus
107
108         // BUILT_IN PARALLEL PORT ON MOTHERBOARD
109         // #define PARPORT_IRQ  7
110         // #define BASEPORT     0x378
111
```

```
112         // HARDWARE 1 = parport0: PC-style at 0xd010 (0xd000), irq 18
113         #define BASEPORT        0xd010
114         #define PARPORT_IRQ     18
115
116         // HARDWARE 2 = parport1: PC-style at 0xe100, irq 17
117         // #define BASEPORT     0xe100
118         // #define PARPORT_IRQ  17
119
120         static volatile int ovr, intcnt, retval, maxcnt;
121
122 // ========================================================
123 // HARDWARE DEVICES PARALLEL PORT SEARCH
124 // ========================================================
125 /* EXECUTE SYSTEM FUNCTION
126 root@dell-ub1604-64b:/home/wruslan# dmesg | grep parp
127
128 KERNEL MESSAGE - FOUND PARALLEL PORT NO. 1
129         [    8.989048] parport0: PC-style at 0xd010 (0xd000), irq 18, using FIFO [PCSPP,TRISTATE,COMPAT,EPP,ECP]
130         [    9.089563] lp0: using parport0 (interrupt-driven).
131
132 KERNEL MESSAGE - FOUND PARALLEL PORT NO. 2
133         [   11.972056] parport1: PC-style at 0xe100, irq 17 [PCSPP,TRISTATE]
134         [   12.073309] lp1: using parport1 (interrupt-driven).
135
136 root@dell-ub1604-64b:/home/wruslan# lspci -v
137
138 LIST PCI DEVICE - FOUND PARALLEL PORT NO. 1
139         03:00.0 Parallel controller: Oxford Semiconductor Ltd Device c110 (prog-if 02 [ECP])
140         Subsystem: Oxford Semiconductor Ltd Device c110
141         Flags: bus master, fast devsel, latency 0, IRQ 18
142         I/O ports at d010 [size=8]
143         I/O ports at d000 [size=4]
144         Capabilities: [40] Power Management version 3
145         Capabilities: [50] MSI: Enable- Count=1/1 Maskable- 64bit+
146         Capabilities: [70] Express Legacy Endpoint, MSI 00
147         Capabilities: [100] Device Serial Number 00-30-e0-11-11-00-01-10
148         Capabilities: [110] Power Budgeting <?>
149         Kernel driver in use: parport_pc
150         Kernel modules: parport_pc
```

```
151
152
153LIST PCI DEVICE - FOUND PARALLEL PORT NO. 2
154         02:00.0 Serial controller: Device 1c00:3250 (rev 10) (prog-if 05 [16850])
155         Subsystem: Device 1c00:3250
156         Flags: fast devsel, IRQ 17
157         I/O ports at e000 [size=256]
158         Memory at f0100000 (32-bit, prefetchable) [size=32K]
159         I/O ports at e100 [size=4]
160         Expansion ROM at f7c00000 [disabled] [size=32K]
161         Capabilities: [60] Power Management version 3
162         Capabilities: [68] MSI: Enable- Count=1/32 Maskable+ 64bit+
163         Capabilities: [80] Express Legacy Endpoint, MSI 00
164         Capabilities: [100] Advanced Error Reporting
165         Kernel driver in use: parport_serial
166         Kernel modules: parport_serial
167
168*/
169// ========================================================
170// REAL TIME OBJECTS
171// ========================================================
172         static RT_TASK*          WRYtask;
173         static RT_TASK*          WRY_RT_task;
174         struct sched_param       WRYsched;
175         struct timeval           WRYstart, WRYfinish;
176         unsigned long            WRY_RT_task_name;
177
178// ========================================================
179// TIMING FOR GENERAL CNC EXECUTION
180// ========================================================
181         static RTIME             CNCunixtime1, CNCunixtime2;
182         static RTIME             CNCstart_time_ns, CNCcurrent_time_ns;
183         static RTIME             CNCend_time_ns, CNCrun_duration_ns;
184         static RTIME             CNCtimer_period_ns; // timer period, in nanoseconds
185         RTIME                    CNCtimer_period_count; //actual timer period, in counts
186
187
188
189
```

```
190 // ================================================================
191 // TIMING FOR INDIVIDUAL TASK EXECUTION
192 // ================================================================
193         struct timeval          TASKstart, TASKfinish;
194         long int                TASKusec_duration;
195         double                  TASKsec_duration;
196
197 // ================================================================
198 // GLOBAL VARIABLES (With KEYBOARD Interrupt)
199 // ================================================================
200         int                     WRYpriority     = 0; // Highest
201         int                     WRYstack_size   = 0; // Using default (512)
202         int                     WRYmax_msg_size = 0; // Using default (256)
203         int                     WRYpolicy       = SCHED_FIFO;
204         int                     WRYcpus_allowed = CPUMAP;
205         long int                WRYusec_duration;
206         double                  WRYsec_duration ;
207         int                     WRYsig;  // Keyboard interrupt signal <Ctrl>-C
208         static volatile int     ovr, intcnt, retval, maxcnt;
209
210 // ================================================================
211 // DEFINE REALTIME TASK STATUS AND SETTINGS
212 // ================================================================
213 // REFERNCE: Initially task status set to non-periodic (value non-zero)
214 // Later on in program we make task periodic (change value to 0)
215
216         int             WRY_RT_task_status = 1;
217         static RTIME    WRYexpected;
218         static RTIME    WRYsampling_interval;
219
220 // SELECT REAL TIME TIMER PERIOD
221         static RTIME    CNCtimer_period_ns = 1000*1000*1000; // means (1 sec) period
222
223 // SET REAL TIME RUN MODE (BOTH CANNOT BE TRUE)
224         int             WRY_set_periodic_mode = 1; // TRUE  = 1
225         int             WRY_set_one_shot_mode = 0; // FALSE = 0
226
227 // SELECTED RUN MODE VALUES (PERIODIC == 1, ONE-SHOT == 2)
228         int             WRY_selected_run_mode;
```

```
229
230 // TASK PERIODIC STATUS VALUES (PERIODIC == 0, NON-PERIODIC != 0)
231         int             WRY_task_periodic_status;
232
233 // TASK ONE-SHOT STATUS VALUES (ONE-SHOT ???, NON-ONE-SHOT ???)
234         int             WRY_task_one_shot_status;
235
236
237
238
239 // ================================================================
240 // GLOBAL DATA STRUCTURES
241 // ================================================================
242 // Used for WRY00_Print_DateTime_Usec(void) ONLY in order
243 // to avoid repetitive declarations (reused)
244
245         time_t          WRYtimer;
246         char            WRYbuffer[26];
247         struct tm*      WRYtm_info;
248         struct timeval  WRYtval_now;
249
250
251
252
253 // ================================================================
254 void WRY00_Print_DateTime_Usec(void) {
255 // ================================================================
256 // EXECUTIONS
257         time(&WRYtimer);
258         WRYtm_info = localtime(&WRYtimer);
259         strftime(WRYbuffer, 26, "%Y-%m-%d %H:%M:%S", WRYtm_info);
260         gettimeofday(&WRYtval_now, NULL);
261         printf("%s", WRYbuffer);
262         printf(".%09ld \t", (long int)WRYtval_now.tv_usec);
263 }
264
265
266
267
```

```
268 // ===================================================
269 void WRY01_Create_and_Initialize_Real_Time_Task_LXRT(void) {
270 // ===================================================
271 WRY00_Print_DateTime_Usec();
272 printf("STARTED  WRY01_Create_and_Initialize_Real_Time_Task-LXRT(void).\n");
273
274 // Create an RTAI task extension for a Linux process/task in user space.
275
276         if (!(WRY_RT_task = rt_task_init(nam2num("CNC06"), WRYpriority, WRYstack_size, WRYmax_msg_size))) {
277                 WRY00_Print_DateTime_Usec();
278                 printf("ERROR : Cannot initialize real time task LXRT.\n");
279                 WRY00_Print_DateTime_Usec();
280                 printf("ERROR DESCRIPTION : %s\n", strerror(errno));
281                 exit(1);
282         } else {
283                 WRY00_Print_DateTime_Usec();
284                 printf("SUCCESS: Completed real time task LXRT initialization.\n");
285         }
286
287 WRY00_Print_DateTime_Usec();
288 printf("FINISHED WRY01_Create_and_Initialize_Real_Time_Task-LXRT(void).\n\n");
289 }
290
291 // ===================================================
292 void WRY02_Get_Address_and_Name_of_Real_Time_Task_LXRT(void) {
293 // ===================================================
294 WRY00_Print_DateTime_Usec();
295 printf("STARTED  WRY02_Get_Address_and_Name_of_Real_Time_Task_LXRT(void).\n");
296
297         // GET RT_TASK NAME BY ADDRESS
298         if (rt_get_name(WRY_RT_task) == 0) {
299                 WRY00_Print_DateTime_Usec();
300                 printf("ERROR : Cannot get name of real time task LXRT.\n");
301                 WRY00_Print_DateTime_Usec();
302                 printf("ERROR DESCRIPTION : %s\n", strerror(errno));
303         //      exit(1);
304         } else {
305                 WRY00_Print_DateTime_Usec();
306                 printf("SUCCESS: NAME of real time task LXRT = %p\n", (void *)WRY_RT_task);
```

```
307                 WRY_RT_task_name = (unsigned long)WRY_RT_task;
308         }
309
310 WRY00_Print_DateTime_Usec();
311 printf("FINISHED WRY02_Get_Address_and_Name_of_Real_Time_Task_LXRT(void).\n\n");
312 }
313
314 // ===================================================
315 void WRY31_Make_Hard_Real_Timer_Run(void) {
316 // ===================================================
317 WRY00_Print_DateTime_Usec();
318 printf("STARTED  WRY31_Make_Hard_Real_Timer_Run(void).\n");
319
320         // EXECUTE HARD REAL TIMER CLOCK
321         rt_make_hard_real_time();
322
323         // IF HARD REAL TIME ALREADY RUNNING, NOTIFY STATUS
324         if (rt_is_hard_timer_running()) {
325                 WRY00_Print_DateTime_Usec();
326                 printf("SUCCESS: Hard real timer is already running.\n");
327         } else {
328                 // IF NOT RUNNING, START WHILE LOOP
329                 while (!(rt_is_hard_timer_running())) {
330
331                         // EXECUTE HARD REAL TIMER CLOCK
332                         rt_make_hard_real_time();
333                         // IF HARD REAL TIMER IS NOT RUNNING, THEN RUN IT BY LOOPING.
334                         if (rt_is_hard_timer_running()) {
335                                 WRY00_Print_DateTime_Usec();
336                                 printf("SUCCESS: Hard real timer is NOW running.\n");
337                         } else {
338                                 WRY00_Print_DateTime_Usec();
339                                 printf("FAILED : Hard real timer is NOT YET running.\n");
340                                 WRY00_Print_DateTime_Usec();
341                                 printf("ERROR DESCRIPTION : %s\n", strerror(errno));
342                                 // exit(1); // DISABLED TO KEEP ON TRYING
343                         } END if else
344                 } // END while loop
345         } // END if else
```

```
346
347 WRY00_Print_DateTime_Usec();
348 printf("FINISHED WRY31_Make_Hard_Real_Timer_Run(void).\n\n");
349 }
350
351 // ============================================================
352 void WRY32_Check_Selected_Hard_Real_Time_Run_Mode(void) {
353 // ============================================================
354 WRY00_Print_DateTime_Usec();
355 printf("STARTED  WRY32_Check_Selected_Hard_Real_Time_Run_Mode(void).\n");
356
357         // BOTH RUN MODES SELECTED
358         if ((WRY_set_periodic_mode == 1) && (WRY_set_one_shot_mode == 1)) {
359                 WRY00_Print_DateTime_Usec();
360                 printf("ERROR : Cannot select both periodic_mode and one-shot mode.\n");
361                 exit(1);
362                 }
363         // NO RUN MODE SELECTED
364         if ((WRY_set_periodic_mode == 0) && (WRY_set_one_shot_mode == 0)) {
365                 WRY00_Print_DateTime_Usec();
366                 printf("ERROR : No hard realtime run mode selected.\n");
367                 exit(1);
368                 }
369         // PERIODIC MODE SELECTED
370         if ((WRY_set_periodic_mode == 1) && (WRY_set_one_shot_mode == 0)) {
371                 WRY00_Print_DateTime_Usec();
372                 printf("SUCCESS: PERIODIC hard realtime run mode selected.\n");
373                 WRY_selected_run_mode = 1;
374                 }
375         // ONE-SHOT MODE SELECTED
376         if ((WRY_set_periodic_mode == 0) && (WRY_set_one_shot_mode == 1)) {
377                 WRY00_Print_DateTime_Usec();
378                 printf("SUCCESS: ONE-SHOT hard realtime run mode selected.\n");
379                 WRY_selected_run_mode = 2;
380                 }
381 WRY00_Print_DateTime_Usec();
382 printf("FINISHED WRY32_Check_Selected_Hard_Real_Time_Run_Mode(void).\n\n");
383 }
384
```

```
385 // ===================================================
386 void WRY33_Execute_Selected_Hard_Real_Time_Run_Mode(void) {
387 // ===================================================
388 WRY00_Print_DateTime_Usec();
389 printf("STARTED  WRY33_Execute_Selected_Hard_Real_Time_Run_Mode(void).\n");
390
391         // FOR PERIODIC-MODE REAL TIME RUN
392         if (WRY_selected_run_mode == 1) {
393
394                 WRY00_Print_DateTime_Usec();
395                 printf("SUCCESS: PERIODIC Set CNCtimer_period_ns \t= %lld \n", CNCtimer_period_ns);
396
397                 rt_set_periodic_mode();
398                 WRY00_Print_DateTime_Usec();
399                 printf("SUCCESS: PERIODIC Execute rt_set_periodic_mode().\n");
400
401                 CNCtimer_period_count=nano2count(CNCtimer_period_ns);
402                 WRY00_Print_DateTime_Usec();
403                 printf("SUCCESS: PERIODIC Execute nano2count(CNCtimer_period_ns).\n");
404
405                 WRY00_Print_DateTime_Usec();
406                 printf("SUCCESS: PERIODIC CNCtimer_period_count \t= %lld \n", CNCtimer_period_count);
407
408                 start_rt_timer(CNCtimer_period_count);
409                 WRY00_Print_DateTime_Usec();
410                 printf("SUCCESS: PERIODIC Execute start_rt_timer(CNCtimer_period_count).\n");
411
412                 rt_task_make_periodic(WRY_RT_task, WRYexpected, WRYsampling_interval);
413                 WRY00_Print_DateTime_Usec();
414                 printf("SUCCESS: PERIODIC Execute rt_task_make_periodic(WRY_RT_task, x, x).\n");
415
416         } // END if PERIODIC-MODE
417
418         // FOR ONE-SHOT MODE REAL TIME RUN
419         if (WRY_selected_run_mode == 2) {
420
421                 rt_set_oneshot_mode();
422                 WRY00_Print_DateTime_Usec();
423                 printf("SUCCESS: ONE-SHOT Execute rt_set_oneshot_mode().\n");
```

```
424
425                 start_rt_timer(0);
426                 WRY00_Print_DateTime_Usec();
427                 printf("SUCCESS: ONE-SHOT Execute start_rt_timer(0).\n");
428
429         } // END if ONE-SHOT MODE
430
431 WRY00_Print_DateTime_Usec();
432 printf("FINISHED WRY33_Execute_Selected_Hard_Real_Time_Run_Mode(void).\n\n");
433 }
434
435 // ========================================================
436 void WRY04_Allow_NonRoot_Users_For_Hard_Real_Time(void) {
437 // ========================================================
438 WRY00_Print_DateTime_Usec();
439 printf("STARTED  WRY04_Allow_NonRoot_Users_For_Hard_Real_Time(void).\n");
440
441         // ALLOW NON-ROOT USERS TO HARD REAL TIME
442         rt_allow_nonroot_hrt();
443         WRY00_Print_DateTime_Usec();
444         printf("SUCCESS: Execute Allow hard realtime for non-root users.\n");
445
446 WRY00_Print_DateTime_Usec();
447 printf("FINISHED WRY04_Allow_NonRoot_Users_For_Hard_Real_Time(void).\n\n");
448 }
449 // ========================================================
450 void WRY05_Lock_Memory_Swapping_For_Hard_Real_Time(void) {
451 // ========================================================
452 WRY00_Print_DateTime_Usec();
453 printf("STARTED  WRY05_Lock_Memory_Swapping_For_Hard_Real_Time(void).\n");
454
455         // LOCK RAM MEMORY FROM MEMORY SWAPPING
456         mlockall(MCL_CURRENT | MCL_FUTURE);
457         WRY00_Print_DateTime_Usec();
458         printf ("SUCCESS: Execute Lock memory and now no memory swapping for hard realtime.\n") ;
459
460 WRY00_Print_DateTime_Usec();
461 printf("FINISHED WRY05_Lock_Memory_Swapping_For_Hard_Real_Time(void).\n\n");
462 }
```

```
463
464 // ===================================================================
465 void WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ(void) {
466 // ===================================================================
467 WRY00_Print_DateTime_Usec();
468 printf("STARTED  WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ(void).\n");
469
470         // The name says it, IRQs managed in user space
471         // REQUEST IRQ FOR PARALLEL PORT TASK (WRY_RT_task)
472                 rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1);
473
474                 if(!(rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1))) {
475                         WRY00_Print_DateTime_Usec();
476                         printf("ERROR  : rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1).\n");
477                         WRY00_Print_DateTime_Usec();
478                         printf("ERROR DESCRIPTION : %s\n", strerror(errno));
479                         exit(1); // Commented during testing only
480                 } else {
481                         WRY00_Print_DateTime_Usec();
482                         printf("SUCCESS: Display PARPORT_IRQ \t= %d\n", PARPORT_IRQ);
483
484                         WRY00_Print_DateTime_Usec();
485                         printf("SUCCESS: Display RT_IRQ_TASK \t= %d\n", RT_IRQ_TASK);
486
487                         WRY00_Print_DateTime_Usec();
488                         printf("SUCCESS: Display BASEPORT \t= 0x%02X\n", BASEPORT);
489
490                         WRY00_Print_DateTime_Usec();
491                         printf("SUCCESS: Display CPUMAP \t= 0x%02X\n", CPUMAP);
492
493                         WRY00_Print_DateTime_Usec();
494                         printf("SUCCESS: Display PERIOD \t= %d (ns)\n", PERIOD);
495
496                         WRY00_Print_DateTime_Usec();
497                         printf("SUCCESS: Display TICK_TIME \t= %d (ns)\n", TICK_TIME);
498
499                         WRY00_Print_DateTime_Usec();
500                         printf("SUCCESS: Execute rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1).\n");
501                 } // END if else (rt_request)
```

```
502
503
504 /* REFERENCE NOTES: ================
505                     https://www.rtai.org/userfiles/documentation/magma/html/api/group__hal.html#ga74
506
507                     Often some of the above functions do equivalent things. Once more there is no way of doing it right
                          except by knowing the hardware you are manipulating. Furthermore you must also remember that when
                          you install a hard real time handler the related interrupt is usually disabled, unless you are
                          overtaking one already owned by Linux which has been enabled by it. Recall that if have done it
                          right, and interrupts do not show up, it is likely you have just to rt_enable_irq() your irq.
508
509         TO SOLVE PROBLEM FOR RTAI VER 5.1 = BOTH LINES BELOW ARE NOT NEEDED
510                     rt_startup_irq(PARPORT_IRQ);
511                     rt_enable_irq(PARPORT_IRQ);
512================ END REFERENCE NOTES */
513
514 WRY00_Print_DateTime_Usec();
515 printf("FINISHED  WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ(void).\n\n");
516 }
517
518 // ========================================================
519 void   WRY07_ParallelPort_Check_RT_Task_Running_Mode(void) {
520 // ========================================================
521 WRY00_Print_DateTime_Usec();
522 printf("STARTED   WRY07_ParallelPort_Check_RT_Task_Running_Mode(void).\n");
523
524         WRY00_Print_DateTime_Usec();
525         printf("SUCCESS: Display WRY_selected_run_mode = %d\n", WRY_selected_run_mode);
526
527         // FOR RUN PERIODIC MODE SELECTED
528         if (WRY_selected_run_mode == 1) {
529
530                 WRY_task_periodic_status = rt_task_make_periodic(WRY_RT_task, WRYexpected, WRYsampling_interval);
531
532                 // IF TASK RUNNING IN PERIODIC MODE
533                 if (WRY_task_periodic_status == 0) {
534                         WRY00_Print_DateTime_Usec();
535                         printf("SUCCESS: Display WRY_RT_task is ALREADY running in periodic mode.\n");
536                 } else {
```

```
537                         // LOOP WHILE NOT RUNNING IN PERIODIC MODE
538                         // MAKE RT_TASK RUN IN PERIODIC MODE
539                         while (WRY_task_periodic_status != 0) {
540
541                                 WRY_task_periodic_status = rt_task_make_periodic(WRY_RT_task, WRYexpected,
                                      WRYsampling_interval);
542                                 WRY00_Print_DateTime_Usec();
543                                 printf("SUCCESS: Execute rt_task_make_periodic(WRY_RT_task, WRYexpected,
                                      WRYsampling_interval).\n");
544
545                                 if (WRY_task_periodic_status == 0) {
546                                         WRY00_Print_DateTime_Usec();
547                                         printf("SUCCESS: Display WRY_RT_task is NOW running in periodic mode.\n");
548                                 } // END if
549
550                         } // END while loop MAKE TASK PERIODIC
551                 } // END if..else TASK RUNNING
552         } END if SELECTED MODE
553
554/* REFERENCE NOTES: ================
555
556         int rt_task_make_periodic(RT_TASK* task, RTIME start_time, RTIME period)
557                 Make a task run periodically.
558
559         rt_task_make_periodic mark the task task, previously created with rt_task_init(), as suitable for a periodic
              execution, with period period, when rt_task_wait_period() is called.
560         The time of first execution is defined through start_time or start_delay. start_time is an absolute value
              measured in clock ticks. start_delay is relative to the current time and measured in nanoseconds.
561
562         Parameters:
563                 task    is a pointer to the task you want to make periodic.
564                 start_time      is the absolute time to wait before the task start running, in clock ticks.
565                 period  corresponds to the period of the task, in clock ticks.
566
567         Return values:
568                 0       on success. A negative value on failure as described below:
569                 EINVAL: task does not refer to a valid task.
570================== END REFERENCE NOTES  */
571
```

```
572 WRY00_Print_DateTime_Usec();
573 printf("FINISHED WRY07_ParallelPort_Check_RT_Task_Running_Mode(void).\n\n");
574 }
575
576 // ============================================================
577 void WRY08_Start_Run_CNC_Machine(void) {
578 // ============================================================
579 WRY00_Print_DateTime_Usec();
580 printf("STARTED  WRY08_Start_Run_CNC_Machine(void).\n");
581
582         // MAP START CNC RUNTIME IN NANO SECONDS
583         CNCstart_time_ns = rt_get_time_ns();
584         WRY00_Print_DateTime_Usec();
585         printf("SUCCESS: Start CNC run at time \t= %lld (ns)\n", CNCstart_time_ns);
586
587         // EXAMPLE EXECUTE OF CNC Signals File RUN HERE
588         int x;
589
590         // START FOR ... CONTROL LOOP THAT READS CNC SIGNALS FILE AND WRITE LINE-BY-LINE
591         // THIS FOR LOOP IS A TEST FOR 10 LINES TO WRITE
592         for (x = 0; x < 10; x++) {
593
594                 // START TIMER
595                 gettimeofday(&TASKstart, NULL); CNCunixtime1 = rt_get_time_ns();
596
597                 // EXECUTE TEST RUN IMPORTANT
598                 // NOTE: rt_sleep suspends execution of the caller task for  a time of delay internal count units.
                      During this time  the CPU is used by other tasks. TIME: 1000000000 (ns) = 2893422000 (internal
                      counts)
599
600                 // FOR COMPILATION TESTING ONLY
601                 // DIFFERENT HARDWARE DEVICES WILL WRITE DIFFERENTLY
602                 outb_p(25, BASEPORT);    // THIS IS WRITING FOR PARALLEL PORT ON LINUX
603
604                 WRY00_Print_DateTime_Usec();
605                 printf("TEST WRITING outb_p(25, BASEPORT) \n");
606
607                 // THIS SLEEP TIME IS FOR PUSLE FEEDRATE
608                 // rt_sleep(1000000000);
```

