Integrated Embedded Microprocessor Development System

Integrated Embedded Microprocessor Development System

Author

Mr. R. J. Spriggs

Course Title

Applied Computing and Electronics
BEng(Hons)

Abstract.

The development of an Embedded Microprocessor Development System is discussed, drawing out the following aspects :-

- The design methodology and implementation used for the system and its subordinate products production.
- How the constraints of limited finance impinge on production of a cost effective and complete Development System Tool Set.

The novel aspects of this project include :-

- The concept of using Single Tool entities which can support a wide range of processors (Past, Present and Future) through self-contained dynamically linked libraries attached at run time.
- The use of Application Specific Languages for system customisation.
- Using Box Concept Design Method as the design tool.

The fully functioning parts of the system are :-

- The Integrated Development Environment for Software production.
- The General Purpose Cross Assembler.
- The General Purpose Macro Processor.
- The Limited Features General Purpose Object Code Linker.
- The General Purpose Processor Simulator.
- The General Purpose Device programmer operating in board level test mode.

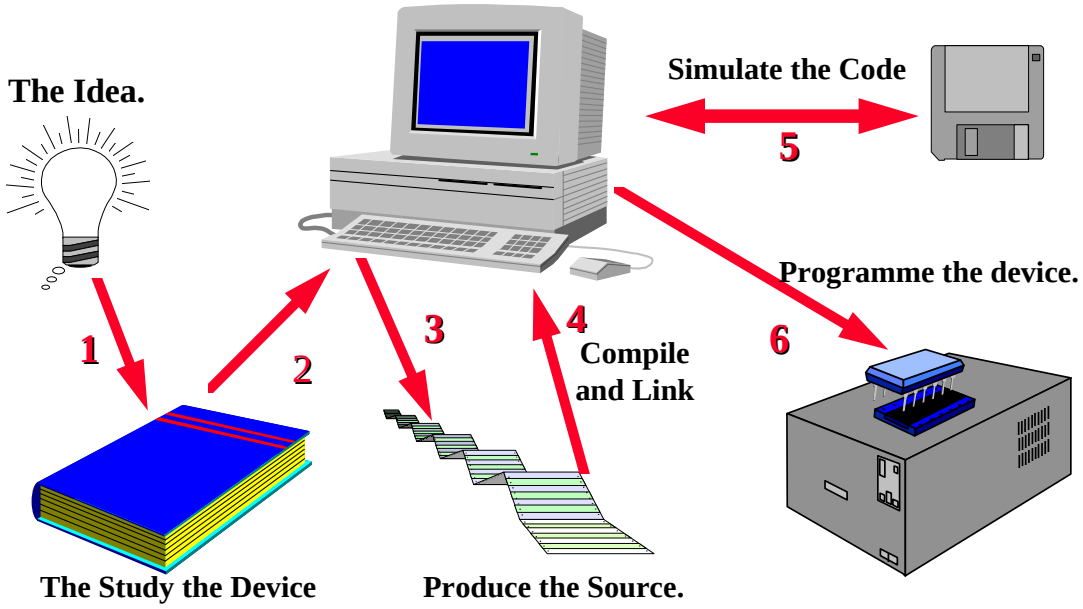
The product set consists of both Software and Hardware designed entities of which many are capable of being used in a fully stand alone mode.
A Significant part of the Tool Set was used to develop the Device Programmer showing the viability of the product package.

Table of Contents. Page

1.0 Introduction.

This dissertation will help to explain the methodology used for creating a fully Integrated Low Cost Embedded Microprocessor Development System. As a general rule, current electronics projects are most likely to involve a microprocessor and some discrete logic elements. The discrete logic elements may well be packaged into one or more programmed logic device/s. When a developer considers the implications of a new design, some of the major constraints are “the cost” and “the availability” of the development tools. In contrast, the advanced modern components remain relatively cheap when compared with the traditional approach components. The high cost of the tools and the experience required are major reasons why a developer will persist in using an older technology rather than innovate and use perhaps more appropriate and modern devices. An additional reason for remaining with the current development methods is that the developer is usually faced with a steep learning curve before he can progress with the project. Therefore, any system that can ease this learning curve and yet allow a rapid forward movement should be seen as an advantageous route to follow.

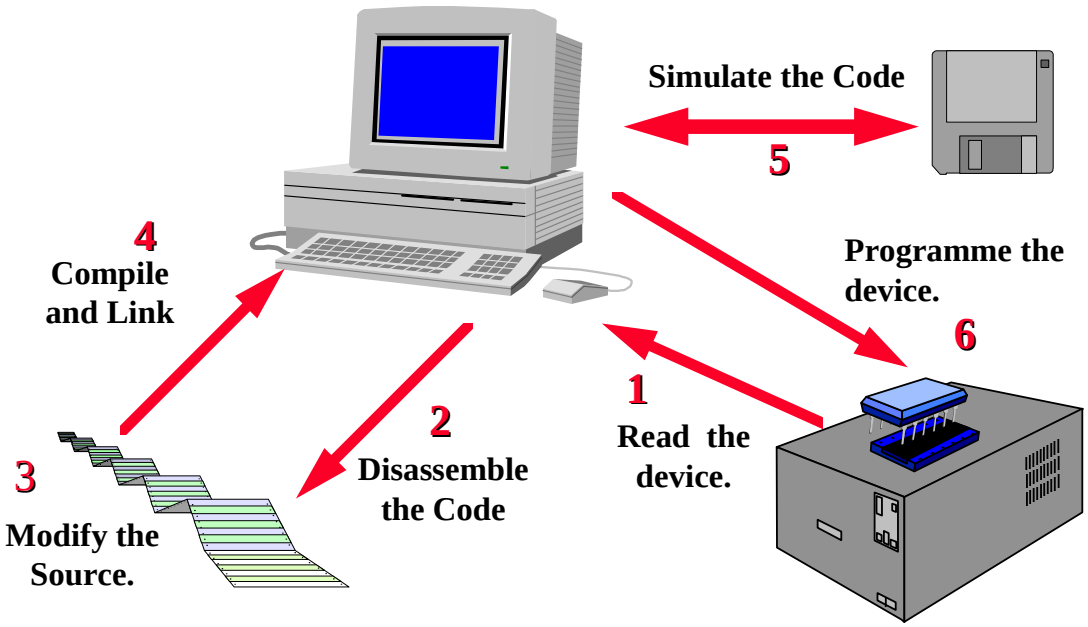
The Basic diagrammatic Summary of the solution view.



The integrated embedded microprocessor development system will address most of the concerns of developers embarking on using alternative products and currently consists of the following entities :-

1. An Integrated Development Environment controlling the following items :-
    - A Screen Text Editor.
    - A General purpose Microprocessor Machine code Assembler System.
    - A General purpose Text Macro Processor System.
    - A General purpose Object Code Linker System.
    - A General purpose Microprocessor Simulator System.
2. A Set of Machine code Dis-Assemblers.
3. A General purpose machine that will be able to programme EPROM, PROM, embedded processors and associated logic devices such as Programmed Array Logic (PAL), Field Programmable Gate Arrays (FPGA) and Generic Array Logic (GAL). The primary hardware configuration is specifically focused on being able to programme just one of the Microchip PIC Family processor chips.
4. A Full Product Text File Based Documentation Manual.

The Reverse Engineering Basic Solution Summary pictorial view.



2.0 System Design.

The development of this System had a number of distinct focal points :-.

- The first part of the system development and production was driven by the immediate need to produce secondary products.
- The remaining development was to enable total product integration with a clear future progression path. A primary goal has always been to leave sufficient hooks in software modules and hardware development in order to allow, where possible, total flexibility of the product. This flexibility covers such areas as :-
    - Introduction of more Processors and Devices being added to the current product range.
    - Introduction of new or extended feature Directives.
    - Systems using a wide instruction code construct width.
    - Features such as Formal or Informal Macro processing.
    - The ability to customise the system with the "INIT.EXE" programme.
- All aspects of the System had to be highly Cost effective.

2.1 Background.

2.11 The Mini Computer Development System.

Some years ago, there was a requirement raised to develop a display system which incorporated a touch sensitive keyboard overlay. The system was to be based upon a Z80 processor. It was considered that it should be possible using the DEC PDP/11 MACRO assembly language to develop a series of macros that could mimic the Z80 instruction set. The Macros that were developed could then translate the Z80 mnemonics into a byte data stream.
The development of the Z80 based Macros generated the templates for a number of other assemblers such as 1802, 6502, 6800, 6809, 8049, 8080/5, Z8 to name but a few. The Macros operated quite effectively, however, they did run rather slowly. In the same period, a number of Dis-Assemblers were developed to match the Macro Cross-Assemblers, one to match each selected processor. It was the development of the Dis-Assemblers that indicated there was a significant amount of commonality between most processors. Hence it was only the unique processor code sections of the Dis-Assembler design that ever needed to be reworked.

2.12 The PC Based Development.

It was realised that any future product development would need to have an effective text editor. Rather than having to rely on editors produced and Licensed by third parties, a new multi feature editor would be created. Hence the first of the development tools was the Screen Editor (SED).

SED was targeted to have the following hybrid functionality :-

- The basic edit command sequences were based totally on one of the more popular PC text screen editors.
- Extensions were added to give almost all of the functionality that the DEC EDT editor supported.

The Screen Editor (SED) was the first of the Executable Shareware products.

2.13 The Dis-Assemblers.

The next phase was to translate the Dis-Assemblers into a language that could be run on a PC. As the original source code was BASIC-11 , GWBASIC became a highly effective migration path with relatively minor amount of code change being needed. The modules were finally compiled using QBASIC for general distribution.

2.14 The Cross Assembler.

The next phase was Cross Assembler support. The original requirements analysis identified that the majority of functionality would be common to all processors. A number of specific entities were created, these being :-

- The Symbol Table Manager.
- An Expression Parser and Evaluator.
- An Object code Manager.
- A Source line Parameter Evaluator.
- A Common Directive Processor.
- A Instruction set Mnemonic Translator.

As the Instruction set Mnemonic translator would need to be unique for each processor, it was decided that this should be developed as an Interpreter controlled by some externally linked mechanism. To enable it to operate in the most efficient manner, the Interpreter was developed to trigger a nested Switch Case with pre-checked parameters targeted at pre-defined operation processes.

2.15 The Library Manager.

The externally linked mechanism is a fixed structure data file and was initially called a “Macro Library .MLB” because it was created from a series of pre-compiled macro type statements. It is now designated as an “Assembler Library .ALB” to avoid name conflicts with other libraries entities that are created by the (LMP) **L**ibrary **M**anager **P**rogramme and other applications.

The Library Manager Programme reads the “**Application Specific Language**” code and generates a library file, or it can merge sets of pre-prepared libraries into a single entity. Hence the Interpreter in the Assembler became a variant macro processor. The system remained static with virtually no development for a significant period.

2.2 Current and Recent Developments.

2.21 Limited Features Object Code Linker.

As a result of once again looking at Embedded systems, the facilities of the Assembler suite have come sharply into focus. The Assembler produced a general hex binary format object code, however, this is unsuitable for downloading to such items as the Flight Boards. To overcome this problem, a Limited features Linker was produced. Its structure was based on the General Purpose Assembler using many of its common functions. The Linker supports the two most popular download formats, these being INTEL HEX and MOTOROLA S FORMAT plus extended features .HEX format.

2.22 System Initialise and Integrated Development Environment.

The **INIT.EXE** programme is common to all programmes used in the development suite. This programme is used to set up most of the control parameters and stores them in a file called ASS.INI in the working directory. This programme has had a significant number of recent updates or major reworks, some of these being :-

- Introduction of a Multiple Configuration Registry.
- Enhanced to configure the Linker.
- Enhanced to configure the Macro Processor.
- Enhanced to configure the Simulator.
- Enhanced to be able to Control all the other Packages (IDE).

2.23 General Purpose Simulator.

This programme is a very recent addition to the overall system. The key requirements of this programme are to be able to :-

- Simulate a range of processors (Past, Present and Future).
- Simulate devices attached to the processor.
- Be configured with different hardware options.
- Record or log simulation progress.
- Stimulate the simulator with a control script.
- Monitor / Time or Interrupt progress as the user requires.
- Operate on a minimal Hardware configuration.

Again due to the nature of the requirements, this programme has many features in common with the General Purpose Assembler. To cope with the requirement of extendible processors or devices, a similar methodology has been used in the form of an interpreted "**Application Specific Language**". The Library Manager Programme (LMP) compiles the code into a dynamic Simulator Linking Library (a .SLB file).
Due to a wide possibility of requirements which may be needed to simulate different processors, the simulator had to be made as flexible as possible.
However, some restrictions have had to be placed on Simulator and these are :-

- Any unique address space is limited to a 32 bit address.
- The maximum width of a single instruction is limited to 128 bits.
- A single processor and its devices are limited to 16 unique address spaces.
- The maximum system map space is 2^31 locations.

2.24 General Purpose Macro Processor.

This programme is a very latest addition to the overall system. The basic Assembler is very successful at processing single line statements and providing the number of bytes created by a statement does not exceed sixty four, there is fundamentally no problem. However, there is a number of situations where data or programme structures logically span across a number of lines. A second scenario is where calculations need to be performed upon and within sequences of data. The final possibility is a structure development where both previous requirements occur.
One solution to the requirement is a source code embedded pre-process editor. However, for total flexibility, the ideal solution is a full features Macro Processor.

So what is a Macro Processor and what does it do ?

At the simplest level, a Macro Processor behaves like a simple text editor substituting one series of character for another sequence. However, a Macro Processor should contain sufficient control structures to allow a significant level of intelligence to be applied to the dynamic editing process.

The intelligence is usually implemented by using structures like :-

- {IF <Condition (True/False)> [Create Block of Text]}
- {REPEAT <Number of times> [Create Block of Text]}

The next important feature that gives the intelligence to the process is the Block of Text that accepts parameters which can be substituted into itself (i.e. A Macro).
A block of text is defined with embedded markers showing where the parameter substitutions should occur. Each time the Macro is exercised, the new parameters are substituted creating text block conforming to a template as defined by the Macro.

It is evident that the overall Macro expansion process can be quite complex and convoluted, especially when we consider that Macros can also call other Macros recursively. They can also contain at all levels both Conditional and Repeat text blocks. It should not be forgotten that Macros usually have the ability to develop dynamic parameters, further adding to the expansion complexity.

So why do we want Macros ?

Quite clearly from the previous description, the creation of Macros can be very complex and debugging them could be a problem. However, in general, there is a number of reasons for using Macros :-

1. Macros can simplify and significantly speed up programme development.
2. Macros can be reusable.
3. Macros can make code Platform Transportable.
4. Macros can make code much easier to read.
   e.g. PRINT “Have A Nice Day”
   It will not take much effort to guess what the above does.

2.3 Design Methodology.

There have been a significant number of Design Methods proposed for Software and Hardware design over the last few decades. UML , YORDON , SSADM , HIPO and Flowcharts to name just a few. However, many of these systems are simply variants on a theme with perhaps the major difference being the diagram symbols. For the purpose of this document, a further two variants are to be added to the list and have been used to develop the majority of this "Development System". These are :-

- The Box Concept Design method.
- The Application Specific Language.

2.31 The Box Concept Design method.

This Design method is a variant of the Block diagram. The System, Sub Systems or lower level entities are defined as a Box or series of boxes. Each Box or Block is then examined in more detail and is either :-

- Further decomposed in a set of lesser Boxes.
- Has a description attached to the block describing its function.
- Has a description attached to the block describing its contents.
- Has a description attached to the block describing its relationship to other entities.

The primary advantages of this system are that :-

- It is very easy to implement.
- The diagrams are fairly intuitive to read or describe.
- Arrows can be added to identify flows or relationships.
- No specialist diagram drawing tools are needed.
- The level of decent detail can easily be adjusted to the presentation or design requirements to give a clear focus point.

2.32 The Application Specific Language.

For some years, there has been a trend to develop computer languages that try to be universal (one language for all applications). Typically "C" , "C++" , "C#" and "JAVA" fall into this category. They have many programme constructs and data types with the opportunity to develop user defined structures. An alternative approach is the "Application Specific Language" which is directly focused at producing one type solution for a limited range of problems. The "Application Specific Language" will tend to have the following features :-

- The Language writing style is likely to be unique.
- Have few or NO definable Data types.
- The programme code can be directly generated from the problem domain.
- The programme code is also the problem solution documentation.

2.4 Future Development

2.41 Enhanced Linker Capability.

The Linker currently only supports absolute binary object images. The next major development phase will include :-

- Re-locatable binary object file support.
- Object Library support.
- Multi Region Image building.
- Checking of duplicate location usage.
- Additional Binary Image Formats.
- Enhanced more meaningful Map analysis.

2.42 Object Library Manager (LIB).

Even from the early days of computing, a need has been seen for the ability to maintain pre-compiled object modules. The Assembler already supports hooks for creating relocatable object modules. The Linker has a menu to support the inclusion of External Libraries into the current build. This will be the next development product.

2.43 General Purpose Dis-Assembler.

This programme will replace all the single processor Dis-Assemblers and the common programme DISEDIT, however, it will give an identical output to the programmes it replaces. It will have a similar front end to the General Purpose Assembler and will select a .DLB Dis-Assembler library to perform the Dis-Assembly process. The .DLB file will be created by (LMP) the Library Manager Programme. The development of this product will basically consist of a collation exercise of pre-existing items and a small amount of integration code.

2.44 Common or Universal Assembler Code.

It has been considered that as many processors have a similar capability instruction set, it may be possible to develop a Common Assembler Code that can be translated or converted to run any processor. This would then only require a few new libraries to be created to give the extra functionality.

2.45 High Level Language Support.

A further level of functionality that could be offered to the suite would be the ability to compile sources in say “C” , “PASCAL” or “BASIC”. The Compilers would then translate the original source codes into an intermediate format which could be converted into an appropriate object code by the Assembler and the selected processor library. The final stage would be to Link the Objects to give the runable binary image.

3.0 System Specification.

The System Specification is separated into the following areas :-
- User Requirements Specification.
- Software Development System Specification.
- Hardware Programmer Specification.

The main reason for part of the separation is that a fully functional and operational Software Development System was a prerequisite for the development of the Hardware Programmer. The Software Development System is required for producing the Drivers and Application Programme for the Hardware Programmer. It is also needed and used for all of the Test and Commissioning software.

3.1 User Requirements Specification.

This section will cover the following topics :-
- User Assumptions.
- Target Users and Requirements.
- Product and Quality.

3.11 User Assumptions..

The system is targeted at users who :-
- Are working to a limited budget but still require product flexibility.
- Have a working knowledge of the device/s they intend to use.
- Are likely to require to work on more than one processor family.
- Are willing to read the user manual rather than expect the system to run in automatic user teaching mode.

The system has a number of user interface modes such as :-
- The Integrated Development Environment (IDE). This mode is targeted for the new or inexperienced user who may be unfamiliar with driving a system from a command line prompt.
- Command line Mode which is for the experienced user.
- Stand Alone mode for applications that only use part of the system suite.

3.12 The Target Users and Requirements.

The system will have an appropriate interface to suit the ability of most users and is particularly focused towards those who are likely to have limited resources.
However, there are users whose requirements are so special that a bespoke solution would normally only be available to organisations with significant funding.

The typical target user groups are :-

- The Hobbyist.
- Education Establishments.
- Organisations with limited resources.
- Private System or Specialist Developers.

The user that requires the bespoke solutions would require two additional programmes currently NOT part of the shareware package, these being :-

- The Library Manager Programme (LMP). This programme allows the user to develop his own specific application solutions to his own requirements. This programme also allows pre-compiled libraries to be linked into a single entity and supports a Library Language Word usage Analyser.
- The Library Scan Programme (LSP). This programme allow patches and minor modifications to be made to libraries. It is also used as a Library debugging tool, and Library Dis-Assembler.

3.13 Quality and Fitness of Purpose.

The system quality checking and fitness of purpose is implemented in the following manner.

The system will support :-

- Test suites issued with the system so that the user can verify that its functionality conforms to specified requirements. The test suites are used for quality checking.
- Default typical system configurations that can be altered to the User specification.
- Options that will allow the user to be able to build the hardware with the specific functionality he requires.
- A System User interface style common across all packages.

3.2 Software Development System Specification.

This section will cover the following entities :-

- The General Purpose Cross Assembler.
- The Macro Processor.
- The General Purpose Linker.
- The General Purpose Simulator.
- The Integrated Development Environment.

3.21 The Cross Assembler Specification.

The Cross Assembler will have the following features :-

- The ability to be controlled by :-
    - Command line message.
    - Menu driven interface.
- The ability to Process a “Main Source” that contains “Include Files” directives.
- It can directly activate the Macro Processor with a pre loaded source image.
- It can be configured to support a range of Processor mnemonics.
- It can be customised for :-
    - Optimum Workspace usage.
    - Pre-selected Default Settings.
- It can support :-
    - Relocatable binary images.
    - Both Numeric and String Symbols.
    - 16 or 32 bit calculation mode.
    - 16 bit High/Low byte reversal display option.
    - Defined symbol name widths of 6 to 24 characters.
    - Positive fraction constants.
    - ASCII character value conversion constants.
    - 8,16 and 32 bit data storage directives.
    - Radix base 2,8,10 and 16 defined values.
    - Local and Global Symbols constructs.

3.22 The Macro Processor Specification.

The Macro Processor will have the following features :-

- The ability to be controlled by :-
    - Command line message.
    - Menu driven interface.
- The ability to Process a “Main Source” that contains “Include Files” directives.
- It can be customised for :-
    - Optimum Workspace usage.
    - Pre-selected Default Settings.
- It can be operated
    - In Stand Alone mode.
    - As a slave process of the Cross Assembler.
- It can transfer Symbol definitions to the Cross Assembler
- It can support :-
    - Macros with or without Parameters.
    - Macros with (dynamic and default parameter) substitution.
    - Conditional assembly “IF” and nested “IF” blocks.
    - REPEAT and Nested REPEAT Code blocks.
    - Global (dynamic and default parameter) substitution.
    - Formal and Informal Macro Expansion mode.

3.23 The General Purpose Linker Specification.

This product will be developed in a number of phases :-

- Phase One The Limited features Linker.
- Phase Two The Full features Linker.

3.231 The Limited features Linker.

This entity has very limited processing capability. It will only process Binary Object files that have all its symbol references resolved.

This Linker will have the following features :-

- The ability to be controlled by :-
    - Command line message.
    - Menu driven interface.
- The ability to process a :-
    - Single Programme Object Source.
    - Definitions file containing a list Programme Object Sources.
- It can be customised for :-
    - Optimum Workspace usage.
    - Pre-selected Default Settings.
- It can produce as an Output file:-
    - INTEL HEX Binary Images.
    - MOTOROLA S Binary Images.
    - Simulator Extended HEX Binary Images.

3.232 The Full features Linker.

This Linker will have the following features :-

- All the features of the Limited variant specification.
- The ability to :-
    - Process relocatable objects files.
    - Support Link Library files.
    - Support multi region images.
- It will identify and error report binary image data clash conflicts.
- It will produce a detailed Linker analysis map output.

3.24 The General Purpose Simulator Specification.

The General Purpose Simulator will have the following features :-

- It will be controlled by Menu Driven Interface.
- It will be able to be customised for :-
    - Optimum Workspace usage.
    - Pre-selected Default Settings.
- It will be able to be configured :-
    - To support a range of Processors and Devices.
    - With Memory Regions set to user requirements.
        - Memory will be able to be :-
            - Mirrored.
            - Mapped to Alternative Regions.
            - Have READ Only access.
            - Have WRITE Only access.
            - Have READ and WRITE access.
            - Have Independent READ and WRITE access at a common data address.
- It will support loading of one or more Simulator Extended HEX Binary images.
- It will have Control Mode support for :-
    - Single Processor Simulation operation mode.
    - Single Processor and Device Simulation interaction environment operation.
    - Master Processor and Slave Processor Simulation interaction operation.
- The Operation Command modes are :-
    - Instruction Single Step
    - Continuous Run
    - Run for a defined number of Instructions
    - Run ignoring any Simulator System Data Access Errors
- It will be able to generate a run time report log to:-
    - A Log Display screen.
    - A Data file.
- It will be able to :-
    - Run timed command script files.
    - Produce a run time code Dis-Assembly.
    - Modify and examine data images under user control.
    - Display a Pseudo Device Animation Screen.
- It will be able to simulate :-
    - DMA transfers.
    - Interrupt operations.
    - I/O interface device interaction.
    - Instruction sets widths up to 128 bits.
    - A maximum of 16 memory space regions.
- All Simulator Regions will have a 32 bit address space limit.

3.25 The Integrated Development Environment Specification.

The Integrated Development Environment (IDE) is a sub entity of the system initialise or configuration programme (INIT.EXE). The INIT.EXE programme can be controlled either by command line prompts or a menu driven interface. The method of operation is purely a matter of choice. see file USER.MAN for details of the various implementation activation options.

The Integrated Development Environment will have the following features :-

- It will be controlled by a Menu Driven Interface.
- It will be able to initiate :-
    - The workspace usage of other programmes in the suite.
    - The customised Default Settings for other programmes in the suite.
- It will be able to :-
    - Verify that processing programmes are available for use and error report if the programmes are unavailable.
    - Select a default preference Editor programme.
    - Display contents of User manual file.
    - Create a basic Assembly Programme Source Template.
    - Activate or Start the Assembler and Linker process.
    - Display the Current Programme Listing output.
    - Support Multi Project working environment by :-.
        - Changing the State of the Current Environment.
        - Saving then Current Environment in a named Registry area.
        - Restoring an Environment from a named Registry area.
    - Enable the Simulator Configuration file to be edited.
    - Display the Current Simulation Library documentation file.
    - Activate or Start the Simulator.

3.3 Hardware Programmer Specification.

3.31 Introduction.

There are hundreds of different field programmable devices types. The specialist equipment needed to support these devices can be very expensive. This is in part primarily due to the fact that many technologies are used to develop the devices and hence the potential for each device gives a limited user base. There is little in the way of a universal hardware programming standard. This is mainly due to the unique nature and construction technologies of the devices. The number of devices available in the market place which can be currently programmed is in the order of hundreds of thousands with more new devices becoming available every day.
The General Purpose Device Programmer was conceived to enable those on a limited budget to enter the programmable device market. These new users will now be able to take advantage of these devices. Furthermore, it will give these innovators the same advantages of electronic circuit designs and products as those who are working in larger and better funded organisations.

The remainder of this section will be subdivided into the following subsections :-
1. The Generic requirements.

2. The User requirements

3. The System requirements.

    a) The detailed device hardware interfacing.

    b) The detailed processor hardware interfacing.

    c) The detailed software overview specification.

3.32 The Generic requirements.

Typical requirements of a General Purpose Programmer Device are :-

1. It needs to have a fast, easy and effective method to "connect to" and "remove from" the Device and the Programmer.
2. It may need to be able to supply power to the device that is to be programmed
3. It may need to be able to supply logic levels to device pins.
4. It may need to be able to supply programming voltages to one or more specific device pin/s. (The programming voltage in this context means a voltage which is different from the devices normal operating range).
5. It may need to be able to read status and verification information from the device being programmed.
6. It may need to supply phased clock signal to specific device pin/s.
7. It will need to ensure that the voltages it supplies to a device remain within the specification of that device being processed in order to maintain quality production.

As there is no particular standard pin layout, all the desired features for programming a device will need to be available on every device pin. However, if a user wishes to limit himself to a particular range of devices, this may allow the overall system functionality to be downgraded to a more appropriate level.

3.33 The User Requirements.

3.331 User Interface.

The GPDP operations will be controlled either via a menu driven interface to a host machine or in a pure "turn key" mode for production work.

The menu driven interface will have two distinct user operating modes :-.

- New user Mode will lead the user step by step through all the appropriate operation stages to get a successful outcome.
- Advanced user Mode will allow appropriate shortcuts to give the desired outcome.

The "turn key" operation mode would typically be :-

1. The System will indicate it is ready for "Device load" or "Programming complete".
2. The user inserts a device in programming socket and then presses programme start button.
3. The System indicates that the Programmer is "busy".
4. On completion, the programme indicates the operation was either "Success" or Failure".
5. The user removes the device and records programming status. (Back to Start)

3.332 Data Creation and conversion.

The Integrated Development Environment (IDE) will initially only be suitable for developing binary images for microprocessors. The IDE is already capable of generating binary images suitable for loading into other systems. It is not unreasonable to expect that the user may wish to continue using tools he is already familiar with which generate binary images. If the user has already been supplied with binary images developed by other tools, he may now need to programme chips from the supplied images.
Therefore, the GPDP will not only accept binary formats from the Integrated Development Environment (IDE) but will also accept most of the industry standards i.e. JEDEC etc. However, if too many formats are presented, then a format conversion module may need to be embedded in the IDE to resolve any overload conflicts.