```
609                 rt_sleep(2893422000);
610
611                 WRY00_Print_DateTime_Usec();
612                 printf("run rt_sleep(2893422000) internal counts for x = %d ", x);
613
614                 // END TIMER
615                 CNCunixtime2 = rt_get_time_ns(); gettimeofday(&TASKfinish,NULL);
616                 printf("run duration = %lld (ns)\n", (CNCunixtime2 - CNCunixtime1));
617
618         } // END FOR ... CONTROL LOOP (MEANS G-CODE PROCESSING ENDED)
619
620         CNCend_time_ns = rt_get_time_ns();
621         WRY00_Print_DateTime_Usec();
622         printf("SUCCESS: End CNC run at time \t= %lld (ns)\n", CNCend_time_ns);
623
624         CNCrun_duration_ns = (CNCend_time_ns - CNCstart_time_ns);
625         WRY00_Print_DateTime_Usec();
626         printf("SUCCESS: CNC total run duration\t= %lld (ns)\n", CNCrun_duration_ns);
627
628
629 /* REFERENCE NOTES ==================
630
631         CONSIDER RTAI SLEEP AND
632                 rt_get_time_ns();    Nanoseconds timing in RTAI
633                 rt_sleep(1000000);   Equivalent to 1 second (rt_sleep is defined in micro seconds).
634                 rt_task_wait_period();
635
636         REALTIME DRIVER FOR PARALLEL PORT OUTPUTS, MODIFIED FOR LINUX BY WRY
637         REFER: Phase1D-CNC-RTOS - Report Page 161 of 204.
638
639         outb_p(valout_pulse + on , BASEPORT);
640         usleep(700);
641         outb_p(valout_dir + on , BASEPORT);
642         usleep(700);
643
644         OUTB(2)                 Linux Programmer's Manual                 OUTB(2)
645
646         NAME
647
```

```
648         outb, outw, outl, outsb, outsw, outsl, inb, inw, inl, insb,
649         insw, insl, outb_p, outw_p, outl_p, inb_p, inw_p, inl_p - port I/O
650
651         SYNOPSIS
652         #include <sys/io.h>
653
654         unsigned char inb(unsigned short int port);
655         unsigned char inb_p(unsigned short int port);
656         unsigned short int inw(unsigned short int port);
657         unsigned short int inw_p(unsigned short int port);
658         unsigned int inl(unsigned short int port);
659         unsigned int inl_p(unsigned short int port);
660
661         void outb(unsigned char value, unsigned short int port);
662         void outb_p(unsigned char value, unsigned short int port);
663         void outw(unsigned short int value, unsigned short int port);
664         void outw_p(unsigned short int value, unsigned short int port);
665         void outl(unsigned int value, unsigned short int port);
666         void outl_p(unsigned int value, unsigned short int port);
667
668         void insb(unsigned short int port, void *addr,
669         unsigned long int count);
670         void insw(unsigned short int port, void *addr,
671         unsigned long int count);
672         void insl(unsigned short int port, void *addr,
673         unsigned long int count);
674         void outsb(unsigned short int port, const void *addr,
675         unsigned long int count);
676         void outsw(unsigned short int port, const void *addr,
677         unsigned long int count);
678         void outsl(unsigned short int port, const void *addr,
679         unsigned long int count);
680
681================== END REFERENCE NOTES */
682
683 WRY00_Print_DateTime_Usec();
684 printf("FINISHED WRY08_Start_Run_CNC_Machine(void).\n\n");
685 }
686
```

```
687 // ================================================================
688 void WRY09_Stop_Run_CNC_Machine(void) {
689 // ================================================================
690 WRY00_Print_DateTime_Usec();
691 printf("STARTED  WRY09_Stop_Run_CNC_Machine(void).\n");
692
693         // RELEASE PARALLEL PORT
694         rt_release_irq_task(PARPORT_IRQ);
695         WRY00_Print_DateTime_Usec();
696         printf("SUCCESS: Execute rt_release_irq_task(PARPORT_IRQ).\n");
697
698         // STOP REALTIME TIMER
699         stop_rt_timer();
700         WRY00_Print_DateTime_Usec();
701         printf("SUCCESS: Execute stop_rt_timer().\n");
702
703         // REVERT TO SOFT REALTIME
704         rt_make_soft_real_time();
705         WRY00_Print_DateTime_Usec();
706         printf("SUCCESS: Execute rt_make_soft_real_time().\n");
707
708         // DELETE REALTIME TASK
709         rt_task_delete(WRY_RT_task);
710         WRY00_Print_DateTime_Usec();
711         printf("SUCCESS: Execute rt_task_delete(WRY_RT_task).\n");
712
713         // NORMAL PROGRAM TERMINATION
714         WRY00_Print_DateTime_Usec();
715         printf("SUCCESS: Finished cleanup. Exiting program normally.\n");
716
717 WRY00_Print_DateTime_Usec();
718 printf("FINISHED WRY09_Stop_Run_CNC_Machine(void).\n\n");
719 }
720
721 // ================================================================
722 void WRYcleanup(int WRYsig) {
723 // ================================================================
724 printf("\n\n");
725 WRY00_Print_DateTime_Usec();
```

```
726 printf("STARTED  USER INTERRUPT RESPONSE TO STOP.\n");
727
728         WRY00_Print_DateTime_Usec();
729         printf("SUCCESS: Activate stop on User <Ctrl>-C Interrupt.\n");
730         WRY00_Print_DateTime_Usec();
731         printf("SUCCESS: Start cleanup sequence to exit program.\n");
732
733         // RELEASE PARALLEL PORT
734         rt_release_irq_task(PARPORT_IRQ);
735         WRY00_Print_DateTime_Usec();
736         printf("SUCCESS: Execute rt_release_irq_task(PARPORT_IRQ).\n");
737
738         // STOP REALTIME TIMER
739         stop_rt_timer();
740         WRY00_Print_DateTime_Usec();
741         printf("SUCCESS: Execute stop_rt_timer().\n");
742
743         // REVERT TO SOFT REALTIME
744         rt_make_soft_real_time();
745         WRY00_Print_DateTime_Usec();
746         printf("SUCCESS: Execute rt_make_soft_real_time().\n");
747
748         // ABNORMAL TERMINATION
749         WRY00_Print_DateTime_Usec();
750         printf("SUCCESS: Abnormal termination. Finished cleanup. Exiting program.\n");
751         WRY00_Print_DateTime_Usec();
752         printf("FINISHED USER INTERRUPT RESPONSE TO STOP.\n");
753
754         // DELETE REALTIME TASK
755         WRY00_Print_DateTime_Usec();
756         printf("SUCCESS: Execute rt_task_delete(WRY_RT_task).\n\n");
757         rt_task_delete(WRY_RT_task);
758
759 exit(1); // QUIT BASED ON KEYBOARD <Ctrl-C>
760 }
761
762
763
764
```

```
765 // ===================================================
766 int main(int argc, char* argv[]) {
767 // ===================================================
768 WRY00_Print_DateTime_Usec();
769 printf("Bismillah from WRY executed in main().\n\n");
770
771 // TO DO: TO INCLUDE AUTOMATIC RTAI MODULES load/unload
772
773         // system("./load-rtai-modules.sh");
774
775 // FOR KEYBOARD INTERRUPT TO STOP (e.g. <Ctrl-C>)
776         signal(SIGTERM, WRYcleanup);
777         signal(SIGINT,  WRYcleanup);
778         signal(SIGKILL, WRYcleanup);
779
780 // INVOKE FUNCTIONS SEQUENTIALLY TO RUN RTAI
781         WRY01_Create_and_Initialize_Real_Time_Task_LXRT();
782         WRY02_Get_Address_and_Name_of_Real_Time_Task_LXRT();
783         WRY31_Make_Hard_Real_Timer_Run();
784         WRY32_Check_Selected_Hard_Real_Time_Run_Mode();
785         WRY33_Execute_Selected_Hard_Real_Time_Run_Mode();
786         WRY04_Allow_NonRoot_Users_For_Hard_Real_Time();
787         WRY05_Lock_Memory_Swapping_For_Hard_Real_Time();
788         WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ();
789         WRY07_ParallelPort_Check_RT_Task_Running_Mode();
790         WRY08_Start_Run_CNC_Machine();
791         WRY09_Stop_Run_CNC_Machine();
792
793         //system("./unload-rtai-modules.sh");
794
795 WRY00_Print_DateTime_Usec();
796 printf("Alhamdulillah from WRY executed in main().\n");
797 return (0);
798 }
799
800 // ===================================================
```

D 4.30 App4-Full execution C/C++ code for Real Time (RTAI)

Listing 1.14: App4-Full execution C/C++ code for Real Time (RTAI)

```
/* COMPILATION AND EXECUTION RESULTS

(1) COMPILATION
wruslan@dell-ub1604-64b:~/ump-project2/test14$ gcc -I./ -I/usr/realtime/include -O2 -pipe -L/usr/realtime/lib -
   lpthread -llxrt -lm -o rtai-CNC14.x rtai-CNC14.c
wruslan@dell-ub1604-64b:~/ump-project2/test14$

(2) DISPLAY COMPILED EXECUTABLE
wruslan@dell-ub1604-64b:~/ump-project2/test14$ ls -al
total 72
drwxrwxr-x  2 wruslan wruslan  4096 May  2 19:59 .
drwxrwxr-x 19 wruslan wruslan  4096 May  2 19:42 ..
-rwxrwxrwx  1 wruslan wruslan  4096 Apr 28 09:23 load-rtai-modules.sh
-rw-rw-r--  1 wruslan wruslan 31513 May  2 19:59 rtai-CNC14.c
-rwxrwxr-x  1 wruslan wruslan 24064 May  2 19:59 rtai-CNC14.x          <==== FOUND COMPILED EXECUTABLE
-rwxrwxrwx  1 wruslan wruslan  4063 Apr 28 09:28 unload-rtai-modules.sh
wruslan@dell-ub1604-64b:~/ump-project2/test14$

(3) EXECUTE COMPILED EXECUTABLE AS ROOT USER
wruslan@dell-ub1604-64b:~/ump-project2/test14$ sudo ./rtai-CNC14.x
[sudo] password for wruslan:
2018-05-02 19:59:42.000944348   Bismillah from WRY executed in main().

2018-05-02 19:59:42.000945009   SUCCESS: Completed real time task LXRT initialization.
2018-05-02 19:59:42.000945019   SUCCESS: NAME of real time task LXRT = 0xffffa08f43997230
2018-05-02 19:59:42.000945038   SUCCESS: Hard real timer is already running.
2018-05-02 19:59:42.000945045   SUCCESS: PERIODIC hard realtime run mode selected.
2018-05-02 19:59:42.000945051   SUCCESS: PERIODIC Set CNCtimer_period_ns        = 1000000000
2018-05-02 19:59:42.000945064   SUCCESS: PERIODIC Execute rt_set_periodic_mode().
2018-05-02 19:59:42.000945076   SUCCESS: PERIODIC Execute nano2count(CNCtimer_period_ns).
2018-05-02 19:59:42.000945083   SUCCESS: PERIODIC CNCtimer_period_count         = 3292519000
2018-05-02 19:59:42.000945102   SUCCESS: PERIODIC Execute start_rt_timer(CNCtimer_period_count).
2018-05-02 19:59:42.000945115   SUCCESS: PERIODIC Execute rt_task_make_periodic(WRY_RT_task, x, x).
2018-05-02 19:59:42.000945128   SUCCESS: Execute Allow hard realtime for non-root users.
2018-05-02 19:59:42.000945274   SUCCESS: Execute Lock memory and now no memory swapping for hard realtime.
```

```
35 2018-05-02 19:59:42.000945281   FINISHED WRY05_Lock_Memory_Swapping_For_Hard_Real_Time(void).
36
37 2018-05-02 19:59:42.000945288   STARTED  WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ(void).
38 2018-05-02 19:59:42.000945301   SUCCESS: Display PARPORT_IRQ     = 18
39 2018-05-02 19:59:42.000945308   SUCCESS: Display RT_IRQ_TASK     = 0
40 2018-05-02 19:59:42.000945315   SUCCESS: Display BASEPORT        = 0xD010
41 2018-05-02 19:59:42.000945322   SUCCESS: Display CPUMAP          = 0x04
42 2018-05-02 19:59:42.000945329   SUCCESS: Display PERIOD          = 500000 (ns)
43 2018-05-02 19:59:42.000945335   SUCCESS: Display TICK_TIME       = 1000000 (ns)
44 2018-05-02 19:59:42.000945342   SUCCESS: Execute rt_request_irq_task(PARPORT_IRQ, WRY_RT_task, RT_IRQ_TASK, 1).
45 2018-05-02 19:59:42.000945348   FINISHED WRY06_ParallelPort_Request_StartUp_and_Enable_IRQ(void).
46
47 2018-05-02 19:59:42.000945355   STARTED  WRY07_ParallelPort_Check_RT_Task_Running_Mode(void).
48 2018-05-02 19:59:42.000945362   SUCCESS: Display WRY_selected_run_mode = 1
49 2018-05-02 19:59:42.000945374   SUCCESS: Display WRY_RT_task is ALREADY running in periodic mode.
50 2018-05-02 19:59:42.000945381   FINISHED WRY07_ParallelPort_Check_RT_Task_Running_Mode(void).
51
52 2018-05-02 19:59:42.000945388   STARTED  WRY08_Start_Run_CNC_Machine(void).
53 2018-05-02 19:59:42.000945400   SUCCESS: Start CNC run at time  = 1528453940458 (ns)
54 2018-05-02 19:59:42.000945418   TEST WRITING outb_p(25, BASEPORT)
55 2018-05-02 19:59:43.000824252   run rt_sleep(2893422000) internal counts for x = 0 run duration = 878838671 (ns)
56 2018-05-02 19:59:43.000824285   TEST WRITING outb_p(25, BASEPORT)
57 2018-05-02 19:59:44.000703121   run rt_sleep(2893422000) internal counts for x = 1 run duration = 878841400 (ns)
58 2018-05-02 19:59:44.000703155   TEST WRITING outb_p(25, BASEPORT)
59 2018-05-02 19:59:45.000581988   run rt_sleep(2893422000) internal counts for x = 2 run duration = 878838807 (ns)
60 2018-05-02 19:59:45.000582021   TEST WRITING outb_p(25, BASEPORT)
61 2018-05-02 19:59:46.000460855   run rt_sleep(2893422000) internal counts for x = 3 run duration = 878839578 (ns)
62 2018-05-02 19:59:46.000460888   TEST WRITING outb_p(25, BASEPORT)
63 2018-05-02 19:59:47.000339721   run rt_sleep(2893422000) internal counts for x = 4 run duration = 878838160 (ns)
64 2018-05-02 19:59:47.000339753   TEST WRITING outb_p(25, BASEPORT)
65 2018-05-02 19:59:48.000218585   run rt_sleep(2893422000) internal counts for x = 5 run duration = 878836793 (ns)
66 2018-05-02 19:59:48.000218617   TEST WRITING outb_p(25, BASEPORT)
67 2018-05-02 19:59:49.000097453   run rt_sleep(2893422000) internal counts for x = 6 run duration = 878841787 (ns)
68 2018-05-02 19:59:49.000097487   TEST WRITING outb_p(25, BASEPORT)
69 2018-05-02 19:59:49.000976322   run rt_sleep(2893422000) internal counts for x = 7 run duration = 878839913 (ns)
70 2018-05-02 19:59:49.000976355   TEST WRITING outb_p(25, BASEPORT)
71 2018-05-02 19:59:50.000855191   run rt_sleep(2893422000) internal counts for x = 8 run duration = 878840543 (ns)
72 2018-05-02 19:59:50.000855223   TEST WRITING outb_p(25, BASEPORT)
73 2018-05-02 19:59:51.000734055   run rt_sleep(2893422000) internal counts for x = 9 run duration = 878835850 (ns)
```

```
74 2018-05-02 19:59:51.000734082    SUCCESS: End CNC run at time     = 1537242485990 (ns)
75 2018-05-02 19:59:51.000734090    SUCCESS: CNC total run duration = 8788545532 (ns)
76 2018-05-02 19:59:51.000734097    FINISHED WRY08_Start_Run_CNC_Machine(void).
77
78 2018-05-02 19:59:51.000734104    STARTED  WRY09_Stop_Run_CNC_Machine(void).
79 2018-05-02 19:59:51.000734116    SUCCESS: Execute rt_release_irq_task(PARPORT_IRQ).
80 2018-05-02 19:59:51.000734131    SUCCESS: Execute stop_rt_timer().
81 2018-05-02 19:59:51.000734141    SUCCESS: Execute rt_make_soft_real_time().
82 2018-05-02 19:59:51.000734150    SUCCESS: Execute rt_task_delete(WRY_RT_task).
83 2018-05-02 19:59:51.000734156    SUCCESS: Finished cleanup. Exiting program normally.
84 2018-05-02 19:59:51.000734162    FINISHED WRY09_Stop_Run_CNC_Machine(void).
85
86 2018-05-02 19:59:51.000734171    Alhamdulillah from WRY executed in main().
87 wruslan@dell-ub1604-64b:~/ump-project2/test14$
88 */
```

D 4.31 App4-C++2011 Example Parallel Multithreading

1. The C++ language offers for the first time support in implementing applications that require concurrent programming, regardless of the development platform.

2. Before the C++11 standard the multi-threaded applications were based on platform-specific extensions, like Intel TBB, OpenMP, Pthreads, etc. Portable applications are the major advantage brought by this new feature (i.e. Windows multi-threaded applications are easily ported to iPhone or Android platforms).

3. An advantage for those familiariazed with the Boost thread library is that many concepts from the standard C++11 library keep the same name and structure as the boost threads library classes.

4. The C++ Standard Library includes classes for thread manipulation and synchronization, common protected data and low-level atomic operations. We will next exemplify and provide a general description of how these concepts occur in the C++11 Standard.

5. REFERENCE: http://en.cppreference.com/w/cpp/thread

6. COMPILATION
   Terminal$ g++ -o multi-threaded.xx multi-threaded.cpp -std=c++11 -pthread -D_GLIBCXX_USE_NANOSLEEP

7. EXECUTION
   Terminal$ ./multi-threaded.xx

The 01-multi-threaded C++ program operations are described as follows:

1. The main C++ program spawn three threads.

2. Thread-1 sleeps for 10 seconds.

3. Thread-2 sleeps for 5 seconds.

4. Thread-3 sleeps for 7 seconds.

5. Thread-1, Thread-2 and Thread-3 are made to run some arbitrary computations to show that individual threads can do some actual useful work. For demonstration, we created three simple functions, one for each thread.

6. The sleep times were assigned to the threads to actually simulate real conditions where actual threads spent times equal to their sleep times when running actual computations. This is used to prove the time parallel nature in executions of threads.

Listing 1.15: App4-C++2011 Example Parallel Multithreading

```
1
2 // File: independent-parallel-threads-06.cpp
3 // Date : Sun Mar  8 12:15:41 MYT 2015
4 // Author: WRY
5
6 // http://en.cppreference.com/w/cpp/thread
7
8 // COMPILATION
9 // THIS DOES NOT WORK
10 // g++  -o independent-parallel-threads-06.xx independent-parallel-threads-06.cpp -std=c++11 -pthread
11
12 // THE FOLLOWING WORKS
13 // g++  -o independent-parallel-threads-06.xx independent-parallel-threads-06.cpp -std=c++11 -pthread -
   D_GLIBCXX_USE_NANOSLEEP
14 // g++  -o independent-parallel-threads-06.xx independent-parallel-threads-06.cpp -std=gnu++11 -pthread -
   D_GLIBCXX_USE_NANOSLEEP
15
16 // ========================================================
17
18         #include <stdio.h>        // To use printf()
19         #include <iostream>       // std::cout
20         #include <thread>         // std::thread
21         #include <ctime>          // To use the C++ timer function
22         #include <chrono>         // To use the system_clock
23         #include <mutex>          // To run mutual exclusion (mutex) locking thread
24
25         using namespace std;
26
27 // Global variables (shared among all threads for Read/Write)
28         double result_wry_thread_01 = 0.0;
29         double result_wry_thread_02 = 0.0;
30         double result_wry_thread_03 = 0.0;
31         double result_total = 0.0;
32
```

```
33 // ============================================================
34 void calculate_norm1(double x, double y, double z) {
35 // ============================================================
36 // OPENING THREAD
37         std::thread::id this_id = std::this_thread::get_id();
38         auto threadstart_time = std::chrono::high_resolution_clock::now();
39         printf(" %lld Start wry_thread_01. Thread_ID = %x Hex. \n", threadstart_time, this_id);
40
41 // PROCESSING THREAD - DELAY (SLEEP) 10 SECONDS
42         int sleep_seconds = 10;
43         printf(" %lld PROCESSING wry_thread_01. Will sleep for %d seconds ... \n", threadstart_time, sleep_seconds);
44
45         std::mutex g_display_mutex;
46         g_display_mutex.lock();
47         std::this_thread::sleep_for(std::chrono::seconds(sleep_seconds));
48         g_display_mutex.unlock();
49
50         result_wry_thread_01 = (x + y + z);
51
52 // CLOSING THREAD
53         auto threadend_time = std::chrono::high_resolution_clock::now();
54         auto duration_time = (threadend_time - threadstart_time);
55         typedef std::chrono::duration<double> double_seconds;
56         double process_seconds = std::chrono::duration_cast<double_seconds>(duration_time).count();
57         printf(" %lld Ended wry_thread_01. Thread_ID = %x Hex. Thread processing time = %lf seconds = %lld ticks \n
             ", threadend_time, this_id, process_seconds, duration_time);
58 }
59 // ============================================================
60 void calculate_norm2(double x, double y, double z) {
61 // ============================================================
62 // OPENING THREAD
63         std::thread::id this_id = std::this_thread::get_id();
64         auto threadstart_time = std::chrono::high_resolution_clock::now();
65         printf(" %lld Start wry_thread_02. Thread_ID = %x Hex. \n", threadstart_time, this_id);
66
67 // PROCESSING THREAD - DELAY (SLEEP) 5 SECONDS
68         int sleep_seconds = 5;
69         printf(" %lld PROCESSING wry_thread_02. Will sleep for %d seconds ... \n", threadstart_time, sleep_seconds);
70
```

```
71          std::mutex g_display_mutex;
72          g_display_mutex.lock();
73          std::this_thread::sleep_for(std::chrono::seconds(sleep_seconds));
74          g_display_mutex.unlock();
75
76          result_wry_thread_02 = (x/0.675 + 15.198*y + 0.005*z);
77
78  // CLOSING THREAD
79          auto threadend_time = std::chrono::high_resolution_clock::now();
80          auto duration_time = (threadend_time - threadstart_time);
81          typedef std::chrono::duration<double> double_seconds;
82          double process_seconds = std::chrono::duration_cast<double_seconds>(duration_time).count();
83          printf(" %lld Ended wry_thread_02. Thread_ID = %x Hex. Thread processing time = %lf seconds = %lld ticks \n
              ", threadend_time, this_id, process_seconds, duration_time);
84  }
85  // ========================================================
86  void calculate_norm3(double x, double y, double z) {
87  // ========================================================
88  // OPENING THREAD
89          std::thread::id this_id = std::this_thread::get_id();
90          auto threadstart_time = std::chrono::high_resolution_clock::now();
91          printf(" %lld Start wry_thread_03. Thread_ID = %x Hex. \n", threadstart_time, this_id);
92
93  // PROCESSING THREAD - DELAY (SLEEP) 7 SECONDS
94          int sleep_seconds = 7;
95          printf(" %lld PROCESSING wry_thread_03. Will sleep for %d seconds ... \n", threadstart_time, sleep_seconds);
96
97          std::mutex g_display_mutex;
98          g_display_mutex.lock();
99          std::this_thread::sleep_for(std::chrono::seconds(sleep_seconds));
100         g_display_mutex.unlock();
101
102         result_wry_thread_03 = (2.56*x + y/1.56 + z*0.017);
103
104 // CLOSING THREAD
105         auto threadend_time = std::chrono::high_resolution_clock::now();
106         auto duration_time = (threadend_time - threadstart_time);
107         typedef std::chrono::duration<double> double_seconds;
108         double process_seconds = std::chrono::duration_cast<double_seconds>(duration_time).count();
```

```
109         printf(" %lld Ended wry_thread_03. Thread_ID = %x Hex. Thread processing time = %lf seconds = %lld ticks \n
              ", threadend_time, this_id, process_seconds, duration_time);
110 }
111
112 // ========================================================
113 void print_current_date_time() {
114 // ========================================================
115         time_t the_time;
116         auto now = std::chrono::system_clock::now();
117         the_time = std::chrono::system_clock::to_time_t(now);
118         // std::cout << "Now : " << ctime(&tt) << std::endl;
119         printf("Date Time: %s", ctime(&the_time));
120 }
121
122 // ========================================================
123 void time_ticks() {
124 // ========================================================
125         auto ticks_wry = std::chrono::high_resolution_clock::now();
126         printf(" %lld ", ticks_wry);
127 }
128
129 // ========================================================
130 int main(int argc, char *argv[]) {
131 // ========================================================
132
133 // OPENING MAIN PROGRAM MESSAGE
134         auto start_time = std::chrono::high_resolution_clock::now();
135         time_ticks(); print_current_date_time();
136         time_ticks(); printf("Bismillah. Main thread started. \n");
137
138 // Threads are started by defining an object std::thread
139 // Spawn and launch three different threads runnung the same function
140         std::thread     wry_thread_01(calculate_norm1, 1.0, 2.0, 3.0);
141         std::thread     wry_thread_02(calculate_norm2, 4.0, 5.0, 6.0);
142         std::thread     wry_thread_03(calculate_norm3, 2.2, 3.2, 6.2);
143
144 // join() - The function returns when the thread execution has completed.
145 // Synchronize all threads, pause until all threads finish execution.
146         wry_thread_01.join();
```

```
147         wry_thread_02.join();
148         wry_thread_03.join();
149
150 // DISPLAY RESULTS OF INDIVIDUAL THREADS
151         time_ticks(); printf("result_wry_thread_01 = %f \n", result_wry_thread_01);
152         time_ticks(); printf("result_wry_thread_02 = %f \n", result_wry_thread_02);
153         time_ticks(); printf("result_wry_thread_03 = %f \n", result_wry_thread_03);
154
155         time_ticks(); printf("result_total = %f \n", result_wry_thread_01 + result_wry_thread_02 +
              result_wry_thread_03);
156
157 // CALCULATE TOTAL PROCESSING TIME
158         auto end_time = std::chrono::high_resolution_clock::now();
159         auto duration_time = (end_time - start_time);
160         typedef std::chrono::duration<double> double_seconds;
161         double process_seconds = std::chrono::duration_cast<double_seconds>(duration_time).count();
162         time_ticks(); printf("TOTAL PROGRAM MULTI-THREADED PROCESSING TIME = %lf seconds = %lld ticks. \n",
              process_seconds, duration_time);
163
164 // CLOSING MAIN PROGRAM MESSAGE
165         time_ticks(); print_current_date_time();
166         time_ticks(); printf("Alhamdulillah. Main thread ended. \n");
167
168 return 0;
169 }
170 // ========================================================
```

D 4.32 App4-C++2011 Execution Parallel Multithreading

Listing 1.16: App4-C++2011 Execution Parallel Multithreading

```
1
2COMPILATION
3=================
4root@hpcompaqdk-ub1004-rtai:~/Documents/Multithreaded-C++11/test2.6# g++  -o independent-parallel-threads-06.xx
   independent-parallel-threads-06.cpp -std=c++11 -pthread -D_GLIBCXX_USE_NANOSLEEP
5
6EXECUTION
7=================
8root@hpcompaqdk-ub1004-rtai:~/Documents/Multithreaded-C++11/test2.6# ./independent-parallel-threads-06.xx
91425871390235109 Date Time: Mon Mar  9 11:23:10 2015
101425871390235280 Bismillah. Main thread started.
111425871390235345 Start wry_thread_02. Thread_ID = b6f28b70 Hex.
121425871390235345 PROCESSING wry_thread_02. Will sleep for 5 seconds ...
131425871390235340 Start wry_thread_01. Thread_ID = b7729b70 Hex.
141425871390235340 PROCESSING wry_thread_01. Will sleep for 10 seconds ...
151425871390235398 Start wry_thread_03. Thread_ID = b6727b70 Hex.
161425871390235398 PROCESSING wry_thread_03. Will sleep for 7 seconds ...
171425871395235469 Ended wry_thread_02. Thread_ID = b6f28b70 Hex. Thread processing time = 5.000124 seconds = 5000124
   ticks
181425871397235500 Ended wry_thread_03. Thread_ID = b6727b70 Hex. Thread processing time = 7.000102 seconds = 7000102
   ticks
191425871400235832 Ended wry_thread_01. Thread_ID = b7729b70 Hex. Thread processing time = 10.000492 seconds =
   10000492 ticks
201425871400235899 result_wry_thread_01 = 6.000000
211425871400235922 result_wry_thread_02 = 81.945926
221425871400235932 result_wry_thread_03 = 7.788682
231425871400235939 result_total = 95.734608
241425871400235963 TOTAL PROGRAM MULTI-THREADED PROCESSING TIME = 10.000849 seconds = 10000849 ticks.
251425871400235969 Date Time: Mon Mar  9 11:23:20 2015
261425871400235995 Alhamdulillah. Main thread ended.
27root@hpcompaqdk-ub1004-rtai:~/Documents/Multithreaded-C++11/test2.6#
28*/
```

D 4.33 App4-C++-MPI Example Parallel Multiprocessing

1. The 03-multi-processing.cpp C++ program is a program based on the OpenMPI library for multi-processing.
2. It requires the installation of this OpenMPI libraries and executables on the particular platform.
   (a) COMPILATION:
   /usr/bin/mpic++ -o 03-multi-processing.xx 03-multi-processing.cpp -lpthread
   (b) EXECUTION:
   /usr/bin/mpirun -np 10 03-multi-processing.xx — sort
   (c) The number of processes to execute np above is 10. It can be changed by the user to any number preferred.
3. Stage 1 : The program calculates the function kuntakinte(x), where x is an integer put as a very large number like 1000000 (one million).
4. Stage 2 : The program calculates the function kamikamu(x), which is the cumulative sum of kuntakinta(x) which is the sum of kuntakinte(1) + kuntakinte(2) + kuntakinte(3) + ... + kuntakinte(x)
5. The calculation of the function kuntakinta(x) itself is very CPU intensive. Study the program code for kuntakinte() function.
6. The computation strategy for this OpenMPI program is "divide and conquer". Break the execution into np individual processes (multi-processing) to handle the large computation. Split the data regions for x into (x/np) individual tasks. This task is then assign to a process.
7. For example, if x = 1,000,000 (one million), and np = 10, then each process will handle (1,000,000)/10 = 100,000 (one hundred thousand).
8. The final result for kamikamu(1,000,000), as an example, must be the same whether we run with np=5, np=10, np=20 or any integer number because np is just the number of processes we choose to split the computation work.
9. Yes. We have proven that it gave the same results for different values of np.
10. The concept is similar to counting a very large pile (mountain) of oranges. The total count must be the same whether we hire 10 people, 50 people or 100 people to do the counting. The total sum cannot change.

11. This is what the current OpenMPI code 03-multi-processing.cpp does. The number of people (processes) to do the counting (computation) is np. Ha ha ha. Ya ya ya. Kah kah kah.

Listing 1.17: App4-C++-MPI Example Parallel Multiprocessing

```
1
2// File: kamikamu-parallel.c
3// Date: Thu Jan  3 21:53:17 MYT 2013
4// Rev1: Mon Nov 18 03:45:54 MYT 2013
5//
6// This is a kamikamu() and kuntakinte() C-program for running parallel OpenMPI routines
7// that run on a local multicore computer or distributed on a computing cluster.
8//
9// COMPILATION
10// /opt/openmpi/bin/mpicc -o kamikamu-parallel.x kamikamu-parallel.c
11// /usr/bin/mpicc -o kamikamu-parallel.x kamikamu-parallel.c
12
13// EXECUTION ON LOCAL MACHINE
14// /opt/openmpi/bin/mpirun -np 10 kamikamu-parallel.x | sort
15// /usr/bin/mpirun -np 10 kamikamu-parallel.x | sort
16
17// EXECUTION ON CLUSTER MACHINE
18// /opt/openmpi/bin/mpirun -np 10 -hostfile machines kamikamu-parallel.x | sort
19
20// ========================================================
21#include <math.h>      // For mathematical functions
22#include <stdio.h>     // For function printf()
23#include <stdlib.h>    // For function calloc()
24#include <string.h>    // For function %s (printf string)
25#include <time.h>      // For datetimestamp() function
26
27// SET THE PATH FOR THIS HEADER FILE TO SUIT ITS LOCATION ON YOUR COMPUTER
28
29// #include "/opt/openmpi/include/mpi.h"
30#include "/usr/lib/openmpi/include/mpi.h"
31
32// ========================================================
33void datetimestamp() {
34// ========================================================
```

```
35          time_t ltime;             /* calendar time */
36          ltime=time(NULL);         /* get current cal time */
37          printf("%s",asctime(localtime(&ltime)));
38  }
39  // ================================================================
40  // DEFINE FUNCTION kuntakinte(n) THE FUNNY FUNCTION
41  // ================================================================
42          unsigned long int kuntakinte(unsigned long int n) {
43          unsigned long int m = 0;
44          while (n != 1) {
45
46                  if ((n%2) == 0) {
47                          n = (n/2);          // FOR EVEN n
48                  } else {
49                          n = (3*n) + 1 ;     // FOR ODD  n
50                  }
51                  m = m + 1;
52
53          // TRACE PRINT
54          //  printf("r = %d \tm = %d ", n, m);
55          }
56
57  return (m);
58  }
59  // ================================================================
60  // DEFINE FUNCTION kamikamu(x) THE SUM ACCUMULATION FUNCTION
61  // ================================================================
62          unsigned long int kamikamu(unsigned long int xstart, unsigned long int xend) {
63
64          unsigned long int n;
65          unsigned long int kamikamu_sum = 0;
66
67          for (n = xstart; n <= xend; n++) {
68                  kamikamu_sum = kamikamu_sum + kuntakinte(n);
69
70                  //TRACE PRINT
71                  //printf("kuntakinte(%d) = %d \t kamikamu(%d) = %d \n", n, kuntakinte(n), n, kamikamu_sum);
72          }
73          return (kamikamu_sum);
```

```
74 }
75 // ============================================================
76 int main(int argc, char **argv) {
77 // ============================================================
78 // VARIABLES FOR CALCULATION
79         int intv_start, intv_end;
80         unsigned long int interval_start, interval_end;
81         unsigned long int sum, result;
82         sum = 0;
83         result = 0;
84
85         double tStart, tEnd, tExecution;
86         double tNow1, tNow2, tNow3;
87         double step;
88         char* cpu_name;
89         int num_steps = 1000000;
90         step = 1.0 / (double)num_steps;
91
92 // VARIABLES FOR OPENMPI ENVIRONMENT
93         int procID, totProcesses;
94
95 // INITIALIZE OPENMPI ENVIRONMENT
96         MPI_Init(&argc, &argv);
97         MPI_Comm_size(MPI_COMM_WORLD, &totProcesses);
98         MPI_Comm_rank(MPI_COMM_WORLD, &procID);
99         cpu_name = (char *)calloc(80,sizeof(char));
100        gethostname(cpu_name,80);
101
102        if (procID == 0) {
103                printf("(Q2.1B) Calculate kamikamu(1000000) using C-code with OpenMPI on local machine.\n");
104                printf("PROGRAM STARTING: "); datetimestamp(); printf("\n");
105        }
106
107        // CALCULATE THE START AND END INTERVAL VALUES TO BE ASSIGNED TO EACH PROCESS
108        intv_start = procID * (num_steps/totProcesses) +1;
109        intv_end   = intv_start + (num_steps/totProcesses) -1;
110
111        // THIS IS FOR THE LAST INTERVAL ONLY
112        if (procID == (totProcesses-1)) {
```

```
113                 intv_end = num_steps;
114         }
115
116         interval_start = (unsigned long int)intv_start;
117         interval_end   = (unsigned long int)intv_end;
118
119         // DISPLAY START TIME FROM ALL PROCESSES
120         tStart = MPI_Wtime();
121         printf("%.16lf \tProcess-%d on %s. STARTED. Assigned range = (%d,%d) \n", MPI_Wtime()-tStart, procID,
              cpu_name, interval_start, interval_end);
122
123         // EACH PROCESS WILL RUN THE AREA COMPUTATION FOR ITS OWN ASSIGNED INTERVAL (START AND END VALUES)
124         tNow2 = MPI_Wtime();
125         sum = kamikamu(interval_start, interval_end);
126         tNow3 = MPI_Wtime();
127
128         // DISPLAY COMPUTING TIME EXECUTED FOR EACH PROCESS
129         printf("%.16lf \tProcess-%d on %s. Execution time = %.16lf \n", MPI_Wtime()-tStart, procID, cpu_name, (tNow3
              -tNow2));
130
131         // DISPLAY COLLECTED PARTIAL SUM FROM EACH PROCESS
132         printf("%.16lf \tProcess-%d on %s. PARTIAL SUM = %ld \n", MPI_Wtime()-tStart, procID, cpu_name, sum);
133
134         // ACCUMULATE SUM FROM EACH PROCESS
135         MPI_Reduce(&sum, &result, 1, MPI_INTEGER, MPI_SUM, 0, MPI_COMM_WORLD);
136
137         // DISPLAY ONLY ON HEADNODE (OTHERWISE WILL RUN ON ALL NODES)
138         if (procID == 0) {
139                 printf("%.16lf \tProcess-%d on %s. Effective Execution Time: %.16lf \n", MPI_Wtime()-tStart, procID
                      , cpu_name, MPI_Wtime()-tStart);
140                 printf("%.16lf \tProcess-%d on %s. kamikamu(1000000) = %ld \n", MPI_Wtime()-tStart, procID, cpu_name
                      , result);
141                 printf("\n"); printf("PROGRAM FINISHED: "); datetimestamp();
142         }
143         MPI_Finalize();
144 return (0);
145 }
146 // ========================================================
```

D 4.34 App4-C++-MPI Execution Parallel Multiprocessing

Listing 1.18: App4-C++-MPI Execution Parallel Multiprocessing

```
1
2COMPILE THE SPMD kamikamu-parallel.c C-PROGRAM
3[clab50@clusterlab tutorial04]$ /opt/openmpi/bin/mpicc -o kamikamu-parallel.x kamikamu-parallel.c
4
5EXECUTE THE SPMD kamikamu-parallel.x EXECUTABLE (MUST HAVE -np 10) WITH sort OUTPUT
6[clab50@clusterlab tutorial04]$ /opt/openmpi/bin/mpirun -np 10 -hostfile machines ./kamikamu-parallel.x | sort
7
8 0.0000008846400306      Process-7 on compute-3-2.local. STARTED. Assigned range = (700001,800000)
9 0.0000009156065062      Process-3 on compute-3-3.local. STARTED. Assigned range = (300001,400000)
10 0.0000009223585948     Process-8 on compute-3-3.local. STARTED. Assigned range = (800001,900000)
11 0.0000009536743164     Process-0 on compute-3-0.local. STARTED. Assigned range = (1,100000)
12 0.0000009541399777     Process-1 on compute-3-1.local. STARTED. Assigned range = (100001,200000)
13 0.0000018946593627     Process-9 on compute-3-4.local. STARTED. Assigned range = (900001,1000000)
14 0.0000019473955035     Process-6 on compute-3-1.local. STARTED. Assigned range = (600001,700000)
15 0.0000019873259589     Process-4 on compute-3-4.local. STARTED. Assigned range = (400001,500000)
16 0.0000020727748051     Process-5 on compute-3-0.local. STARTED. Assigned range = (500001,600000)
17 0.0000021163141355     Process-2 on compute-3-2.local. STARTED. Assigned range = (200001,300000)
18 0.0973579536657780     Process-0 on compute-3-0.local. Execution time = 0.0973269939422607
19 0.0973799537168816     Process-0 on compute-3-0.local. PARTIAL SUM = 10753840
20 0.1112939541926607     Process-1 on compute-3-1.local. Execution time = 0.1111660003662109
21 0.1113159541273490     Process-1 on compute-3-1.local. PARTIAL SUM = 12184824
22 0.1144861163338646     Process-2 on compute-3-2.local. Execution time = 0.1144130229949951
23 0.1145081162685528     Process-2 on compute-3-2.local. PARTIAL SUM = 12731265
24 0.1205389156239107     Process-3 on compute-3-3.local. Execution time = 0.1204419136047363
25 0.1205629155738279     Process-3 on compute-3-3.local. PARTIAL SUM = 13092861
26 0.1223229473689571     Process-6 on compute-3-1.local. Execution time = 0.1222000122070312
27 0.1223449473036453     Process-6 on compute-3-1.local. PARTIAL SUM = 13721423
28 0.1263560727238655     Process-5 on compute-3-0.local. Execution time = 0.1262848377227783
29 0.1263840727042407     Process-5 on compute-3-0.local. PARTIAL SUM = 13534059
30 0.1266208846354857     Process-7 on compute-3-2.local. Execution time = 0.1265420913696289
31 0.1266438846942037     Process-7 on compute-3-2.local. PARTIAL SUM = 13884901
32 0.1267469873419032     Process-4 on compute-3-4.local. Execution time = 0.1266419887542725
33 0.1267699872842059     Process-4 on compute-3-4.local. PARTIAL SUM = 13372761
34 0.1284559223568067     Process-8 on compute-3-3.local. Execution time = 0.1283540725708008
35 0.1284809224307537     Process-8 on compute-3-3.local. PARTIAL SUM = 14000042
```

```
0.1290498946327716      Process-9 on compute-3-4.local. Execution time = 0.1289429664611816
0.1290728946914896      Process-9 on compute-3-4.local. PARTIAL SUM = 14159263
0.1308339537354186      Process-0 on compute-3-0.local. Effective Execution Time: 0.1308329537278041
0.1308569536777213      Process-0 on compute-3-0.local. kamikamu(1000000) = 131435239
PROGRAM FINISHED: Mon Nov 18 05:32:27 2013
PROGRAM STARTING: Mon Nov 18 05:32:27 2013
(Q2.1B) Calculate kamikamu(1000000) using C-code with OpenMPI on local machine.
[clab50@clusterlab tutorial04]$
*/
// ============================================================
```

D 4.35 App4-Rust Parallel Multithreading Codes

1. We provided the Rust program source code: rust-multi-threaded.rs

2. The file suffix (.rs) is for Rust program code. We compile this Rust code using the Rust compiler (rustc)

3. Rust Compiler:
   wruslanHP-ELBook8470p-ub1604-64b: $ which rustc
   /home/wruslan/.cargo/bin/rustc
   wruslanHP-ELBook8470p-ub1604-64b: $ rustc -version
   rustc 1.28.0 (9634041f0 2018-07-30)

4. The resulting compiled Rust code does not have a suffix, just rust-multi-threaded.

5. Child threads: In this multi-threaded Rust program, the main thread spawned three(3) child threads: thread01(x), thread02(y) and thread03(z), where x, y and z, are random integers between 1 and 20 (seconds), to simulate the sleep times of each individual child thread.

6. Main thread: The main thread is made to run beyond the longest time for any one of the three(3) child threads to finish. Since each child thread will have a maximum assigned finished time of 20 seconds (sleep), we made the main thread sleep for 21 seconds. This is to ensure that all child threads will finish early and join the main thread.

7. If we reduce the main thread sleep to less than 20 seconds, then some child threads would not have finished by this time. This means when the main thread dies, all the child threads get pulled and die together (whether finished or not yet finished). This is multi-threading.

8. Note that the total execution time is the time the main thread runs active, which is set at 21 seconds. You can reduce this setting, to prove that it is multi-threading (as explained above).

9. Proof : This total time is not the sum of the sleep times of the child threads. If it is the sum, then we have sequential-in-time execution not parallel-in-time execution. This is the proof that all child threads run independently and parallel-in-time.

10. Note that the codes for Rust is much more compact than C/C++ codes. Rust was designed from the ground up with Safety, Security and Concurrency (SSC) built-into the Rust engine, not many separate software libraries, unlike C/C++ language.

Listing 1.19: App4-Rust Parallel Multithreading Codes

```
// File: rust-multi-threaded.rs
// Date: Sat Aug 25 15:42:16 +08 2018
// Author: WRY
// ==========================================================

        use std::thread;
        use std::time::{Duration, SystemTime, UNIX_EPOCH};

// ==========================================================
// SEE SOLUTION (1) BY WRY HOW TO SOLVE THIS PROBLEM
// https://crates.io/crates/rand
// About generating random numbers in Rust

        extern crate rand;
        use rand::prelude::*;

// ==========================================================
// SEE SOLUTION (2) BY WRY HOW TO SOLVE THIS PROBLEM
// https://crates.io/crates/chrono
// About getting current date and time in Rust

        extern crate chrono;
        use chrono::prelude::*;

// ======================
// HIGH RESOLUTION TIME
// ======================

fn hires_time() {
        let start = SystemTime::now();
        let since_the_epoch = start.duration_since(UNIX_EPOCH).expect("Time went backwards");
        print!("{:?} ", since_the_epoch);
}

// SIMPLE THREAD 1
// ========================
fn my_thread01(x: u64) {
```

```
38 // ============================
39         hires_time(); print!("\tStarting child thread01({:?}).\n", x);
40         thread::sleep(Duration::from_millis(1000*x));
41         hires_time(); print!("\tFinished child thread01({:?}).\n", x);
42 }
43
44 // SIMPLE THREAD 2
45 // ============================
46 fn my_thread02(y: u64) {
47 // ============================
48         hires_time(); print!("\tStarting child thread02({:?}).\n", y);
49         thread::sleep(Duration::from_millis(1000*y));
50         hires_time(); print!("\tFinished child thread02({:?}).\n", y);
51 }
52
53 // SIMPLE THREAD 3
54 // ============================
55 fn my_thread03(z: u64) {
56 // ============================
57         hires_time(); print!("\tStarting child thread03({:?}).\n", z);
58         thread::sleep(Duration::from_millis(1000*z));
59         hires_time(); print!("\tFinished child thread03({:?}).\n", z);
60 }
61
62 // ============================================================
63 fn main() {
64 // ============================================================
65
66 //  TESTED GOOD RANDOM NUMBERS BASED ON SOLUTION(1) ABOVE
67 //
68         //  let x: u8 = random();
69         //  println!("random u8 = {}", x);
70
71         //  let y = random::<f64>();
72         //  println!("random::<f64> = {}", y);
73
74         //  let sleep_time: u8 = thread_rng().gen_range(1, 21);
75         //  println!("sleep_time u8 value = {:?}", sleep_time);
76
```

```
77 // TESTED GOOD DATE RUN TODAY NOW BASED ON SOLUTION(2) ABOVE
78         // let utc: DateTime<Utc> = Utc::now();
79         // println!("DateTime<Utc> value = {}", utc);
80
81         // let local: DateTime<Local> = Local::now();
82         // println!("DateTime<Local> value = {:?}", local);
83
84 // ============================================================
85 // IMPLEMENTATION
86
87         let local: DateTime<Local> = Local::now();
88         hires_time(); print!("\tBismillah from WRY. DateTime now = {:?} \n", local);
89
90         let sleep_time01: u64 = thread_rng().gen_range(1, 21);
91
92         println!("Assigned sleep_time01 u64 value = {:?}", sleep_time01);
93         thread::spawn(move || {
94                 hires_time();
95                 print!("\tMain thread spawning child thread01({:?})\n", sleep_time01);
96                 my_thread01(sleep_time01);
97         });
98
99         let sleep_time02: u64 = thread_rng().gen_range(1, 21);
100        println!("Assigned sleep_time02 u64 value = {:?}", sleep_time02);
101
102        thread::spawn(move || {
103                hires_time();
104                print!("\tMain thread spawning child thread02({:?})\n", sleep_time02);
105                my_thread02(sleep_time02);
106        });
107
108        let sleep_time03: u64 = thread_rng().gen_range(1, 21);
109        println!("Assigned sleep_time03 u64 value = {:?}", sleep_time03);
110
111        thread::spawn(move || {
112                hires_time();
113                print!("\tMain thread spawning child thread03({:?})\n", sleep_time03);
114                my_thread03(sleep_time03);
115        });
```

```
116
117// IMPT: Main to sleep longer than all child threads, otherwise the remaining
118// child threads die together when main thread dies.
119// Try reducing this sleep time and see. Ha ha ha.
120
121         hires_time();
122         print!("\tMain thread starting to sleep for max XXX seconds.\n");
123
124         thread::sleep(Duration::from_millis(21000)); // Wait for 21 seconds.
125         hires_time();print!("\tMain thread finished sleep time max for 21 seconds.\n");
126
127         hires_time(); print!("\tAlhamdulillah from WRY.\n");
128
129} // END main()
130
131// ============================================================
132/* COMPILATION
133
134wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp3$ rustc rust-multi-threaded.rs
135
136EXECUTION NO. 1
137====================
138wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp3$ ./rust-multi-threaded
1391535187474.12119863s     Bismillah from WRY. DateTime now = 2018-08-25T16:57:54.121130393+08:00
1401535187474.123016112s    Main thread spawning child thread01(4)
1411535187474.12304328s     Starting child thread01(4).
1421535187474.123467714s    Main thread starting to sleep for max 21 seconds.
1431535187474.123703762s    Main thread spawning child thread02(2)
1441535187474.123724902s    Starting child thread02(2).
1451535187474.123972992s    Main thread spawning child thread03(18)
1461535187474.123986326s    Starting child thread03(18).
1471535187476.123839552s    Finished child thread02(2).
1481535187478.123198653s    Finished child thread01(4).
1491535187492.124127672s    Finished child thread03(18).
1501535187495.123586666s    Main thread finished sleep time max for 21 seconds.
1511535187495.123627182s    Alhamdulillah from WRY.
152wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp3$
153
154ANALYSIS
```

```
1535187492.124127672s   Finished child thread03(18).
-1535187474.123986326s  Starting child thread03(18).
==========
18.001  seconds HA HA HA Time for longest thread to finish.