3.333 Hardware Construction and maintenance.

The General Purpose Device Programmer (GPDP) will be designed using a fully modular approach. This will enable :-

1. The GPDP user to start with a limited features machine and expand it as their budget permits or programming needs and requirements change.
2. Low cost specialist device GPDP could be created for the production environment.
3. Sub modules can be built, tested independently or even sold as a kit of parts.
4. In the event of a sub module failure, maintenance or replacement will be an easy process.
5. Franchise, production, maintenance or development could be offered to appropriate organisations.

3.34 The System requirements.

3.341 The detailed device hardware interfacing.

The design of the General Purpose Device Programmer programming module will :-

1. be able to be controlled from a series of parallel TTL Level Interface signals.
2. be expandable in steps of 8 Pins to a maximum of 64 Pins.
3. attach all device pins via a ZIF (Zero Insertion Force) Socket (maximum size 64 Pins).
4. support a fixed “default +5Volts source” to any pin.
5. support changing the “default +5Volt source” with a supply from a digitally controlled analogue voltage generator with a range of 0 to +25Volt in 64 or more incremental voltage steps.
6. support device pins being used in a high speed clock mode.

3.342 The detailed processor hardware interfacing.

The Hardware design requirements of the General Purpose Device Programmer Processor will include :-

1. A common interface interconnection to the General Purpose Device Programmer module.
2. A master control RS232 serial interface.
3. An optional USB master control interface to be able to replace the RS232 serial interface.
4. An optional Flash Memory Disk Interface.
5. A battery backup real time clock to enable automatic date and time stamping .
6. An onboard configuration memory store.
7. A optional direct connection programme configuration and data entry interface for stand alone operations.

3.343 The detailed software overview specification.

The Software design of the Overall General Purpose Device Programmer will eventually incorporate such features as the ability to :-

1. automatically detect its own hardware configuration.
2. regularly check its own basic calibration with a log of its status.
3. keep an upload quality assurance log of its programming activities with the outcomes.
4. accept downloads of programming methods.
5. hold a range of direct select preferred programming methods.
6. hold a range of direct select preferred programme images.
7. verify that its own hardware configuration is compatible with the device being requested to be programmed.
8. Verify that data and protocol downloads have not been corrupted.
9. Verify that downloaded protocols have not date expired. (Author comment: “It has been noticed that as manufacturers develop or enhance their products, the recommended algorithms necessary to programme their devices may alter. This is a method that could catch obsolete algorithms. However, support of past algorithms may be necessary if batches of the older device still exist in the market place”.)

4.0 The Project Plan.

4.1 Project Activities.

The initial development stages viewed the whole project and decomposed it into a number of main identifiable activities. These were :-

- Prepare the Development System.
    - This would consist of the Assembler , Simulator and other Programmes.
- Develop System Hardware.
    - This is specifically the Programmer developed in a modular manner.
    - It was conceived as consisting of :-
        - A Processor Board/Section.
        - A Main Interface Board/Section.
        - A number of sub function control Board/Sections.
- Develop Main Processor.
    - This processor would perform the following activities
        - Act as interface between the programmer and the Host System.
        - Control the functionality of the programmer.
        - Would perform application specific activities.
- Develop System Software.
    - This is the Software of Main Processor.
- Develop Mother Board.
    - This would be the Main Programmer Control and Interface Board.
- Develop Daughter Boards.
    - These would add the overall functionality to the Mother Board.
- Feasibility Study.

4.2 Programme Evaluation and Review Technique (PERT).

This section will cover :-

- Detailed Work Breakdown.
- Detailed Activity List Breakdown.
- Estimation of work per activity.
- Identification of the basis of assumptions and estimations.
- Resource Profile and impact upon estimated times.

The following items are covered in Appendix PERTCALC.doc

- Layered PERT Diagram.
- Calculations of estimated times and variances.
- Estimated time for completion.
- Calculations of probabilities of project overrun.

4.21 Detailed Work Breakdown.

The production of a detailed work-break-down chart was not possible as this feature has been removed from the processing package. However, the detailed PERT diagram gives the same information and has been structured to show breakdown and linkages of the following top level development work package tasks :-

- Prepare the Development System.
- Develop System Hardware.
- Develop Main Processor.
- Develop System Software.
- Develop Mother Board.
- Develop Daughter Boards.
- Feasibility Study.

4.22 Detailed Activity List Breakdown.

The detailed activity lists are shown in :-

Appendix Filename PERT_EIE.XLS.
Appendix GANTT Chart.
Appendix PERT Diagram.

and is segmented as follows :-

- Prepare the Development System.
  - Entries 2 to 9.
  - Entries 11 to 18 resulted from need to rebuild a previously designed processor monitor programme.
- Develop System Hardware.
  - Entries 21 to 22.
- Develop Main Processor.
  - Entries 23 to 27.
- Develop System Software.
  - Entries 29 to 34.
- Develop Mother Board.
  - Entries 36 to 38.
- Develop Daughter Boards.
  - Entries 40 to 49.
- Feasibility Study.
  - Entries 51 to 52.

4.23 Estimation of work per activity.

The document Appendix Filename PERT_EIE.XLS shows the estimates in man-days for the most likely, optimistic and pessimistic times for each activity.

4.231 Identification of the basis of assumptions and estimations.

Many of the assumptions that have been made are based on past experience of similar work, however, there is always an element of educated guess. This becomes a particular problem when working in unknown areas. In these cases, there is a tendency to err on the side of caution especially when dealing with entities that are known to be potentially complex. It is the complex sub sections' duration which is the hardest to predict due to the potential internal and external interactions. This is particularly so with the layout of printed circuit boards. When a small number of components is required, this presents only a minimal problem i.e. a few hours of layout at most. However, once the number of through board holes approaches 200, significant difficulties are usually experienced with the layout. The auto routing and auto placement options may help, however, they usually route the power lines first, resulting in an excess of vias between layers of the signal lines. This creates its own unique problems when it comes to board production i.e. the accuracy required to place the vias and the time involved in soldering them in place. When laying out the PCB, the manual approach would be to keep as many tracks as possible on the copper side of the board and to keep the via count to a minimum. These techniques do require a certain level of intuition as ideal component placement is paramount. If these constraints are not enough, the designer will also need to take into account component direction placement for ease of construction and track widths for power distribution.

4.232 Resource Profile and impact upon estimated times.

The decomposition of the project has identified that there is a number of parallel work paths and normally, different sections of the project would be allocated to different individuals or groups of individuals (Human Resources). The resources could consist of both internals staff and persons contracted specifically for certain activities. A second group of resources that needs to be considered is equipment, workspace and services that will need to be available for completion of the task. The non human resources may also raise allocation and requirement conflicts. The staff using these shared resources therefore need to compromise with other staff when making use of limited facilities. It may be necessary to arrange specific scheduling of some resources to overcome the allocation overloads. The result of the rescheduled work is usually to extend the estimated time of the planned requirement for a specific task. This becomes all the more important if these specific tasks are on the critical path.
The resource profile for this specific project causes one of the major problems. There is only one resource to complete all activities and hence by definition all activities must be on the critical path no matter which parallel route is being examined. Normally, a project would have more than one resource and critical path becomes more meaningful. The critical path is therefore by definition in this specific case the sum of all the working time for all projects.

The resources available to the project are rather limited and hence effective use of available resources is paramount. For the majority of time, there is only one individual to complete all the activities. However, there will be a significant number of activities that could sensibly run in parallel. There will be a few activities that will require a small amount of third party assistance or safety monitoring.
These activities specifically are :-

- Printing of photo masks for PCB manufacture.
- Supply and selection of PCB board material.
- Arranging appropriate access times for the PCB manufacture.

4.3 Risk Analysis

This section will cover :-

- Risk categories and consequences.
- Type of Risk.
- Causes of risk.
- Constraints and their impact on risk.
- Strategy for risk management.

4.31 Risk categories and consequences.

Risks basically fall into three categories:

Type 1 Normal or typical day to day problems.
These are the common occurrences and will be typically covered in the allocation of the pessimistic prediction of task duration.

Type 2 Rare events.
These are derived from investigating assumptions made about the project. They may need some level of contingency planning and perhaps insurance to hedge against potential loss or incurred penalty expenses.

Type 3 Project fails, Never finished or Rejected by sponsor (Act of God).
This is the case when the project delivers nothing so all investment is lost. The level of failure may also incur penalties due to lack of delivery. So not only is the investment lost but the penalties may force the provider into receivership.

This project will only consider the implications of Type 1 risks as Type 2 & 3 are more appropriate to a commercial organisation. In the event of a Type 2 or 3 occurrence, then it is back to square one. Start again with a new project or salvage redefined elements of current project.

4.32 Types of Risk.

The types of risks for this project can be sub divided into distinct areas and will require different strategies to achieve a successful outcome specifically :-

- The risks associated with producing a final output from the project.
- The risks associated with the product being fit for purpose.

The Risk management strategies used to produce a final output from the project will be resolved by :-

- Making effective use of all available resources.
- Have an alternative approach option for all high risk activities.
- Keeping Backups archives of documentation and planning activities.
- Monitoring Progress, Finance, Resource allocation and ensuring all Milestones achieved on Time and within the Budget allowed.

The Risk management strategies used to ensure the product is fit for purpose will be resolved by :-

- By using incremental development and ensuring that all prototype interfaces meet the Customer requirements.
- The System interfaces are easy to use and are supported by clearly defined procedures and User documentation.
- The System is incrementally validated against the specification ensuring that each subordinate entity is fully functional before submission to full system integration.
- The system will be developed ensuring that all appropriate current Safety features and Radiation shielding requirements are included within the product.

4.33 Causes of risk.

The most significant risk and problem areas of this project will be :-

1. Getting the programmer to be recognised by chip manufacturers.
2. Finding a highly cost effective plug and socket system for sub module docking.
3. Safety issues of using a programmer without a case. The board is a low voltage device, however, desk contamination could damage the programmer rather than being a particular threat to the user.
4. System reliability due to high component count.
5. On the development front the main risk areas will be :-
    a) Large PCB Layouts (Processor Board and Mother Board).
    b) Complex Application Software programme implementation.
    c) Development of an Integrated Macro Processor programme.
    d) Component sourcing as many standard items are NOW only available in surface mount format.

4.34 Constraints and their impact on risk.

The major constraining factor of this project is COST and has the most significant effect of possible development routes. Due to lack of funding, the surface mount production option is effectively shut off (cost of purchase of equipment). Long term, this will have a significant effect mainly due to the fact that fewer and fewer components are available for through board mounting. A secondary factor is that the project is envisaged to be a long running activity and as such, any components used need to be available long term. Usually, the most effective method of ensuring continuity of supply is to select generic products that have been in the market place for many years. This unfortunately means that the design will potentially require more components than a customised design has and also this impinges on the PCB design layout.

Other significant risk and problem areas of this project will be :-

- Getting the programmer to be recognised by chip manufacturers.
    - (Solution) Initially only devices that have published programming algorithms will be able to be programmed. This still represents a significant number of devices.
- Finding a highly cost effective plug and socket system for sub module docking.
    - (Solution) Initially sub module boards may be hard wired.
- Safety issues of using a programmer without a case. The board is a low voltage device, however, desk contamination could damage the programmer rather than being a particular threat to the user.
    - (Solution) allow legs and protection sheet to be fitted to PCB to give required isolation.
- System reliability due to high component count.
    - (A future solution) route would be to develop an ASIC (Application Specific Integrated Circuit) that could replace the contents of the whole daughter board. However, for this project, the development costs alone of an ASIC, rule it out as a possible solution option.

- On the development front, the main risk areas will be :-
    - Large PCB Layouts (Processor Board and Mother Board).
        - (Solution) Use Auto route and vias (or spend lots of time resolving).
    - Complex Application Software programme implementation.
        - (Solution) Decompose use effective analysis techniques.
    - Development of an Integrated Macro Processor programme.
        - (Solution) Decompose use effective analysis techniques
    - Component sourcing due to standard item NOW only available in surface mount format.
        - (Solution) Migrate to surface mount route when funding available.

4.35 Strategy for risk management.

One of the simplest methods to identify risk areas is to allocate a risk factor quotient to each activity in the project. For purpose of ease, the following method has been used :-
Each task is allocated a potential risk factor from 1 to 5 where 1 is considered to be a low risk task , whereas 5 is considered to be cause for concern.
The following table categorises the guidelines used.

Risk Level	Category of Task Risk Level Allocation.	Difficulty
1	Task expected not to overrun by more than 25%	Easy
2	Task expected not to overrun by more than 50%	Medium
3	Task expected not to overrun by more than 100%	
4	Task expected not to overrun by more than 200%	
5	Task expected to be a potential major problem	Concerned
+1	Task expected to have increased likelihood of difficulty.	
+2	Task expected to have significant likelihood of difficulty	

Note
Any task that exceeds level 5 needs to be re-evaluated as its outcome may seriously effect the projects viability.

The document Appendix Filename PERT_EIE.XLS shows the Risk Allocation levels and summation averaged calculation. As the average value of all risks is below 3 then it could be expected that the project should succeed within the project constraints.

4.4 Decision Analysis

This section will cover :-
Improving decision making process.
Decision making and its impact on risk.
Decision tree.
Forecasting.

4.41 Improving decision making process.

To make effective decisions, it is necessary to collate ideas , information and resource availability from as many sources as possible. No ideas should be rejected without prior consideration, however, a time limit should be pre-set on the collection procedure. The next phase is to identify where the possible risks associated with the project are and their implications. Having identified the risks and grouped the ideas, it is now important to identify the activities, setting realistic milestones and to outline resource requirements.
Having got a basic plan, the first review will check feasibility and viability of continuing. At this stage, very little costs should have been incurred and abandoning now will normally cause minimal disruption.
If the review indicates that the project is viable (i.e. potentially profitable), then a second review is implemented to clearly identify the following items :-

- The project specification.
- The project activity plan .
- The project funding.
- The availability of the required resources.
- The milestones and individuals responsible for them.
- The contingency procedures and insurance.

At this stage, providing the project plan is followed and milestones met, then the project should have a successful outcome.

4.411 Decision making and its impact on risk.

All decisions involve some level of risk. By developing a diagram or model like the decision tree, then the project manager should be able to quickly see the implications of each decision and its potential outcome. Obviously, as the decisions become more complex then more sophisticated decision making tools may be needed. These tools will assist with the decision making and decomposing of the decision options to more manageable levels. As the levels are decomposed, the associated level of risk should also be more easily definable and quantifiable. Once the project manager has been able to quantify the risks associated with any decision, a decision table should easily be constructed. Even relatively arbitrary decision about perceived risk magnitude of a particular activity can cumulatively identify an unacceptable route when the project is viewed as whole.

4.42 Decision tree.

The document Appendix Filename DTREE.DOC shows an example of the decision tree that has effected the direction of the project. It can be seen that as cost is a major constraint, most alternative options have been ruled out as a possible route.

4.421 Forecasting.

Prediction of the future is always a difficult activity as we are dealing with the unknown. However, we can use past performance as a guide to how things may develop provided all things remain the same as in the past. If we are dealing with the unpredictable i.e. like the stock exchange, then we have little to base our decisions on other than intuition and insider information. However, with a project, hopefully we have some experience or can call upon experience to base our decisions on. With past experience and some of the modelling software tools like UML , MS PROJECT and YOURDON, we should be able to make moderately accurate predictions of expected work loads and resource requirements. The GANTT chart document Appendix Project GANTT Chart shows the effect of the single resource being multi-tasked. The time scales have been expanded into September, however, this does show the implications of over allocation of that single resource. It is fortunate that a number of duration estimated have so far been generous or there would be potentially a need to recruit extra resources to complete the project by the deadline.

5.0 Hardware Design.

5.1 Design Plans.

The initial project product design phases focused on the ability to programme a limited range of devices from the PIC processor family.
The design was left sufficiently open so that it could easily be upgraded to extend from the initial design processor range to other processor families and logic devices. Typically, these upgrades needed to include features such as :-

- more active pin connections.
- all pins supporting full range programming voltage capability.
- all pins configurable as Power or Ground connections.
- all pins being dynamically configurable as Inputs or Outputs.
- every pin capable of behaving as a master device clock.

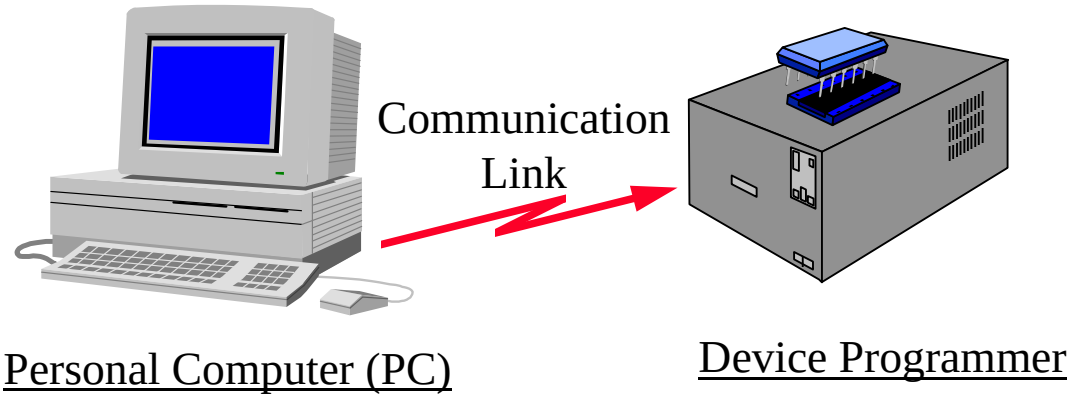
The machine's configuration and programming methods should be implemented through a dynamic structure which would be downloaded from a central store. The central store typically would be an Open Access Public Domain Internet site. The central store of dynamic structures could easily be upgraded as "change request" requirements dictate.
The whole system will be controlled and run from a standard DOS or Windows based Personal Computer. Future consideration would be given to the possibility of running the applications from other platforms i.e. LINUS , Apple Mac. However, the amount of resources that these options would involve put it beyond the scope of this project.
Due to a limited project budget, the programmer would need to be built in various "flavours" i.e. The professional version would have the smarter casing, additional interface options (Serial / Parallel Port plus USB), a stand alone control mode and multiple programme device sockets. The hobbyist (or custom build) version would support the same functionality, however, with none of the frills.

5.2 General Requirements.

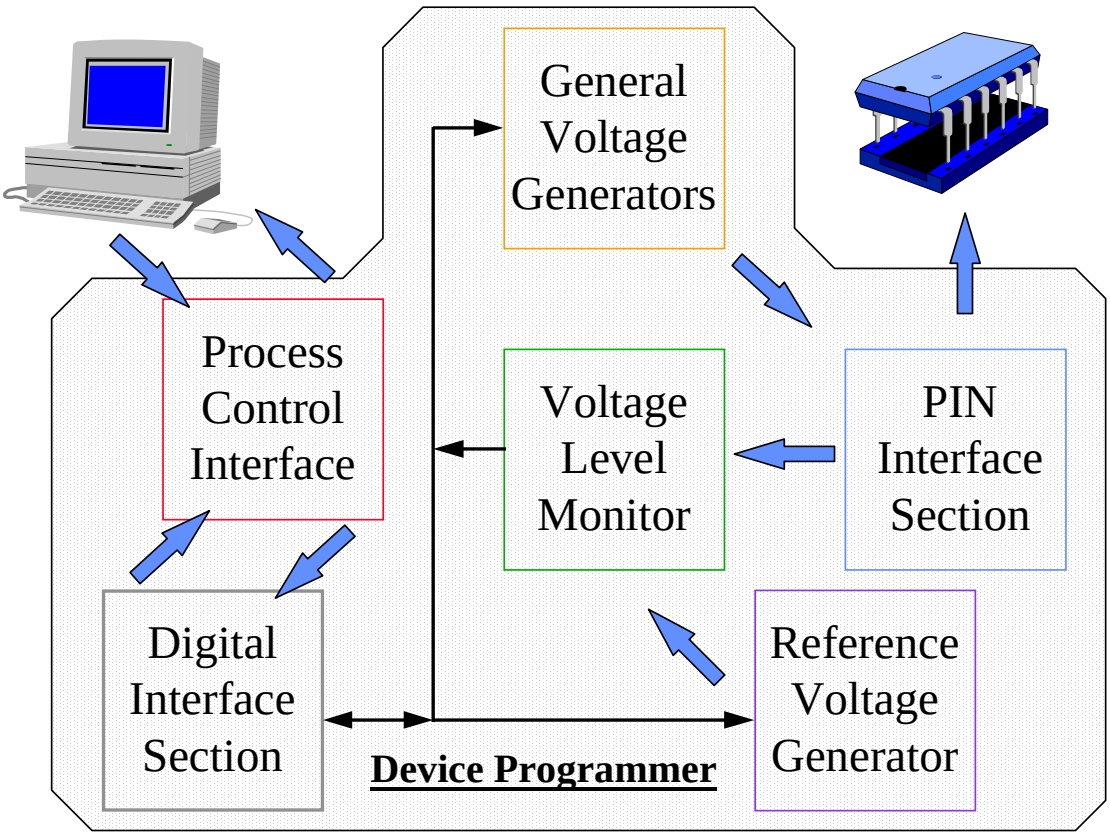
The Device programmer will be controlled by a number of constraints. These are :-

- The System has to be low cost , price competitive and market place viable.
- The System has to be highly flexible to cope with past, present and future devices.
- The System life time expectancy of the product is anticipated to be many years.

5.3 Initial Block Diagram.



5.4 Detail Block Diagram.



5.5 Detailed Requirements and Implementation.

This section covers the following areas :-

- The ZIF (Zero Insertion Force) Pin Interface.
- The Voltage generators.
- The Reference Voltage Generator.
- The Voltage Monitors.
- The Digital Interface Section.
- The Process Control Interface.

5.51 The ZIF Pin Interface.

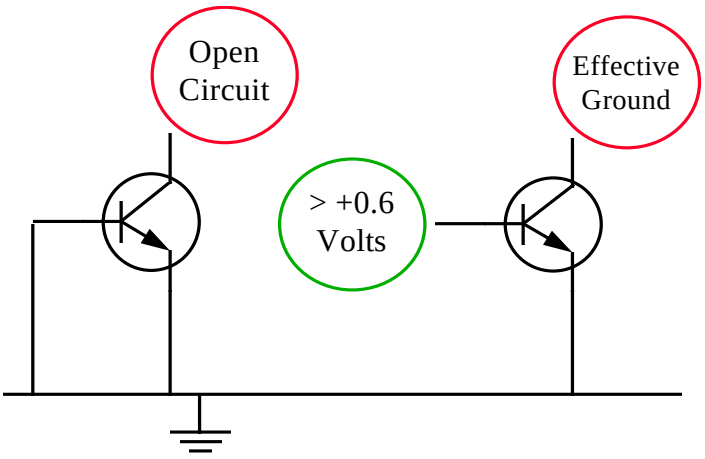
The ZIF Zero Insertion Force) Pin interface has the following requirements :-

- Each Pin must be able to tie a pin to ground.
- Each pin must be able to supply a Voltage.
- Each pin must be able to be an Input.

To enable these features, the following discrete transistors switching circuits were implemented.

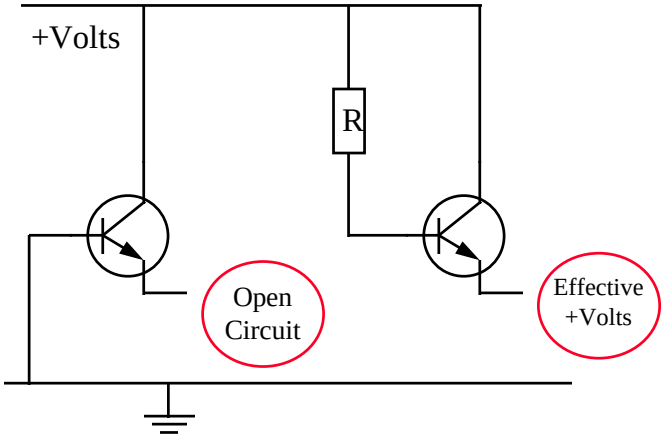
The Ground Pin Tie configuration.

Whilst the Base of the transistor is tied to ground, the collector is effectively an open circuit with the transistor shut off. However, when the Base is driven with a voltage above +0.6 Volts, then the transistor can start to conduct. When sufficient current is driving the base, this effectively ties the collector to ground.



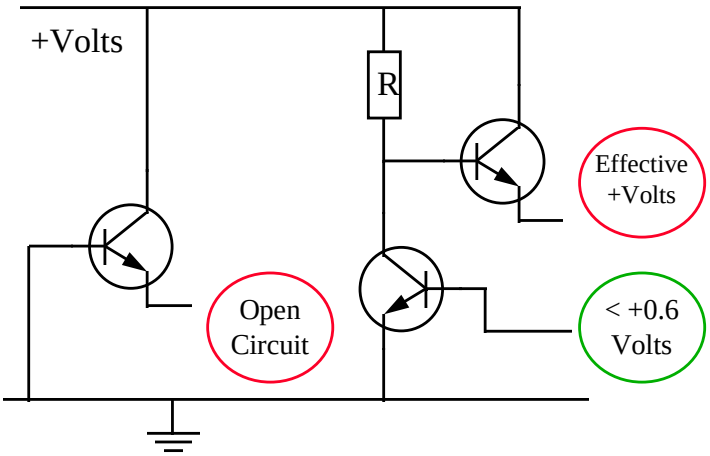
The Voltage supply Pin.

In this configuration, when the Base of the Transistor is tied to ground, then the Emitter is effectively an open circuit with the transistor shut off. However, when the Base is driven with a voltage above +0.6 Volts, then the transistor can start to conduct. With sufficient current driving the transistor, it will effectively appear as a short circuit hence switching a supply voltage. As with the previous circuit, there will be a small voltage difference due to the saturation voltage of the transistors. However, for all practical purposes, it can be ignored as it will typically be only circa 0.2 Volts.

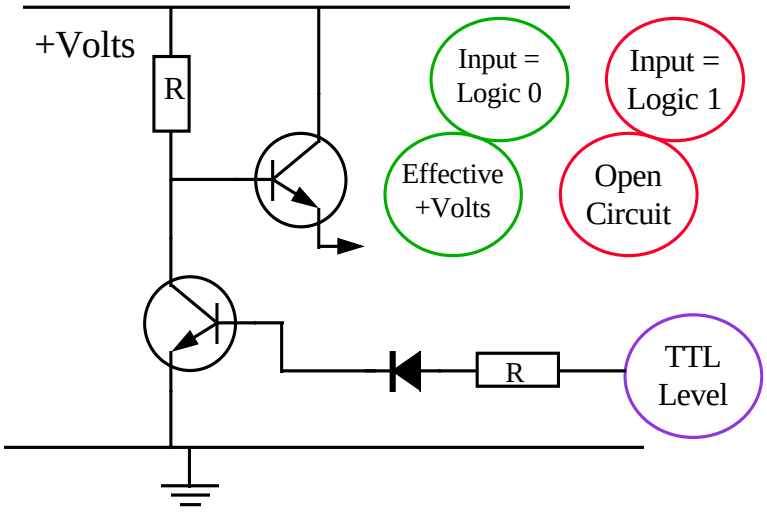


The Basic Control Circuit.

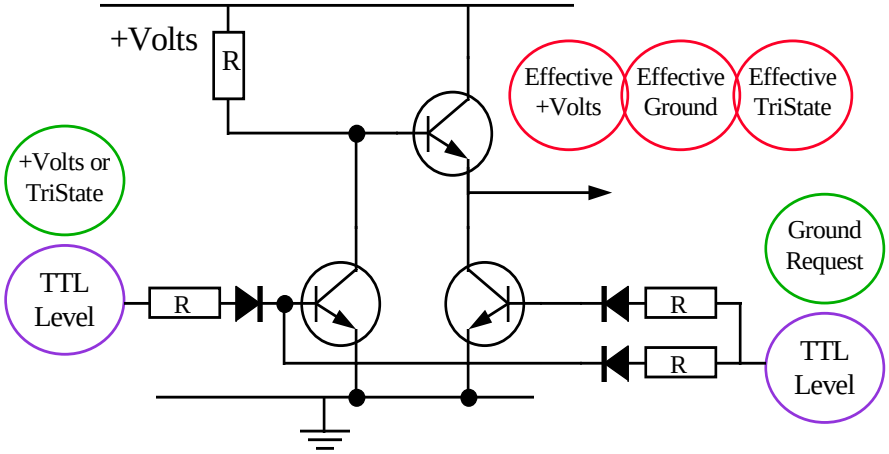
To avoid the necessity of requiring large voltage swings to control the system, the following circuit was developed to enable a simple option to be implemented.
A second transistor was added to enable the base of the control transistor to be either tied to Ground or allowed to be pulled to +VOLTS through the Resistor R. While the control voltage is less the +0.6 Volts, the lower transistor will be effectively shut “off” and the control transistor will be driven into saturation. When the control voltage switches the lower transistor fully “on”, then the control transistors base will be held effectively at Ground and the Emitter will appear as a high impedance.



The circuit described so far covers all the basic requirements of the Pin interface requirements, however, refinements were still required. The Diode and Resistor chain were added for the following reasons :-
The Resistor limits the drive current and loading of the Transistor on the external control circuits.
The Diode is used to isolate the Transistor circuit from the Control circuit.



To give all the required features of the pin interface, the following circuit is built from a combination of the previous circuits. The circuit has the two digital control signals that in combination can present the three distinct required output states.

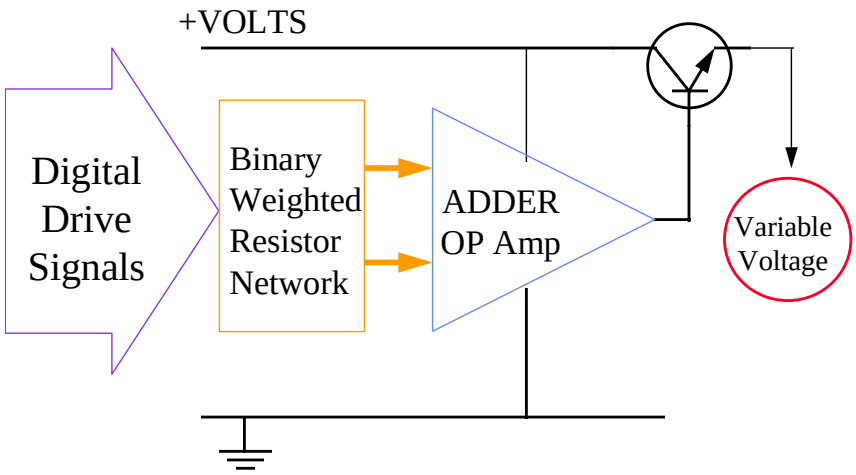


Safety Aspects.

As a safety feature when “Ground Request” is asserted, it also asserts an overriding forced “TriState” request via the additional Resistor and Diode chain. Therefore, the destructive Ground and Voltage option can never occur even if the request is selected or driven by the programmer.

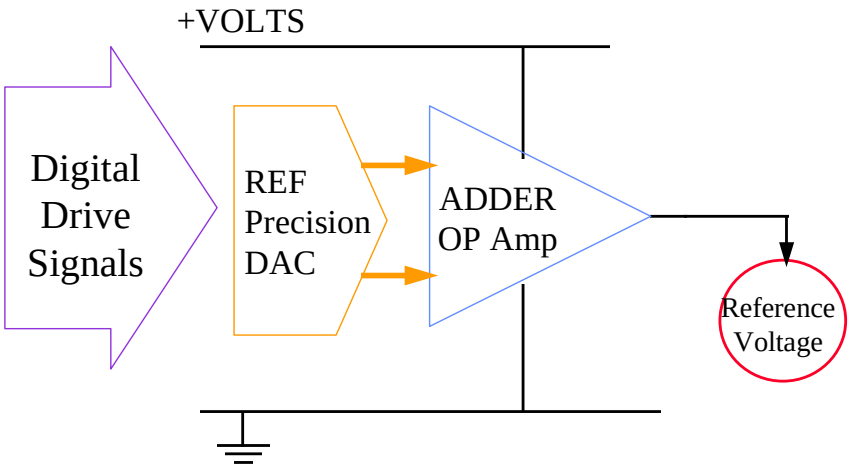
5.52 The Voltage Generators.

The Voltage generators are of a standard basic design in that the operational amplifiers are configured as a simple multiplying adder circuit. The value input resistors are adjusted in conjunction with the feedback resistor so that each input has a power of two gain higher than its predecessor. The additional Transistor is used to increase the power output capability of the operational Amplifier. To get the required voltage range i.e. Zero to Thirty volts, the operational amplifiers are configured in the single supply mode with the negative rail voltage being used as the common earth line. The resistors were selected to give as close as possible a monotonic output. Only seven of the Digital drive signals are used so as to keep the resistor range with reasonable and usable bounds.

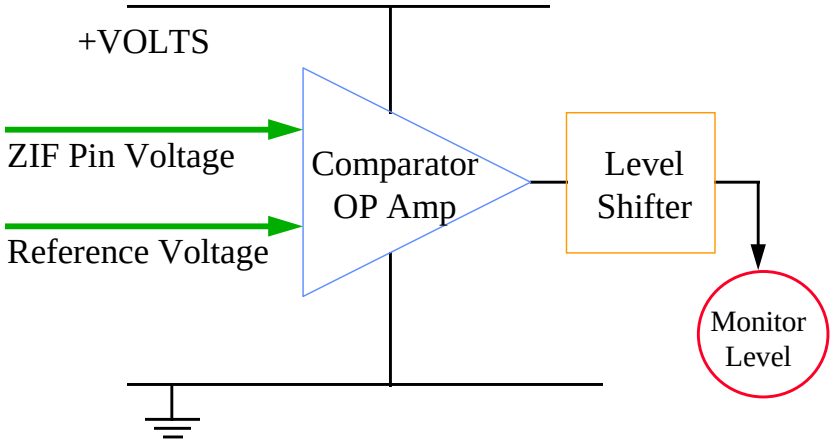


5.53 The Reference Voltage Generator.

The Reference Voltage generator was constructed in a similar manner, however, due to the limited supply current requirements, there was no need for the additional transistor for increase power capability. An alternative method was also implemented using commercial 8 bit DAC to drive the amplifier. A precision 5.6Volt zenner diode is used as a reference for the DAC device. The calibration procedure offers the opportunity to adjust the amplifier gain for correcting any minor voltage errors.



5.54 The Voltage Monitors.



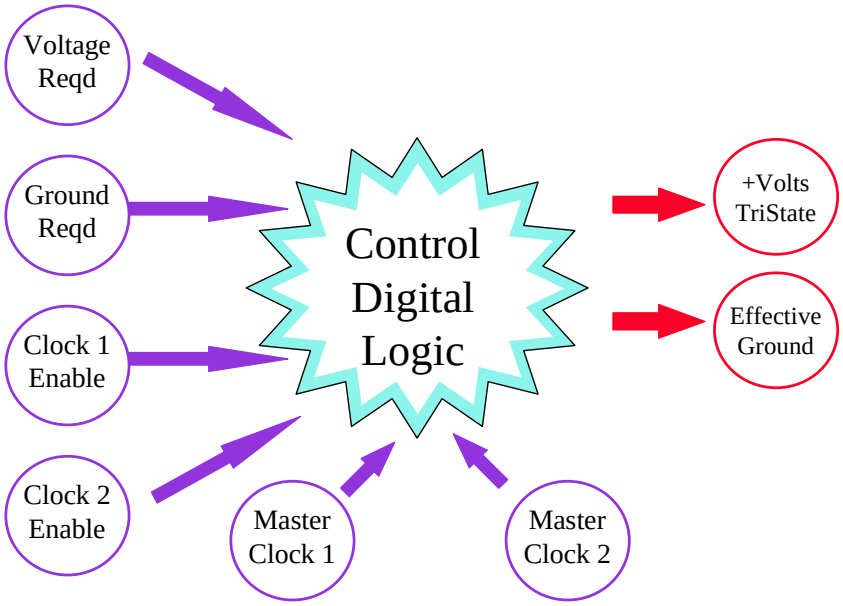
A Voltage Monitor compares the actual ZIF pin voltage with the Master reference Voltage. The level shifter network configures the output to a valid TTL Monitor level. The Monitor Level is used for calibration procedures. Any instability of the Monitor Level output is removed by latching the Reference voltage up or down an increment.

5.55 The Digital Interface.

This interface implements the majority of the programming capability. Each ZIF (Zero Insertion Force) Pin has the following control inputs :-

- A Signal that requires a voltage to present on a pin.
- A Signal that ties a pin to ground.
- A Signal that enables the Master Clock 1 to control a voltage present on a pin.
- A Signal that enables the Master Clock 2 to control a voltage present on a pin.

The Master Clock signals 1 and 2 are both TTL levels and are either generated by the Master Clock module or supplied from an external source. The main reason that there are two clock signals is that some devices need to be stimulated with clock signals in anti-phase. There is also a possibility that some processors may need more than one control clocking process to be effectively programmed.



The Digital logic implement all the required functionality with the two output signals which are :-

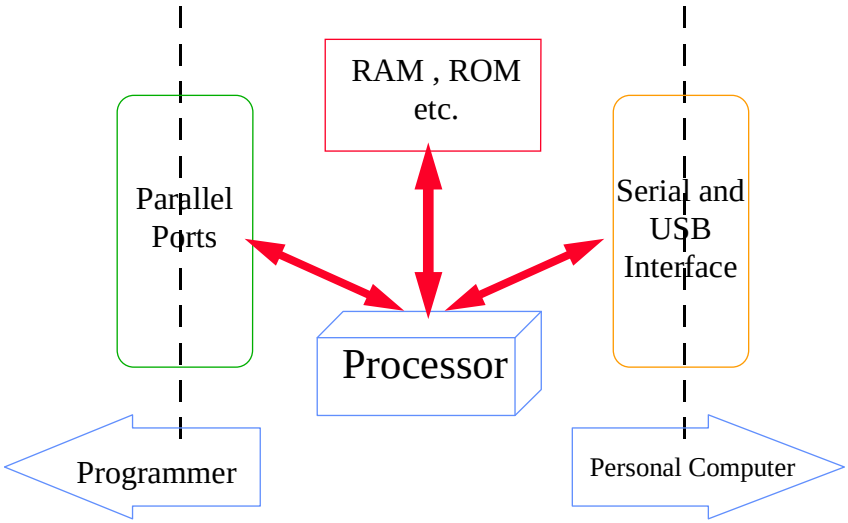
- Set pin to Voltage Ground.
- Voltage select or pin is in TriState mode (input mode).

5.56 The Process Control Interface.

The process control interface is where the majority of the system intelligence resides. The processor will control the following functions :-

- Data transfer to and from the Personal Computer (PC).
- All the timing functions of the Programme Process logic via the Parallel Port lines.
- The Implementation of the programming algorithms.
- The Implementation of Device read algorithms.
- The System Diagnostics and Calibration Procedures.
- Manage Flash disk Memory interface.
- Verification of Algorithm data expiry

The Process Control Interface (Basic Version).



A more detailed Block diagram of the Process Control Interface can be located in the Appendix. The present Process Control Interface is based around a Hitachi 64180 processor, however, the basic design of the programming elements ensures that they are fully compatible with any processor that can support a TTL interface.

6.0 Software Design.

The Software design falls into two distinct areas :-

- Software for the Development System.
- Software for the Device Programmer.

6.1 Software for the Development System.

The Software for the Development system consists of the following programmes :-
The General Purpose Cross Assembler.
The General Purpose Macro Processor.
The General Purpose Linker.
The General Purpose Simulator.

6.11 The General Purpose Cross Assembler.

The basis of this system is that one tool set should be suitable to enable the development of any embedded style processor. When viewing Assemblers in particular, there is a number of characteristics which are noticeably common between all systems. When looking at a typical Assembler programme source, we can identify a number of distinct areas of commonality.

```
        .TITLE Example Programme
        .ORG Here+0CH          ;Programme Start Address
X1:     LOAD (J1),Y2           ;Store Data
        SUB Y2,#1.             ;Change Data
        INC J1                 ;Change Data Pointer
        IF_ZERO Y2  X1         ;Loop
        .END                   ;End of Programme
```

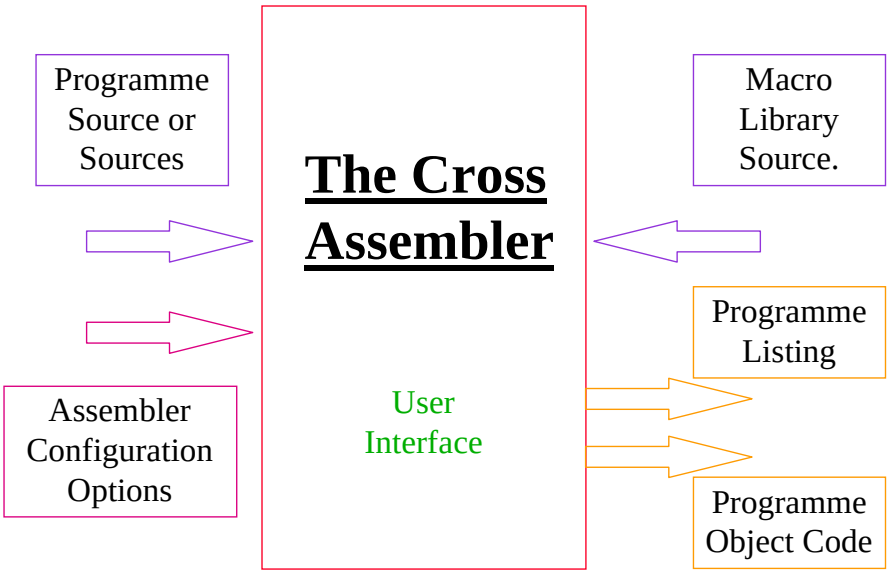
The source above can be broken down into the following distinct areas :-

- The “Blue” Text identifies Directives.
- The “Green” Text identifies Comments.
- The “Orange” Text is a Label (A Special Symbol).
- The “Violet” Text is an Expression.
- The “Grey” Text is a Symbol or a Data Store.

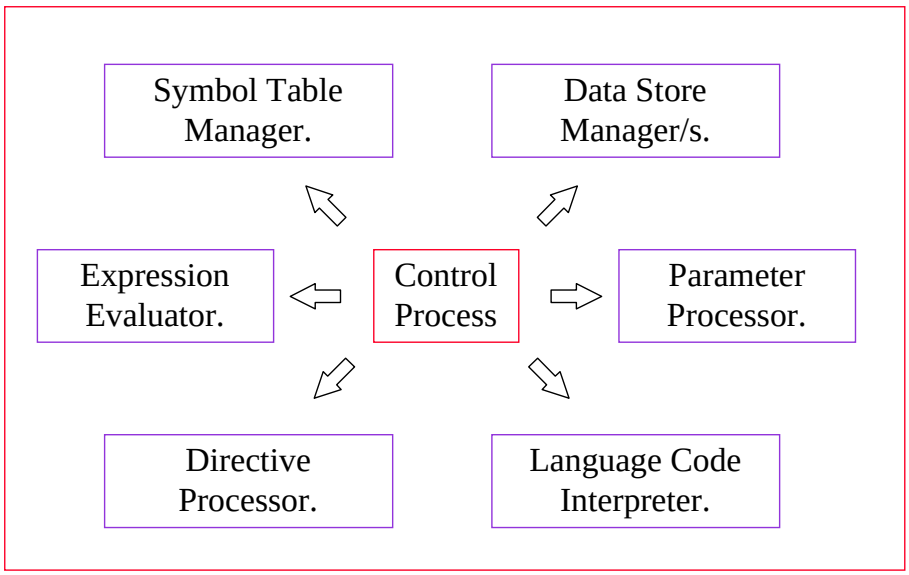
Most systems therefore need the following facilities :-

- The ability to process “Directives”.
- Support for “Labels” and “Symbols”.
- An “Expression Evaluator”.
- The ability to specify “Data Stores”.
- An Instruction Set Code Translator.

Basic Data Flow Diagram of the Cross Assembler.



The Basic control Mechanism of the Cross Assembler.



Main operations of the basic features.

The **Control Process** is Broken down into the following passes :-
1. Process Source Directives , Basic Translation of Mnemonics.
2. Resolve Label addresses and Symbols, Substitute Symbol values into Object code.
3. Dump Object or Report errors.

The **Symbol Table Manager** needs to be able to :-
- Initialise the Symbol Table structure.
- Store or Add contents a Symbol.
- Locate or Get contents a Symbol.
- Update a Symbol store contents.
- Evaluate or resolve Symbol contents.
- Display or Dump Symbol table contents

The **Data Store Manager** needs to be able to :-
- Locate appropriate Data Store access regions.
- Write data to its Data store location.
- Read data from its Data store location.
- Error report bad Data store access.

The **Expression Evaluator** needs to be able to :-
- Translate values using differing radix or formats.
- Perform arithmetic and logic operations on values.
- Access contents symbol in Symbol table.

The **Parameter Processor** needs to be able to :-
- Divide a text line into individual entities using known terminators.
- Remove white space and redundant controls.

The **Directive Processor** needs to be able to :-
- Purge comments and redundant information from a text string.
- Recognise specific text strings.
- Extract any associated parameters with recognised string.
- Perform the appropriate action of detected directive.

The **Language Code Interpreter** needs to be able to :-
- Initialise “Application Specific Language” library pointers.
- Repeat activities till EXIT Encountered.
    - Access next statement.
    - Access command features of “Application Specific Language” switch table.
    - Process selected “Application Specific Language” command.
- End Repeat.
- Exit Language Code Interpreter.

The only item that is specifically unique in all Assemblers is the Instruction Set Code Translator. With this concept in mind, if the translation of the Instruction Set can be processed via some externally linked entity, then all the remaining items could be processed by a single package. The methodology used to resolve this problem was to develop an interpreter that could perform the translation process. The translation interpreter process was developed around the concept of using an “Application Specific Language” for both the “Process” and “The Documentation”.

The General Purpose Cross Assembler Shell process separates the functionality of the system in the following manner :-

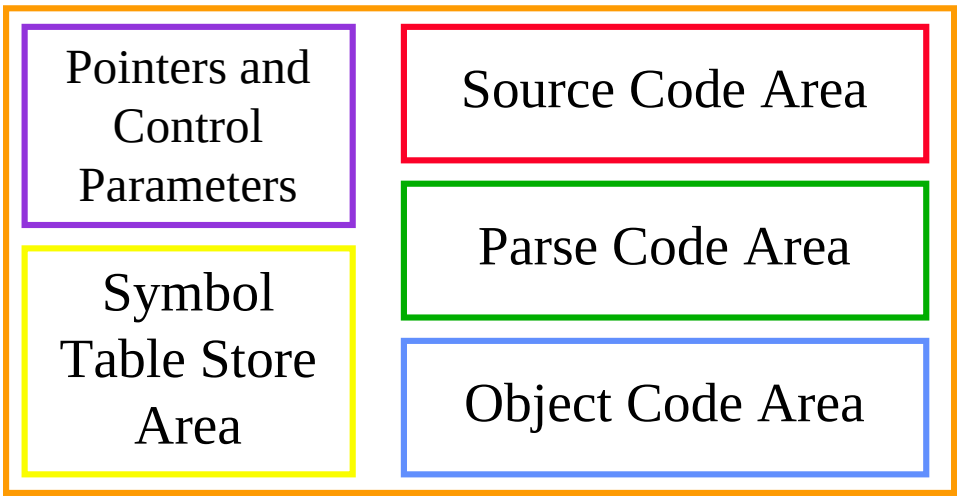
The General aspects of the Cross Assembler :-

- Processes all the Directives.
- Evaluates the Expressions.
- Maintains Symbols in Symbol Table.
- Prepares the Data Stores.
- Creates Parameters from any remaining programme Statements.
- Runs the “Application Specific Language” Interpreter with supplied Parameters.
- Displays and Outputs Processed Results.

The “Application Specific Language”.

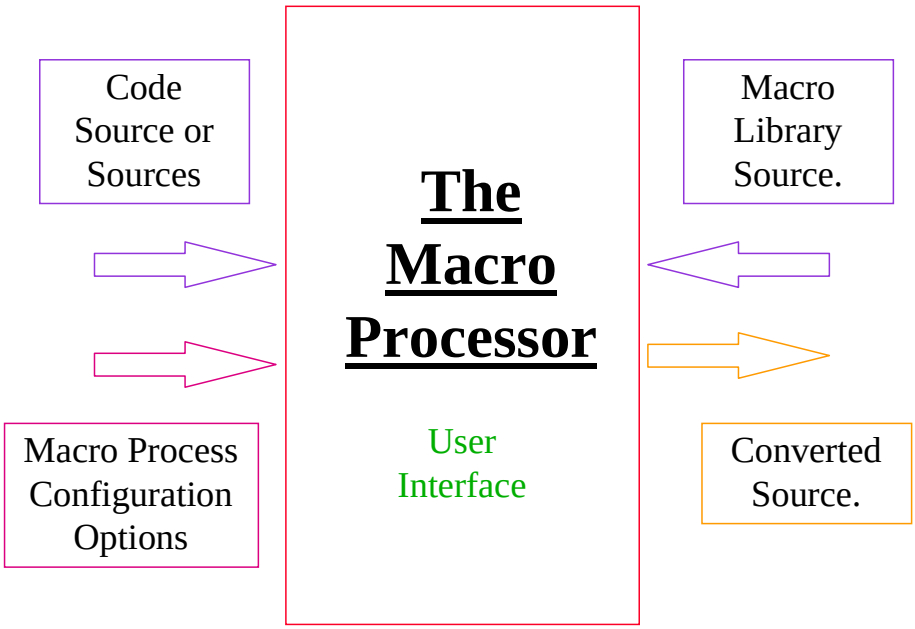
- Initialises the Assembler default settings.
- Translates Parameters into a valid Data Store Format. (No Checking needed).
- Adds Appropriate Range checking parameters and data size specifications to Data Store Statements.
- Creates Error messages as appropriate.

Basic Data structure sections of the Cross Assembler.

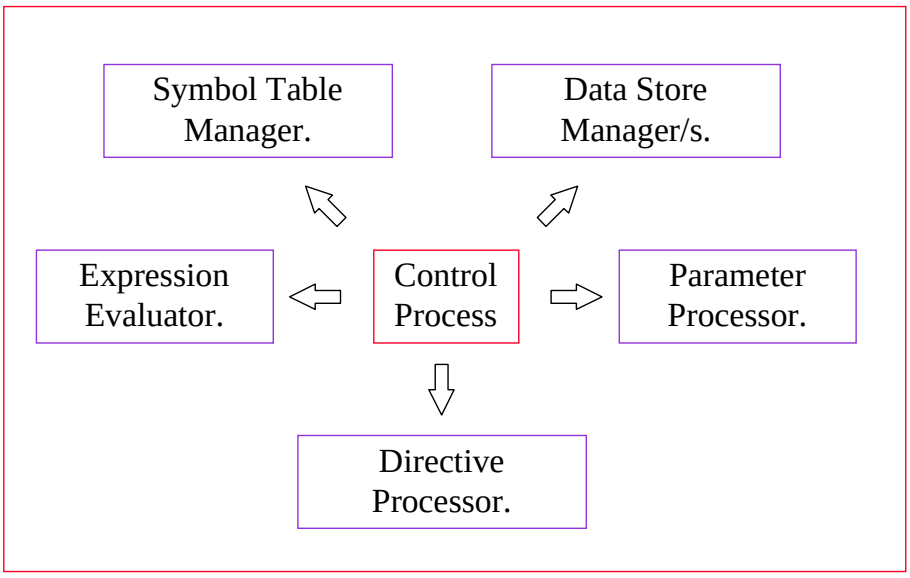


6.12 The General Purpose Macro Processor.

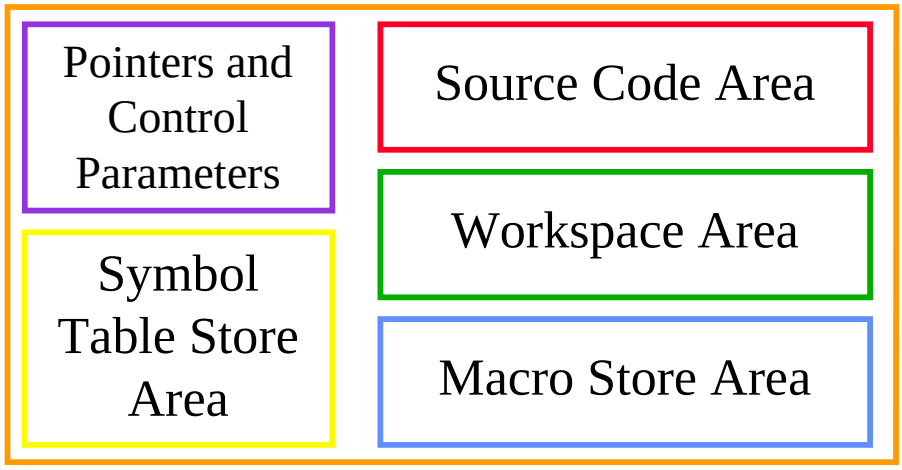
Basic Data Flow Diagram of the Macro Processor.



The Basic control Mechanism of the Macro Processor.



Basic Data structure sections of the Macro Processor.



The Macro Processor has an identical data structure to the Cross Assembler, however, some areas are used for slightly different purposes. The majority of the Macro Processor common named entities operate identically as for those in the Cross Assembler, the only exception being the **Control Process**.

The **Control Process** is Broken down into the following passes :-

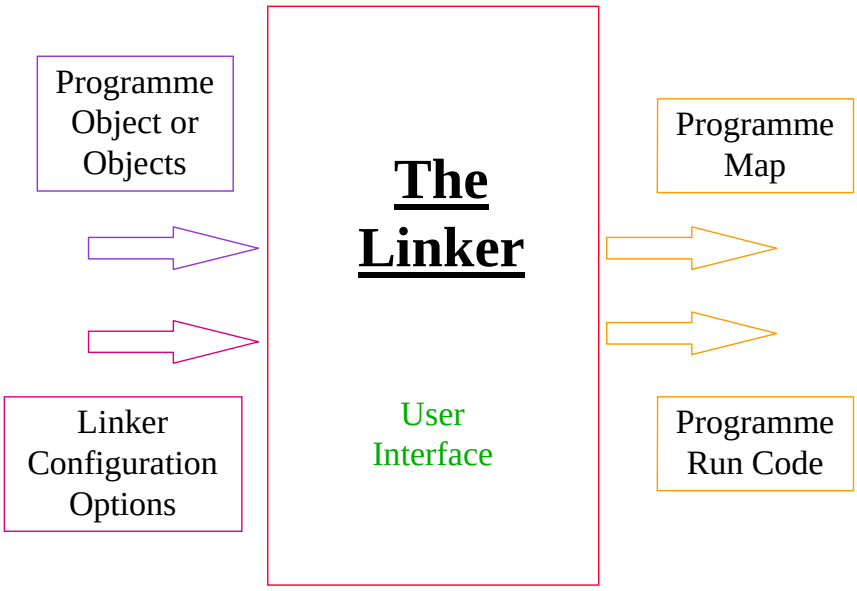
Pass 1.
Processes the unique operation Directives.
Extracts any Embedded Macros and Stores them in the Macro Area.

Pass 2. The Recursive section.
Implement Repeats , Expands Macros,
Substitutes Parameters, Processes “IF” blocks
Implemented by :-
Read data from Source area.
The Processed or modified data is loaded into Workspace area.
A tidy process copies the updated data back to Source area.
If Recursive count NOT expired and WORK still to do
Continues till all processes complete.