EXECUTION No. 2
==========
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp3$ ./rust-multi-threaded
1535187502.325989233s   Bismillah from WRY. DateTime now = 2018-08-25T16:58:22.325920986+08:00
1535187502.32760476s    Main thread starting to sleep for max 21 seconds.
1535187502.32786682s    Main thread spawning child thread01(20)
1535187502.327892862s   Starting child thread01(20).
1535187502.328241946s   Main thread spawning child thread02(10)
1535187502.328266967s   Starting child thread02(10).
1535187502.328634609s   Main thread spawning child thread03(17)
1535187502.328649296s   Starting child thread03(17).
1535187512.328427258s   Finished child thread02(10).
1535187519.328750721s   Finished child thread03(17).
1535187522.328023394s   Finished child thread01(20).
1535187523.327788422s   Main thread finished sleep time max for 21 seconds.
1535187523.32783525s    Alhamdulillah from WRY.
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp3$

ANALYSIS
1535187522.328023394s   Finished child thread01(20).
-1535187502.327892862s  Starting child thread01(20).
==========
20.001  seconds HA HA HA Time for longest thread to finish.

ALHAMDULILLAH 3 TIMES.
*/
// ====================================================
```

D 4.36 App4-Julia Parallel Programming Codes

1. **Parallel computing in Julia**
2. https://github.com/JuliaLang/julia/blob/master/doc/src/manual/parallel-computing.md
3. For newcomers to multi-threading and parallel computing it can be useful to first appreciate the different levels of parallelism offered by Julia. We can divide them in three main categories :
   (a) Julia Coroutines (Green Threading)
   (b) Multi-Threading
   (c) Multi-Core or Distributed Processing
4. We will first consider Julia [Tasks (aka Coroutines)](@ref man-tasks) and other modules that rely on the Julia runtime library, that allow to suspend and resume computations with full control of inter-Tasks communication without having to manually interface with the operative system's scheduler.
5. Julia also allows to communicate between Tasks through operations like wait and fetch. Communication and data synchronization is managed through Channels, which are the conduit that allows inter-Tasks communication.
6. Julia also supports experimental multi-threading, where execution is forked and an anonymous function is run across all threads. Described as a fork-join approach, parallel threads are branched off and they all have to join the Julia main thread to make serial execution continue.
7. Multi-threading is supported using the Base.Threads module that is still considered experimental, as Julia is not fully thread-safe yet. In particular segfaults seem to emerge for IØoperations and task switching. As an un up-to-date reference, keep an eye on the issue tracker. Multi-Threading should only be used if you take into consideration global variables, locks and atomics, so we will explain it later.
8. In the end we will present Julia's way to distributed and parallel computing. With scientific computing in mind, Julia natively implements interfaces to distribute a process through multiple cores or machines. Also we will mention useful external packages for distributed programming like MPI.jl and DistributedArrays.jl.
9. **Coroutines and Channels in Julia**

Page 160 of 230

10. Julia's parallel programming platform uses [Tasks (aka Coroutines)](@ref man-tasks) to switch among multiple computations.

11. To express an order of execution between lightweight threads communication primitives are necessary.

12. Julia offers Channel(func::Function, ctype=Any, csize=0, taskref=nothing) that creates a new task from func, binds it to a new channel of type ctype and size csize and schedule the task.

13. Channels can serve as a way to communicate between tasks, as ChannelT(sz::Int) creates a buffered channel of type T and size sz.

14. Whenever code performs a communication operation like fetch or wait, the current task is suspended and a scheduler picks another task to run. A task is restarted when the event it is waiting for completes.

15. For many problems, it is not necessary to think about tasks directly. However, they can be used to wait for multiple events at the same time, which provides for dynamic scheduling. In dynamic scheduling, a program decides what to compute or where to compute it based on when other jobs finish. This is needed for unpredictable or unbalanced workloads, where we want to assign more work to processes only when they finish their current tasks.

16. The section on Tasks in Control Flow discussed the execution of multiple functions in a co-operative manner. Channels can be quite useful to pass data between running tasks, particularly those involving I/O operations.

17. Examples of operations involving I/O include reading/writing to files, accessing web services, executing external programs, etc. In all these cases, overall execution time can be improved if other tasks can be run while a file is being read, or while waiting for an external service/program to complete.

18. A channel can be visualized as a pipe, i.e., it has a write end and a read end :

    (a) Multiple writers in different tasks can write to the same channel concurrently via put! calls.

    (b) Multiple readers in different tasks can read data concurrently via take! calls.

19. Channels are created via the ChannelT(sz) constructor. The channel will only hold objects of type T. If the type is not specified, the channel can hold objects of any type. sz refers to the maximum number of elements that can be held in the channel at any time. For example, Channel(32) creates a channel that can hold a maximum of 32 objects of any type. A ChannelMyType(64) can hold up to 64 objects of MyType at any time.

20. If a Channel is empty, readers (on a take! call) will block until data is available.

21. If a Channel is full, writers (on a put! call) will block until space becomes available.

22. isready tests for the presence of any object in the channel, while wait waits for an object to become available.

23. A Channel is in an open state initially. This means that it can be read from and written to freely via take! and put! calls. close closes a Channel. On a closed Channel, put! will fail.

24. The current version of Julia multiplexes all tasks onto a single OS thread. Thus, while tasks involving I/O operations benefit from parallel execution, compute bound tasks are effectively executed sequentially on a single OS thread.

25. Future versions of Julia may support scheduling of tasks on multiple threads, in which case compute bound tasks will see benefits of parallel execution too.

Listing 1.20: App4-Julia Parallel Programming Codes

```
# File: julia-script-04.jl
# Date: Fri Aug 31 16:02:56 +08 2018
# Author: WRY
# ============================================================
# Comparing sequential and parallel code execution.
# PARALLEL COMPUTING IN JULIA LANGUAGE, developed at MIT, USA.
# ============================================================

        using Dates
        print(Dates.time(), " Current date today() = ", Dates.today(), "\n")
        print(Dates.time(), " Current date time now: ", Dates.now(), "\n")
        print(Dates.time(), " Bismillah from WRY in Julia script. \n")

        using PyCall
        @pyimport psutil
        nCPU = psutil.cpu_count()
        print(Dates.time(), " PyCall: Value of psutil.cpu_count() = $nCPU \n")

        using Hwloc
        topology = Hwloc.topology_load()
        counts = Hwloc.histmap(topology)
        ncores = counts[:Core]
        npus = counts[:PU]
```

```
24
25          print(Dates.time(), " Hwloc: This machine has $ncores cores and $npus PUs (processing units). \n")
26          nphysicalcores = Hwloc.num_physical_cores()
27          print(Dates.time(), " Hwloc: This machine has $nphysicalcores physical cores. \n")
28
29# ============================================================
30# PARALLEL COMPUTING IN JULIA
31# ============================================================
32# Create n workers at start of Julia session
33# MUST RUN "julia -p n" TO GET nprocs(), IF NOT ERROR
34
35# View number of workers + master process
36          my_nprocs = nprocs();
37          print(Dates.time(), " Number of workers + master process = $my_nprocs \n")
38
39# Create n workers during a session if required
40         # addprocs(n);
41
42# workers are addressed by numbers (PIDs)
43# master process had PID =1, the rest are PID of workers
44          my_wpid = workers();
45          print(Dates.time(), " List of workers PIDs = $my_wpid \n")
46
47# ============================================================
48function fxn02(input02, sleeprand02)
49# ============================================================
50          print(Dates.time(), " Started  running fxn02 \n")
51          print(Dates.time(), " Value input02 = $input02 \n")
52          print(Dates.time(), " Value sleeprand02 = $sleeprand02 \n")
53          sleep(sleeprand02)
54          str02 = " Result $input02 returned from fxn02"
55          print(Dates.time(), " Finished running fxn02 \n")
56          return (str02)
57end
58
59# ============================================================
60function fxn03(input03, sleeprand03)
61# ============================================================
62          print(Dates.time(), " Started  running fxn03 \n")
```

```
63          print(Dates.time(), " Value input03 = $input03 \n")
64          print(Dates.time(), " Value sleeprand03 = $sleeprand03 \n")
65          sleep(sleeprand03)
66          str03 = " Result $input03 returned from fxn03"
67          print(Dates.time(), " Finished running fxn03 \n")
68          return (str03)
69  end
70
71  % ========================================================
72  function fxn04(input04, sleeprand04)
73  # ========================================================
74          print(Dates.time(), " Started  running fxn04 \n")
75          print(Dates.time(), " Value input04 = $input04 \n")
76          print(Dates.time(), " Value sleeprand04 = $sleeprand04 \n")
77          sleep(sleeprand04)
78          str04 = " Result $input04 returned from fxn04"
79          print(Dates.time(), " Finished running fxn04 \n")
80          return (str04)
81  end
82
83  # ========================================================
84  function fxn05(input05, sleeprand05)
85  # ========================================================
86          print(Dates.time(), " Started  running fxn05 \n")
87          print(Dates.time(), " Value input05 = $input05 \n")
88          print(Dates.time(), " Value sleeprand05 = $sleeprand05 \n")
89          sleep(sleeprand05)
90          str05 = " Result $input05 returned from fxn05"
91          print(Dates.time(), " Finished running fxn05 \n")
92          return (str05)
93  end
94
95  # ========================================================
96  # DATA ASSIGNED TO FUNCTIONS
97          input02 = 2.2222; sleeprand02 = rand(1:10)
98          input03 = 3.3333; sleeprand03 = rand(1:10)
99          input04 = 4.4444; sleeprand04 = rand(1:10)
100         input05 = 5.5555; sleeprand05 = rand(1:10)
101
```

```
102# ============================================================
103# SEQUENTIAL FUNCTIONS EXECUTION
104# ============================================================
105
106         print(Dates.time(), " ======= START SEQUENTIAL FUNCTIONS EXECUTION ======= \n")
107         runfxn02 = fxn02(input02, sleeprand02)
108         runfxn03 = fxn03(input03, sleeprand03)
109         runfxn04 = fxn04(input04, sleeprand04)
110         runfxn05 = fxn05(input05, sleeprand05)
111
112# DISPLAY RESULTS OF EXECUTIONS
113         print(Dates.time(), " In Main runfxn02 = $runfxn02 \n")
114         print(Dates.time(), " In Main runfxn03 = $runfxn03 \n")
115         print(Dates.time(), " In Main runfxn04 = $runfxn04 \n")
116         print(Dates.time(), " In Main runfxn05 = $runfxn05 \n")
117
118# ============================================================
119# PARALLEL FUNCTIONS EXECUTIONS (ASYNCHRONOUSLY, OR NON BLOCKING)
120# ============================================================
121
122         print(Dates.time(), " ======= START PARALLEL FUNCTIONS EXECUTION ======= \n")
123         const jobs = Channel{Int}(32);
124         const results = Channel{Tuple}(32);
125
126# ============================================================
127# DO WORK
128# ============================================================
129         function do_work()
130                 for job_id in jobs
131                         print(Dates.time(), " job_id $job_id started do_work().\n")
132                         exec_time = rand(1:10)
133
134                         # DO SOMETHING USEFUL HERE
135                         sleep(exec_time)     # SIMULATION FOR ELAPSED TIME DOING REAL WORK
136
137                         print(Dates.time(), " job_id $job_id set to finish in $exec_time seconds. \n")
138                         put!(results, (job_id, exec_time))
139                         print(Dates.time(), " job_id $job_id finished do_work(). \n")
140                 end
```

```
141         end;
142
143 ============================================================
144 # MAKE JOBS
145 # ============================================================
146         function make_jobs(n)
147                 for i in 1:n
148                         put!(jobs, i)
149                 end
150         end;
151
152 # ============================================================
153 # FEED THE JOBS CHANNELS WITH "n" JOBS
154 # ============================================================
155         n = 4;
156         @async make_jobs(n);
157
158 # ============================================================
159 # START 4 TASKS TO PROCESS REQUESTS IN PARALLEL
160 # ============================================================
161         for i in 1:4
162                 @async do_work()
163         end
164
165 # ============================================================
166 # print out results
167 # ============================================================
168         @elapsed while n > 0
169                 job_id, exec_time = take!(results)
170                 print(Dates.time(), " job_id $job_id actually finished in $(round(exec_time; digits=4)) seconds \n")
171                 global n = n - 1
172         end
173
174 # ============================================================
175 print(Dates.time(), " Alhamdulillah from WRY in Julia script. \n")
176 # ============================================================
177
178 ALHAMDULILLAH 3 TIMES. WRY.
179 """
```

```
EXECUTION 1

wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp julia -p 4 julia-script04.jl
1.535760901044821e9 Current date today() = 2018-09-01
1.535760901176391e9 Current date time now: 2018-09-01T08:15:01.176
1.535760901266606e9 Bismillah from WRY in Julia script.
1.535760910499787e9 PyCall: Value of psutil.cpu_count() = 4
1.535760918305807e9 Hwloc: This machine has 2 cores and 4 PUs (processing units).
1.535760918463921e9 Hwloc: This machine has 2 physical cores.
1.535760918464832e9 Number of workers + master process = 5
1.535760918473897e9 List of workers PIDs = [2, 3, 4, 5]
1.535760918762892e9 ======= START SEQUENTIAL FUNCTIONS EXECUTION =======
1.5357609187739e9 Started  running fxn02
1.535760918774005e9 Value input02 = 2.2222
1.535760918774057e9 Value sleeprand02 = 1
1.535760919776272e9 Finished running fxn02
1.535760919798129e9 Started  running fxn03
1.535760919798278e9 Value input03 = 3.3333
1.535760919798374e9 Value sleeprand03 = 2
1.535760921801578e9 Finished running fxn03
1.535760921820814e9 Started  running fxn04
1.535760921820915e9 Value input04 = 4.4444
1.535760921820977e9 Value sleeprand04 = 1
1.535760922823194e9 Finished running fxn04
1.535760922847947e9 Started  running fxn05
1.535760922848116e9 Value input05 = 5.5555
1.535760922848213e9 Value sleeprand05 = 3
1.535760925852463e9 Finished running fxn05
1.53576092585304e9 In Main runfxn02 =  Result 2.2222 returned from fxn02
1.535760925853588e9 In Main runfxn03 =  Result 3.3333 returned from fxn03
1.535760925854025e9 In Main runfxn04 =  Result 4.4444 returned from fxn04
1.535760925854435e9 In Main runfxn05 =  Result 5.5555 returned from fxn05
1.535760925854789e9 ======= START PARALLEL FUNCTIONS EXECUTION =======
1.535760926140693e9 job_id 1 started
1.535760926140750e9 job_id 2 started
1.535760926140759e9 job_id 3 started
1.535760926140762e9 job_id 4 started
1.535760930146089e9 job_id 2 set to finish in 4 seconds.
1.535760930146177e9 job_id 4 set to finish in 4 seconds.
```

```
219 1.535760930146448e9 job_id 2 actually finished in 4.0 seconds
220 1.535760930178436e9 job_id 4 actually finished in 4.0 seconds
221 1.535760932143593e9 job_id 3 set to finish in 6 seconds.
222 1.535760932143825e9 job_id 3 actually finished in 6.0 seconds
223 1.535760935145178e9 job_id 1 set to finish in 9 seconds.
224 1.535760935145356e9 job_id 1 actually finished in 9.0 seconds
225 1.535760935145937e9 Alhamdulillah from WRY in Julia script.
226 wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp
227
228 ANALYSIS 1.1
229 1.535760,935,145,356e9  job_id 1 actually finished in 9.0 seconds
230 -1.535760,926,140,693e9 job_id 1 started
231 ========================
232 9,004,663e9 CONFIRMED 9 seconds,
233
234 ANALYSIS 1.2
235
236 1.535760932,143,825e9  job_id 3 actually finished in 6.0 seconds
237 -1.535760926,140,759e9 job_id 3 started
238 ========================
239 6,003,066e9 CONFIRMED 6 seconds.
240
241 EXECUTION 2
242
243 wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp julia -p 4 julia-script04.jl
244 1.535761821515869e9 Current date today() = 2018-09-01
245 1.535761821656556e9 Current date time now: 2018-09-01T08:30:21.656
246 1.535761821745387e9 Bismillah from WRY in Julia script.
247 1.535761830903611e9 PyCall: Value of psutil.cpu_count() = 4
248 1.535761838714373e9 Hwloc: This machine has 2 cores and 4 PUs (processing units).
249 1.535761838876378e9 Hwloc: This machine has 2 physical cores.
250 1.535761838876955e9 Number of workers + master process = 5
251 1.535761838901269e9 List of workers PIDs = [2, 3, 4, 5]
252 1.535761839174190e9 ======= START SEQUENTIAL FUNCTIONS EXECUTION =======
253 1.535761839185062e9 Started  running fxn02
254 1.535761839185163e9 Value input02 = 2.2222
255 1.535761839185216e9 Value sleeprand02 = 7
256 1.535761846192202e9 Finished running fxn02
257 1.535761846214519e9 Started  running fxn03
```

```
1.535761846214672e9 Value input03 = 3.3333
1.535761846214757e9 Value sleeprand03 = 4
1.535761850219946e9 Finished running fxn03
1.535761850239776e9 Started  running fxn04
1.535761850239921e9 Value input04 = 4.4444
1.535761850240005e9 Value sleeprand04 = 10
1.535761860247521e9 Finished running fxn04
1.535761860269229e9 Started  running fxn05
1.535761860269385e9 Value input05 = 5.5555
1.535761860269468e9 Value sleeprand05 = 9
1.535761869279662e9 Finished running fxn05
1.535761869280255e9 In Main runfxn02 =  Result 2.2222 returned from fxn02
1.535761869280763e9 In Main runfxn03 =  Result 3.3333 returned from fxn03
1.535761869281252e9 In Main runfxn04 =  Result 4.4444 returned from fxn04
1.535761869281725e9 In Main runfxn05 =  Result 5.5555 returned from fxn05
1.535761869282145e9 ====== START PARALLEL FUNCTIONS EXECUTION ======
1.535761869574609e9 job_id 1 started do_work().
1.535761869574664e9 job_id 2 started do_work().
1.535761869574676e9 job_id 3 started do_work().
1.535761869574683e9 job_id 4 started do_work().
1.535761872579027e9 job_id 4 set to finish in 3 seconds.
1.535761872579328e9 job_id 4 finished do_work().
1.535761872579457e9 job_id 4 actually finished in 3.0 seconds
1.535761873576851e9 job_id 2 set to finish in 4 seconds.
1.535761873577079e9 job_id 2 finished do_work().
1.535761873577144e9 job_id 2 actually finished in 4.0 seconds
1.535761875577793e9 job_id 1 set to finish in 6 seconds.
1.535761875578067e9 job_id 1 finished do_work().
1.535761875578148e9 job_id 1 actually finished in 6.0 seconds
1.535761876576700e9 job_id 3 set to finish in 7 seconds.
1.535761876576916e9 job_id 3 finished do_work().
1.535761876576973e9 job_id 3 actually finished in 7.0 seconds
1.535761876577773e9 Alhamdulillah from WRY in Julia script.
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/temp

ANALYSIS 2.1

1.535761,876,576,973e9  job_id 3 actually finished in 7.0 seconds
1.535761,876,576,916e9  job_id 3 finished do_work().
```

```
-1.535761,869,574,676e9 job_id 3 started do_work().
==========
7,002,240e9 CONFIRMED 7 seconds.

"""
ALHAMDULILLAH 3 TIMES.
# ============================================================
```

D 4.37 App4-Python Parallel Multithreading Codes

Listing 1.21: App4-Python Parallel Multithreading Codes

```
1
2#! /usr/bin/env python
3
4# File: 03_simple_python_multi_threading.py
5# Date: Thu Nov 29 12:23:16 MYT 2012
6# Author: WRY
7# Envn: Linux wruslan-ubuntu1004-rtai 2.6.32-122-rtai #rtai SMP Tue Jul 27 12:44:07 CDT 2010 i686 GNU/Linux
8
9"""
10NOTES ABOUT THIS PYTHON CODE:
11This program is about multi-threading implemented using the python programming language. Multi-threaded programs "
   share everything" in its environment. This is how simple and short the code looks like. For more information go to
    : http://www.doughellmann.com/PyMOTW/threading/
12"""
13import threading
14import datetime
15import time
16import logging
17import random
18
19logging.basicConfig(level=logging.DEBUG, format='(%(threadName)-10s) %(message)s', )
20
21def worker():
22"""thread worker function"""
23        t = threading.currentThread()
24        maxworktime = 30
25        pause = random.randint(1, maxworktime)
26        time.sleep(pause)
27
28        print "%18.12f" %(time.time() - globalStartTime), "\t", t.getName(), \
29        "worker already finished its assigned work(",pause,") time."
30
31        return
32
33# ===================================================================
```

```
34# MAIN PROGRAM
35# ================================================================
36globalStartTime = time.time()
37print "PROGRAM STARTING:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
38
39main_thread = threading.currentThread()
40numThreads = 10
41
42print "Number of worker threads to create = ", numThreads
43print "For mock execution each worker thread is assigned a random duration (1 to 30) seconds.\n"
44
45for i in range(numThreads):
46        t = threading.Thread(target=worker)
47        t.setDaemon(True)
48        t.start()
49
50for t in threading.enumerate():
51        if t is main_thread:
52                continue
53
54        t.join()
55
56print "\nThe last thread will never finish later than 30 seconds. This is multi-threading."
57print "PROGRAM FINISHED:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
58
59# ================================================================
60# EXECUTION
61# ================================================================
62"""
63wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$ python
64Python 2.7.12 (default, Dec  4 2017, 14:50:18)
65[GCC 5.4.0 20160609] on linux2
66Type "help", "copyright", "credits" or "license" for more information.
67>>>
68>>> import threading
69>>> quit()
70
71================================================================-
72wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$ python 02-multi-thread.py
```

```
PROGRAM STARTING: 2018-08-05 17:37:45.672530 at 7.91549682617e-05

Number of worker threads to create =  10
For mock execution each worker thread is assigned a random duration (1 to 30) seconds.

1.002974033356  Thread-8 worker already finished its assigned work( 1 ) time.
2.002993106842  Thread-6 worker already finished its assigned work( 2 ) time.
4.003006935120  Thread-1 worker already finished its assigned work( 4 ) time.
6.003776073456  Thread-2 worker already finished its assigned work( 6 ) time.
17.006188154221         Thread-10 worker already finished its assigned work( 17 ) time.
21.007598161697         Thread-3 worker already finished its assigned work( 21 ) time.
25.009006023407         Thread-7 worker already finished its assigned work( 25 ) time.
26.027007102966         Thread-4 worker already finished its assigned work( 26 ) time.
28.031121969223         Thread-9 worker already finished its assigned work( 28 ) time.
29.027016162872         Thread-5 worker already finished its assigned work( 29 ) time.