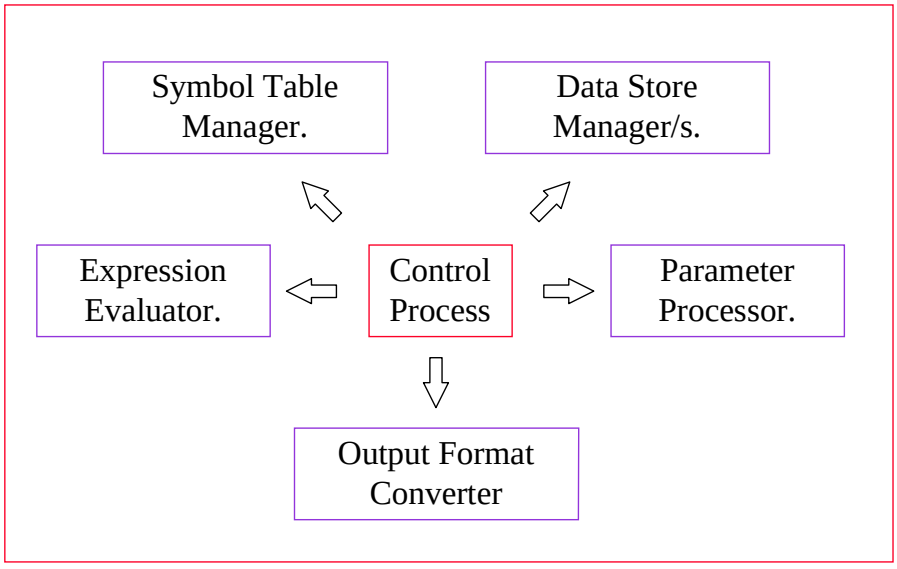
Pass 3. The Termination Options :-
Option 1. Output processed Source.
Option 2. Return control to Cross Assembler.
Option 3. Quit the Macro Processor.

6.13 The General Purpose Linker.

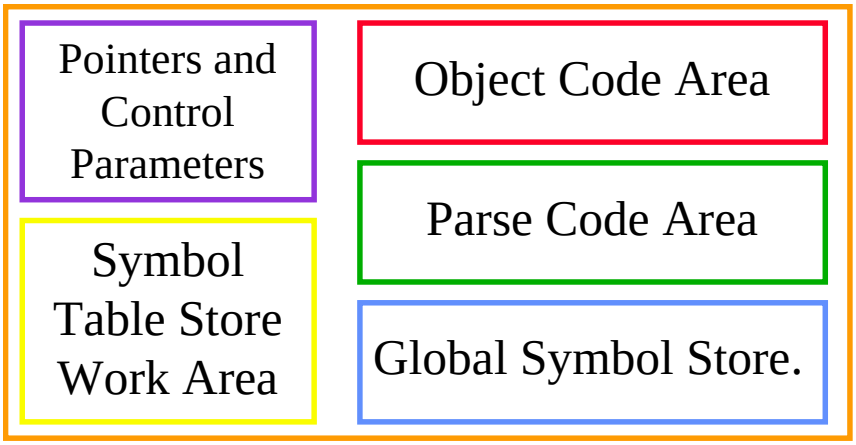
Basic Data Flow Diagram of the Linker.



The Basic control Mechanism of the Linker.



Basic Data structure sections of the Linker.



The Linker has a similar data structure to the Cross Assembler, however, some areas are used for slightly different purposes. The majority of the Linker common named entities operate identically as for those in the Cross Assembler, the main exception being the **Control Process**.

The **Control Process** is Broken down into the following passes :-

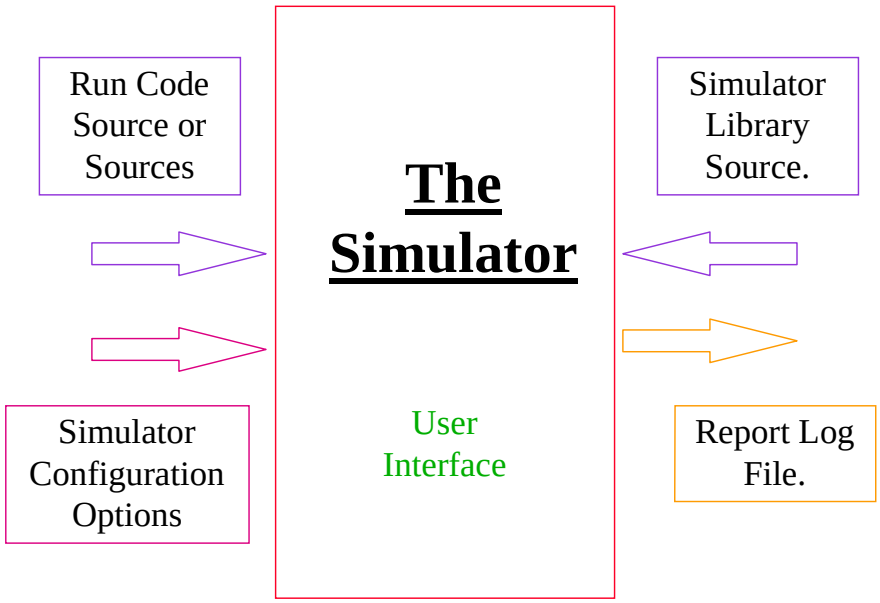
Pass 1.
Remove all redundant information.
Extracts Object Binary and Convert Address Directives.

Pass 2.
Tidy Symbol Table.

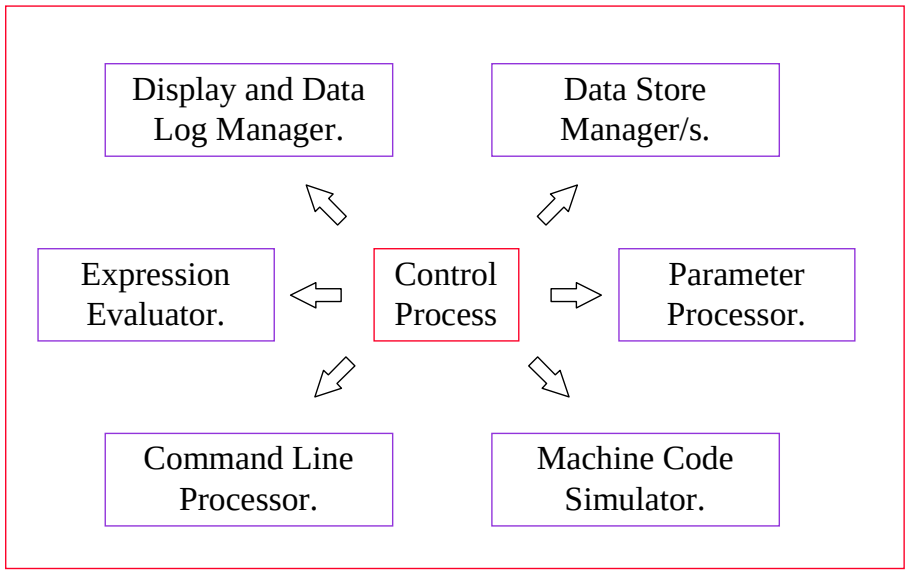
Pass 3.
Convert Binary to required Output format.

6.14 The General Purpose Simulator.

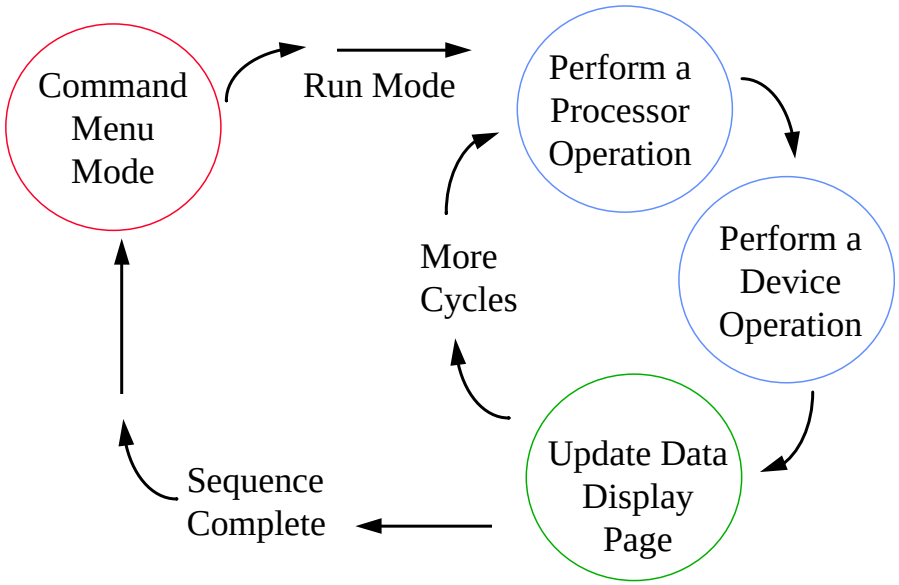
Basic Data Flow Diagram of the Simulator.



The Basic control Mechanism of the Simulator.



The Control Process of the Simulator.



The Simulator uses a number of common entities with other packages in the system, however, there is a major processing concept differences with the **Control Process** :-

- The "**Command Menu Mode**" identifies how many Cycles are to be processed based on the User response. It is only a QUIT response in this mode that will terminate any further the Simulator activities. This section also controls the ability to include executable programme code and to change the status of system data store contents.
- "**Perform a Processor operation**" activates the Simulator Interpreter to process a Processor Machine code operation.
- "**Perform a Device operation**" activates the Simulator Interpreter to process a Device activity.
- "**Update Data Display Page**" redraws the currently selected information screen.

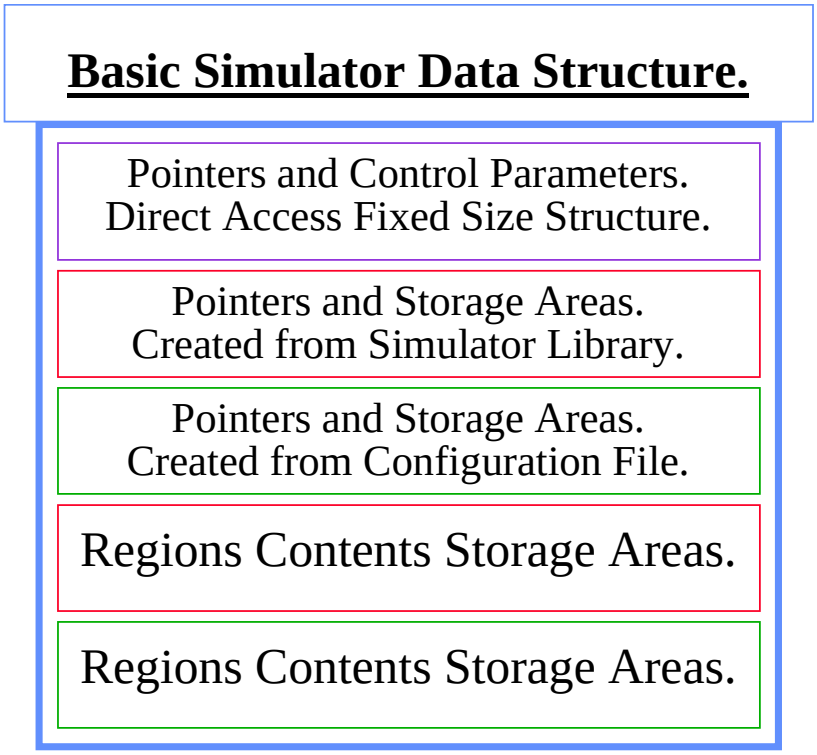
The **Data Store Manager/s** needs to be able to :-

- Create the appropriate Dynamic Data structure and Mapping references.
- Read or Write to an individual Region elements.
- Error report if Region access location invalid.

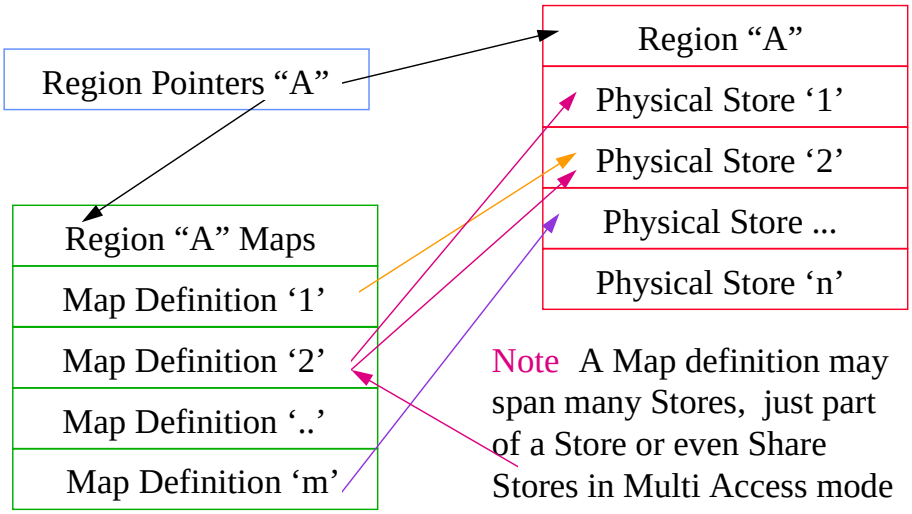
The **Machine Code Simulator Interpreter** needs to be able to :-

- Initialise "Application Specific Language" library pointers.
- Repeat activities till EXIT Encountered.
    - Access next statement.
    - Access command features of "Application Specific Language" switch table.
    - Process selected "Application Specific Language" command.
- End Repeat.
- Exit Machine Code Simulator Interpreter.

Basic Data structure sections of the Simulator.



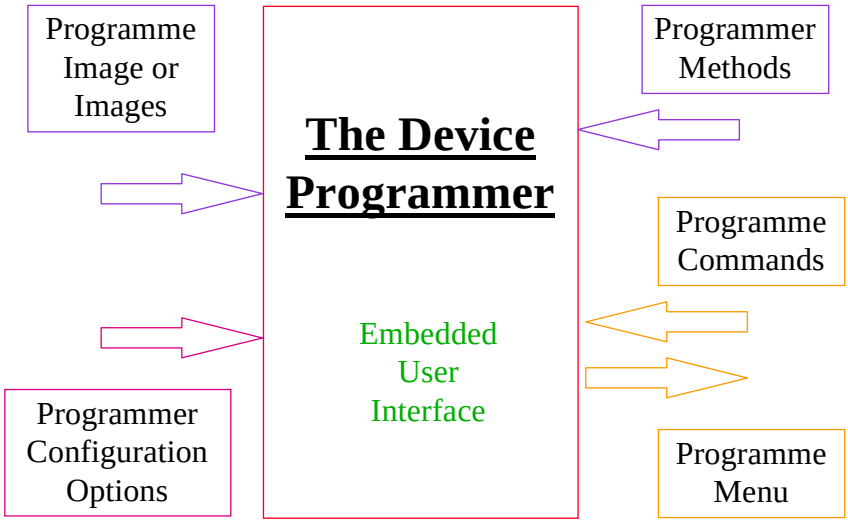
Basic Region Mapping Method of the Simulator.



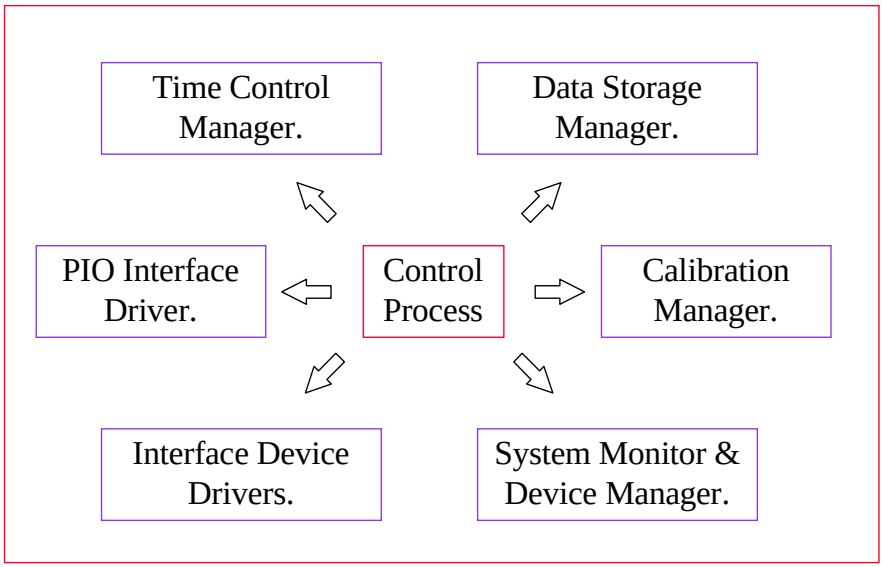
6.2 Software for the Device Programmer.

This Software entity is expected to be an ongoing project for a number of years. It is likely to experience a significant number of development extensions before it can be considered to be a stable product.

Basic Data Flow Diagram of the Device Programmer.



The Basic control Mechanism of the Device Programmer.



7.0 User Interface.

There are two distinct interfaces to the overall system. These are :-
- The Software development route.
- The Hardware or Device programming route.

As these interface are quite significantly different they will be discussed as two separate unique entities.

7.1 The Software Interface.

This section specifically focuses on the user interface programme development environment aspects of the Universal Programming development system.

7.11 The Integrated Development Environment.

The Integrated Development Environment has been developed especially for the new user to the Universal Programming development system. Many new programme developers due to their prior background experience are somewhat unfamiliar with software that is driven by command prompts. This lack of familiarity can be overcome by offering the user an on screen menu display where a limited number of options are presented. The user selects his required option to achieve his required outcome. There is one potential danger with the excessively user friendly interface and that is, many users begin to expect that all application systems are going to tell them what to do. The very nature of developing software for embedded system should imply that the user knows his device and its constraints. The Universal Programming development system cannot and should not be expected to be a teaching application. The user should understand basic requirements and development sequence process before embarking their first programme.

The normal development sequence is :-
- DEFINE the work environment.
- EDIT programme source or CREATE a new programme source.
- ASSEMBLE Source (Back to EDIT if Errors detected).
- LINK appropriate Object (Back to EDIT if Errors detected).
- SIMULATE task image (Back to EDIT if functionality incorrect).
- Submit task image to device programmer.

7.12 The Manual Method.

This mode is perhaps more appropriate for the more experienced programme developer who is working with multiple programme sources. However, continually switching between manual command mode and IDE mode may not be an unreasonable working practice approach, especially as the INIT.EXE programme can set up most of the appropriate default response replies. INIT.EXE is the first programme that is used because it prepares the environment. The rest of the process is as follows :-

- EDIT programme source or CREATE a new programme source.
- ASSEMBLE Source (Back to EDIT if Errors detected).
- LINK appropriate Object (Back to EDIT if Errors detected).
- SIMULATE task image (Back to EDIT if functionality incorrect).
- Submit task image to device programmer.

7.2 The Hardware Interface.

The device programmer is connected to the host system either by :-

- Using a Serial Interface Link.
- Using a USB port.

If the system is connected using a Serial Interface Link when running under WINDOWS 9x or similar system, then one appropriate programme to control the Hardware is the “Hyper Terminal” terminal emulator programme. Once the communication link is established, then the Programmer Hardware will display a menu that will define what functions are available to the user.
Note: As the interface options to the programmer is a universal format, then the control options should be viable on any system that support that interface. The only difficulty may be with the transfer of the data file structures. All communications to and from the programmer will be in an ASCII text format.

Typical operations that should be available are :-

- The option to Download the programming algorithm.
- The option to Download the data to be programmed.
- The option to Upload the data from a programmed device.
- Configuration and Calibration options.

8.0 System testing.

Due to the very large scope and nature of the project a number of different strategies have been implemented. The typical approach for most **Software** entities was to test sub modules for validity. When the basic functionality was proven, the sub modules were integrated into the complete system where the next level in the test sequence was implemented.

8.1 Hardware Testing Strategy.

The whole product was made from a large number of common entities. The complexity of the system was not in the individual items but in the fact that there is a significant quantity of each item. The complexity is also present in the way the sub entities can interact with each other.

8.11 The Analogue Hardware.

The Analogue **Hardware** entities in general were initially developed and tested in a simulated mode using the CADENCE evaluation version of PSPICE. Once the Simulated versions gave the required results, then some of the circuits were built up using a standard prototype bread board circuit. When the Bread board prototypes gave proven results, the circuits were offered for inclusion into the main system. The system is a hybrid Analogue and Digital system and it was necessary at times to rely on the prototype printed circuit boards for the final testing stage. It is for this reason that the test procedures have been so extensively developed, however, the test procedures are now ready when and if the product goes into production.
See appendix item marked Analogue Simulation Results detailing some of the hybrid circuit design aspects.

8.12 The Digital Hardware.

The Digital **Hardware** entities are in general initially checked with simple continuity checks verifying that signal and power routing is correct. Power is applied to the board and voltage checks are taken to verify power routing is correct. The remaining checks are normally software driven and selected pins are monitored with an oscilloscope.

8.2 Software Testing Strategy.

The testing of the Software entities i.e. The Assembler , The Linker , The Macro Processor, The Dis-Assemblers and The Simulator are usually tested as a group. Each processor configuration has an example source that contains one or more variants of every style of instruction that processor should be able to perform. Usually the test suite has all the instructions sorted in alphabetical order. The test suite is submitted to the Assembler, the source is Linked and then Dis-Assembled. Manual checks are then performed to verify that the Dis-Assembly output matches the original source. A second source is then developed that will also attempt to use an example of all the instructions of a processor, but this time, the source is expected to produce some known expected result. The second source is Assembled , Linked and submitted to the Simulator. The Simulator will process the programme and should give the expected results that the second source was expected to produce. The software is then submitted for usage testing. Whenever faults or anomalies are identified, they are corrected and the new software is released with the next system revision update.

The test procedures for the Programmer Hardware is based very much around a production line testing. The actual test procedures are described in the document TESTPROC.DOC and a copy can be found in the Appendix. The majority of the Hardware checks are controlled by a series of mini test programmes. An example of a mini test programme suite can be seen in the document BOARDTST.ASM also located in the Appendix.

9.0 Conclusion.

9.1 The Successes.

With hindsight, this project was significantly larger than had been initially envisaged. After the final update of the GANTT chart had been made and the resource allocation levelled, the GANTT chart indicated that the project was at least five months ahead of schedule (based on 20 hours being allocated to the project per week).
The Software development system has shown itself to be a proven and viable entity if only by the fact that it has been used exclusively to develop the software for the General Purpose Programmer. The introduction of the Macro Processor into the development suite has significantly enhanced Data and Programme structures options that can be used within the development programme source code. It is the Macro Processor specifically that has enabled the ZMON source to be built and hence permitting it to control the whole of the General Purpose Programmer Test Suites and its Main Application Programme. The Simulator has proven to be an invaluable tool during the system fault finding if only to confirm that the binary image was valid and was functioning in the manner specified. It has also been used as a debugging aid when code did not perform quite as expected.

9.2 The Design Methods.

The Application Specific Language concept has shown by example how a PIC Cross Assembler can be developed in an extremely short time period. This concept shows the total flexibility and efficiency it can offer to a system developer.
The use of Box Concept analysis has also shown itself to be an effective methodology for designing small systems both for the hardware and software aspects of the project.

9.3 The Hardware learning activities.

During the project design phase the PSPICE Simulation tool has been highly useful in evaluating much of the analogue circuitry, however, it does not permit one to ignore the need to build some circuits on a breadboard just to verify results. This was brought sharply into focus when a μ741 chip was used instead of a section of an LM348 which was supposed to have identical characteristics.
One of the main lessons learnt during the processor hardware development exercise was that Test Equipment can also go faulty. It was as a result of running the test programmes in the General Purpose Simulator showing them to be viable that the fault finding focus changed. Rather than examining the new hardware, it was considered that the Test equipment may not be performing correctly. The problem was eventually tracked down to be an earth loop fault located on the EPROM Emulator socket connector. The effects of that fault did stall the project for a significant time period as the interactions were of an intermittent and inconsistent nature.

9.4 The Hardware Units.

Although a complete set of boards (some 30 in total) have not been produced, a prototype of every variant has been fabricated. The Test procedures designed for the production phase of the manufacturing process have shown that the design of every board is sound. All the boards perform to specification, however, some minor modifications to them may be implemented to help reduce the manufacturing time. Once sufficient quantities have been manufactured, then the system will be ready to begin the next phase in the process which is the implementation of the device programming algorithms.

9.5 Future Development.

Now that the development prototype modules have proved themselves, a second phase design review needs to take place. This will specifically look at the implementation of using :-

- Programmable logic devices to minimise the component count.
- Surface mount components to allow more compact PCB to be designed.
- Alternative interconnection methodology to reduce overall plug and socket costs.
- An appropriate casing to enclose the product.
- Alternative cost effective methods to implement the Analogue circuitry.
- An Alternative cost effective and labour efficient method of producing Printed Circuit Boards.
- The use of the prototype programmer for programming the devices in the next generation of programmer.

The General Purpose Assembler and Simulator now supports a limited environment and programming capability for PIC processor range. However, a fully viable market-ready embedded microprocessor development system for PIC processors will still require considerably more functionality to be added to the system. Specifically, the Simulator Library code needs the implementation of the Watchdog timer and some appropriate devices so that more practical demonstration programmes can be presented.

9.6 Alternative Development Directions.

9.61 FPGA Processing.
In pure concept terms, the General Purpose Programmer accepts a binary image pattern and burns that pattern into a specified device. One type of pattern may be used to implement some logic functionality in a programmable logic device. The next binary image could be some machine code to implement an application in a microprocessor. Only the method of production of the binary image differs. The General Purpose Assembler basically translates mnemonics of a specified processor to binary codes. It may be possible to develop a series of assembler style languages targeted at a specific range of logic devices. The General Purpose Assembler could then create the appropriate binary patterns from a source code necessary to configure the selected logic devices. This may be a future development path for the General Purpose Assembler.

9.62 Device Recognition.
By connecting some of the programming pins of the General Purpose Programmer via discrete resistors to previously allocated connections on the ZIF socket, it should be possible to use the programmer to analyse the functionality of an unknown device by stimulation of its pins. Once the programmer has captured a signature from the unknown device, it could compare that signature against known templates for part code recognition.

9.63 Device Emulation.
The General Purpose Programmer can derive logic signals to a pin and read levels present on that pin. Therefore, with a suitable adapter, the programmer could become a General Purpose Device Emulator. Obviously, the operational speed of the programmer is currently relatively slow, however, with some design changes a better performance should be realisable. Hence the programmer could not only program devices but also emulate them in real time prior to the burning process. This could be a highly cost effective option for future system developers.

List of Useful References.

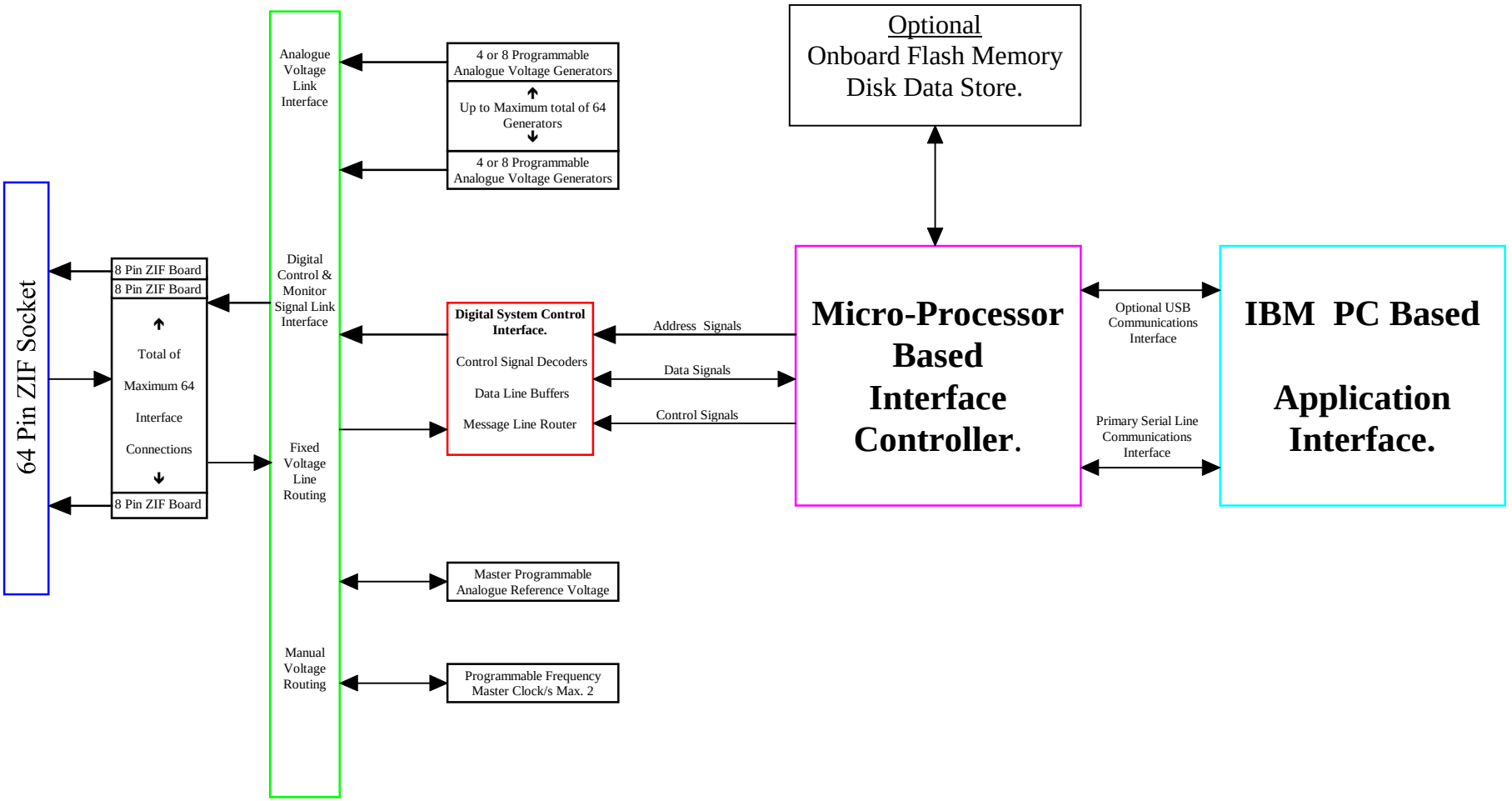
Title of Document	References
HD64180 Hardware User Manual. 1987	UM64180/88/03 Hitachi Ltd
PIC16C84 Data Sheet	DS30445A ©1996 Microchip Technology Inc.
PIC16F87X Data Sheet	DS3029C ©2001 Microchip Technology Inc.
Programmable Logic Devices 1986	Mulard Publication no. 806352
PAL Device Handbook AMD and MMI	1988 Advanced Micro Devices Inc. 10173A/0
PAL Device Handbook AMD and MMI	1988 Advanced Micro Devices Inc. 10206A/0
The TTL Data Book for design Engineers 1984	Vol. 1 , ISBN 3-88078-037-4
The TTL Data Book for design Engineers.	Vol. 2 , ISBN 3-88078-042-0
TI High Speed CMOS Logic Data Book 1988	ISBN 3-88078-071-4
8 Bit Multiplying Digital to Analogue Converters TLC7524C, TLC7524E, TLC7524I	SLAS061C Revised November 1998 Texas Instruments

APPENDIX Section

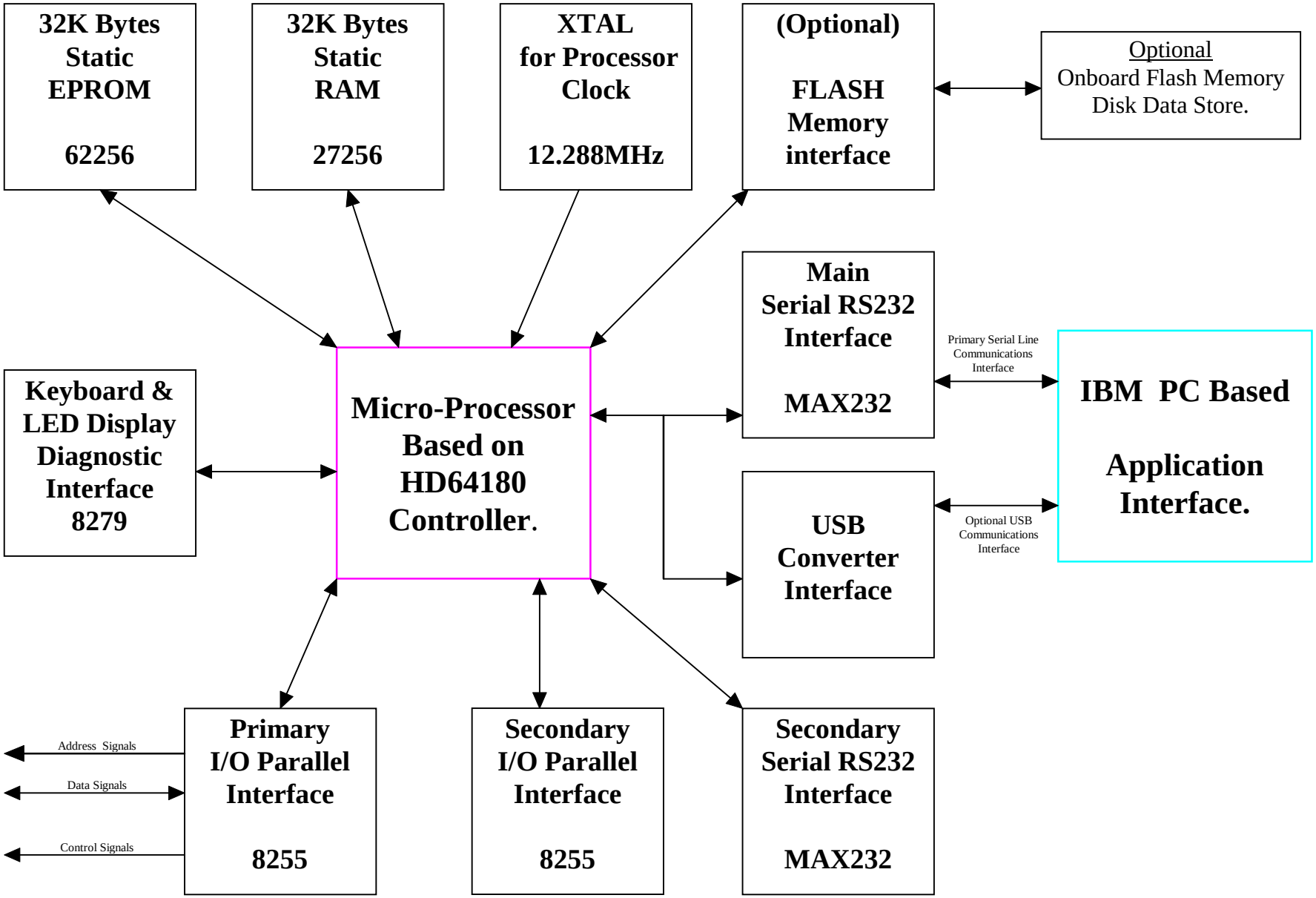
This section consists of the following pages

Title of Pages	Number of Pages
Block Diagram of General Purpose Programmer.	1
Block Diagram of General Purpose Microprocessor Module.	1
Programmer Interface Block Diagram.	1
The Project GANTT Chart	3
The Project PERT Chart.	3
Activity calculation spreadsheet Filename PERT_EIE.XLS.	1
Work Analysis Calculations Filename PERTCALC.doc	2
Decision Tree Diagram DTREE.DOC	1
Analogue Simulation Results.	A Sub-Section
Circuit Specifics of the General Purpose Programmer.	A Sub-Section
Documentation, Demonstration and Test Programmes.	A Sub-Section
Application Specific Language Source code for PIC16 .	A Sub-Section
Application Specific Language Source code for PIC16SIM.	A Sub-Section

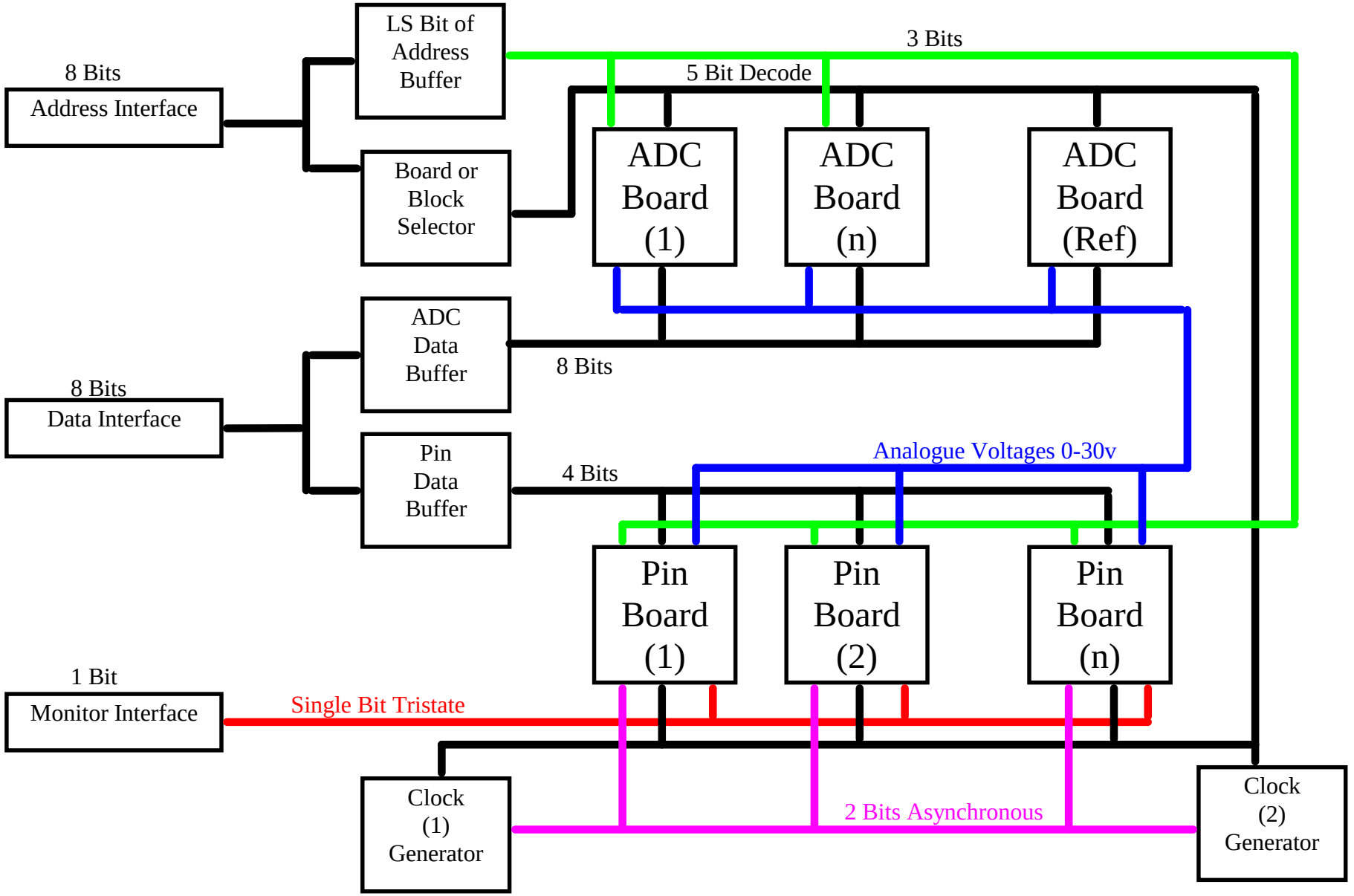
Block Diagram of the General Purpose Device Programmer.

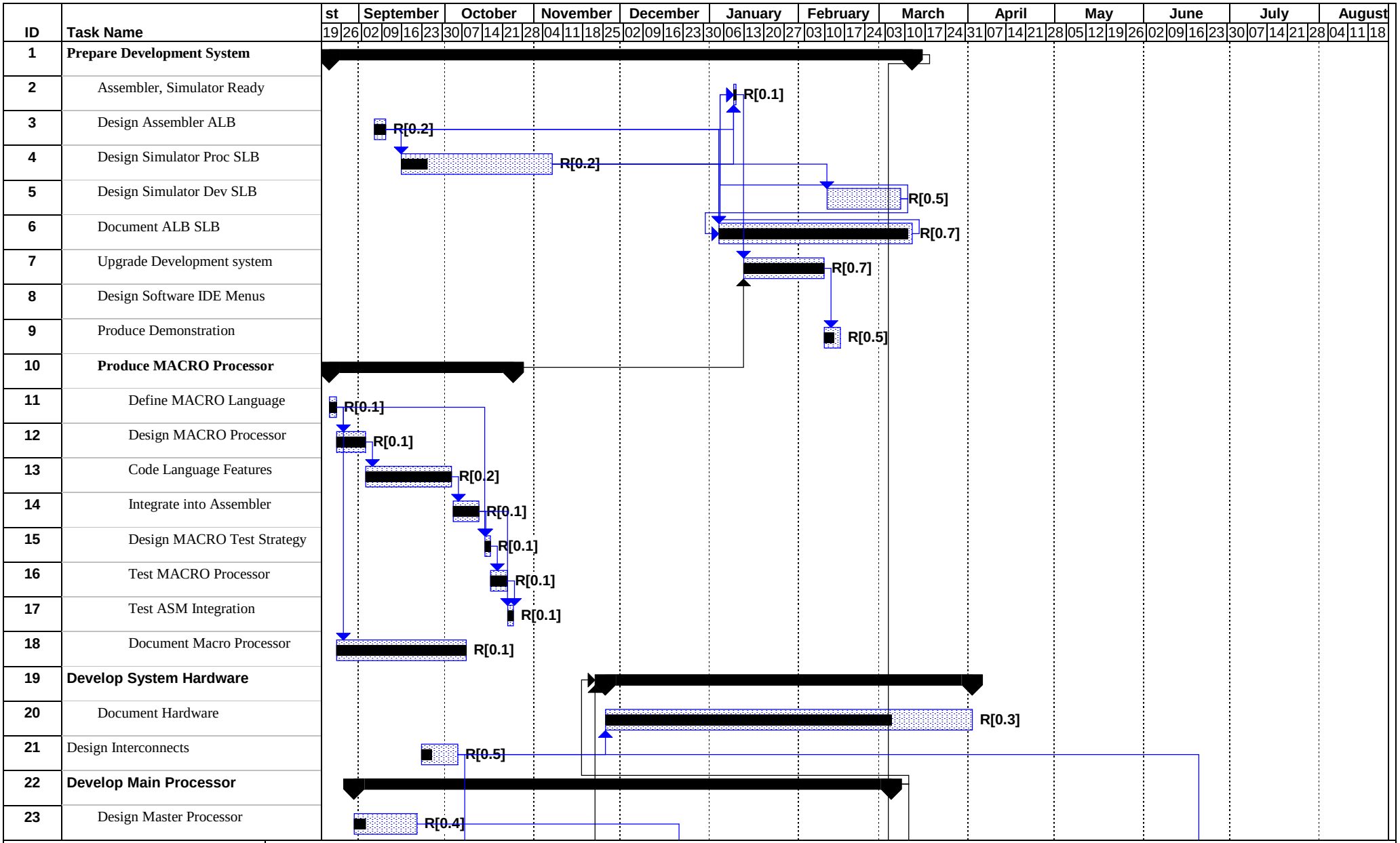


Block Diagram of the General Purpose Microprocessor Module.



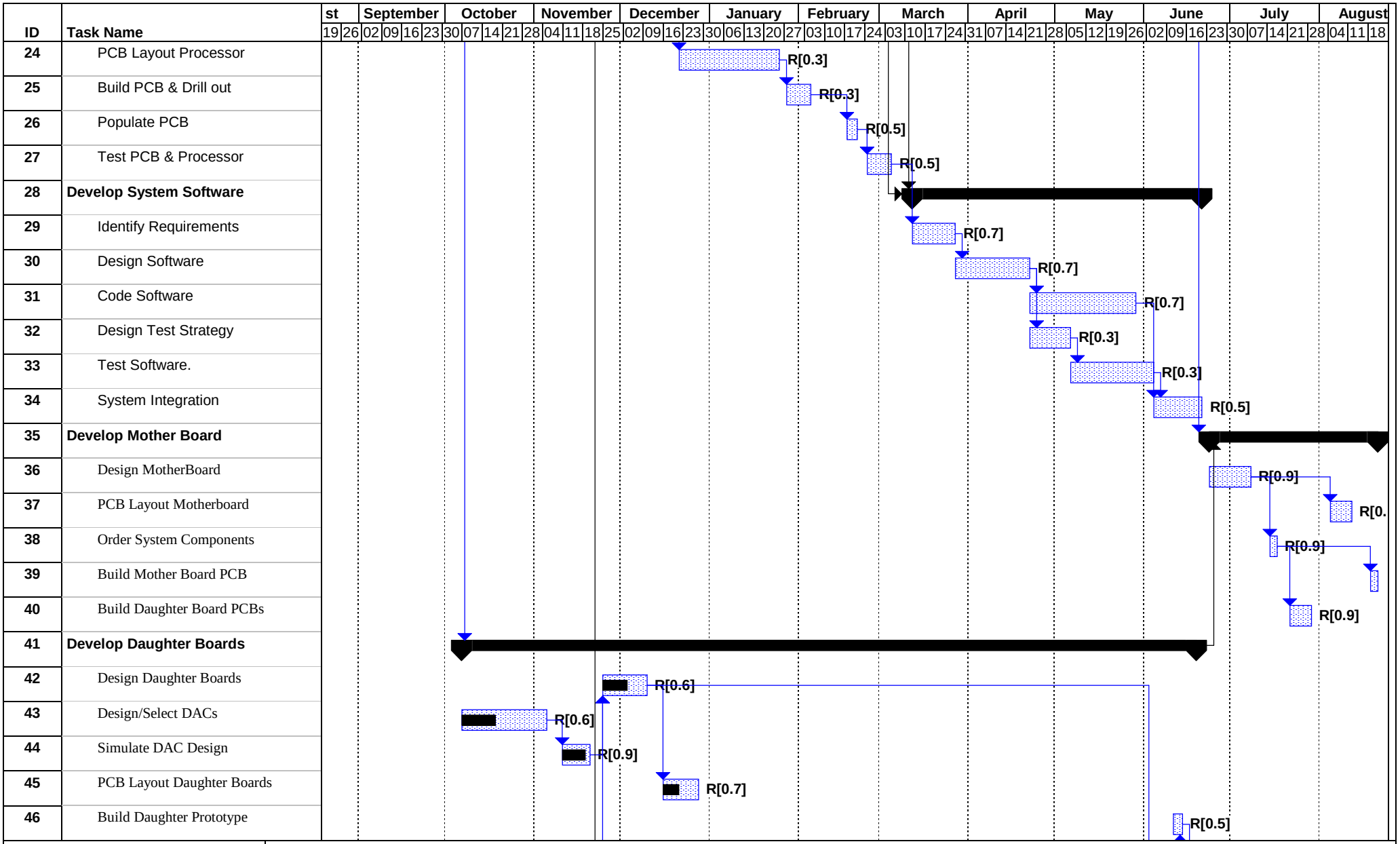
Programmer Interface Block Diagram.



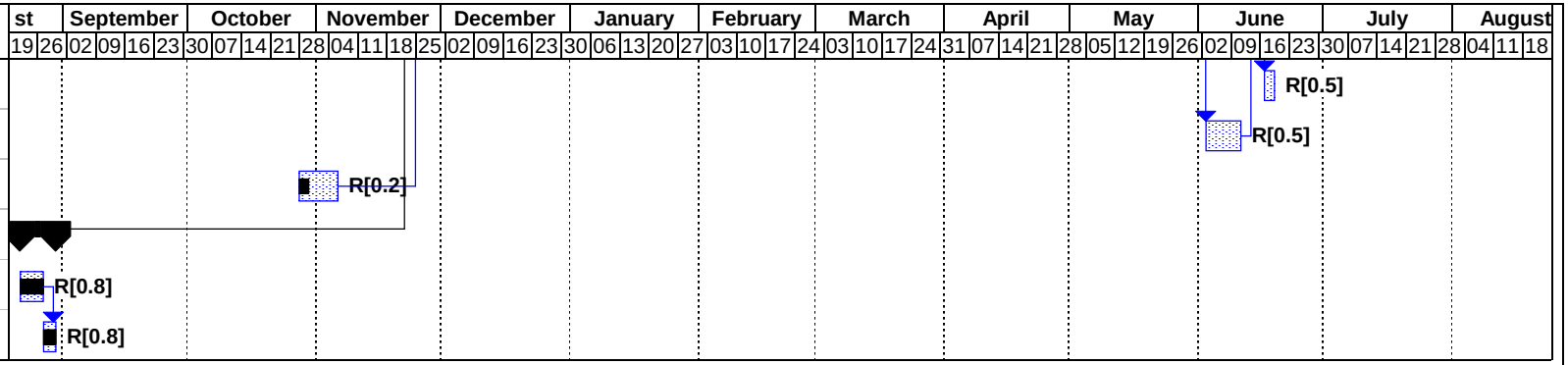


Project: General Purpose Programme
Date: Tue 23/08/11

Task
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Milestone
Rolled Up Progress

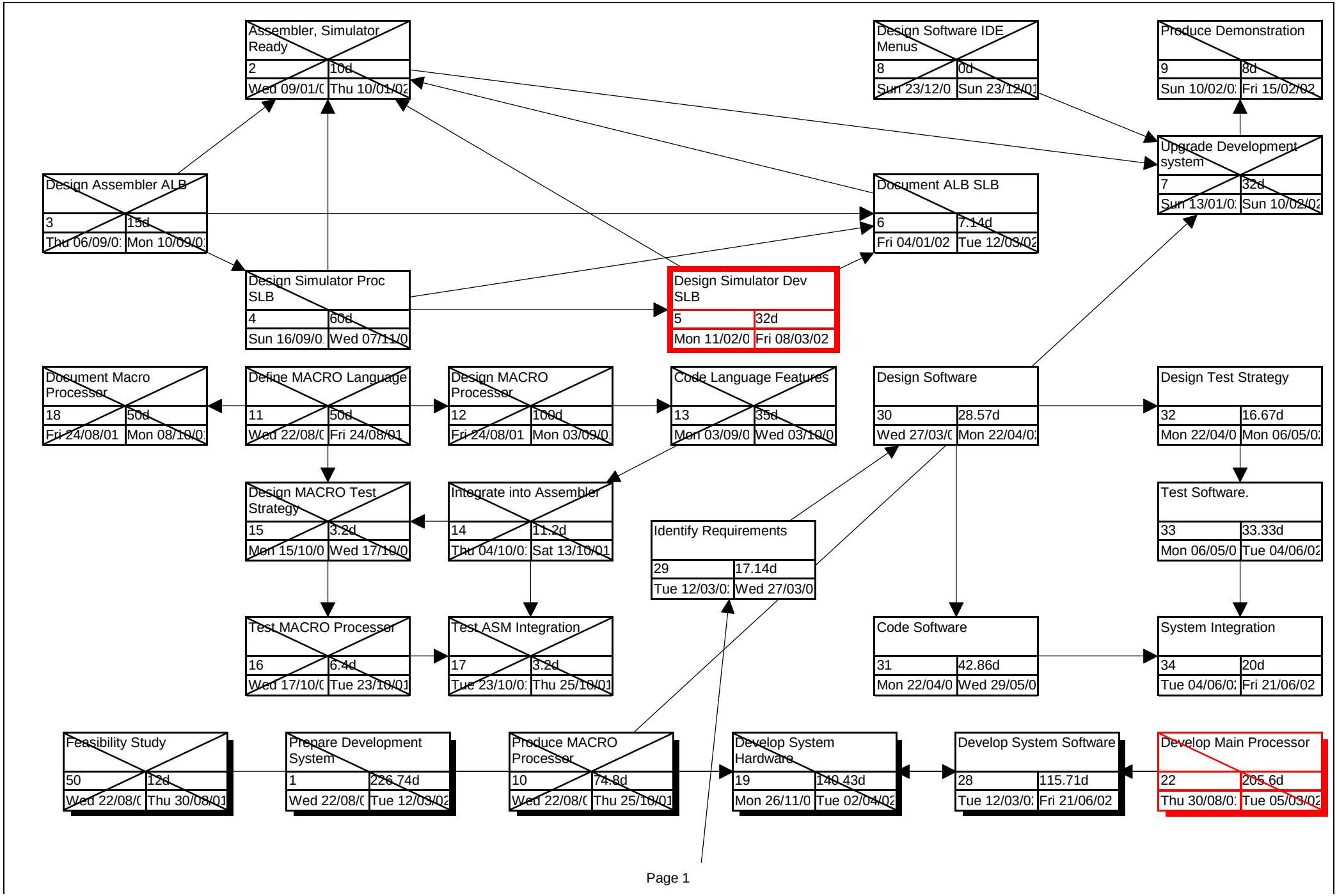

st
September
October
November
December
January
February
March
April
May
June
July
August
19 26 02 09 16 23 30 07 14 21 28 04 11 18 25 02 09 16 23 30 06 13 20 27 03 10 17 24 03 10 17 24 31 07 14 21 28 05 12 19 26 02 09 16 23 30 07 14 21 28 04 11 18
ID
Task Name
24 PCB Layout Processor
25 Build PCB & Drill out
26 Populate PCB
27 Test PCB & Processor
28 Develop System Software
29 Identify Requirements
30 Design Software
31 Code Software
32 Design Test Strategy
33 Test Software.
34 System Integration
35 Develop Mother Board
36 Design MotherBoard
37 PCB Layout Motherboard
38 Order System Components
39 Build Mother Board PCB
40 Build Daughter Board PCBs
41 Develop Daughter Boards
42 Design Daughter Boards
43 Design/Select DACs
44 Simulate DAC Design
45 PCB Layout Daughter Boards
46 Build Daughter Prototype
R[0.3]
R[0.3]
R[0.5]
R[0.5]
R[0.7]
R[0.7]
R[0.7]
R[0.3]
R[0.3]
R[0.5]
R[0.9]
R[0.
R[0.9]
R[0.9]
R[0.6]
R[0.6]
R[0.9]
R[0.7]
R[0.5]
Project: General Purpose Programme
Date: Tue 23/08/11
Task
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Milestone
Rolled Up Progress

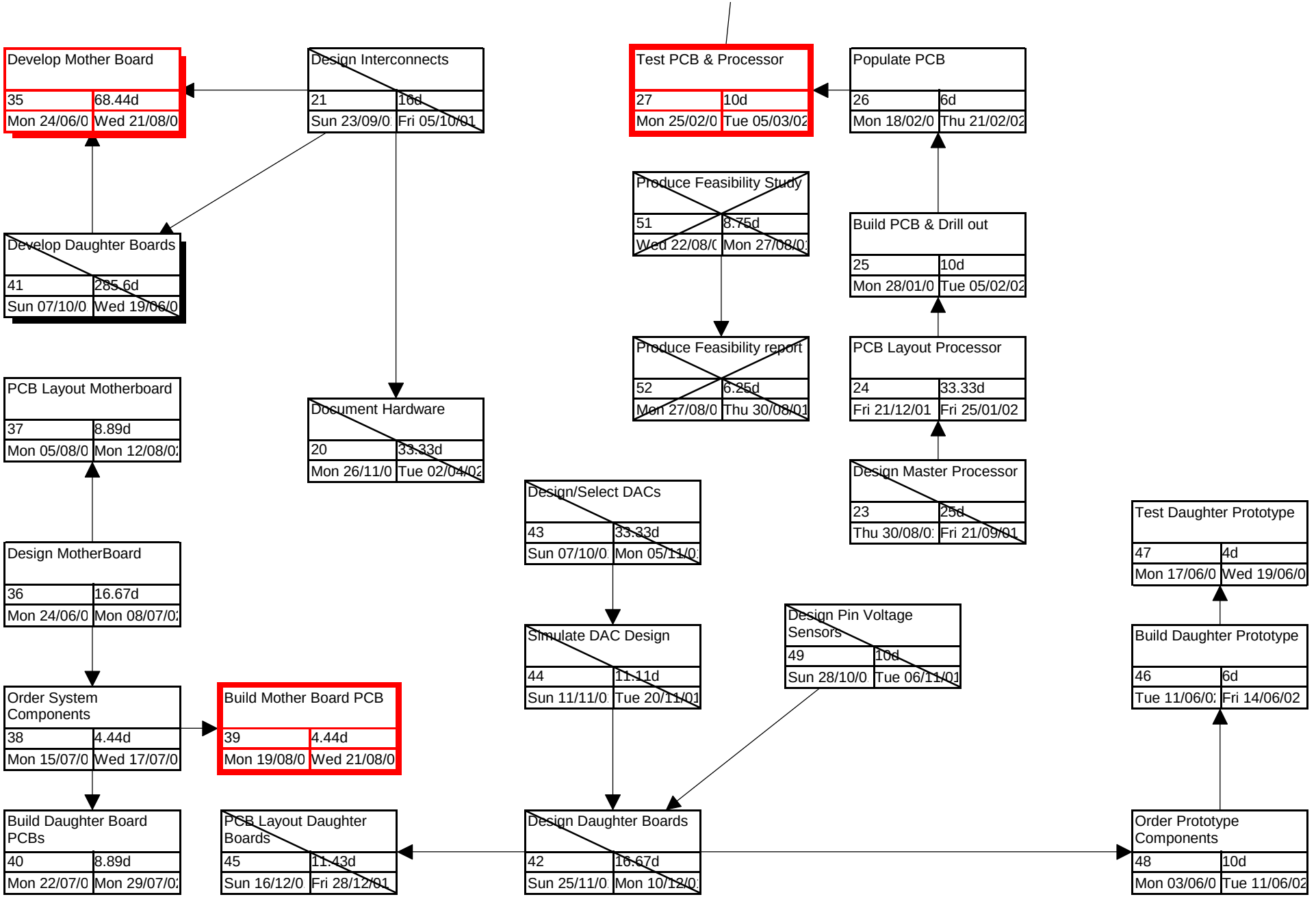
ID	Task Name
47	Test Daughter Prototype
48	Order Prototype Components
49	Design Pin Voltage Sensors
50	**Feasibility Study**
51	Produce Feasibility Study
52	Produce Feasibility report