The last thread will never finish later than 30 seconds. This is multi-threading.
PROGRAM FINISHED: 2018-08-05 17:38:14.699948 at 29.0275480747

wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$
"""
# ================================================================
```

D 4.38 App4-Python Parallel Multiprocessing Codes

Listing 1.22: App4-Python Parallel Multiprocessing Codes

```
1
2#! /usr/bin/env python
3
4# File: 02_simple_python_multi_processing.py
5# Date: Thu Nov 29 16:38:40 MYT 2012
6# Author: WRY
7# Envn: Linux wruslan-ubuntu1004-rtai 2.6.32-122-rtai #rtai SMP Tue Jul 27 12:44:07 CDT 2010 i686 GNU/Linux
8
9"""
10NOTES ABOUT THIS PYTHON CODE:
11
12(1) This program is about multi-processing implemented using the python programming language. It is an example
    showing how to use queues to feed tasks to a collection of worker processes running in parallel and then collect
    the results.
13
14(2) Multi-processing programs "share nothing" in its environment. This means we must have mechanisms to exchange
    data, read and write data, etc, among the many simultaneously running processes. In this multi-processing program,
     we will use a "queue".
15
16(3) We are now talking about "creating processes" in multi-processing, instead of "creating threads" in multi-
    threading.
17
18(4) In this program, a "worker" means a child process. The "MainProcess" means the parent process. Both child and
    parent processes have a process ID. For example, a "pid" (for a child) and a "ppid" (for the parent). On the Linux
     terminal, run the command "ps -ef", and look for the ppid and the pid.
19
20(5) To execute this program, the command is: "python 01_simple_python_multi_processing.py | sort ". It is important
    to put in the sort command. Try without it. This will give a time ordered display of results.
21
22(6) What happens if you purposely omit the "sort" command? Do it and see for yourselves. Is there any order? Do this
     for all three programs given here.
23
24(7) Can you explain why "FINISHED:  bla bla bla" and "ALHAMDULILLAH WRY." are not the last two(2) statements to be
    displayed in the results, even though they are the last two(2) closing statements sent to the computer?
25
```

```
(8) In this program, we have purposely "commented many print lines" to avoid confusion. In the next program we open them all. Go and run it, i.e. 02_simple_python_multi_processing.py.

(9) In the third program, we cut away all the comments, i.e. 03_simple_python_multi_processing.py, and you should see how short the actual program really is (may not be clear). Ha ha ha. Now, you will recall about SE documentation and SE coding standards.

For more information go to :
http://docs.python.org/2/library/multiprocessing.html#multiprocessing-examples

"""
# ====================================================
# IMPORT THE REQUIRED PYTHON MODULES
# ====================================================
import multiprocessing
import datetime
import time
import random
import os
from    multiprocessing import Process, Queue, current_process, freeze_support

# ====================================================
def worker(input, output):

        for func, args in iter(input.get, 'STOP'):
                worker_result = calculate(func, args)
                output.put(worker_result)

def calculate(func, args):
        result = func(*args)

        # Note that this calculate_result is NOT a print statement (think)
        calculate_result = "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name,\
                                                    current_process().pid), \
                                                    "Child %s%s=%s" %((func.__name__), args, result)


        return calculate_result

def mult(a, b):
```

```
62          time.sleep(0.5*random.random())
63          return a * b
64
65  def plus(a, b):
66          time.sleep(0.5*random.random())
67          return a + b
68
69  def run_test():
70
71          TASKS1 = [(mult, (i, 7)) for i in range(10)]
72          TASKS2 = [(plus, (i, 8)) for i in range(5)]
73
74          task_queue = multiprocessing.Queue()
75          done_queue = multiprocessing.Queue()
76
77          for task in TASKS1:
78                  task_queue.put(task)
79
80          procs = []
81          for i in range(numProcesses):
82
83                  # Create processes by calling the python library
84                  p = multiprocessing.Process(target=worker, args=(task_queue, done_queue))
85                  p.start()
86                  procs.append(p)
87
88          for i in range(len(TASKS1)):
89                  print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name,\
90                  current_process().pid), \
91                  "done_queue = ", done_queue.get()
92
93          for task in TASKS2:
94                  task_queue.put(task)
95
96          for i in range(len(TASKS2)):
97                  print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name,\
98                  current_process().pid), \
99                  "done_queue = ", done_queue.get()
100
```

```
101         # Tell child processes to stop
102         for i in range(numProcesses):
103                 task_queue.put('STOP')
104
105# ====================================================
106# MAIN PROGRAM ENTRY
107# ====================================================
108if __name__ == '__main__':
109
110         globalStartTime = time.time()
111         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
112         "BISMILLAH WRY"
113         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
114         "STARTING:", datetime.datetime.now(), "at program run time %s " %(time.time() - globalStartTime), "seconds."
115
116         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
117         "Actual computer cpu_count()  = %d " % multiprocessing.cpu_count()
118
119         # TRY 4, 6, 20, 25 or 3
120         numProcesses = 15
121         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
122         "Number of processes to create:", numProcesses
123
124         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
125         "Begin multi-processing to run concurrent processes not concurrent threads."
126
127         # The freeze_support() line can be omitted if the program is not going
128         # to be frozen to produce a Windows executable.
129
130         # COMMENT THE NEXT LINE. USED ONLY FOR WINDOWS.
131         # freeze_support()
132
133         # THE RUN_TEST() ACTIVATES MULTI-PROCESSING
134         run_test()
```

```
135
136         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
137         "FINISHED:", datetime.datetime.now(), "at program run time %s " %(time.time() - globalStartTime), "seconds."
138         print "%18.12f %11s pid=%s," %(time.time() - globalStartTime, current_process().name, current_process().pid)
              , \
139         "ALHAMDULILLAH WRY."
140
141# ============================================================
142"""
143EXECUTION
144============================================================
145wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$ python
146Python 2.7.12 (default, Dec  4 2017, 14:50:18)
147[GCC 5.4.0 20160609] on linux2
148Type "help", "copyright", "credits" or "license" for more information.
149>>>
150>>> import multiprocessing
151>>> quit()
152
153EXECUTION 1
154============================================================
155wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$ python 04-multi-processing.py
1560.000000953674 MainProcess pid=16602, BISMILLAH WRY
1570.000068902969 MainProcess pid=16602, STARTING: 2018-08-05 19:15:59.476303 at program run time 0.000150918960571
    seconds.
1580.000164031982 MainProcess pid=16602, Actual computer cpu_count()  = 4
1590.000237941742 MainProcess pid=16602, Number of processes to create: 15
1600.000259876251 MainProcess pid=16602, Begin multi-processing to run concurrent processes not concurrent threads.
1610.026196002960 MainProcess pid=16602, Print from done_queue  =  ('0.245332956314   Process-7 pid=16610,', 'Finish
    child with result mult(7, 7) = 49')
1620.246630907059 MainProcess pid=16602, Print from done_queue  =  ('0.261003971100   Process-3 pid=16606,', 'Finish
    child with result mult(1, 7) = 7')
1630.261963844299 MainProcess pid=16602, Print from done_queue  =  ('0.282203912735   Process-2 pid=16605,', 'Finish
    child with result mult(2, 7) = 14')
1640.283308029175 MainProcess pid=16602, Print from done_queue  =  ('0.290824890137   Process-4 pid=16607,', 'Finish
    child with result mult(3, 7) = 21')
1650.291895866394 MainProcess pid=16602, Print from done_queue  =  ('0.378902912140   Process-6 pid=16609,', 'Finish
    child with result mult(4, 7) = 28')
```

```
166 0.380208015442 MainProcess pid=16602, Print from done_queue = ('0.403285026550   Process-5 pid=16608,', 'Finish
    child with result mult(5, 7) = 35')
167 0.404357910156 MainProcess pid=16602, Print from done_queue = ('0.487457036972   Process-9 pid=16612,', 'Finish
    child with result mult(9, 7) = 63')
168 0.488544940948 MainProcess pid=16602, Print from done_queue = ('0.494517803192  Process-10 pid=16613,', 'Finish
    child with result mult(6, 7) = 42')
169 0.495476961136 MainProcess pid=16602, Print from done_queue = ('0.508390903473   Process-1 pid=16604,', 'Finish
    child with result mult(0, 7) = 0')
170 0.509133815765 MainProcess pid=16602, Print from done_queue = ('0.521643877029  Process-13 pid=16616,', 'Finish
    child with result mult(8, 7) = 56')
171 0.523227930069 MainProcess pid=16602, Print from done_queue = ('0.587677001953  Process-14 pid=16617,', 'Finish
    child with result plus(3, 8) = 11')
172 0.588639020920 MainProcess pid=16602, Print from done_queue = ('0.634513854980  Process-11 pid=16614,', 'Finish
    child with result plus(2, 8) = 10')
173 0.635685920715 MainProcess pid=16602, Print from done_queue = ('0.718482971191   Process-8 pid=16611,', 'Finish
    child with result plus(1, 8) = 9')
174 0.719624996185 MainProcess pid=16602, Print from done_queue = ('0.930598974228  Process-15 pid=16618,', 'Finish
    child with result plus(4, 8) = 12')
175 0.931499004364 MainProcess pid=16602, Print from done_queue = ('0.957461833954  Process-12 pid=16615,', 'Finish
    child with result plus(0, 8) = 8')
176 0.958902835846 MainProcess pid=16602, FINISHED: 2018-08-05 19:16:00.435185 at program run time 0.959066867828
    seconds.
177 0.959134817123 MainProcess pid=16602, ALHAMDULILLAH WRY.
178 wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$
179
180 EXECUTION 2
181 ================================================================
182 wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$ python 04-multi-processing.py  | sort
183 0.000000953674 MainProcess pid=17070, BISMILLAH WRY
184 0.000041961670 MainProcess pid=17070, STARTING: 2018-08-05 20:17:56.402341 at program run time 0.000102043151855
    seconds.
185 0.000107049942 MainProcess pid=17070, Actual computer cpu_count()  = 4
186 0.000159025192 MainProcess pid=17070, Number of processes to create: 15
187 0.000170946121 MainProcess pid=17070, Begin multi-processing to run concurrent processes not concurrent threads.
188 0.024323940277 MainProcess pid=17070, Print from done_queue = ('0.042391061783   Process-5 pid=17077,', 'Finish
    child with result mult(5, 7) = 35')
189 0.043257951736 MainProcess pid=17070, Print from done_queue = ('0.103610992432   Process-8 pid=17080,', 'Finish
    child with result mult(7, 7) = 49')
```

```
190 0.104824066162 MainProcess pid=17070, Print from done_queue = ('0.140308856964   Process-4 pid=17076,', 'Finish
    child with result mult(3, 7) = 21')
191 0.141304969788 MainProcess pid=17070, Print from done_queue = ('0.254593849182   Process-2 pid=17074,', 'Finish
    child with result mult(1, 7) = 7')
192 0.255899906158 MainProcess pid=17070, Print from done_queue = ('0.265964031219   Process-3 pid=17075,', 'Finish
    child with result mult(2, 7) = 14')
193 0.267159938812 MainProcess pid=17070, Print from done_queue = ('0.321465015411   Process-6 pid=17078,', 'Finish
    child with result mult(4, 7) = 28')
194 0.322818994522 MainProcess pid=17070, Print from done_queue = ('0.378835916519   Process-9 pid=17081,', 'Finish
    child with result mult(8, 7) = 56')
195 0.380008935928 MainProcess pid=17070, Print from done_queue = ('0.379969835281   Process-7 pid=17079,', 'Finish
    child with result mult(6, 7) = 42')
196 0.380954980850 MainProcess pid=17070, Print from done_queue = ('0.478909969330   Process-1 pid=17073,', 'Finish
    child with result mult(0, 7) = 0')
197 0.479954004288 MainProcess pid=17070, Print from done_queue = ('0.513936042786  Process-13 pid=17085,', 'Finish
    child with result mult(9, 7) = 63')
198 0.515058040619 MainProcess pid=17070, Print from done_queue = ('0.721731901169  Process-10 pid=17082,', 'Finish
    child with result plus(2, 8) = 10')
199 0.722784042358 MainProcess pid=17070, Print from done_queue = ('0.736804962158  Process-11 pid=17083,', 'Finish
    child with result plus(3, 8) = 11')
200 0.738076925278 MainProcess pid=17070, Print from done_queue = ('0.776344060898  Process-14 pid=17086,', 'Finish
    child with result plus(0, 8) = 8')
201 0.777623891830 MainProcess pid=17070, Print from done_queue = ('0.782939910889  Process-15 pid=17087,', 'Finish
    child with result plus(1, 8) = 9')
202 0.784394979477 MainProcess pid=17070, Print from done_queue = ('0.818614006042  Process-12 pid=17084,', 'Finish
    child with result plus(4, 8) = 12')
203 0.820370912552 MainProcess pid=17070, FINISHED: 2018-08-05 20:17:57.222743 at program run time 0.820557832718
    seconds.
204 0.820574998856 MainProcess pid=17070, ALHAMDULILLAH WRY.
205 wruslan@HP-ELBook8470p-ub1604-64b:~/Documents/temp1$
206 """
207 # ============================================================
```

D 4.39 App4-Concurrent Writes to Parallel and Serial Ports

Listing 1.23: App4-Concurrent Writes to Parallel and Serial Ports

```
1
2// File: Serial_Parallel_Write.c
3// Date: Tue Nov 27 10:17:36 +08 2018
4// Author: WRY
5// ================================================================
6/* Description
7
8This C-program drives ASCII characters (8-bit integer numbers, 0 .. 255),
9and generates TTL electrical pulses through two(2) USB ports on the personal
10computer or notebook running Ubuntu Linux. The two USB attached devices
11basically extend the Linux computer system by:
12
13(1) USB-to-Serial PL2303 cable, making it a RS-232 serial port device.
14(2) USB-to-Parallel PL2305 cable, making it a IEEE-1284 parallel port device.
15
16Both cables are USB version 1.1 specification compliant.
17
18In this C-code, as each character from a predefined character array, TheChar[],
19is read, we write the character to the serial port and parallel port,
20sequentially, that is, one after another.
21
22Later on, depending on design, we may consider writing the character to the
23two devices (serial and parallel ports) simultaneously, meaning, execution
24parallel in time.
25
26The TTL electrical pulses are essentially the "8 bit" binary representation
27of each character sent to the respective devices. We can observe these bits
28using an oscilloscope. The more USB ports we extend on our computer (USB hosts),
29the more we can attach serial and parallel port devices.
30
31The PL2303 chip on the Prolific Technology USB-to-Serial cable device,
32is a full-duplex transmitter and receiver (TXD and RXD) for read and
33write serial communications.
34
35The PL2305 chip on the USB-to-Parallel cable emulates a Bidirectional
```

```
printer device so can also be used for read and write communications.

Later on, depending on design, we may consider both read and write
communications using these devices.

================================================================
COMPUTING ENVIRONMENT
================================================================
wruslan@HP-ELBook8470p-ub1604-64b:~$ date
Tue Nov 27 10:17:36 +08 2018

wruslan@HP-ELBook8470p-ub1604-64b:~$ uname -a
Linux HP-ELBook8470p-ub1604-64b 4.9.80rtai-5.1 #1 SMP Mon Apr 16 02:36:19 +08 2018 x86_64 x86_64 x86_64 GNU/Linux

wruslan@HP-ELBook8470p-ub1604-64b:~$ lsb_release -a
LSB Version:    core-9.20160110ubuntu0.2-amd64:core-9.20160110ubuntu0.2-noarch:security-9.20160110ubuntu0.2-amd64:
    security-9.20160110ubuntu0.2-noarch
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.5 LTS
Release:        16.04
Codename:       xenial

================================================================
SERIAL PORT SOFTWARE DRIVER (Cable USB-to-Serial PL2303)
================================================================
wruslan@HP-ELBook8470p-ub1604-64b:~$ ls -al /dev/ | grep ttyU
crw-rw----   1 root     dialout   188,   0 Nov 27 10:19 ttyUSB0
wruslan@HP-ELBook8470p-ub1604-64b:~$

wruslan@HP-ELBook8470p-ub1604-64b:~$ lsmod | grep pl
pl2303                 20480  0
usbserial              53248  1 pl2303

wruslan@HP-ELBook8470p-ub1604-64b:~$ dmesg
....
[ 6428.591444] usb 1-1.6: new full-speed USB device number 24 using ehci-pci
[ 6428.702458] usb 1-1.6: New USB device found, idVendor=067b, idProduct=2303
[ 6428.702464] usb 1-1.6: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 6428.702467] usb 1-1.6: Product: USB-Serial Controller
```

```
[ 6428.702470] usb 1-1.6: Manufacturer: Prolific Technology Inc.
[ 6428.703108] pl2303 1-1.6:1.0: pl2303 converter detected
[ 6428.705060] usb 1-1.6: pl2303 converter now attached to ttyUSB0
....
================================================================
PARALLEL PORT SOFTWARE DRIVER (Cable USB-to-Parallel PL2305)
================================================================
wruslan@HP-ELBook8470p-ub1604-64b:~$ ls -al /dev/usb/lp0
crw-rw---- 1 root lp 180, 0 Nov 26 22:43 /dev/usb/lp0

wruslan@HP-ELBook8470p-ub1604-64b:~$ lsmod | grep usbl
usblp                  20480  0

wruslan@HP-ELBook8470p-ub1604-64b:~$ dmesg
....
[ 6451.118420] usb 1-1.2: new full-speed USB device number 26 using ehci-pci
[ 6451.228661] usb 1-1.2: New USB device found, idVendor=067b, idProduct=2305
[ 6451.228667] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 6451.228670] usb 1-1.2: Product: IEEE-1284 Controller
[ 6451.228673] usb 1-1.2: Manufacturer: Prolific Technology Inc.
[ 6451.235287] usblp 1-1.2:1.0: usblp0: USB Bidirectional printer dev 26 if 0 alt 1 proto 2 vid 0x067B pid 0x2305
....

// REFERENCES: LINUX SERIAL PORT:
// https://stackoverflow.com/questions/13474923/read-dev-ttyusb0-with-a-c-program-on-linux
// https://stackoverflow.com/questions/10710083/serial-programming-on-linux-in-c
// https://www.codeproject.com/Questions/718340/C-program-to-Linux-Serial-port-read-write
// http://xanthium.in/Serial-Port-Programming-on-Linux
// https://github.com/xanthium-enterprises/Serial-Port-Programming-on-Linux/blob/master/USB2SERIAL_Write/Transmitter%20(PC%20Side)/SerialPort_write.c

*/
// ===================================================
// C-PROGRAM HEADER FILES
#include <stdio.h>
#include<string.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
```

```
112#include <sys/fcntl.h>
113#include <termios.h>
114
115// DEVICE: For-Cable-USB-to-Serial-PL2303.
116#define DEVICE_PORT_SER "/dev/ttyUSB0"
117
118// DEVICE: For-Cable-USB-to-Parallel-PL2305.
119#define DEVICE_PORT_PAR "/dev/usb/lp0"
120
121// PROTOTYPE FUNCTION DEFINITIONS
122void convert_integer_to_binary8 (int input_int, char *output_bin);
123void display_SERIAL_write_to_screen(int intIndex, int intWrite) ;
124void display_PARALLEL_write_to_screen(int intIndex, int intWrite);
125
126// GLOBAL VARIABLE DECLARATIONS
127// FOLLOW THE ORDER OF USING int main(int argc, char *argv[])
128char *TheChar = "Bismillah Alhamdulillah !";
129
130// BINARY REPRESENTATION FOR PRINTING (DISPLAY) TO SCREEN
131char bin8_output[33] = {0};      // LAST CHAR IS NULL "\0"
132
133// ================================================
134void convert_integer_to_binary8(int input_int, char *output_bin8) {
135// ================================================
136        unsigned int mask8;
137        mask8  = 0b10000000;
138        // printf("INTEGER INPUT = %d \n", input_int);
139
140        int bitposition = 0;
141        while (mask8)  {                        // Loop until MASK is empty
142
143                bitposition++;
144                if (input_int & mask8) {        // Bitwise AND => test the masked bit
145                        *output_bin8 = '1';     // if true, binary value 1 is appended to output array
146
147                } else {
148                        *output_bin8 = '0';     // if false, binary value 0 is appended to output array
149                }
150
```

```
151                 output_bin8++;                  // next character
152                 mask8 >>= 1;                    // shift the mask variable 1 bit to the right
153         }
154         *output_bin8 = 0;                       // add the trailing null
155}
156
157// =============================================================
158void display_SERIAL_write_to_screen(int intIndex, int intWrite) {
159// =============================================================
160
161         if (intWrite >= 1 && intWrite <=255) {
162                 convert_integer_to_binary8(intWrite, bin8_output);
163                 printf("SERIAL   write TheChar[%d] INT= %d \tHEX= 0x%x \tBIN= %s \tASCII= %c \n", intIndex, intWrite
                      , intWrite, bin8_output, intWrite);
164         } else {
165                 printf("ERROR: Integer to write is outside of range (0..255) \n");
166         }
167         // usleep(100000); // 0.1 sec delay (usleep = microsecond)
168}
169// =============================================================
170void display_PARALLEL_write_to_screen(int intIndex, int intWrite) {
171// =============================================================
172
173         if (intWrite >= 1 && intWrite <=255) {
174                 convert_integer_to_binary8(intWrite, bin8_output);
175                 printf("PARALLEL write TheChar[%d] INT= %d \tHEX= 0x%x \tBIN= %s \tASCII= %c \n", intIndex, intWrite
                      , intWrite, bin8_output, intWrite);
176         } else {
177                 printf("ERROR: Integer to write is outside of range (0..255) \n");
178         }
179         // usleep(100000); // 0.1 sec delay (usleep = microsecond)
180}
181// =============================================================
182int main(int argc, char *argv[]) {
183// =============================================================
184printf("Bismillah from WRY executed in main().\n\n");
185
186         // OPEN SERIAL PORT
187         // ================
```

```
188         // Set file descriptor device to write: Cable USB-to-Serial PL2303
189         int intFD_SER = open(DEVICE_PORT_SER, O_WRONLY | O_NOCTTY | O_NDELAY);
190         if(intFD_SER < 0) {
191                 perror(DEVICE_PORT_SER);
192                 printf("FAILED: SERIAL PORT. Cannot open file descriptor %s in main().\n", DEVICE_PORT_SER);
193                 printf("Value int file descriptor (fd_SER = %d) \n\n", intFD_SER);
194                 // exit(1);
195         } else {
196                 printf("SUCCESS: SERIAL PORT. Opened file descriptor %s in main().\n", DEVICE_PORT_SER);
197                 printf("Value int file descriptor (fd_SER = %d) \n\n", intFD_SER);
198         }
199
200         // OPEN PARALLEL PORT
201         // ==========================
202         // Set file descriptor device to write: Cable USB-to-Parallel PL2305
203         int intFD_PAR = open(DEVICE_PORT_PAR, O_WRONLY | O_NOCTTY | O_NDELAY);
204         if(intFD_PAR < 0) {
205                 perror(DEVICE_PORT_PAR);
206                 printf("FAILED: PARALLEL PORT. Cannot open file descriptor %s in main().\n", DEVICE_PORT_PAR);
207                 printf("Value int file descriptor (fd_PAR = %d) \n\n", intFD_PAR);
208                 // exit(1);
209         } else {
210                 printf("SUCCESS: PARALLEL PORT. Opened file descriptor %s in main().\n", DEVICE_PORT_PAR);
211                 printf("Value int file descriptor (fd_PAR = %d) \n\n", intFD_PAR);
212         }
213
214         // PERFORM WRITE LOOP FOR ELECTRICAL PULSE OUTPUTS
215         int intIndex;
216         int intToWrite = 0;
217
218         for (intIndex = 0; intIndex < strlen(TheChar); intIndex++) {
219
220                 // GRAB INTEGER TO WRITE (8-BITS) TO DEVICE
221                 intToWrite = TheChar[intIndex];
222
223                 // (1) EXECUTE DEVICE write() ELECTRICAL PULSES TO SERIAL PORT
224                 if (intFD_SER != -1) {
225                         write(intFD_SER, (const void *)TheChar[intIndex], 1);
226                         display_SERIAL_write_to_screen(intIndex, intToWrite);
```

```
227                 }
228
229                 // (2) EXECUTE DEVICE write() ELECTRICAL PULSES TO PARALLEL PORT
230                 if (intFD_PAR != -1) {
231                         write(intFD_PAR, (const void *)TheChar[intIndex], 1);
232                         display_PARALLEL_write_to_screen(intIndex, intToWrite);
233                 }
234
235         } // END FOR..LOOP
236
237         // CLOSE SERIAL AND PARALLEL PORTS
238         if (intFD_SER >= 0) { close(intFD_SER); }
239         if (intFD_PAR >= 0) { close(intFD_PAR); }
240
241 printf("\nAlhamdulillah from WRY executed in main().\n\n");
242 return(0);
243 }
244 // ============================================================
245 /* COMPILATION
246
247 wruslan@HP:~$ gcc -w -o Serial_Parallel_Write.x Serial_Parallel_Write.c
248
249 -rw-rw-r-- 1 wruslan wruslan 14183 Nov 27 14:25 Serial_Parallel_Write.c
250 -rwxrwxr-x 1 wruslan wruslan 13264 Nov 27 14:27 Serial_Parallel_Write.x
251
252 EXECUTION
253 ============================================================
254 wruslan@HP:~$ sudo ./Serial_Parallel_Write.x
255
256 [sudo] password for wruslan:
257 Bismillah from WRY executed in main().
258
259 SUCCESS: SERIAL PORT. Opened file descriptor /dev/ttyUSB0 in main().
260 Value int file descriptor (fd_SER = 3)
261
262 SUCCESS: PARALLEL PORT. Opened file descriptor /dev/usb/lp0 in main().
263 Value int file descriptor (fd_PAR = 4)
264
265 SERIAL    write TheChar[0] INT= 66        HEX= 0x42        BIN= 01000010    ASCII= B
```