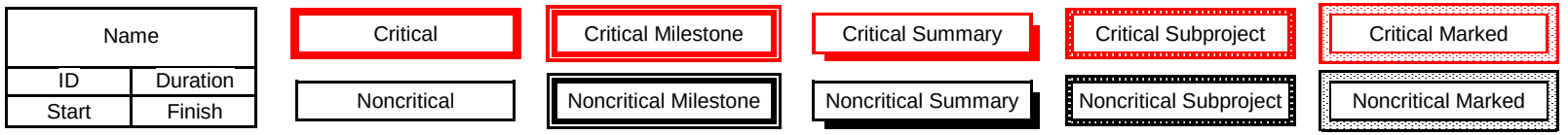
Project: General Purpose Programme
Date: Tue 23/08/11

Task
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Milestone
Rolled Up Progress


Assembler, Simulator Ready
2
10d
Wed 09/01/0
Thu 10/01/02
Design Software IDE Menus
8
0d
Sun 23/12/0
Sun 23/12/01
Produce Demonstration
9
8d
Sun 10/02/0
Fri 15/02/02
Upgrade Development system
7
32d
Sun 13/01/0
Sun 10/02/02
Design Assembler ALB
3
15d
Thu 06/09/0
Mon 10/09/0
Document ALB SLB
6
7.14d
Fri 04/01/02
Tue 12/03/02
Design Simulator Proc SLB
4
60d
Sun 16/09/0
Wed 07/11/0
Design Simulator Dev SLB
5
32d
Mon 11/02/0
Fri 08/03/02
Document Macro Processor
18
50d
Fri 24/08/01
Mon 08/10/0
Define MACRO Language
11
50d
Wed 22/08/0
Fri 24/08/01
Design MACRO Processor
12
100d
Fri 24/08/01
Mon 03/09/0
Code Language Features
13
35d
Mon 03/09/0
Wed 03/10/0
Design Software
30
28.57d
Wed 27/03/0
Mon 22/04/0
Design Test Strategy
32
16.67d
Mon 22/04/0
Mon 06/05/0
Design MACRO Test Strategy
15
3.2d
Mon 15/10/0
Wed 17/10/0
Integrate into Assembler
14
11.2d
Thu 04/10/0
Sat 13/10/01
Identify Requirements
29
17.14d
Tue 12/03/0
Wed 27/03/0
Test Software.
33
33.33d
Mon 06/05/0
Tue 04/06/02
Test MACRO Processor
16
6.4d
Wed 17/10/0
Tue 23/10/01
Test ASM Integration
17
3.2d
Tue 23/10/0
Thu 25/10/01
Code Software
31
42.86d
Mon 22/04/0
Wed 29/05/0
System Integration
34
20d
Tue 04/06/0
Fri 21/06/02
Feasibility Study
50
12d
Wed 22/08/0
Thu 30/08/01
Prepare Development System
1
226.74d
Wed 22/08/0
Tue 12/03/02
Produce MACRO Processor
10
74.8d
Wed 22/08/0
Thu 25/10/01
Develop System Hardware
19
140.43d
Mon 26/11/0
Tue 02/04/02
Develop System Software
28
115.71d
Tue 12/03/0
Fri 21/06/02
Develop Main Processor
22
205.6d
Thu 30/08/0
Tue 05/03/02


Develop Mother Board
35
68.44d
Mon 24/06/0
Wed 21/08/0
Design Interconnects
21
16d
Sun 23/09/0
Fri 05/10/01
Test PCB & Processor
27
10d
Mon 25/02/0
Tue 05/03/02
Populate PCB
26
6d
Mon 18/02/0
Thu 21/02/02
Produce Feasibility Study
51
8.75d
Wed 22/08/(
Mon 27/08/0
Develop Daughter Boards
41
285.6d
Sun 07/10/0
Wed 19/06/0
Build PCB & Drill out
25
10d
Mon 28/01/0
Tue 05/02/02
Produce Feasibility report
52
6.25d
Mon 27/08/0
Thu 30/08/01
PCB Layout Processor
24
33.33d
Fri 21/12/01
Fri 25/01/02
PCB Layout Motherboard
37
8.89d
Mon 05/08/0
Mon 12/08/0
Document Hardware
20
33.33d
Mon 26/11/0
Tue 02/04/02
Design Master Processor
23
25d
Thu 30/08/0
Fri 21/09/01
Design/Select DACs
43
33.33d
Sun 07/10/0
Mon 05/11/0
Test Daughter Prototype
47
4d
Mon 17/06/0
Wed 19/06/0
Design MotherBoard
36
16.67d
Mon 24/06/0
Mon 08/07/0
Simulate DAC Design
44
11.11d
Sun 11/11/0
Tue 20/11/01
Design Pin Voltage Sensors
49
10d
Sun 28/10/0
Tue 06/11/01
Build Daughter Prototype
46
6d
Tue 11/06/0
Fri 14/06/02
Order System Components
38
4.44d
Mon 15/07/0
Wed 17/07/0
Build Mother Board PCB
39
4.44d
Mon 19/08/0
Wed 21/08/0
Build Daughter Board PCBs
40
8.89d
Mon 22/07/0
Mon 29/07/0
PCB Layout Daughter Boards
45
11.43d
Sun 16/12/0
Fri 28/12/01
Design Daughter Boards
42
16.67d
Sun 25/11/0
Mon 10/12/0
Order Prototype Components
48
10d
Mon 03/06/0
Tue 11/06/02

Project: General Purpose Programme
Date: Tue 23/08/11



General Purpose Programmer Project Activity Analysis

Number	Activity	Optimistic	Most Likely	Pessimistic	Estimate	Varience	Risk
2	Assembler, Simulator Ready	1	1	1	1.00	0.00	1
3	Design Assembler ALB	2	3	5	3.17	0.25	1
4	Design Simulator Proc SLB	7	12	30	14.17	14.69	3
5	Design Simulator Dev SLB	9	16	30	17.17	12.25	3
6	Document ALB SLB	4	5	7	5.17	0.25	1
7	Upgrade Development system	4	20	30	19.00	18.78	2
8	Design Software IDE Menus	2	5	9	5.17	1.36	2
9	Produce Demonstration	3	4	5	4.00	0.11	1
11	Define MACRO Language	3	5	6	4.83	0.25	1
12	Design MACRO Processor	2	10	25	11.17	14.69	4
13	Code Language Features	2	7	15	7.50	4.69	4
14	Integrate into Assembler	2	4	8	4.33	1.00	4
15	Design MACRO Test Strategy	1	3	4	2.83	0.25	2
16	Test MACRO Processor	2	5	6	4.67	0.44	1
17	Test ASM Integration	1	2	3	2.00	0.11	1
18	Document Macro Processor	1	5	6	4.50	0.69	1
20	Document Hardware	8	10	12	10.00	0.44	1
21	Design Interconnects	7	8	18	9.50	3.36	3
23	Design Master Processor	8	10	20	11.33	4.00	3
24	PCB Layout Processor	9	10	30	13.17	12.25	5
25	Build PCB & Drill out	2	3	4	3.00	0.11	2
26	Populate PCB	2	3	4	3.00	0.11	1
27	Test PCB & Processor	4	5	8	5.33	0.44	3
29	Identify Requirements	7	12	20	12.50	4.69	3
30	Design Software	11	20	30	20.17	10.03	3
31	Code Software	15	30	70	34.17	84.03	5
32	Design Test Strategy	3	5	9	5.33	1.00	3
33	Test Software.	7	10	18	10.83	3.36	3
34	System Integration	8	10	20	11.33	4.00	4
36	Design MotherBoard	10	15	20	15.00	2.78	4
37	PCB Layout Motherboard	7	8	25	10.67	9.00	5
38	Build Mother Board PCB	3	4	5	4.00	0.11	2
40	Design Daughter Boards	7	10	14	10.17	1.36	2
41	Design/Select DACs	10	20	22	18.67	4.00	3
42	Simulate DAC Design	8	10	12	10.00	0.44	2
43	PCB Layout Daughter Boards	7	8	12	8.50	0.69	3
44	Build Daughter Prototype	2	3	4	3.00	0.11	2
45	Test Daughter Prototype	2	2	5	2.50	0.25	3
46	Order Prototype Components	2	5	7	4.83	0.69	2
47	Order System Components	2	4	5	3.83	0.25	2
48	Build Daughter Board PCBs	7	8	12	8.50	0.69	2
49	Design Pin Voltage Sensors	2	2	6	2.67	0.44	3
51	Produce Feasibility Study	6	7	12	7.67	1.00	3
52	Produce Feasibility report	4	5	8	5.33	0.44	3
	Totals	216.00	354.00	622.00	375.67	219.94	112.00
	Averages					5.11	2.80
	Overall using Totals					9450.78	

Work Analysis Calculations and PERT.

Program Evaluation and Review Technique.

Calculations of estimated times and variances.

The document Appendix Filename PERT_EIE.XLS shows the calculations of the estimated times and the variances for the project.

The estimated times are calculated from the following formula

$$\frac{OptimisticTime + (4 * MostLikelyTime) + PessimisticTime)}{6}$$

The variance is calculated from the following formula

$$\frac{(PessimisticTime - OptimisticTime)^2}{36}$$

Layered PERT Diagram.

The document Appendix PERT Diagram is based on the Appendix Project GANNT Chart and has been organised to show the linkage between task activities. The all red boxes show the sections that have specifically identified to be critical path. Note at this level "all resource overload" had not been fully resolved.

Estimated time for completion.

The document Appendix Filename PERT_EIE.XLS shows the summation calculation of the expected completion times.

Work Analysis Calculations and PERT.

Program Evaluation and Review Technique.

Calculations of probabilities of project overrun.

$$\text{variance} = \sigma^2 = \frac{(PessimisticTime - OptimisticTime)^2}{36}$$

The following formulae can be used for the calculation of the variance does appear to ensure that the calculated "*deviate value z"* will also lie in the bounds of standard deviation table.

$$\text{variance} = \sigma^2 = \frac{(\sum PessimisticTimes - \sum OptimisticTimes)^2}{36}$$

From the spreadsheet, we can see that the most likely completion time is 354 days.

The probability of 10%,20% and 50% overruns are calculated as follows :-

the 10% overrun period is 354 + 10% = 389.4 days
the 20% overrun period is 354 + 20% = 424.8 days
the 50% overrun period is 354 + 50% = 531 days

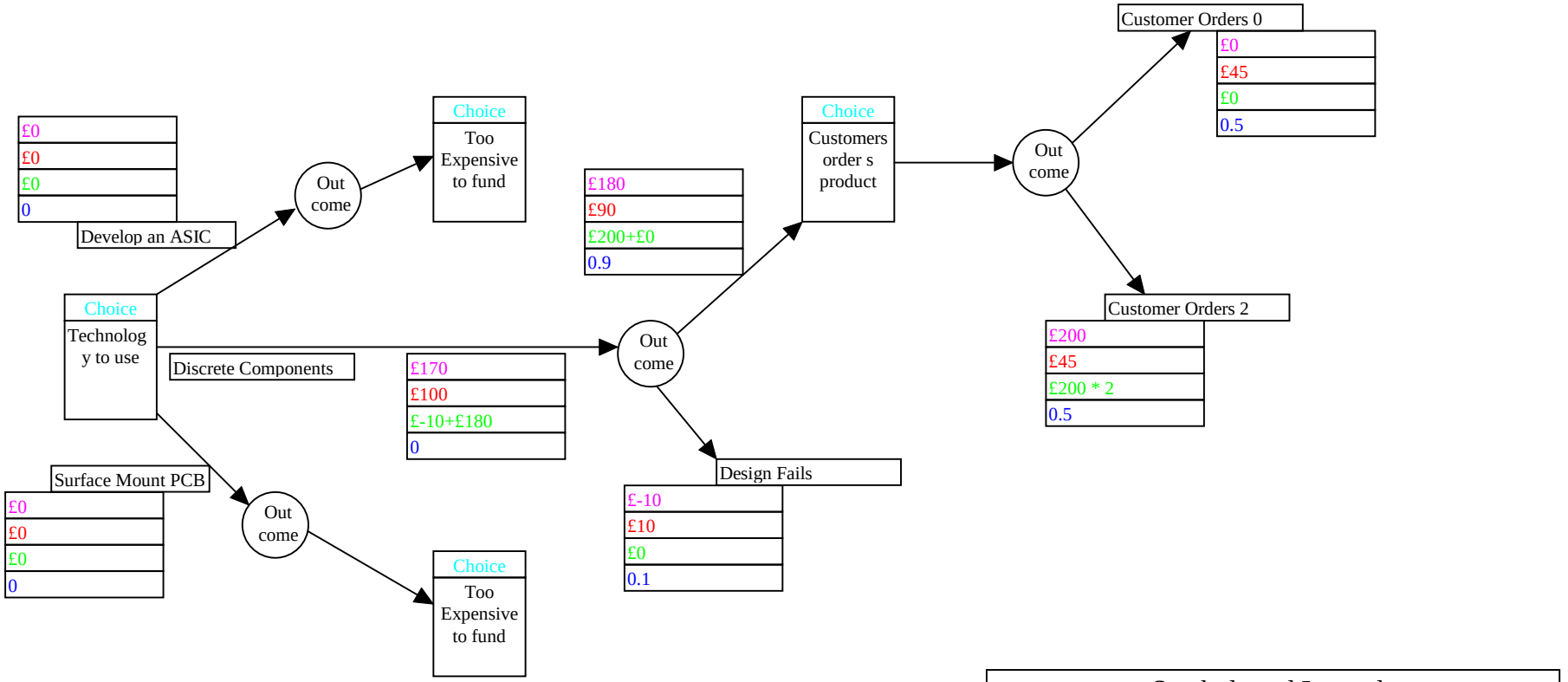
$$normal\_deviate\_z = \frac{T\arg et - mean(\mu)}{\sqrt{\sigma^2}} \quad \text{where } \sigma^2 = 9450.78$$

normal deviate for 10% = $\frac{389.4 - 354}{97.215}$ = 0.36414 the tables give a value of 0.1406
so probability of 10% overrun is = (1-(.5+0.1406))*100% =35.94%

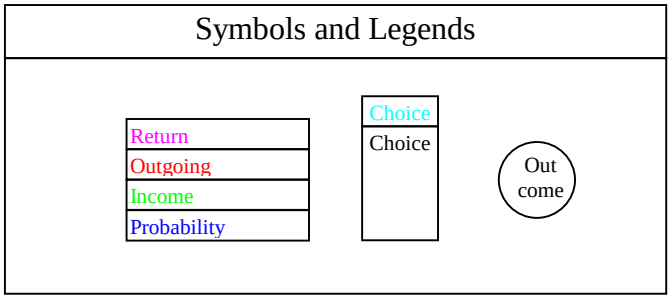
normal deviate for 20% = $\frac{424.8 - 354}{97.215}$ = 0.7282 the tables give a value of 0.2673
so probability of 20% overrun is = (1-(.5+0.2673))*100% =23.27%

normal deviate for 50% = $\frac{531 - 354}{97.215}$ = 1.8207 the tables give a value of 0.4656
so probability of 50% overrun is = (1-(.5+0.4656))*100% =3.44%

Decision Tree for General Purpose Programmer Project



Symbols and Legends



APPENDIX
Sub-Section

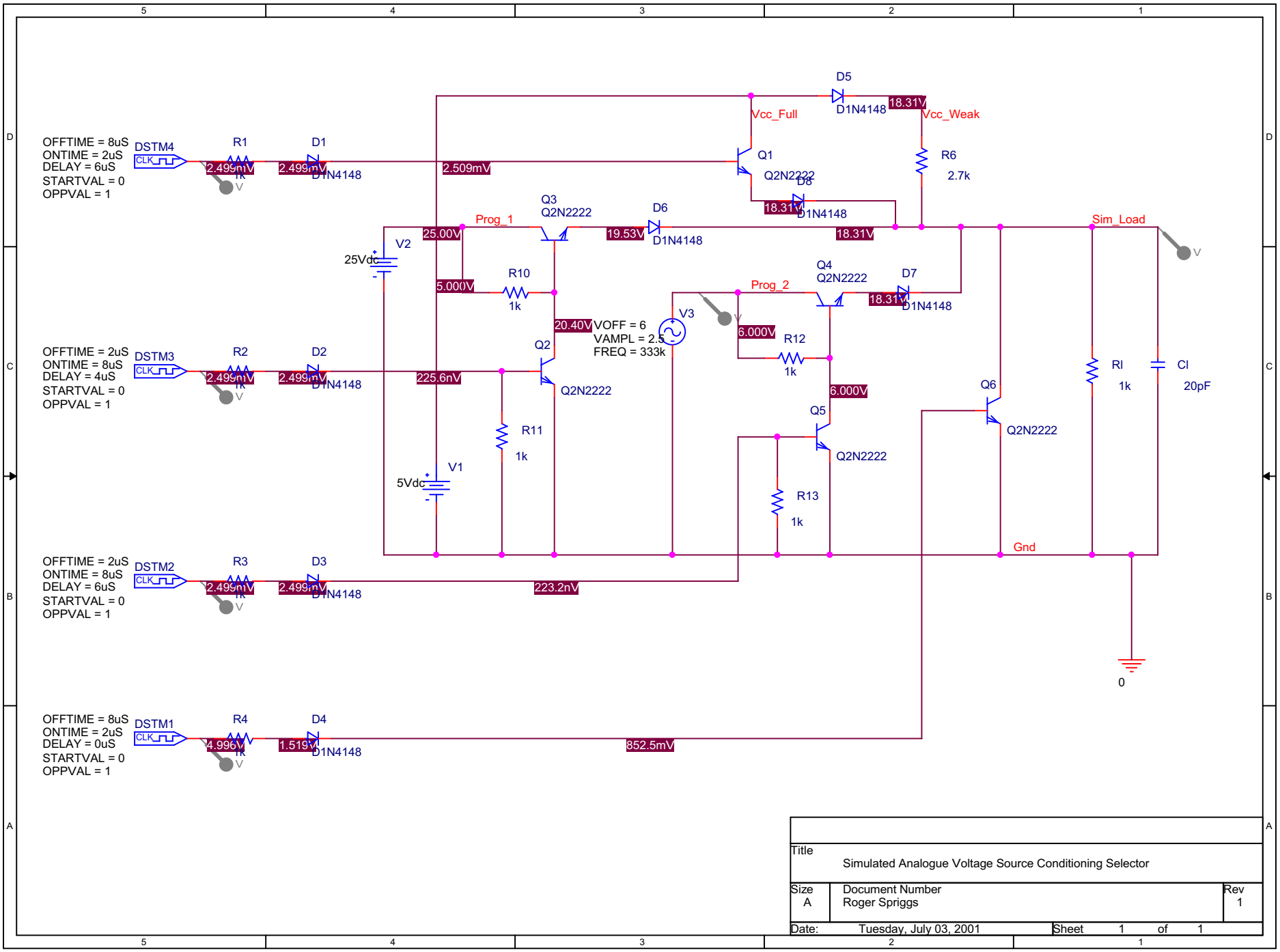
This section contains the following items:

Analogue Simulation Results.

Description :

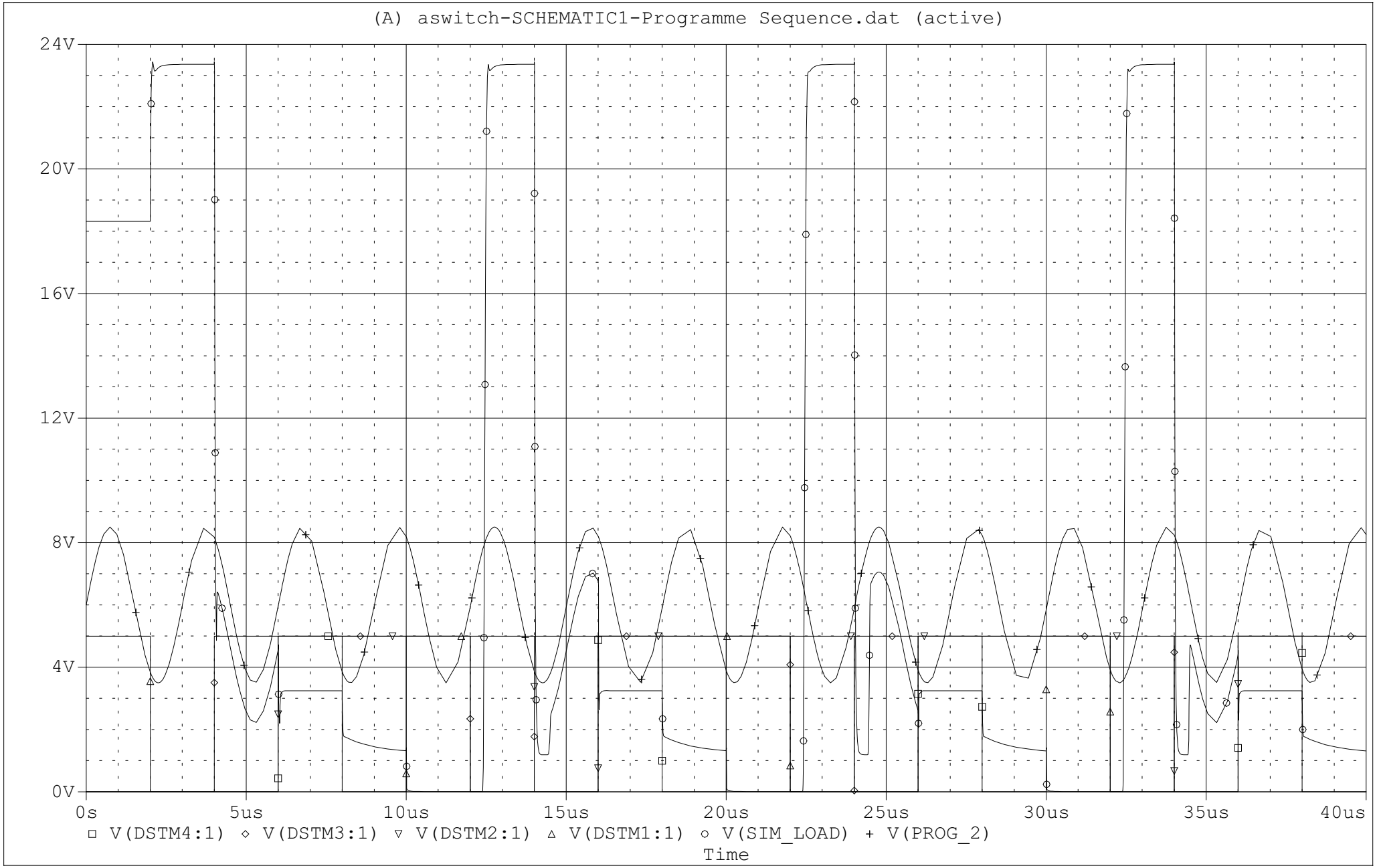
This is the Analogue Simulation Results section. The section contains specifically the following items :-

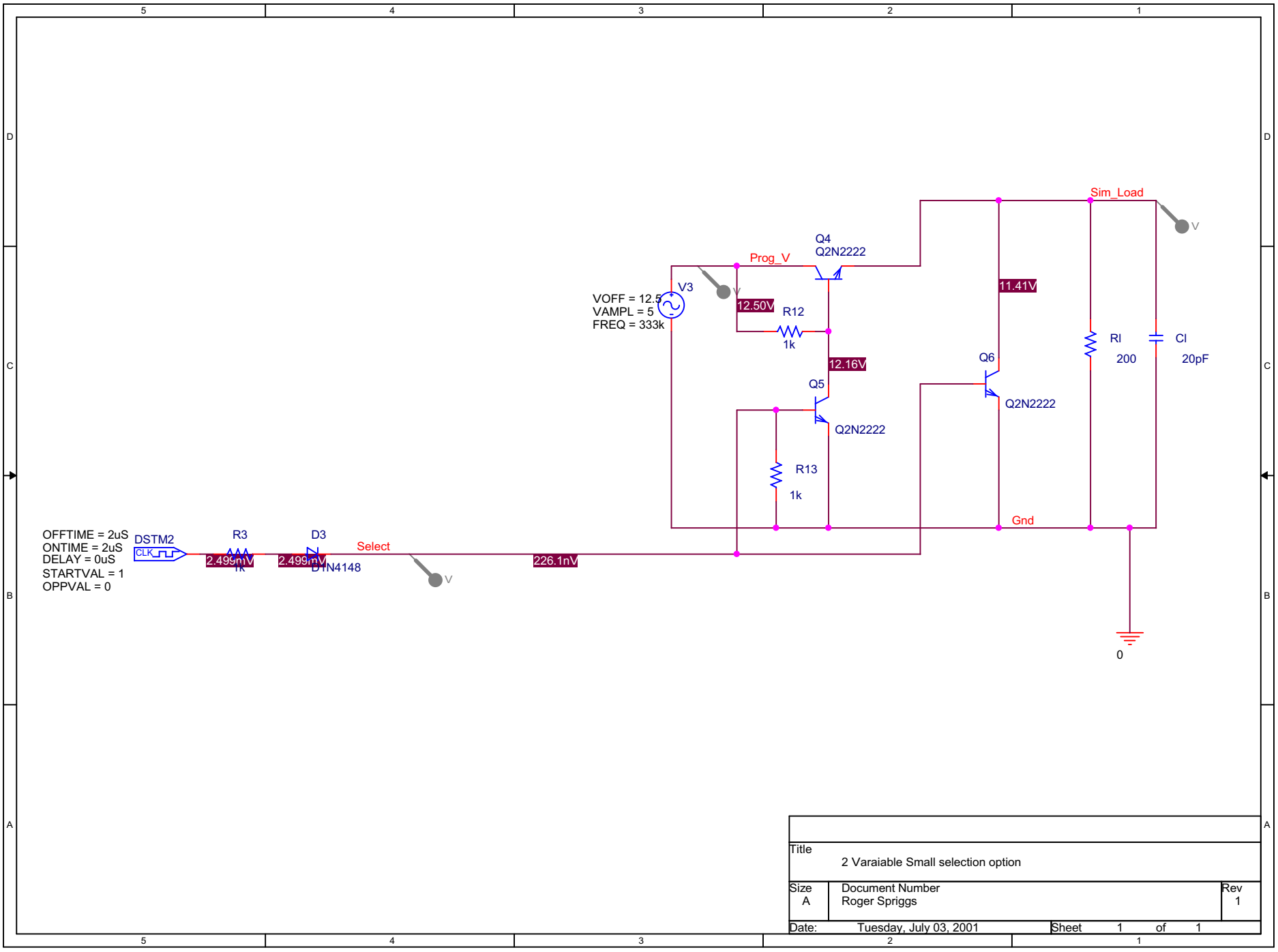
Title of Pages and additional notes.	Number of Pages
The initial design Analogue Voltage Source Conditioning Selector. Circuit basis assumed discrete programming and chip supply voltages available.	2
The evaluation of final circuit in switching mode, 2 Variable Small selection option.	2
Pin Monitor and calibration circuit.	2
Basic Configuration Level sensor with Pulse Switch feature.	2
Single Supply 7 Bit DAC. This is an abnormal operating mode for this type of device.	2
Combined D/A Converter with Pulse switching Circuit. This checked the system performance and loading when applying a high voltage programming pulse to simulated device.	2
OP Amp Buffer Design for DAC. It was necessary for the circuit to act both as a Buffer and multiply the voltage of the Reference DAC to give the required range. The DAC had to be isolated from feedback voltages higher than +15 Volts. The OP Amp needed to operate in Single Supply Mode to give the required 0 to 30 Volt swing.	2


OFFTIME = 8uS
ONTIME = 2uS
DELAY = 6uS
STARTVAL = 0
OPPVAL = 1
DSTM4
CLK
R1
1k
D1
D1N4148
2.499mV
2.509mV
Vcc_Full
D5
D1N4148
18.31V
Vcc_Weak
Q1
Q2N2222
R6
2.7k
D8
D1N4148
18.31V
Q3
Q2N2222
Prog_1
25.00V
D6
D1N4148
19.53V
18.31V
Sim_Load
V2
25Vdc
R10
1k
5.000V
Q4
Q2N2222
Prog_2
D7
D1N4148
18.31V
V3
20.40V
VOFF = 6
VAMPL = 2.5
FREQ = 333k
6.000V
R12
1k
OFFTIME = 2uS
ONTIME = 8uS
DELAY = 4uS
STARTVAL = 0
OPPVAL = 1
DSTM3
R2
D2
D1N4148
2.499mV
225.6nV
Q2
Q2N2222
R11
1k
V1
5Vdc
6.000V
Q5
Q2N2222
Q6
Q2N2222
RI
1k
CI
20pF
R13
1k
OFFTIME = 2uS
ONTIME = 8uS
DELAY = 6uS
STARTVAL = 0
OPPVAL = 1
DSTM2
R3
D3
D1N4148
2.499mV
223.2nV
Gnd
0
OFFTIME = 8uS
ONTIME = 2uS
DELAY = 0uS
STARTVAL = 0
OPPVAL = 1
DSTM1
R4
D4
D1N4148
4.996V
1.519V
852.5mV
Title
Simulated Analogue Voltage Source Conditioning Selector
Size
A
Document Number
Roger Spriggs
Rev
1
Date:
Tuesday, July 03, 2001
Sheet 1 of 1

** Profile: "SCHEMATIC1-Programme Sequence" [E:\SCHOOL7\Dups\aswitch-SCHEMATIC1-Programme Sequence.sim]

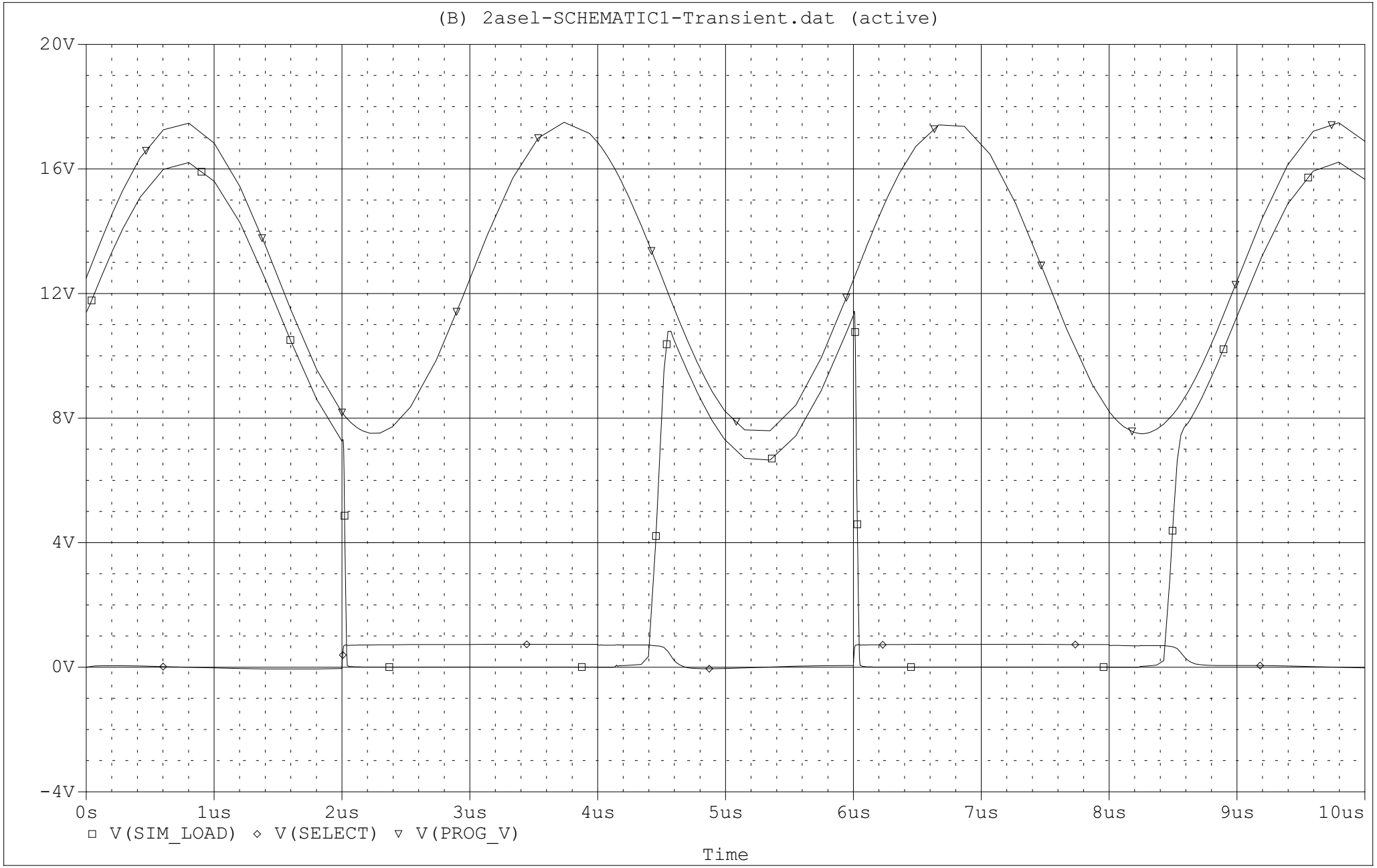
Date/Time run: 09/13/11 14:58:24 Temperature: 27.0

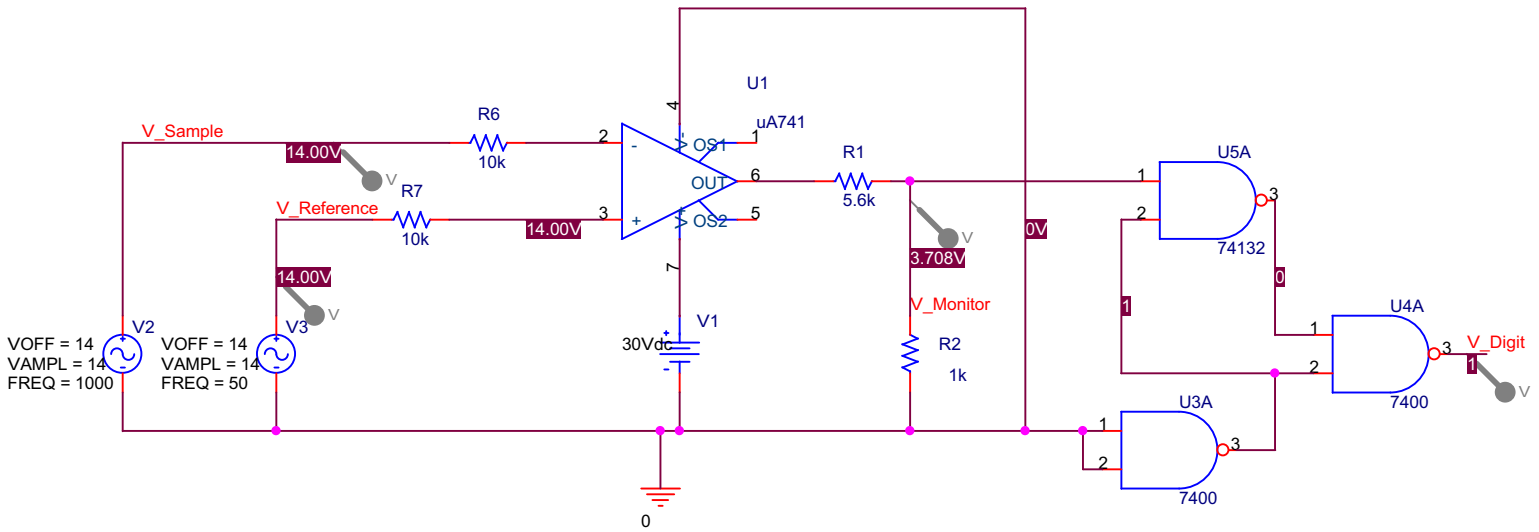



OFFTIME = 2uS
ONTIME = 2uS
DELAY = 0uS
STARTVAL = 1
OPPVAL = 0
DSTM2
CLK
R3
1k
2.499nV
D3
DTN4148
2.499nV
Select
226.1nV
V
V3
VOFF = 12.5
VAMPL = 5
FREQ = 333k
Prog_V
12.50V
R12
1k
Q4
Q2N2222
12.16V
Q5
Q2N2222
R13
1k
11.41V
Q6
Q2N2222
Sim_Load
V
Rl
200
Cl
20pF
Gnd
0
Title
2 Varaiable Small selection option
Size
A
Document Number
Roger Spriggs
Rev
1
Date:
Tuesday, July 03, 2001
Sheet
1
of
1

** Profile: "SCHEMATIC1-Transient" [E:\SCHOOL7\Dups\2asel-schematic1-transient.sim]

Date/Time run: 09/13/11 15:28:39 Temperature: 27.0

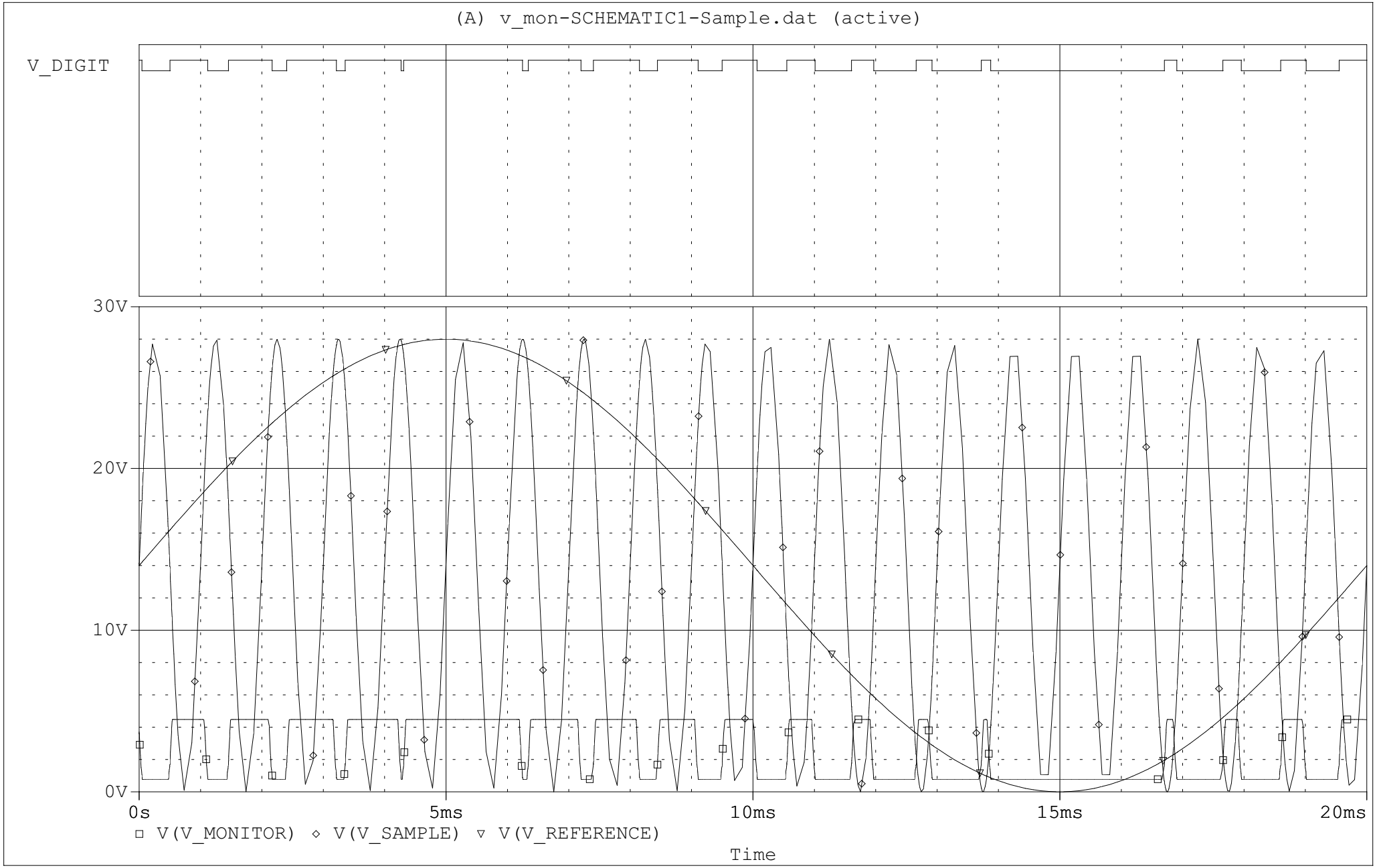


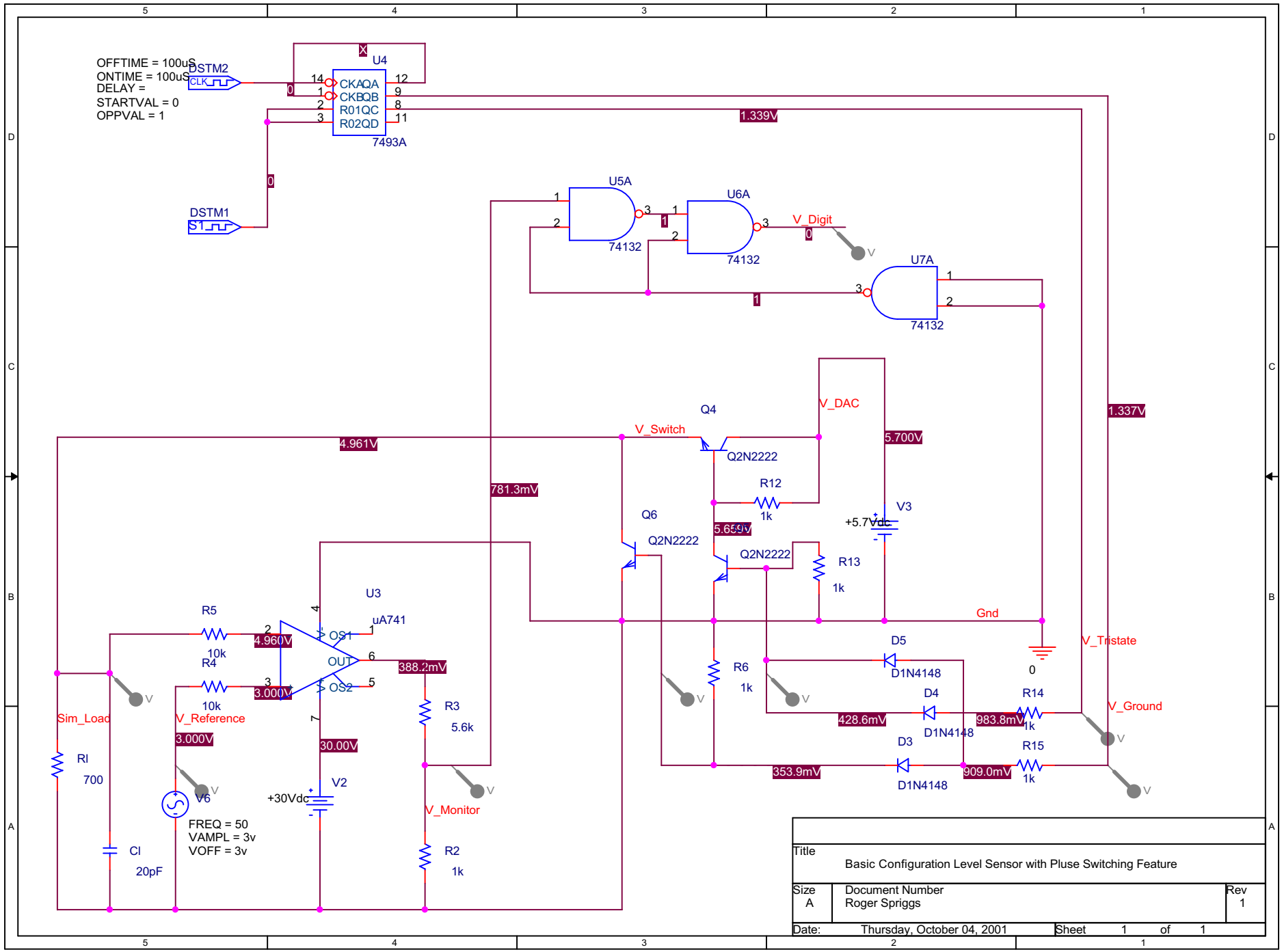


Title	Pin Monitor and Calibrator Circuit	
Size A	Document Number Roger Spriggs	Rev 1
Date: Wednesday, September 26, 2001	Sheet 1 of 1	

** Profile: "SCHEMATIC1-Sample" [E:\SCHOOL7\Dups\v_mon-schematic1-sample.sim]

Date/Time run: 09/13/11 15:40:24 Temperature: 27.0

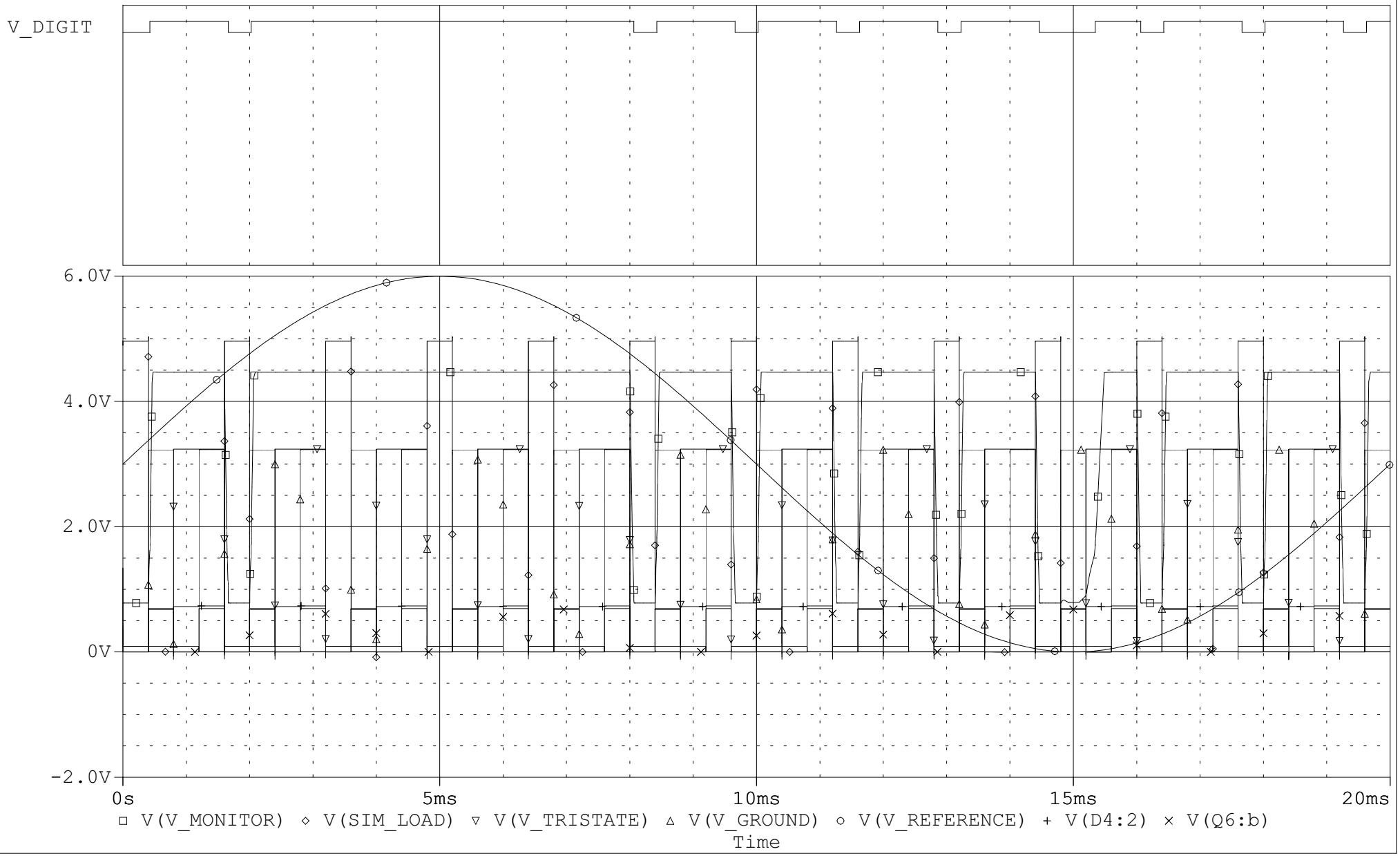


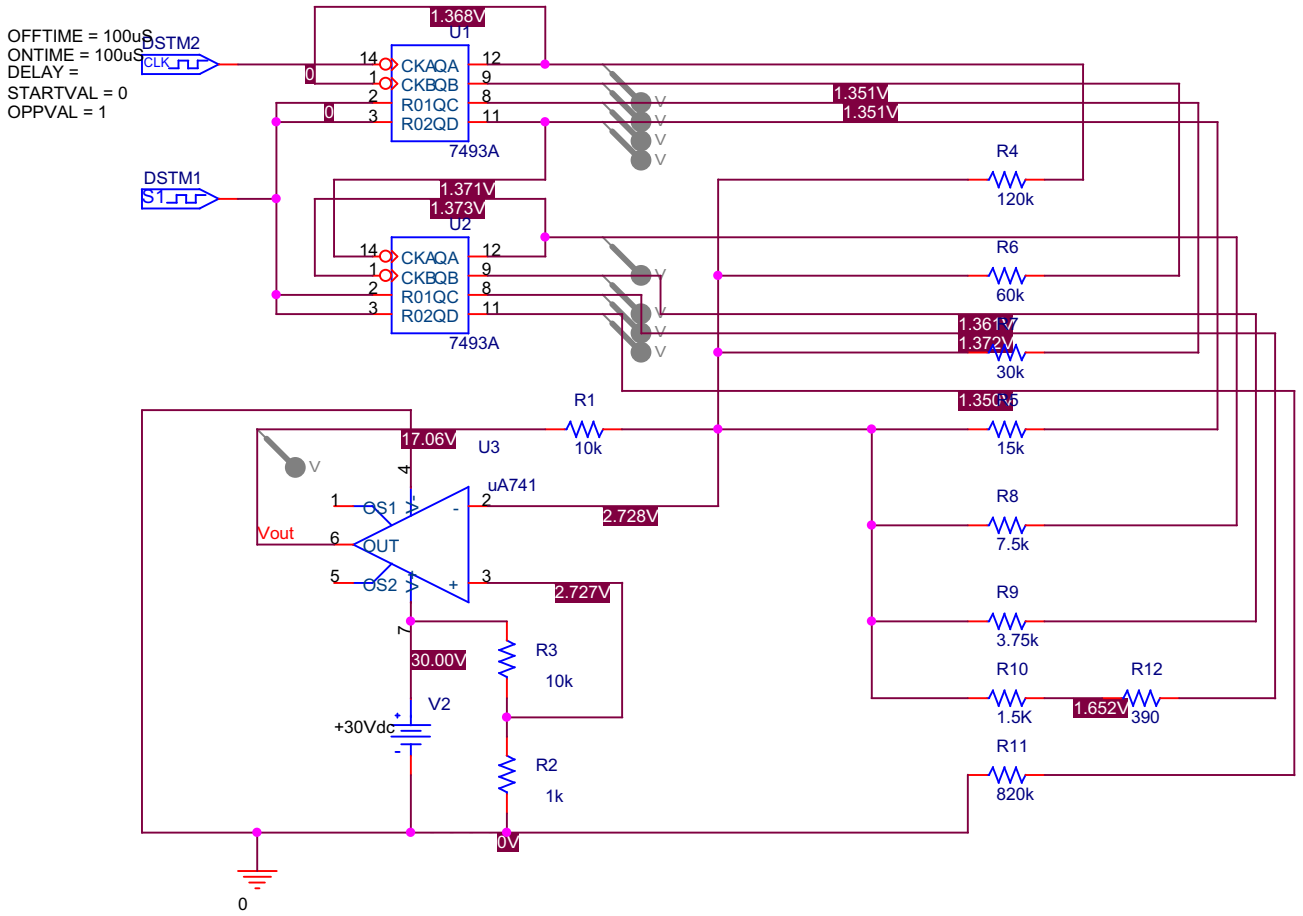

OFFTIME = 100uS
ONTIME = 100uS
DELAY =
STARTVAL = 0
OPPVAL = 1
DSTM2
CLK
DSTM1
ST
U4
7493A
CKAQA
CKBQB
R01QC
R02QD
U5A
U6A
U7A
74132
V_Digit
1.339V
1.337V
4.961V
781.3mV
Q4
Q2N2222
V_Switch
V_DAC
5.700V
R12
1k
V3
+5.7Vdc
Q6
5.659V
R13
Gnd
U3
uA741
R5
10k
R4
4.960V
3.000V
388.2mV
R3
5.6k
R6
D5
D1N4148
D4
D3
428.6mV
353.9mV
983.8mV
909.0mV
R14
R15
V_Tristate
V_Ground
Sim_Load
V_Reference
3.000V
30.00V
V2
+30Vdc
V6
FREQ = 50
VAMPL = 3v
VOFF = 3v
RI
700
CI
20pF
V_Monitor
R2
1k
Title
Basic Configuration Level Sensor with Pluse Switching Feature
Size
A
Document Number
Roger Spriggs
Rev
1
Date:
Thursday, October 04, 2001
Sheet 1 of 1

** Profile: "SCHEMATIC1-Simulate" [E:\SCHOOL7\Dups\bsens-schematic1-simulate.sim]

Date/Time run: 09/13/11 15:46:04 Temperature: 27.0

(A) bsens-SCHEMATIC1-Simulate.dat (active)