```
266PARALLEL write TheChar[0] INT= 66  HEX= 0x42 BIN= 01000010 ASCII= B
267SERIAL   write TheChar[1] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
268PARALLEL write TheChar[1] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
269SERIAL   write TheChar[2] INT= 115 HEX= 0x73 BIN= 01110011 ASCII= s
270PARALLEL write TheChar[2] INT= 115 HEX= 0x73 BIN= 01110011 ASCII= s
271SERIAL   write TheChar[3] INT= 109 HEX= 0x6d BIN= 01101101 ASCII= m
272PARALLEL write TheChar[3] INT= 109 HEX= 0x6d BIN= 01101101 ASCII= m
273SERIAL   write TheChar[4] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
274PARALLEL write TheChar[4] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
275SERIAL   write TheChar[5] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
276PARALLEL write TheChar[5] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
277SERIAL   write TheChar[6] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
278PARALLEL write TheChar[6] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
279SERIAL   write TheChar[7] INT= 97  HEX= 0x61 BIN= 01100001 ASCII= a
280PARALLEL write TheChar[7] INT= 97  HEX= 0x61 BIN= 01100001 ASCII= a
281SERIAL   write TheChar[8] INT= 104 HEX= 0x68 BIN= 01101000 ASCII= h
282PARALLEL write TheChar[8] INT= 104 HEX= 0x68 BIN= 01101000 ASCII= h
283SERIAL   write TheChar[9] INT= 32  HEX= 0x20 BIN= 00100000 ASCII=
284PARALLEL write TheChar[9] INT= 32  HEX= 0x20 BIN= 00100000 ASCII=
285SERIAL   write TheChar[10] INT= 65  HEX= 0x41 BIN= 01000001 ASCII= A
286PARALLEL write TheChar[10] INT= 65  HEX= 0x41 BIN= 01000001 ASCII= A
287SERIAL   write TheChar[11] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
288PARALLEL write TheChar[11] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
289SERIAL   write TheChar[12] INT= 104 HEX= 0x68 BIN= 01101000 ASCII= h
290PARALLEL write TheChar[12] INT= 104 HEX= 0x68 BIN= 01101000 ASCII= h
291SERIAL   write TheChar[13] INT= 97  HEX= 0x61 BIN= 01100001 ASCII= a
292PARALLEL write TheChar[13] INT= 97  HEX= 0x61 BIN= 01100001 ASCII= a
293SERIAL   write TheChar[14] INT= 109 HEX= 0x6d BIN= 01101101 ASCII= m
294PARALLEL write TheChar[14] INT= 109 HEX= 0x6d BIN= 01101101 ASCII= m
295SERIAL   write TheChar[15] INT= 100 HEX= 0x64 BIN= 01100100 ASCII= d
296PARALLEL write TheChar[15] INT= 100 HEX= 0x64 BIN= 01100100 ASCII= d
297SERIAL   write TheChar[16] INT= 117 HEX= 0x75 BIN= 01110101 ASCII= u
298PARALLEL write TheChar[16] INT= 117 HEX= 0x75 BIN= 01110101 ASCII= u
299SERIAL   write TheChar[17] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
300PARALLEL write TheChar[17] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
301SERIAL   write TheChar[18] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
302PARALLEL write TheChar[18] INT= 105 HEX= 0x69 BIN= 01101001 ASCII= i
303SERIAL   write TheChar[19] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
304PARALLEL write TheChar[19] INT= 108 HEX= 0x6c BIN= 01101100 ASCII= l
```

```
305SERIAL    write TheChar[20] INT= 108     HEX= 0x6c      BIN= 01101100   ASCII= l
306PARALLEL  write TheChar[20] INT= 108     HEX= 0x6c      BIN= 01101100   ASCII= l
307SERIAL    write TheChar[21] INT= 97      HEX= 0x61      BIN= 01100001   ASCII= a
308PARALLEL  write TheChar[21] INT= 97      HEX= 0x61      BIN= 01100001   ASCII= a
309SERIAL    write TheChar[22] INT= 104     HEX= 0x68      BIN= 01101000   ASCII= h
310PARALLEL  write TheChar[22] INT= 104     HEX= 0x68      BIN= 01101000   ASCII= h
311SERIAL    write TheChar[23] INT= 32      HEX= 0x20      BIN= 00100000   ASCII=
312PARALLEL  write TheChar[23] INT= 32      HEX= 0x20      BIN= 00100000   ASCII=
313SERIAL    write TheChar[24] INT= 33      HEX= 0x21      BIN= 00100001   ASCII= !
314PARALLEL  write TheChar[24] INT= 33      HEX= 0x21      BIN= 00100001   ASCII= !
315
316Alhamdulillah from WRY executed in main().
317
318wruslan@HP:~$
319*/
```

D 4.40 App4-Converting software codes to binary bit pulses

Listing 1.24: App4-Converting-software-codes-to-binary-bits-pulses

```
1
2// File: Binary-data-string-representation.c
3// Date: Wed May  1 04:07:15 MYT 2013/2018 WRY
4// Author: WRY
5// Envn: Linux wruslan-rtai-bismillah 2.6.32-122-rtai #rtai SMP Tue Jul 27 12:44:07 CDT 2010 i686 GNU/Linux
6// Path: /home/wruslan/Documents/WRY-Hardware-ALL/aC-Programming
7//
8// ================================================================
9/*
10// http://tigcc.ticalc.org/doc/opers.html#assign
11The C language offers these bitwise and logical operators:
12
13&  (bitwise AND)
14^  (bitwise exclusive OR)
15|  (bitwise inclusive OR)
16
17&& (logical AND)
18|| (logical OR)
19
20They use the following syntax:
21
22expr1 & expr2
23expr1 ^ expr2
24expr1 | expr2
25expr1 && expr2
26expr1 || expr2
27
28In first three expressions, both operands must be of integral type.
29In fourth and fifth expressions, both operands must be of scalar type.
30
31The usual arithmetical conversions are performed on expr1 and expr2.
32
33// http://www.cprogramming.com/tutorial/bitwise_operators.html
34
35Thinking about Bits
```

```
36
37The byte is the lowest level at which we can access data; there's no "bit"
38type, and we can't ask for an individual bit. In fact, we can't even perform
39operations on a single bit — every bitwise operator will be applied to, at
40a minimum, an entire byte at a time. This means we'll be considering the whole
41representation of a number whenever we talk about applying a bitwise operator.
42
43(Note that this doesn't mean we can't ever change only one bit at a time; it
44just means we have to be smart about how we do it.)
45
46Understanding what it means to apply a bitwise operator to an entire string
47of bits is probably easiest to see with the shifting operators. By convention,
48in C and C++ you can think about binary numbers as starting with the most
49significant bit to the left (i.e., 10000000 is 128, and 00000001 is 1).
50
51Regardless of underlying representation, you may treat this as true. As a
52consequence, the results of the left shift (<<) and right shift (>>) operators
53are not implementation dependent for unsigned numbers (for signed numbers, the
54right shift operator is implementation defined).
55
56The leftshift operator is the equivalent of moving all the bits of a number
57a specified number of places to the left:
58
59[variable]<<[number of places]
60
61*/
62// ===========================================================
63#include <stdio.h>
64#include <string.h>
65#include <limits.h>
66
67char x, z;
68int y;
69
70// BINARY REPRESENTATION FOR PRINTING (DISPLAY) TO TERMINAL
71char bin8_output[33] = {0};      // LAST CHAR IS NULL "\0"
72char bin16_output[33] = {0};     // LAST CHAR IS NULL "\0"
73char bin32_output[33] = {0};     // LAST CHAR IS NULL "\0"
74
```

```
75 // FUNCTION PROTOTYPES
76 void convert_integer_to_binary8 (int input_int, char *output_bin);
77 void convert_integer_to_binary16(int input_int, char *output_bin);
78 void convert_integer_to_binary32(int input_int, char *output_bin);
79
80 // ============================================================
81 void convert_integer_to_binary8(int input_int, char *output_bin8) {
82 // ============================================================
83 unsigned int mask8;
84 unsigned int mask16;
85 unsigned int mask32;
86 //       used to check each individual bit,
87 //       unsigned to alleviate sign extension problems
88
89 //       Set only the high-end bit, 4-bits/byte
90 //       8-bit  = 2 bytes,  mask8  = 1000 0000
91 //       16-bit = 4 bytes,  mask16 = 1000 0000 0000 0000
92 //       32-bit = 8 bytes,  mask32 = 1000 0000 0000 0000 0000 0000 0000 0000
93
94 //       mask8  = 0x80;
95          mask8  = 0b10000000 ;
96 //       mask16 = 0x8000;
97          mask16 = 0b1000000000000000;
98 //       mask32 = 0x80000000;
99          mask32 = 0b10000000000000000000000000000000;
100
101         char *masked8[9]={"", "10000000","01000000","00100000", "00010000", \
102                               "00001000","00000100","00000010", "00000001"};
103
104         printf("INTEGER INPUT = %d \n", input_int);
105         int bitposition = 0;
106         while (mask8)  {                        // Loop until MASK is empty
107
108                 bitposition++;
109                 if (input_int & mask8) {        // Bitwise AND => test the masked bit
110                         *output_bin8 = '1';     // if true, binary value 1 is appended to output array
111
112                         /*  DEBUGGING
113                         printf("bitposition = %d \tTRUE \tmask8(BIN, DEC, HEX) = (%s, %d, %x) ", \
```

```
114                     bitposition, masked8[bitposition], mask8, mask8);
115                     printf("\toutput_bin8 = %s \n", output_bin8);
116                     */
117             } else {
118                     *output_bin8 = '0';     // if false, binary value 0 is appended to output array
119
120                     /* DEBUGGING
121                             printf("bitposition = %d \tFALSE \tmask8(BIN, DEC, HEX) = (%s, %d, %x) ", \
122                             bitposition, masked8[bitposition], mask8, mask8);
123                             printf("\toutput_bin8 = %s \n", output_bin8);
124                     */
125             } END if .. else
126
127             output_bin8++;                  // next character
128             mask8 >>= 1;                    // shift the mask variable 1 bit to the right
129
130     } // END while loop
131
132     *output_bin8 = 0;                       // add the trailing null to return value
133
134} // END void function()
135
136// ========================================================
137void convert_integer_to_binary16(int input_int, char *output_bin16) {
138// ========================================================
139unsigned int mask16;
140mask16 = 0x8000;
141
142printf("INTEGER INPUT = %d \n", input_int);
143
144char *masked16[17]={"", "1000000000000000", "0100000000000000", "0010000000000000", "0001000000000000", \
145                        "0000100000000000", "0000010000000000", "0000001000000000", "0000000100000000", \
146                        "0000000010000000", "0000000001000000", "0000000000100000", "0000000000010000", \
147                        "0000000000001000", "0000000000000100", "0000000000000010", "0000000000000001"};
148
149     int bitposition = 0;
150     while (mask16)  {          // Loop until MASK is empty
151
152             bitposition++;
```

```
153                 if (input_int & mask16) {      // Bitwise AND => test the masked bit
154
155                         *output_bin16 = '1';       // if true, binary value 1 is appended to output array
156
157                         /* DEBUG
158                         printf("bitposition = %d \tTRUE \tmask16(BIN, DEC, HEX) = (%s, %d, %x) ", \
159                         bitposition, masked16[bitposition], mask16, mask16);
160                         printf("\toutput_bin16 = %s \n", output_bin16);
161                         */
162                 } else {
163                         *output_bin16 = '0';      // if false, binary value 0 is appended to output array
164
165                         /* DEBUG
166                         printf("bitposition = %d \tFALSE \tmask16(BIN, DEC, HEX) = (%s, %d, %x) ", \
167                         bitposition, masked16[bitposition], mask16, mask16);
168                         printf("\toutput_bin16 = %s \n", output_bin16);
169                         */
170                 } END if ..else
171
172                 output_bin16++;                 // next character
173                 mask16 >>= 1;                   // shift the mask variable 1 bit to the right
174         } // END while .. loop
175
176         *output_bin16 = 0;                  // add the trailing null to return
177}
178
179// ================================================================
180void convert_integer_to_binary32(int input_int, char *output_bin32) {
181// ================================================================
182unsigned int mask32;
183mask32 = 0x80000000;
184
185printf("INTEGER INPUT = %d \n", input_int);
186
187char *masked32[33]={"", "1000000000000000", "0100000000000000", "0010000000000000", "0001000000000000", \
188                        "0000100000000000", "0000010000000000", "0000001000000000", "0000000100000000", \
189                        "0000000010000000", "0000000001000000", "0000000000100000", "0000000000010000", \
190                        "0000000000001000", "0000000000000100", "0000000000000010", "0000000000000001"};
191
```

```
192
193         int bitposition = 0;
194         while (mask32)  {          // Loop until MASK is empty
195
196                 bitposition++;
197                 if (input_int & mask32) {      // Bitwise AND => test the masked bit
198                         *output_bin32 = '1';      // if true, binary value 1 is appended to output array
199
200                         /* DEBUG
201                         printf("bitposition = %d \tTRUE \tmask32(DEC, HEX) = (%d, %x) ", \
202                         bitposition, mask32, mask32);
203                         printf("\toutput_bin32 = %s \n", output_bin32);
204                         */
205
206                 } else {
207                         *output_bin32 = '0';      // if false, binary value 0 is appended to output array
208
209                         /* DEBUG
210                         printf("bitposition = %d \tFALSE \tmask32(DEC, HEX) = (%d, %x) ", \
211                         bitposition, mask32, mask32);
212                         printf("\toutput_bin32 = %s \n", output_bin32);
213                         */
214                 } // END if .. else
215
216                 output_bin32++;                 // next character
217                 mask32 >>= 1;                   // shift the mask variable 1 bit to the right
218
219         } // END while .. loop
220
221 *output_bin32 = 0;                  // add the trailing null to return value
222 }
223 // ========================================================
224 void main(int argc, char* argv[]) {
225 // ========================================================
226 printf("\nBismillah.\n\n");
227
228         // READ INTEGER FROM TERMINAL
229         printf("Input a positive integer to convert to binary. Maximum = 2147483647: ");
230         scanf("%d", &y);          // store an int as y
```

```
231
232         // y = 2147483647;   // MAXIMUM FOR SIGNED INTEGER
233         printf("\nINT_MAX = %d \tINT_MIN = %d \n", INT_MAX, INT_MIN);
234         printf("CHAR_MAX = %d \tCHAR_MIN = %d \n\n", CHAR_MAX, CHAR_MIN);
235
236         if (y >= 1 && y <=255) {
237                         convert_integer_to_binary8(y, bin8_output);      // AN EXECUTION NOT AN ASSIGNMENT,
238                         printf("BINARY OUTPUT = %s \n", bin8_output);   // CORRECT
239                         printf("\n");
240
241         } else if (y >= 256 && y <=65535) {
242                         convert_integer_to_binary16(y, bin16_output);    // AN EXECUTION NOT AN ASSIGNMENT,
243                         printf("BINARY OUTPUT = %s \n", bin16_output);   // CORRECT
244                         printf("\n");
245
246         } else if (y >= 65536 && y <=2147483647) {
247                         convert_integer_to_binary32(y, bin32_output);    // AN EXECUTION NOT AN ASSIGNMENT,
248                         printf("BINARY OUTPUT = %s \n", bin32_output);   // CORRECT
249                         printf("\n");
250
251         } else  if (y > 2147483647) {
252                         printf("Invalid positive integer input. Maximum = 2147483647\n");
253
254         } else  {
255                         printf("Invalid positive integer input. Maximum = 2147483647\n");
256         }
257
258 printf("\nAlhamdulillah.\n\n");
259 return(0);
260 }
261 // ========================================================================
262 // COMPILATION AND EXECUTION
263 // ========================================================================
264 /*
265 COMPILATION
266 ============
267 wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/Temp/temp1$ gcc -o test1-convert-binary.x test1-convert-binary.c
268 wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/Temp/temp1$ ls -al
269 ....
```

```
-rw-rw-r-- 1 wruslan wruslan   6928 Dec 13 00:01 test1-convert-binary.c
-rwxrwxr-x 1 wruslan wruslan  13280 Dec 13 00:02 test1-convert-binary.x
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/Temp/temp1$

EXECUTION
=================
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/Temp/temp1$ ./test1-convert-binary.x
Bismillah.

Input a positive integer to convert to binary. Maximum = 2147483647: 255

INT_MAX = 2147483647     INT_MIN = -2147483648
CHAR_MAX = 127   CHAR_MIN = -128

INTEGER INPUT = 255
BINARY OUTPUT = 11111111

Alhamdulillah.
wruslan@HP-ELBook8470p-ub1604-64b:~/Downloads/Temp/temp1$

================================================
COLLECTING RESULTS ONLY

INTEGER INPUT = 14 BINARY OUTPUT = 00001110
INTEGER INPUT = 15 BINARY OUTPUT = 00001111
INTEGER INPUT = 16 BINARY OUTPUT = 00010000

INTEGER INPUT = 254 BINARY OUTPUT = 11111110
INTEGER INPUT = 255 BINARY OUTPUT = 11111111
INTEGER INPUT = 256 BINARY OUTPUT = 0000000100000000

INTEGER INPUT = 510 BINARY OUTPUT = 0000000111111110
INTEGER INPUT = 511 BINARY OUTPUT = 0000000111111111
INTEGER INPUT = 512 BINARY OUTPUT = 0000001000000000

INTEGER INPUT = 1022 BINARY OUTPUT = 0000001111111110
INTEGER INPUT = 1023 BINARY OUTPUT = 0000001111111111
INTEGER INPUT = 1024 BINARY OUTPUT = 0000010000000000

```

```
309INTEGER INPUT = 2147483646 BINARY OUTPUT = 01111111111111111111111111111110
310INTEGER INPUT = 2147483647 BINARY OUTPUT = 01111111111111111111111111111111
311
312INTEGER INPUT = 2147483648
313Invalid positive integer input. Maximum = 2147483647
314*/
315// ==========================================================
```

D 4.41 App4-Creation of 2D Model

1. **About 2D/3D model drawings and G-Code files**

   In summary, we say that a 2D or 3D model drawing file is about providing information on what we want to produce, meaning. the visual look of the model, whereas a G-Code command file for the 2D or 3D model drawing is about providing information on specific machine instructions on how to produce, meaning, the required machining steps to produce the model.

   Similarly, we say that CAD applications generate 2D or 3D model drawings as outputs while CAM applications generate G-Code file outputs from the 2D or 3D model drawings as inputs.

   The inputs to the CAD application are essentially, virtual and non-tangible ideas formed inside the human mind, that the designer wants to translate into models and drawings in software terms.

2. **Creation of a drawing 2D or 3D model**

   Many software applications can create 2D or 3D model drawings as described at the list of best 3D CAD modeling software.

   Some examples of commercial and high end applications are [AutoCAD], [Solid Works], [Fusion 360] and [BrisCAD]. Some examples of cost-free and open source applications are [Inkscape], [OpenSCAD], [FreeCAD], [BRL-CAD] and [Blender].

   The 2D or 3D model files are normally represented as standardized graphic vector files. A graphic vector file is a software file that represents its graphical elements in terms of text, and this text is editable manually with a text editor application. Some examples of graphic vector files are SVG, STL, PDF (Tikz), DXF, EPS, AI, and so on.

   On the other hand, graphical images that are saved as non-graphic files (scalar bitmap) are not represented in text and so cannot be edited using a text editor. Some examples of graphic bitmap files are BMP, JPEG/JPG, PNG, GIF, TIFF/TIF and so on. Non graphic files can only be edited using special GUI applications for example, Inkscape, Adobe Illustrator, Graphic Image Manipulation Program (GIMP), PaintBrush, OpenSCAD, Blender and so on.

3. **Example of a 2D model drawing using Inkscape**

   As an example in our CNC research work, a **2D KSG drawing** was created using the open source Inkscape graphic software application. This drawing was saved in SVG/XML textual vector format.

   A visual display image of the 2D KSG drawing is provided in the next figure at the link [1.39]. A textual(SVG/XML) listing display of the same 2D KSG drawing is provided in the next figure at the link **??**.

   This 2D KSG drawing was selected for illustration purposes because the drawing requires both linear (straight line) and circular arc (curve) movements. The character K is made up of totally linear movements, while the characters S and G require both linear and curve movements.

Example of a 2D model KSG drawing using Inkscape

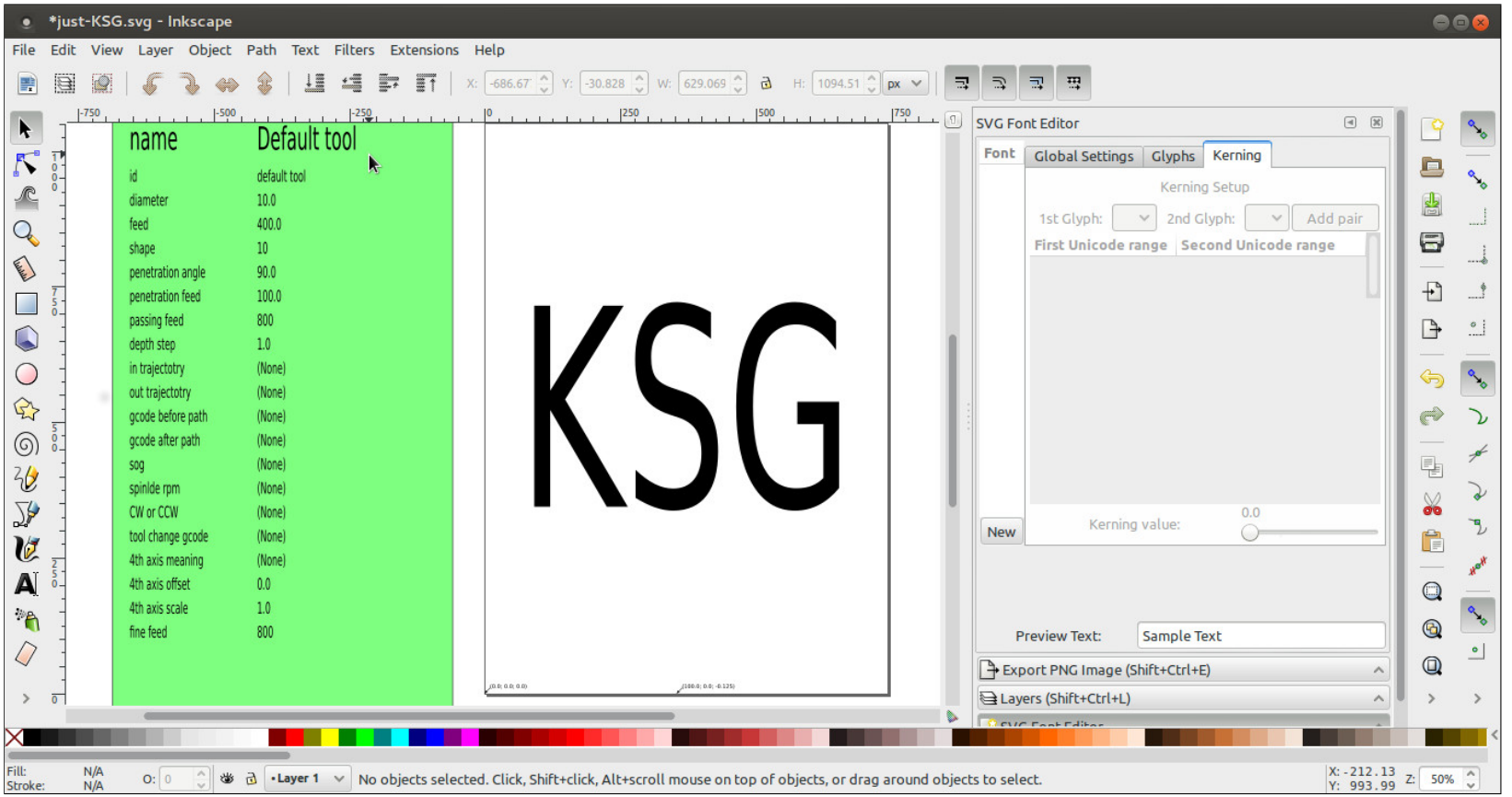


Figure 1.39: App4-Inkscape display of 2D KSG jpg file.

Display Text for 2D KSG model SVG/XML (Clipped text from 671 lines total)

Listing 1.25: App4-Display Text for 2D KSG model SVG/XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:cc="http://creativecommons.org/ns#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:svg="http://www.w3.org/2000/svg"
xmlns="http://www.w3.org/2000/svg"
xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
          width="210mm"
          height="297mm"
          viewBox="0 0 744.09448819 1052.3622047"
          id="svg4749"
          version="1.1"
          inkscape:version="0.91 r13725"
          sodipodi:docname="just-KSG.svg">
         <sodipodi:namedview
          id="base"
          pagecolor="#ffffff"
          bordercolor="#666666"
          borderopacity="1.0"
          inkscape:pageopacity="0.0"
          inkscape:pageshadow="2"
          inkscape:zoom="0.35"
          inkscape:cx="375"
          inkscape:cy="520"
          inkscape:document-units="px"
          inkscape:current-layer="layer1"
          showgrid="false"
          inkscape:window-width="1366"
          inkscape:window-height="689"
          inkscape:window-x="0"
          inkscape:window-y="24"
          inkscape:window-maximized="1" />
```

```
37          <defs
38          id="defs4751" />
39          <metadata
40          id="metadata4754">
41          <rdf:RDF>
42          <cc:Work
43          rdf:about="">
44          <dc:format>image/svg+xml</dc:format>
45          <dc:type
46          rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
47          <dc:title></dc:title>
48          </cc:Work>
49          </rdf:RDF>
50          </metadata>
51          <g
52          id="layer1"
53          inkscape:groupmode="layer"
54          inkscape:label="Layer 1">
55          <text
56          transform="scale(0.76565133,1.3060775)"
57          sodipodi:linespacing="125%"
58          id="text4757"
59          y="554.98346"
60          x="81.874046"
61          style="font-style:normal;font-weight:normal;font-size:389.44628906px;line-height:125%;font-family:sans-serif
              ;letter-spacing:0px;word-spacing:0px;fill:#000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-
              linecap:butt;stroke-linejoin:miter;stroke-opacity:1"
62          xml:space="preserve"><tspan
63          y="554.98346"
64          x="81.874046"
65          id="tspan4759"
66          sodipodi:role="line">KSG</tspan></text>
67          <g
68          id="g4787"
69          gcodetools="Gcodetools orientation group">
70          <g
71          id="g4789"
72          gcodetools="Gcodetools orientation point (2 points)">
73          <path
```

```
74          id="path4791"
75          style="stroke:none;fill:#000000;"
76          gcodetools="Gcodetools orientation point arrow"
77          d="m 0.0,1052.36220472 2.9375,-6.343750000001 0.8125,1.90625 6.843748640396,-6.84374864039 0,0 0.6875,0.6875
                -6.84375,6.84375 1.90625,0.812500000001 z z" />
78          <text
79          id="text4793"
80          xml:space="preserve"
81          y="1042.36220472"
82          x="10.0"
83          style="font-family:DejaVu Sans;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;
              font-family:DejaVu Sans;fill:#000000;fill-opacity:1;stroke:none;font-size:10.000000px;"
84          gcodetools="Gcodetools orientation point text"><tspan
85          id="tspan4795"
86          sodipodi:role="line"
87          y="1042.36220472"
88          x="10.0">(0.0; 0.0; 0.0)</tspan></text>
89          </g>
90
91 .... BEGIN CLIPPED TEXT
92 ....
93 .... END CLIPPED TEXT
94          x="150"
95          style="font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;font-size:10px;font-
              family:'DejaVu Sans';fill:#000000;fill-opacity:1;stroke:none"
96          gcodetools="Gcodetools tool definition field value"><tspan
97          id="tspan5017"
98          sodipodi:role="line"
99          y="305"
100         x="150">800</tspan></text>
101         </g>
102         </g>
103         </g>
104         <g
105         id="g5019" />
106</svg>
107THE END AT LINE 671
```

D 4.42 App4-Creation of G-Code for 2D Model

1. **CAM software tracing 2D model KSG drawing**

   To convert the model into G-Code, some software tool, like CAM, must trace the 2D model drawing (SVG or STL) for lines, curves and surfaces, then capture and save those point traces as numerical data into a file. These points saved into the file are only geometrical points. There are no instruction commands for a CNC machine to follow inside this file.

2. **CAM software added machining instructions to make G-Code**

   The next step for CAM, is to read the numerical data file, determine how a machine should trace the geometrical codes, and add these machine instructions into the data file. Now the file becomes a G-code file.

   This CAM file trace includes not only captures of the points as numerical path data (G series), but also CNC tool movements like speeds (F series) to move along the data points as well as associated CNC tool jumps from point-to-point. In addition, CNC service commands are also added into the G-Code file, like (M series) to start and stop the tool, to start and stop lubrication fluids, to pause and change tool cutters, and so on.

   Typically, the (G00, G01. G02 and G03) are the primary movement commands generated in conversion from models to G-Code files.

   Note that, a CAD model drawing file only contains path data and geometric elements. It does not contain CNC machine movements and other machine service instructions.

3. **Intensive CAM software computations to produce G-Code**

   Just consider a simple 2D model of the letters "KSG" as we have shown in the previous section. For that, the CAM engine must process about 700 lines of SVG text codes to produce the G-Code file. Imagine, for very large and complex 2D and 3D model drawings, for example, large SVG, STL files. The conversion of these model drawings to G-Code can be very time consuming, like hours to complete. And the resulting G-Code files can run into thousands of G-Code command lines. vspace0.2cm

   A single command instruction in G-Code file is usually written as a one line entry. This varies depending on the G-Codes standard in practice. Because of efficient curve interpolation, NURBS entries in G-Code like G06 series, make the overall G-Code file size smaller. However, NURBS G-Code entries are generally more complicated to interpret and execute by the CNC Interpreter and CNC Interpolator software, compared to common G01, G02 and G03 codes. High end FANUC CNC machines implement NURBS interpolation and they have their own software (proprietary) to accomplish that.

   In our illustration example, the KSG 2D drawing is simple, and only contains linear and arc elements with an SVG file consisting of about 700 lines. This number varies depending on software graphic applications and their vector format. For this drawing, we generated a G-Code file in RS274D NGC format with just 200 lines. This file does not contain NURBS G-Codes.

Page 204 of 230

An excerpt showing typical motion commands like G00, G01, G02 and G03 in G-Code for the KSG 2D model can be seen following that at Listing 1.26 .

The full textual contents of the G-Code generated file for our KSG 2D model drawing can be seen in the next section at Listing 1.27 .

Display excerpt listing for 2D KSG G-Code

Notice that from lines no. 27 until line no. 51 in this excerpt, we can see typical motion commands like G00, G01, G02 and G03 in G-Code.

Listing 1.26: App4-Display excerpt listing for 2D KSG G-Code

```
1 %
2 (Header)
3 (Generated by gcodetools from Inkscape.)
4 (Using default header. To add your own header create file "header" in the output dir.)
5 M3
6 (Header end.)
7 G21 (All units in mm)
8 #8  = 0 (Z axis offset)
9 #6  = 0 (X axis offset)
10 #7  = 0 (Y axis offset)
11 #10 = 1 (XY Scale factor)
12 #11 = 1 (Z Scale factor)
13 #21 = 400.000000 (Feed definition)
14 #20 = 100.000000 (Feed definition)
15
16 (Start cutting path id: path5167)
17 (Change tool to Default tool)
18
19 .... CLIPPED SECTION ----
20
21 (Start cutting path id: path5165)
22 (Change tool to Default tool)
23
24 G00 Z[5.000000*#11+#8]
25 G00 X[117.952117*#10+#6] Y[193.645846*#10+#7]
26
27 G01 Z[1.000000*#11+#8] F [#20](Penetrate)
28 G01 X[117.952117*#10+#6] Y[179.837424*#10+#7] Z[1.000000*#11+#8] F [#21]
29 G03 X[113.046989*#10+#6] Y[183.402174*#10+#7] Z[1.000000*#11+#8] I[-34.318323*#10] J[-42.065200*#10]
```

```
G03 X[109.035511*#10+#6] Y[185.585093*#10+#7] Z[1.000000*#11+#8] I[-16.685118*#10] J[-25.884338*#10]
G03 X[104.749703*#10+#6] Y[187.038280*#10+#7] Z[1.000000*#11+#8] I[-9.591941*#10] J[-21.242475*#10]
G03 X[100.940711*#10+#6] Y[187.477618*#10+#7] Z[1.000000*#11+#8] I[-3.808992*#10] J[-16.292016*#10]
G03 X[94.758678*#10+#6] Y[186.209903*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[-15.707261*#10]
G03 X[90.462671*#10+#6] Y[182.991632*#10+#7] Z[1.000000*#11+#8] I[4.472674*#10] J[-10.446948*#10]
G03 X[87.912396*#10+#6] Y[178.111685*#10+#7] Z[1.000000*#11+#8] I[10.615970*#10] J[-8.654302*#10]
G03 X[86.805629*#10+#6] Y[170.234616*#10+#7] Z[1.000000*#11+#8] I[27.477917*#10] J[-7.877068*#10]
G03 X[87.568633*#10+#6] Y[163.529681*#10+#7] Z[1.000000*#11+#8] I[29.841487*#10] J[0.000000*#10]
G03 X[89.229960*#10+#6] Y[159.720590*#10+#7] Z[1.000000*#11+#8] I[11.027232*#10] J[2.542666*#10]
G03 X[92.213942*#10+#6] Y[156.922563*#10+#7] Z[1.000000*#11+#8] I[8.237512*#10] J[5.794811*#10]
G03 X[98.516380*#10+#6] Y[154.043014*#10+#7] Z[1.000000*#11+#8] I[15.074024*#10] J[24.655545*#10]
G01 X[103.529404*#10+#6] Y[152.290678*#10+#7] Z[1.000000*#11+#8]
G02 X[111.731935*#10+#6] Y[147.939660*#10+#7] Z[1.000000*#11+#8] I[-8.284484*#10] J[-25.525105*#10]
G02 X[117.212491*#10+#6] Y[141.636464*#10+#7] Z[1.000000*#11+#8] I[-13.341928*#10] J[-17.134888*#10]
G02 X[120.313088*#10+#6] Y[133.703323*#10+#7] Z[1.000000*#11+#8] I[-23.402927*#10] J[-13.719316*#10]
G02 X[121.650249*#10+#6] Y[121.309349*#10+#7] Z[1.000000*#11+#8] I[-56.770488*#10] J[-12.393974*#10]
G02 X[119.784929*#10+#6] Y[106.730189*#10+#7] Z[1.000000*#11+#8] I[-57.907303*#10] J[-0.000000*#10]
G02 X[115.651058*#10+#6] Y[98.248584*#10+#7] Z[1.000000*#11+#8] I[-23.094899*#10] J[6.008071*#10]
G02 X[108.584210*#10+#6] Y[92.631505*#10+#7] Z[1.000000*#11+#8] I[-14.532534*#10] J[11.029440*#10]
G02 X[98.146568*#10+#6] Y[90.398113*#10+#7] Z[1.000000*#11+#8] I[-10.437642*#10] J[23.273202*#10]
G02 X[93.854953*#10+#6] Y[90.783690*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[24.076414*#10]
G02 X[88.860148*#10+#6] Y[92.080356*#10+#7] Z[1.000000*#11+#8] I[6.155305*#10] J[33.978893*#10]

--- CLIPPED SECTION ---

(Footer)
M5
G00 X0.0000 Y0.0000
M2
(Using default footer. To add your own footer create file "footer" in the output dir.)
(end)
%
```

Display full listing for 2D KSG G-Code

Listing 1.27: App4-Display full listing for 2D KSG G-Code

```
%
(Header)
(Generated by gcodetools from Inkscape.)
(Using default header. To add your own header create file "header" in the output dir.)
M3
(Header end.)
G21 (All units in mm)
#8  = 0 (Z axis offset)
#6  = 0 (X axis offset)
#7  = 0 (Y axis offset)
#10 = 1 (XY Scale factor)
#11 = 1 (Z Scale factor)
#21 = 400.000000 (Feed definition)
#20 = 100.000000 (Feed definition)

(Start cutting path id: path5167)
(Change tool to Default tool)

G00 Z[5.000000*#11+#8]
G00 X[176.423691*#10+#6] Y[107.360743*#10+#7]

G01 Z[1.000000*#11+#8] F [#20](Penetrate)
G01 X[176.423691*#10+#6] Y[135.468236*#10+#7] Z[1.000000*#11+#8] F [#21]
G01 X[162.863873*#10+#6] Y[135.468236*#10+#7] Z[1.000000*#11+#8]
G01 X[162.863873*#10+#6] Y[147.103758*#10+#7] Z[1.000000*#11+#8]
G01 X[184.641760*#10+#6] Y[147.103758*#10+#7] Z[1.000000*#11+#8]
G01 X[184.641760*#10+#6] Y[102.173824*#10+#7] Z[1.000000*#11+#8]
G02 X[179.391923*#10+#6] Y[96.903039*#10+#7] Z[1.000000*#11+#8] I[-30.942596*#10] J[25.569747*#10]
G02 X[174.040450*#10+#6] Y[93.342040*#10+#7] Z[1.000000*#11+#8] I[-19.182522*#10] J[23.025911*#10]
G02 X[168.066958*#10+#6] Y[91.148675*#10+#7] Z[1.000000*#11+#8] I[-12.219785*#10] J[24.048923*#10]
G02 X[161.672253*#10+#6] Y[90.398113*#10+#7] Z[1.000000*#11+#8] I[-6.394705*#10] J[26.865808*#10]
G02 X[149.370262*#10+#6] Y[93.974078*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[22.948566*#10]
G02 X[139.154740*#10+#6] Y[104.697188*#10+#7] Z[1.000000*#11+#8] I[16.440628*#10] J[25.889933*#10]
G02 X[133.559873*#10+#6] Y[119.722073*#10+#7] Z[1.000000*#11+#8] I[44.792682*#10] J[25.233731*#10]
G02 X[131.059941*#10+#6] Y[144.650484*#10+#7] Z[1.000000*#11+#8] I[123.038549*#10] J[24.928411*#10]
```

```
36 G02 X[133.566449*#10+#6] Y[169.644463*#10+#7] Z[1.000000*#11+#8] I[125.868687*#10] J[0.000000*#10]
37 G02 X[139.154740*#10+#6] Y[184.603785*#10+#7] Z[1.000000*#11+#8] I[49.856648*#10] J[-10.101281*#10]
38 G02 X[149.381921*#10+#6] Y[195.385610*#10+#7] Z[1.000000*#11+#8] I[26.785332*#10] J[-15.165982*#10]
39 G02 X[161.672253*#10+#6] Y[198.972953*#10+#7] Z[1.000000*#11+#8] I[12.290332*#10] J[-19.259830*#10]
40 G02 X[167.528571*#10+#6] Y[198.331573*#10+#7] Z[1.000000*#11+#8] I[0.000000*#10] J[-27.057176*#10]
41 G02 X[173.054281*#10+#6] Y[196.449585*#10+#7] Z[1.000000*#11+#8] I[-5.749598*#10] J[-25.934418*#10]
42 G02 X[178.118029*#10+#6] Y[193.430835*#10+#7] Z[1.000000*#11+#8] I[-12.194513*#10] J[-26.211890*#10]
43 G02 X[183.039237*#10+#6] Y[189.019673*#10+#7] Z[1.000000*#11+#8] I[-21.688851*#10] J[-29.147349*#10]
44 G01 X[183.039237*#10+#6] Y[173.949571*#10+#7] Z[1.000000*#11+#8]
45 G03 X[177.959907*#10+#6] Y[180.117456*#10+#7] Z[1.000000*#11+#8] I[-39.092615*#10] J[-27.017866*#10]
46 G03 X[173.259733*#10+#6] Y[183.972941*#10+#7] Z[1.000000*#11+#8] I[-19.689245*#10] J[-19.210212*#10]
47 G03 X[167.837734*#10+#6] Y[186.512086*#10+#7] Z[1.000000*#11+#8] I[-11.832629*#10] J[-18.208427*#10]
48 G03 X[162.370790*#10+#6] Y[187.337430*#10+#7] Z[1.000000*#11+#8] I[-5.466944*#10] J[-17.693398*#10]
49 G03 X[152.786893*#10+#6] Y[184.557277*#10+#7] Z[1.000000*#11+#8] I[0.000000*#10] J[-17.909143*#10]
50 G03 X[145.441564*#10+#6] Y[176.613122*#10+#7] Z[1.000000*#11+#8] I[11.266522*#10] J[-17.785156*#10]
51 G03 X[141.629050*#10+#6] Y[165.440943*#10+#7] Z[1.000000*#11+#8] I[33.955998*#10] J[-17.824110*#10]
52 G03 X[139.812185*#10+#6] Y[144.650484*#10+#7] Z[1.000000*#11+#8] I[118.044621*#10] J[-20.790459*#10]
53 G03 X[141.626890*#10+#6] Y[123.929590*#10+#7] Z[1.000000*#11+#8] I[119.206343*#10] J[-0.000000*#10]
54 G03 X[145.441564*#10+#6] Y[112.757940*#10+#7] Z[1.000000*#11+#8] I[37.812227*#10] J[6.674267*#10]
55 G03 X[152.786892*#10+#6] Y[104.813789*#10+#7] Z[1.000000*#11+#8] I[18.611844*#10] J[9.841002*#10]
56 G03 X[162.370790*#10+#6] Y[102.033636*#10+#7] Z[1.000000*#11+#8] I[9.583897*#10] J[15.128996*#10]
57 G03 X[166.706157*#10+#6] Y[102.379890*#10+#7] Z[1.000000*#11+#8] I[0.000000*#10] J[27.314245*#10]
58 G03 X[170.219047*#10+#6] Y[103.295320*#10+#7] Z[1.000000*#11+#8] I[-3.020477*#10] J[18.788763*#10]
59 G03 X[173.456867*#10+#6] Y[104.956924*#10+#7] Z[1.000000*#11+#8] I[-6.197101*#10] J[16.061169*#10]
60 G03 X[176.423691*#10+#6] Y[107.360743*#10+#7] Z[1.000000*#11+#8] I[-10.386640*#10] J[15.852077*#10]
61 G01 X[176.423691*#10+#6] Y[107.360743*#10+#7] Z[1.000000*#11+#8]
62 G00 Z[5.000000*#11+#8]
63
64 (End cutting path id: path5167)
65
66 (Start cutting path id: path5165)
67 (Change tool to Default tool)
68
69 G00 Z[5.000000*#11+#8]
70 G00 X[117.952117*#10+#6] Y[193.645846*#10+#7]
71
72 G01 Z[1.000000*#11+#8] F [#20](Penetrate)
73 G01 X[117.952117*#10+#6] Y[179.837424*#10+#7] Z[1.000000*#11+#8] F [#21]
74 G03 X[113.046989*#10+#6] Y[183.402174*#10+#7] Z[1.000000*#11+#8] I[-34.318323*#10] J[-42.065200*#10]
```

```
G03 X[109.035511*#10+#6] Y[185.585093*#10+#7] Z[1.000000*#11+#8] I[-16.685118*#10] J[-25.884338*#10]
G03 X[104.749703*#10+#6] Y[187.038280*#10+#7] Z[1.000000*#11+#8] I[-9.591941*#10] J[-21.242475*#10]
G03 X[100.940711*#10+#6] Y[187.477618*#10+#7] Z[1.000000*#11+#8] I[-3.808992*#10] J[-16.292016*#10]
G03 X[94.758678*#10+#6] Y[186.209903*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[-15.707261*#10]
G03 X[90.462671*#10+#6] Y[182.991632*#10+#7] Z[1.000000*#11+#8] I[4.472674*#10] J[-10.446948*#10]
G03 X[87.912396*#10+#6] Y[178.111685*#10+#7] Z[1.000000*#11+#8] I[10.615970*#10] J[-8.654302*#10]
G03 X[86.805629*#10+#6] Y[170.234616*#10+#7] Z[1.000000*#11+#8] I[27.477917*#10] J[-7.877068*#10]
G03 X[87.568633*#10+#6] Y[163.529681*#10+#7] Z[1.000000*#11+#8] I[29.841487*#10] J[0.000000*#10]
G03 X[89.229960*#10+#6] Y[159.720590*#10+#7] Z[1.000000*#11+#8] I[11.027232*#10] J[2.542666*#10]
G03 X[92.213942*#10+#6] Y[156.922563*#10+#7] Z[1.000000*#11+#8] I[8.237512*#10] J[5.794811*#10]
G03 X[98.516380*#10+#6] Y[154.043014*#10+#7] Z[1.000000*#11+#8] I[15.074024*#10] J[24.655545*#10]
G01 X[103.529404*#10+#6] Y[152.290678*#10+#7] Z[1.000000*#11+#8]
G02 X[111.731935*#10+#6] Y[147.939660*#10+#7] Z[1.000000*#11+#8] I[-8.284484*#10] J[-25.525105*#10]
G02 X[117.212491*#10+#6] Y[141.636464*#10+#7] Z[1.000000*#11+#8] I[-13.341928*#10] J[-17.134888*#10]
G02 X[120.313088*#10+#6] Y[133.703323*#10+#7] Z[1.000000*#11+#8] I[-23.402927*#10] J[-13.719316*#10]
G02 X[121.650249*#10+#6] Y[121.309349*#10+#7] Z[1.000000*#11+#8] I[-56.770488*#10] J[-12.393974*#10]
G02 X[119.784929*#10+#6] Y[106.730189*#10+#7] Z[1.000000*#11+#8] I[-57.907303*#10] J[-0.000000*#10]
G02 X[115.651058*#10+#6] Y[98.248584*#10+#7] Z[1.000000*#11+#8] I[-23.094899*#10] J[6.008071*#10]
G02 X[108.584210*#10+#6] Y[92.631505*#10+#7] Z[1.000000*#11+#8] I[-14.532534*#10] J[11.029440*#10]
G02 X[98.146568*#10+#6] Y[90.398113*#10+#7] Z[1.000000*#11+#8] I[-10.437642*#10] J[23.273202*#10]
G02 X[93.854953*#10+#6] Y[90.783690*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[24.076414*#10]
G02 X[88.860148*#10+#6] Y[92.080356*#10+#7] Z[1.000000*#11+#8] I[6.155305*#10] J[33.978893*#10]
G02 X[84.109236*#10+#6] Y[94.053384*#10+#7] Z[1.000000*#11+#8] I[13.446482*#10] J[39.084622*#10]
G02 X[78.710831*#10+#6] Y[97.056998*#10+#7] Z[1.000000*#11+#8] I[26.242970*#10] J[53.519667*#10]
G01 X[78.710831*#10+#6] Y[111.636444*#10+#7] Z[1.000000*#11+#8]
G03 X[84.025978*#10+#6] Y[107.168820*#10+#7] Z[1.000000*#11+#8] I[37.287981*#10] J[38.966103*#10]
G03 X[88.613605*#10+#6] Y[104.346723*#10+#7] Z[1.000000*#11+#8] I[19.239604*#10] J[26.136178*#10]
G03 X[93.599617*#10+#6] Y[102.469141*#10+#7] Z[1.000000*#11+#8] I[11.130303*#10] J[21.997980*#10]
G03 X[98.146568*#10+#6] Y[101.893449*#10+#7] Z[1.000000*#11+#8] I[4.546951*#10] J[17.668599*#10]
G03 X[104.614313*#10+#6] Y[103.237701*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[16.231605*#10]
G03 X[109.117692*#10+#6] Y[106.659805*#10+#7] Z[1.000000*#11+#8] I[-4.741455*#10] J[10.913805*#10]
G03 X[111.808038*#10+#6] Y[111.841476*#10+#7] Z[1.000000*#11+#8] I[-11.297040*#10] J[9.154726*#10]
G03 X[112.980184*#10+#6] Y[120.257946*#10+#7] Z[1.000000*#11+#8] I[-29.630701*#10] J[8.416470*#10]
G03 X[112.127054*#10+#6] Y[127.717636*#10+#7] Z[1.000000*#11+#8] I[-33.039967*#10] J[0.000000*#10]
G03 X[110.186041*#10+#6] Y[132.314028*#10+#7] Z[1.000000*#11+#8] I[-14.213271*#10] J[-3.294098*#10]
G03 X[106.818823*#10+#6] Y[135.882125*#10+#7] Z[1.000000*#11+#8] I[-10.873106*#10] J[-6.888092*#10]
G03 X[101.105073*#10+#6] Y[138.832725*#10+#7] Z[1.000000*#11+#8] I[-12.895623*#10] J[-17.964440*#10]
G01 X[96.050959*#10+#6] Y[140.514968*#10+#7] Z[1.000000*#11+#8]
G02 X[87.715741*#10+#6] Y[144.862945*#10+#7] Z[1.000000*#11+#8] I[9.895237*#10] J[29.132934*#10]
```

```
G02 X[82.614413*#10+#6] Y[150.398154*#10+#7] Z[1.000000*#11+#8] I[11.687787*#10] J[15.889965*#10]
G02 X[79.737752*#10+#6] Y[157.488256*#10+#7] Z[1.000000*#11+#8] I[19.564586*#10] J[12.066546*#10]
G02 X[78.464288*#10+#6] Y[169.113120*#10+#7] Z[1.000000*#11+#8] I[52.422259*#10] J[11.624864*#10]
G02 X[80.176218*#10+#6] Y[182.442710*#10+#7] Z[1.000000*#11+#8] I[52.750021*#10] J[0.000000*#10]
G02 X[84.175848*#10+#6] Y[190.982294*#10+#7] Z[1.000000*#11+#8] I[25.128284*#10] J[-6.562740*#10]
G02 X[91.023399*#10+#6] Y[196.835582*#10+#7] Z[1.000000*#11+#8] I[15.406407*#10] J[-11.091406*#10]
G02 X[99.995634*#10+#6] Y[198.972953*#10+#7] Z[1.000000*#11+#8] I[8.972235*#10] J[-17.763096*#10]
G02 X[104.263355*#10+#6] Y[198.651605*#10+#7] Z[1.000000*#11+#8] I[-0.000000*#10] J[-28.499848*#10]
G02 X[108.788968*#10+#6] Y[197.641175*#10+#7] Z[1.000000*#11+#8] I[-5.009916*#10] J[-33.078997*#10]
G02 X[113.155807*#10+#6] Y[196.046682*#10+#7] Z[1.000000*#11+#8] I[-10.854275*#10] J[-36.503587*#10]
G02 X[117.952117*#10+#6] Y[193.645846*#10+#7] Z[1.000000*#11+#8] I[-20.261739*#10] J[-46.469596*#10]
G01 X[117.952117*#10+#6] Y[193.645846*#10+#7] Z[1.000000*#11+#8]
G00 Z[5.000000*#11+#8]

(End cutting path id: path5165)

(Start cutting path id: path5163)
(Change tool to Default tool)

G00 Z[5.000000*#11+#8]
G00 X[25.950818*#10+#6] Y[197.080429*#10+#7]

G01 Z[1.000000*#11+#8] F [#20](Penetrate)
G01 X[34.251068*#10+#6] Y[197.080429*#10+#7] Z[1.000000*#11+#8] F [#21]
G01 X[34.251068*#10+#6] Y[152.851428*#10+#7] Z[1.000000*#11+#8]
G01 X[61.781606*#10+#6] Y[197.080429*#10+#7] Z[1.000000*#11+#8]
G01 X[72.465097*#10+#6] Y[197.080429*#10+#7] Z[1.000000*#11+#8]
G01 X[42.017145*#10+#6] Y[148.295349*#10+#7] Z[1.000000*#11+#8]
G01 X[74.642884*#10+#6] Y[92.430825*#10+#7] Z[1.000000*#11+#8]
G01 X[63.712853*#10+#6] Y[92.430825*#10+#7] Z[1.000000*#11+#8]
G01 X[34.251068*#10+#6] Y[142.828054*#10+#7] Z[1.000000*#11+#8]
G01 X[34.251068*#10+#6] Y[92.430825*#10+#7] Z[1.000000*#11+#8]
G01 X[25.950818*#10+#6] Y[92.430825*#10+#7] Z[1.000000*#11+#8]
G01 X[25.950818*#10+#6] Y[197.080429*#10+#7] Z[1.000000*#11+#8]
G01 X[25.950818*#10+#6] Y[197.080429*#10+#7] Z[1.000000*#11+#8]
G00 Z[5.000000*#11+#8]

(End cutting path id: path5163)

```

```
(Footer)
M5
G00 X0.0000 Y0.0000
M2

(Using default footer. To add your own footer create file "footer" in the output dir.)
(end)
%
```

E 5 Appendix-E5 Research Implementation Plan

E 5.1 App5-PhD Proposal Document Preparation

	Name	Duration	Start
1	**⊟Research Implementation Plan**	356 days	**11/20/18 8:00 AM**
2	**⊟PhD Proposal Preparation**	20 days	**11/20/18 8:00 AM**
3	Title Cover Page	2 days	11/20/18 8:00 AM
4	Abstract	2 days	11/20/18 8:00 AM
5	Acknowledgement	2 days	11/20/18 8:00 AM
6	Table of Contents	2 days	11/20/18 8:00 AM
7	Chapter-1 Introduction	10 days	11/20/18 8:00 AM
8	Chapter-2 Literature Survey	20 days	11/20/18 8:00 AM
9	Chapter-3 Research Methodology	10 days	11/20/18 8:00 AM
10	Chapter-4 Related Research Work	20 days	11/20/18 8:00 AM
11	Chapter-5 Research Implementation Plan	5 days	11/20/18 8:00 AM
12	Chapter-6 Conclusion	2 days	11/20/18 8:00 AM
13	Bibliography	10 days	11/20/18 8:00 AM
14	Chapter-7 Appendices	20 days	11/20/18 8:00 AM
15	Acronyms	5 days	11/20/18 8:00 AM
16	Glossary	5 days	11/20/18 8:00 AM
17	Index	5 days	11/20/18 8:00 AM
18	**⊟Issue Drafts PhD Proposal**	26 days	**11/20/18 8:00 AM**
19	Issue Draft No 6	1 day	11/20/18 8:00 AM
20	Issue Draft No 14	1 day	11/23/18 8:00 AM
21	Issue Draft No 24	1 day	12/7/18 8:00 AM
22	Issue Draft No 25 Semester 2 Submission	1 day	12/9/18 8:00 AM
23	Issue Chapter-1 Introduction	1 day	12/25/18 8:00 AM
24	Issue Chapter-1 Introduction Appendix	1 day	12/25/18 8:00 AM
25	Issue Chapter-3 Research Methodology	1 day	12/25/18 8:00 AM
26	Issue Chapter-3 Research Methodology Appendix	1 day	12/25/18 8:00 AM

Figure 1.40: App5-PhD Proposal Document Preparation

Note that the contents of a proposal document do not include the chapter on implementation design and the chapter on results and discussion, which are both required for a thesis document.

E 5.2 App5-Procurement of Long Lead Items and Project Setup

		Name	Duration	Start
29		**Procurement Long Lead Items**	30 days	**11/20/18 8:00 AM**
30		**PICO Universal PWM Servo Controller**	30 days	**11/20/18 8:00 AM**
31		Place Order	1 day	12/4/18 8:00 AM
32		Received Order	30 days	11/20/18 8:00 AM
33		Test Item	2 days	11/20/18 8:00 AM
34		**Microchip 28-Pin LIN Dev Demo Board**	30 days	**11/20/18 8:00 AM**
35		Place Order	1 day	12/18/18 8:00 AM
36		Received Order	30 days	11/20/18 8:00 AM
37		Test Item	2 days	11/20/18 8:00 AM
38		**Microchip Curiosity Dev Demo Board**	30 days	**11/20/18 8:00 AM**
39		Place Order	1 day	12/18/18 8:00 AM
40		Received Order	30 days	11/20/18 8:00 AM
41		Test Item	2 days	11/20/18 8:00 AM
42		**Solid State Relays MP240D2 SSR 2A, 8 nos.**	30 days	**11/20/18 8:00 AM**
43		Place Order	1 day	11/20/18 8:00 AM
44		Received Order	30 days	11/20/18 8:00 AM
45		Test Item	2 days	11/20/18 8:00 AM
46		**Setup CNC Research Machine**	5 days	**11/20/18 8:00 AM**
47		Tabletop preparation	2 days	11/20/18 8:00 AM
48		Rewiring Checks	2 days	11/20/18 8:00 AM
49		Power Supply	1 day	11/20/18 8:00 AM
50		Test CNC Machine	5 days	11/20/18 8:00 AM
51		Record CNC Auto Calibration	1 day	11/20/18 8:00 AM
52		Record CNC Machine Specifications	2 days	11/20/18 8:00 AM
53		Servo-Motor specifications	1 day	11/20/18 8:00 AM
54		Physical Dimension specifications	1 day	11/20/18 8:00 AM

Figure 1.41: App5-Procurement of Long Lead Items and Project Setup

The long lead items above are hardware resources that must be procured for the project.

E 5.3 App5-Computer 64bit Resources and Software Installations

		Name	Duration	Start
55		**Other Hardware Resources**	**5 days**	**11/26/18 8:00 AM**
56		Electrical 32V DC Power Supply	5 days	11/26/18 8:00 AM
57		Electronic Signal Generator (Analog and Digital)	5 days	11/26/18 8:00 AM
58		Dual Trace Analog/Digital Capture Oscilloscope	5 days	11/26/18 8:00 AM
59		Eight Trace Analog/Digital Software Oscilloscope	5 days	11/26/18 8:00 AM
60		**Computer Notebooks Setup**	**5 days**	**11/20/18 8:00 AM**
61		**HPEliteBook 8470p 64bit OS 4-cores**	**5 days**	**11/20/18 8:00 AM**
62		Ubuntu 16.04 LTS 64bit OS	5 days	11/20/18 8:00 AM
63		RTAI 5.1 Realtime	5 days	11/20/18 8:00 AM
64		C/C++ Compilers	5 days	11/20/18 8:00 AM
65		Python2 and Python3	5 days	11/20/18 8:00 AM
66		Julia	5 days	11/20/18 8:00 AM
67		Rust	5 days	11/20/18 8:00 AM
68		Scilab-NURBS	5 days	11/20/18 8:00 AM
69		Scilab XCos	5 days	11/20/18 8:00 AM
70		Octave-NURBS	5 days	11/20/18 8:00 AM
71		HDF5	5 days	11/20/18 8:00 AM
72		OpenMPI	5 days	11/20/18 8:00 AM
73		OpenMP	5 days	11/20/18 8:00 AM
74		LinuxCNC	5 days	11/20/18 8:00 AM

Figure 1.42: App5-Computer 64bit Resources and Software Installations

We will be implementing the CNC Control software on both 64bit and 32bit systems. The Linux kernel for the 64bit system is of version 4.X, while that for the 32bit system is of version 3.X. The Realtime RTAI library for the 64bit system is of version 5.1, while that for the 32bit system is version 2.7.

The above tasks in the figure identify the 64bit hardware resources and their associated software that are required for the project.

E 5.4 App5-Computer 32bit Resources and Software Installations

		Name	Duration	Start
75		⊟**HPEliteBook 8470p 32bit OS 4-Cores**	5 days	**11/20/18 8:00 AM**
76		Debian Wheezy 7.11 32 bit OS	5 days	11/20/18 8:00 AM
77		RTAI 2.7 Realtime	5 days	11/20/18 8:00 AM
78		C/C++ Compilers	5 days	11/20/18 8:00 AM
79		Python2 and Python3	5 days	11/20/18 8:00 AM
80		Julia	5 days	11/20/18 8:00 AM
81		Rust	5 days	11/20/18 8:00 AM
82		Scilab-NURBS	5 days	11/20/18 8:00 AM
83		Scilab XCos	5 days	11/20/18 8:00 AM
84		Octave-NURBS	5 days	11/20/18 8:00 AM
85		HDF5	5 days	11/20/18 8:00 AM
86		OpenMPI	5 days	11/20/18 8:00 AM
87		OpenMP	5 days	11/20/18 8:00 AM
88		LinuxCNC	5 days	11/20/18 8:00 AM
89		⊞**HPProBook 440G 64bit OS 4-Cores**	5 days	**11/20/18 8:00 AM**
103		⊞**HPProBook 440G 32bit OS 4-Cores**	5 days	**11/20/18 8:00 AM**
117		⊟**Computer Desktops Setup**	7 days	**12/25/18 8:00 AM**
118		⊞**SGI Desktop 64bit OS 8-cores**	7 days	**12/25/18 8:00 AM**
132		⊞**Intel Desktop 64bit OS 4-Cores**	7 days	**12/25/18 8:00 AM**

Figure 1.43: App5-Computer 32bit Resources and Software Installations

We will be implementing the CNC Control software on four(4) different computer hardware configurations (number of CPU cores, memory speeds, hard disk speeds, etc) for the purposes of performance comparisons. The multi-core feature of the computers is for true parallel computation design. The different computers are as follows:

1. Notebook HP EliteBook 8470p, 4-cores, installed with both 64bit and 32bit Linux operating systems.
2. Notebook HP ProBook 440G, 4-cores, installed with both 64bit and 32bit Linux operating systems.
3. Desktop SGI, 8-cores, installed with only 64bit Linux operating system. Does not accept 32bit OS.
4. Desktop Intel, 4-cores, installed with both 64bit and 32bit Linux operating systems.

The above tasks in the figure identify the 32bit hardware resources and their associated software that are required for the project.

E 5.5 App5-UseCase Design and Software Architecture Design

		Name	Duration	Start
146		**⊟UseCase Design - CNC Control Software**	**30 days**	**1/1/19 8:00 AM**
147		UseCase - System Logging Module	30 days	1/1/19 8:00 AM
148		UseCase - System Timing Module	30 days	1/1/19 8:00 AM
149		UseCase - CNC Control Loop	30 days	1/1/19 8:00 AM
150		UseCase - CNC Interpreter	30 days	1/1/19 8:00 AM
151		UseCase - CNC Interpolator	30 days	1/1/19 8:00 AM
152		UseCase - Signal Driver	30 days	1/1/19 8:00 AM
153		UseCase - CNC Motion Controller	30 days	1/1/19 8:00 AM
154		UseCase - CNC Error Controller	30 days	1/1/19 8:00 AM
155		UseCase - CNC Services Controller	30 days	1/1/19 8:00 AM
156		UseCase - CNC Human-Interface Controller	30 days	1/1/19 8:00 AM
157		**⊟Architecture Design - CNC Control Software**	**15 days**	**2/1/19 8:00 AM**
158		System Logging Module	15 days	2/1/19 8:00 AM
159		System Timing Module	15 days	2/1/19 8:00 AM
160		CNC Control Loop	15 days	2/1/19 8:00 AM
161		CNC Interpreter Module	15 days	2/1/19 8:00 AM
162		CNC Interpolator Module	15 days	2/1/19 8:00 AM
163		CNC Signal Driver Module	15 days	2/1/19 8:00 AM
164		CNC Motion Controller Module	15 days	2/1/19 8:00 AM
165		CNC Error Controller Module	15 days	2/1/19 8:00 AM
166		CNC Services Controller Module	15 days	2/1/19 8:00 AM
167		CNC Human-Interface Controller Module	15 days	2/1/19 8:00 AM

Figure 1.44: App5-UseCase Design and Software Architecture Design

E 5.6 App5-Process and Data Flow Design for the Control Loop

		Name	Duration	Start
168		⊟**Process and Data Flow Design**	15 days	**2/15/19 8:00 AM**
169		System Logging Module	15 days	2/15/19 8:00 AM
170		CNC Control Loop	15 days	2/15/19 8:00 AM
171		CNC Interpreter Module	15 days	2/15/19 8:00 AM
172		CNC Interpolator Module	15 days	2/15/19 8:00 AM
173		CNC Signal Driver Module	15 days	2/15/19 8:00 AM
174		CNC Motion Controller Module	15 days	2/15/19 8:00 AM
175		CNC Error Controller Module	15 days	2/15/19 8:00 AM
176		CNC Services Controller Module	15 days	2/15/19 8:00 AM
177		CNC Human-Interface Controller Module	15 days	2/15/19 8:00 AM
178		CNC Control Loop Development	20 days	**3/1/19 8:00 AM**
179		Design SigSlot Architecture	20 days	3/1/19 8:00 AM
180		Design SigSlot Algorithm	20 days	3/1/19 8:00 AM
181		Design SigSlot with Encoder Feedback	20 days	3/1/19 8:00 AM
182		Write SigSlot C/C++ Component Test Codes	10 days	3/1/19 8:00 AM
183		Test SigSlot Algorithm	5 days	3/1/19 8:00 AM
184		Write SigSlot C/C++ Realtime Codes	10 days	3/1/19 8:00 AM
185		Test SigSlot C/C++ Realtime Algorithm	5 days	3/1/19 8:00 AM
186		Write SigSlot C/C++ Realtime + Parallel Codes	10 days	3/1/19 8:00 AM
187		Test SigSlot C/C++ Realtime + Parallel Algorithm	5 days	3/1/19 8:00 AM
188		Write SigSloc C/C++ Realtime + Parallel + Feedback	10 days	3/1/19 8:00 AM
189		Test SigSloc C/C++ Realtime + Parallel + Feedback	5 days	3/1/19 8:00 AM

Figure 1.45: App5-Process and Data Flow Design for the Control Loop

E 5.7 App5-Test-Case Based Design in Software Construction

		Name	Duration	Start
190		**⊟CNC Control Loop Integration Testing**	20 days	**3/15/19 8:00 AM**
191		Test Control Loop Logging and Timing	20 days	3/15/19 8:00 AM
192		Test Interpreter Logging and Timing	20 days	3/15/19 8:00 AM
193		Test Interpolator Logging and Timing	20 days	3/15/19 8:00 AM
194		Test Signal Driver Logging and Timing	20 days	3/15/19 8:00 AM
195		Test Motion Controller Logging and Timing	20 days	3/15/19 8:00 AM
196		Test Error Controller Logging and Timing	20 days	3/15/19 8:00 AM
197		Test Services Controller Logging and Timing	20 days	3/15/19 8:00 AM
198		Test Human Interface Logging and Timing	20 days	3/15/19 8:00 AM
199		**⊟Create Test Models G-Codes**	3 days	**4/12/19 8:00 AM**
200		Circle, Square, Semi-Circle, Rectangle, Hexagon	3 days	4/12/19 8:00 AM
201		Starfish, Complex objects, Curves	3 days	4/12/19 8:00 AM
202		**⊟Execute Test Models without Error Compensation**	3 days	**4/15/19 8:00 AM**
203		Run CNC on Test Models	3 days	4/15/19 8:00 AM
204		Measure and capture test results	3 days	4/15/19 8:00 AM
205		**⊟Contour Error Module Development**	40 days	**4/1/19 8:00 AM**
206		Incorporate NURBS	40 days	4/1/19 8:00 AM
207		Integrate with Scilab-NURBS	40 days	4/1/19 8:00 AM
208		Integrate with Octave-NURBS	40 days	4/1/19 8:00 AM
209		Integrate with realtime computations	40 days	4/1/19 8:00 AM
210		Integrate with parallel computations	40 days	4/1/19 8:00 AM
211		integrate with HDF5 high speed capture	40 days	4/1/19 8:00 AM
212		One step-look-ahead	40 days	4/1/19 8:00 AM
213		Two step-look-ahead	40 days	4/1/19 8:00 AM
214		Compute contour errors	40 days	4/1/19 8:00 AM
215		Adjust pulse feedrate (machine velocity)	40 days	4/1/19 8:00 AM

Figure 1.46: App5-Test-Case Based Design in Software Construction

E 5.8 App5-Completion Testing, Publication Plan and Project Closing

	Name	Duration	Start
216	**⊟Execute Test Models with Error Compensation**	4 days	**5/25/19 8:00 AM**
217	Run CNC on Test Models	3 days	5/25/19 8:00 AM
218	Measure and capture test results	3 days	5/28/19 8:00 AM
219	**⊟Test with signal generator devices**	20 days	**6/3/19 8:00 AM**
220	Parallel Port on Desktops	5 days	6/3/19 8:00 AM
221	Pico Universal PWM Servo Board	5 days	6/8/19 8:00 AM
222	MCU 28-Pin LIN Demo Board	5 days	6/15/19 8:00 AM
223	MCU Curiosity Demo Board	5 days	6/22/19 8:00 AM
224	**⊟Publications Plan**	140 days	**6/30/19 8:00 AM**
225	Publication No 1 Using the Universal PWM with LinuxCNC	30 days	6/30/19 8:00 AM
226	Publication No 2 Using MCU Curiosity + MCU 28-Pin LIN	30 days	9/30/19 8:00 AM
227	Publication No 3 Contour Error CNC Compensation	30 days	11/30/19 8:00 AM
228	**⊟Project Closing**	110 days	**10/30/19 8:00 AM**
229	Write PhD Thesis	20 days	10/30/19 8:00 AM
230	Submit PhD Thesis WriteUp	2 days	11/30/19 8:00 AM
231	Perform Thesis Corrections	10 days	1/31/20 8:00 AM
232	Resubmit Thesis	2 days	2/27/20 8:00 AM
233	Complete Viva Voce	2 days	3/30/20 8:00 AM

Figure 1.47: App5-Completion Testing, Publication Plan and Project Closing

E 5.9 App5-Computer Notebook Specifications

Table 1.4: App5-Computer Notebook Specifications

Computer Notebook Specifications			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.10 App5-Computer Desktop Specifications

Table 1.5: App5-Computer Desktop Specifications

Computer Desktop Specifications			
No	**Specifications**	**WRY Intel Desktop**	**WRY SGI Desktop**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.11 App5-Pulse Generator Interface Boards

Table 1.6: App5-Pulse Generator Interface Boards

Pulse Generator Interface Boards				
No	**Specifications**	**Pico Universal PWM Servo Controller Board**	**MCU Microchip 28-Pin LIN Development Board**	**MCU Microchip Curiosity Development Board**
1	Name of Student	Wan Ruslan bin W Yusoff	hello	Hello
2	Student ID	aaa	Hello	hello
3	National Reg. ID	bbb	Hello	Hello
4	Faculty	ccc	Hello	Hello

E 5.12 App5-Software Programming Language Features

Table 1.7: App5-Software Programming Language Features

Software Programming Language Features			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.13 App5-Software Programming Paradigms

Table 1.8: App5-Software Programming Paradigms

Software Programming Paradigms			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.14 App5-Scilab NURBS versus Octave NURBS packages

Table 1.9: App5-Scilab NURBS versus Octave NURBS packages

Scilab NURBS versus Octave NURBS packages			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.15 App5-Python versus Julia programming languages

Table 1.10: App5-Python versus Julia programming languages

Python versus Julia programming languages			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

E 5.16 App5-C/C++ versus Rust system programming languages

Table 1.11: App5-C/C++ versus Rust system programming languages

C/C++ versus Rust system programming languages			
No	**Specifications**	**Hewlett Packard EliteBook 8470p**	**Hewlett Packard ProBook 440G**
1	Name of Student	Wan Ruslan bin W Yusoff	hello
2	Student ID	aaa	Hello
3	National Reg. ID	bbb	Hello
4	Faculty	ccc	Hello

Acronyms

This document is incomplete. The external file associated with the glossary 'acronym' (which should be called `main-phd-proposal-WRY.acr`) hasn't been created.

Check the contents of the file `main-phd-proposal-WRY.acn`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

You may need to rerun LaTeX. If you already have, it may be that TeX's shell escape doesn't allow you to run makeindex. Check the transcript file `main-phd-proposal-WRY.log`. If the shell escape is disabled, try one of the following:

- Run the external (Lua) application:

  ```
  makeglossaries-lite "main-phd-proposal-WRY"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "main-phd-proposal-WRY"
  ```

Then rerun LaTeX on this document.
This message will be removed once the problem has been fixed.

Glossary

This document is incomplete. The external file associated with the glossary 'main' (which should be called `main-phd-proposal-WRY.gls`) hasn't been created.

Check the contents of the file `main-phd-proposal-WRY.glo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

You may need to rerun LaTeX. If you already have, it may be that TeX's shell escape doesn't allow you to run makeindex. Check the transcript file `main-phd-proposal-WRY.log`. If the shell escape is disabled, try one of the following:

- Run the external (Lua) application:

  ```
  makeglossaries-lite "main-phd-proposal-WRY"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "main-phd-proposal-WRY"
  ```

Then rerun LaTeX on this document.
This message will be removed once the problem has been fixed.

Index