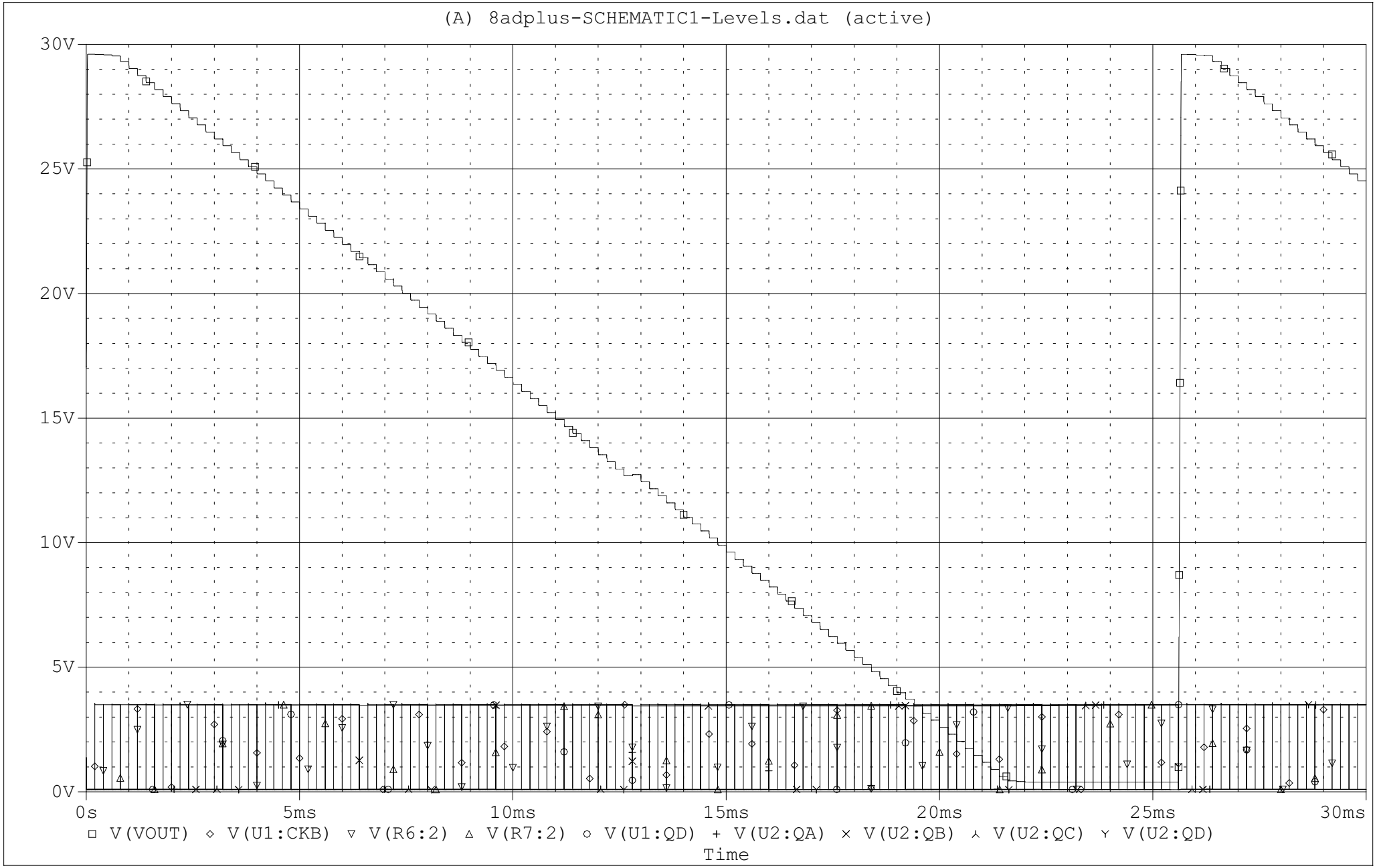


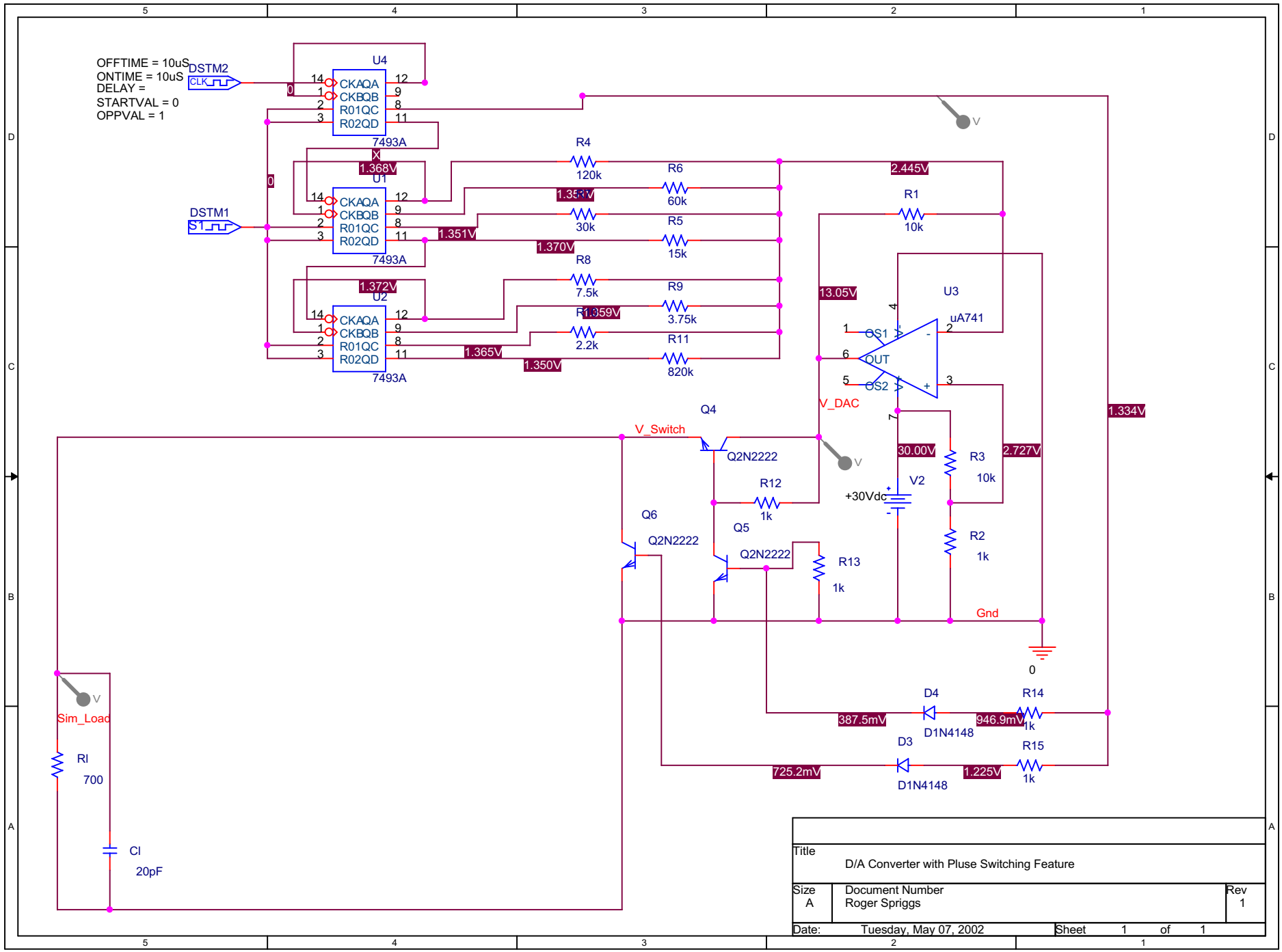


Title	Single Supply 7 Bit DAC	
Size A	Document Number Roger Spriggs	Rev 1
Date: Tuesday, May 07, 2002	Sheet 1 of 1	

** Profile: "SCHEMATIC1-Levels" [E:\SCHOOL7\Dups\8adplus-schematic1-levels.sim]

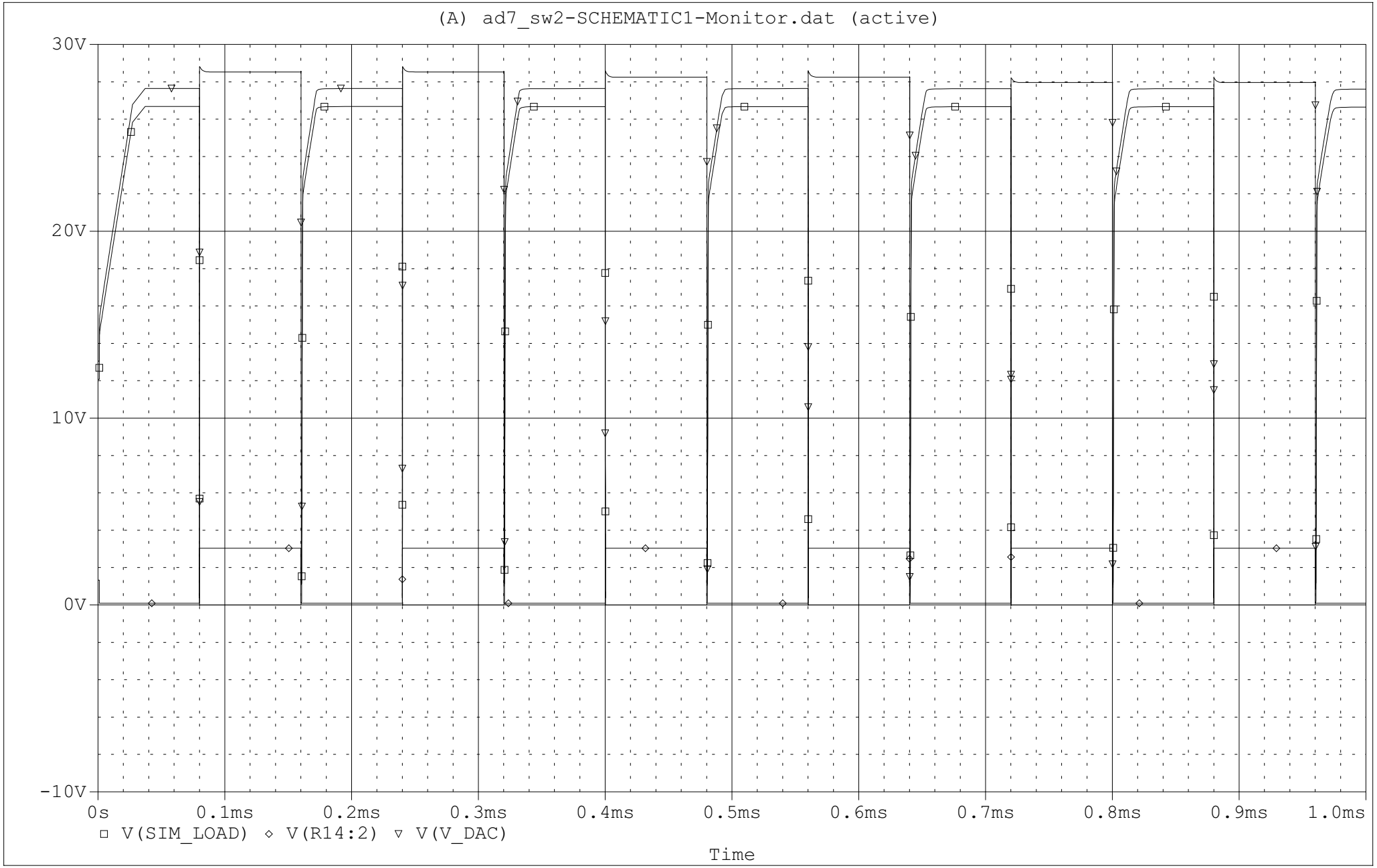
Date/Time run: 09/13/11 15:50:30 Temperature: 27.0

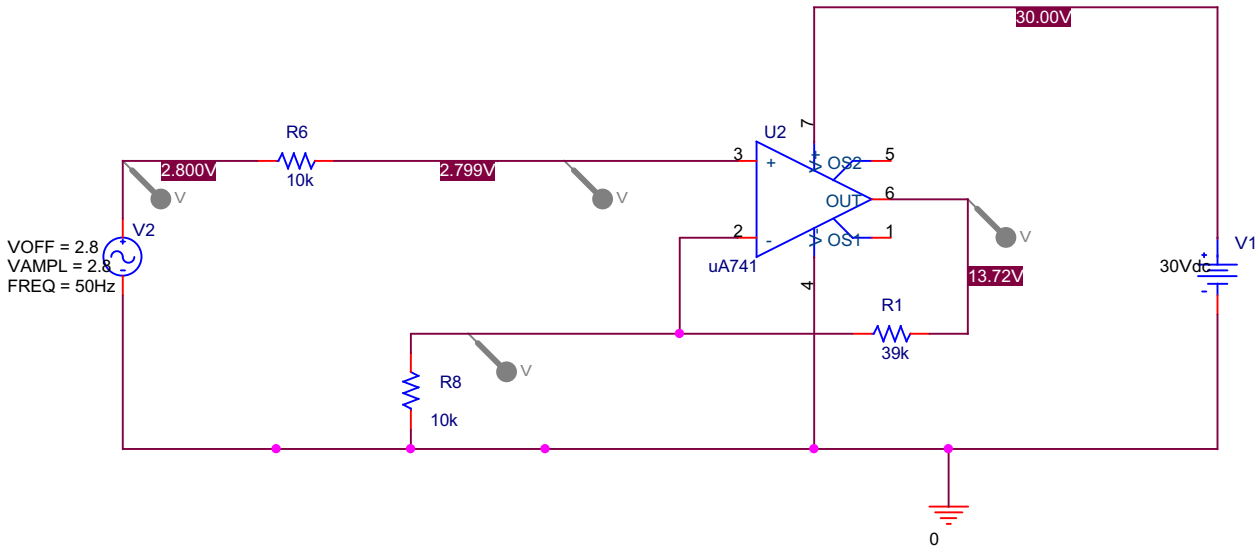



OFFTIME = 10uS
ONTIME = 10uS
DELAY =
STARTVAL = 0
OPPVAL = 1
DSTM2
CLK
DSTM1
ST
U4
U1
U2
7493A
CKAQA
CKBQB
R01QC
R02QD
1.368V
1.351V
1.370V
1.372V
1.365V
1.350V
1.359V
R4
120k
R6
60k
R5
30k
R8
15k
R9
7.5k
R10
3.75k
R11
2.2k
820k
2.445V
R1
10k
13.05V
U3
uA741
OS1
OUT
OS2
V_DAC
1.334V
Q4
V_Switch
Q2N2222
R12
1k
Q6
Q5
R13
30.00V
R3
10k
2.727V
V2
+30Vdc
R2
1k
Gnd
D4
D1N4148
R14
387.5mV
946.9mV
D3
R15
725.2mV
1.225V
Sim_Load
RI
700
CI
20pF
Title
D/A Converter with Pluse Switching Feature
Size
A
Document Number
Roger Spriggs
Rev
1
Date:
Tuesday, May 07, 2002
Sheet 1 of 1

** Profile: "SCHEMATIC1-Monitor" [E:\SCHOOL7\Dups\ad7_sw2-schematic1-monitor.sim]

Date/Time run: 09/13/11 15:55:45 Temperature: 27.0

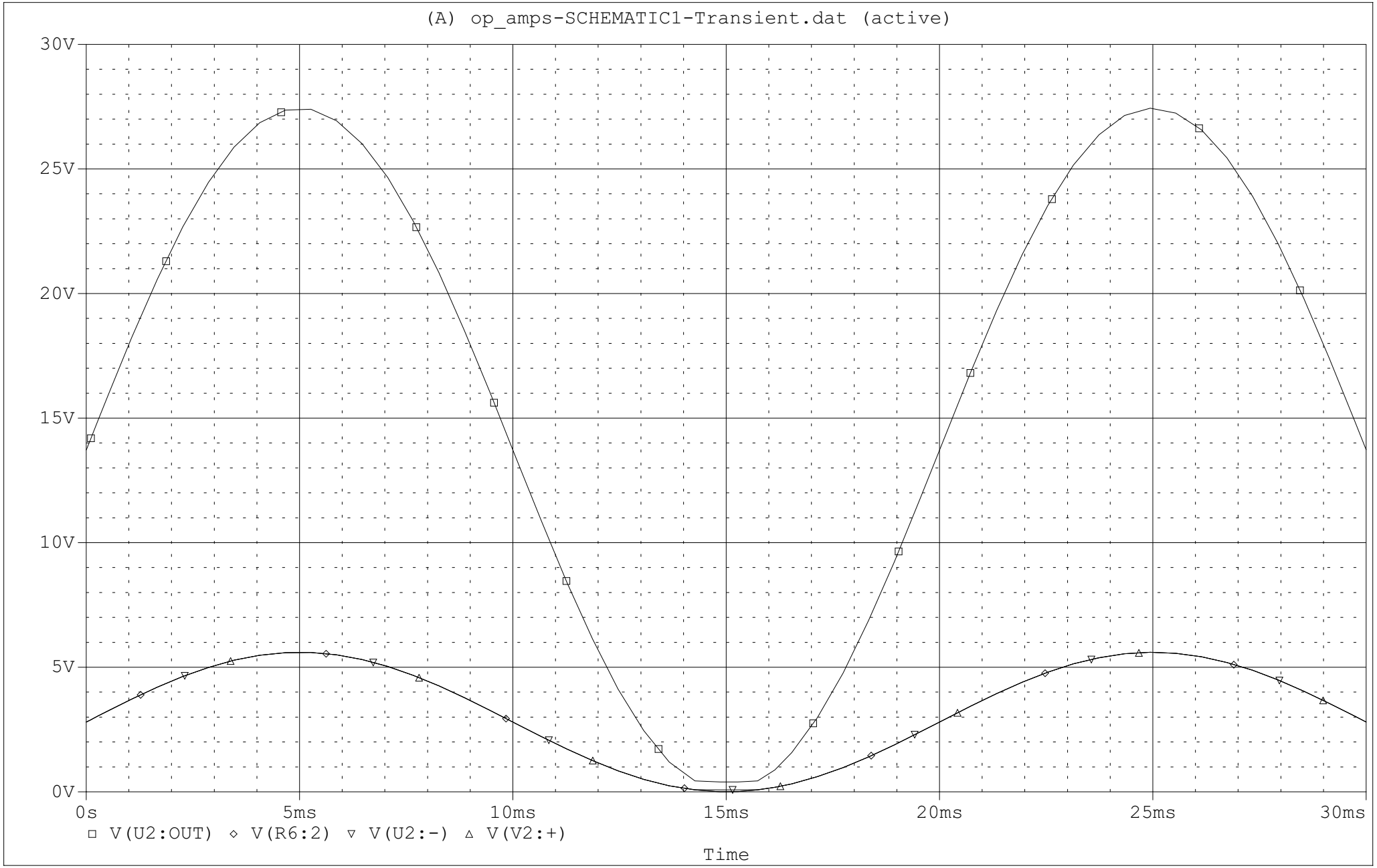




Title	OP Amp Buffer Design for DAC	
Size A	Document Number Roger Spriggs	Rev 1.1
Date: Tuesday, May 07, 2002	Sheet 1 of 1	

** Profile: "SCHEMATIC1-Transient" [E:\SCHOOL7\Dups\op_amps-schematic1-transient.sim]

Date/Time run: 09/13/11 16:00:25 Temperature: 27.0



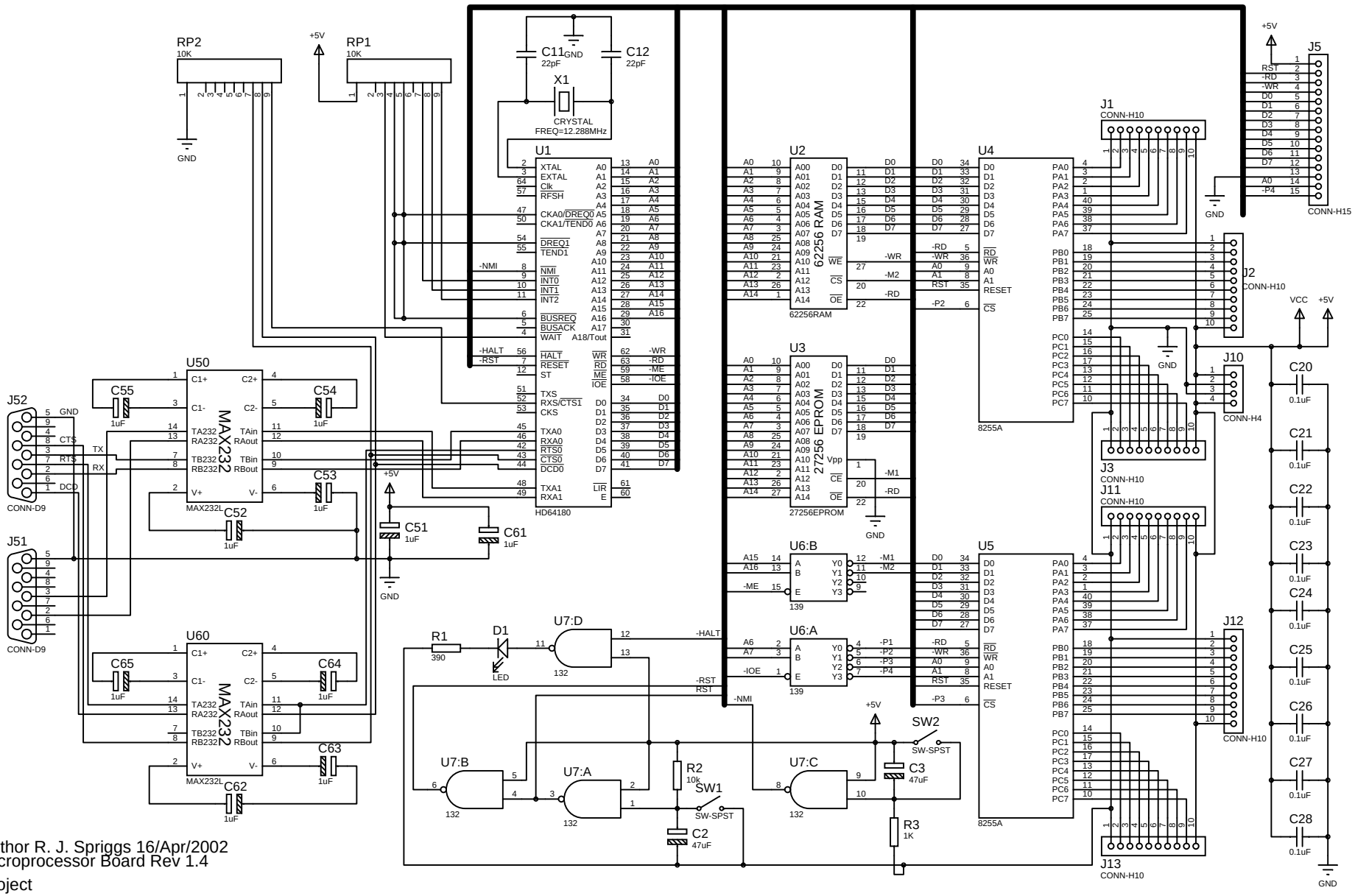
APPENDIX
Sub-Section

This section contains the following items:

Circuit Specifics of the General Purpose Programmer.

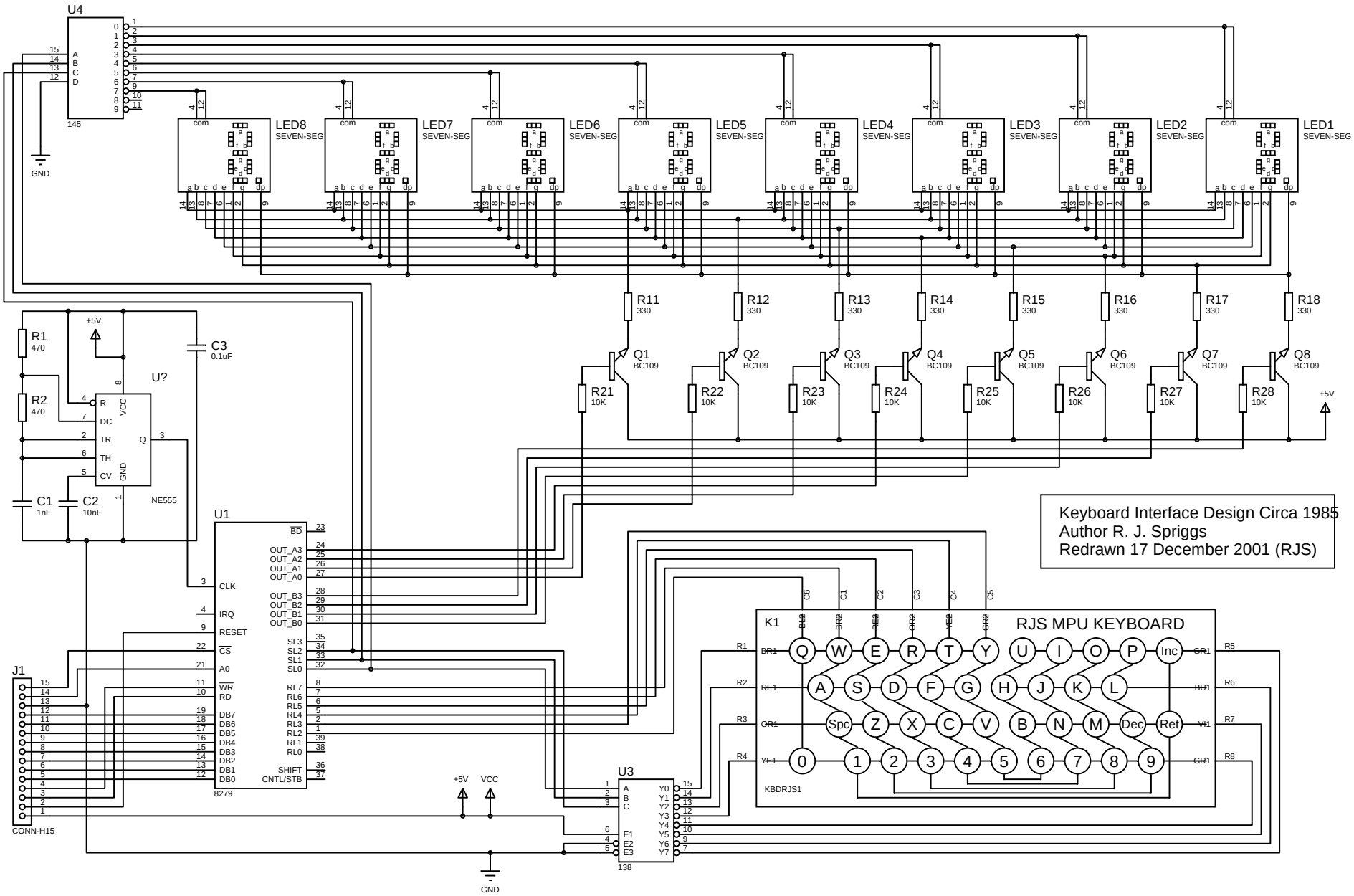
Description :

This section contains Circuit and Layout diagrams or Calculations used for the development of the General Purpose Programmer and consists of the following items :-

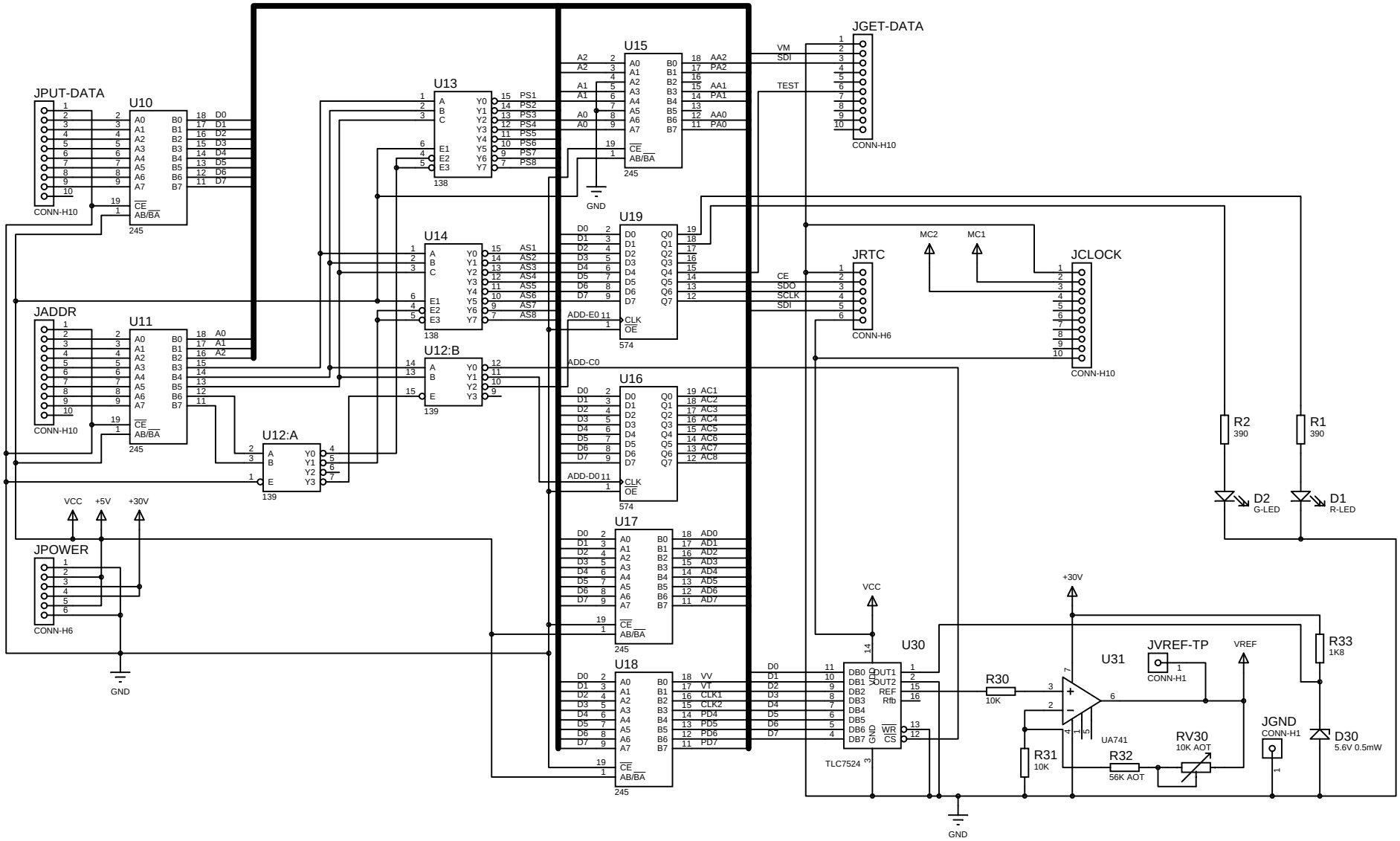


Author R. J. Spriggs 16/Apr/2002
Microprocessor Board Rev 1.4

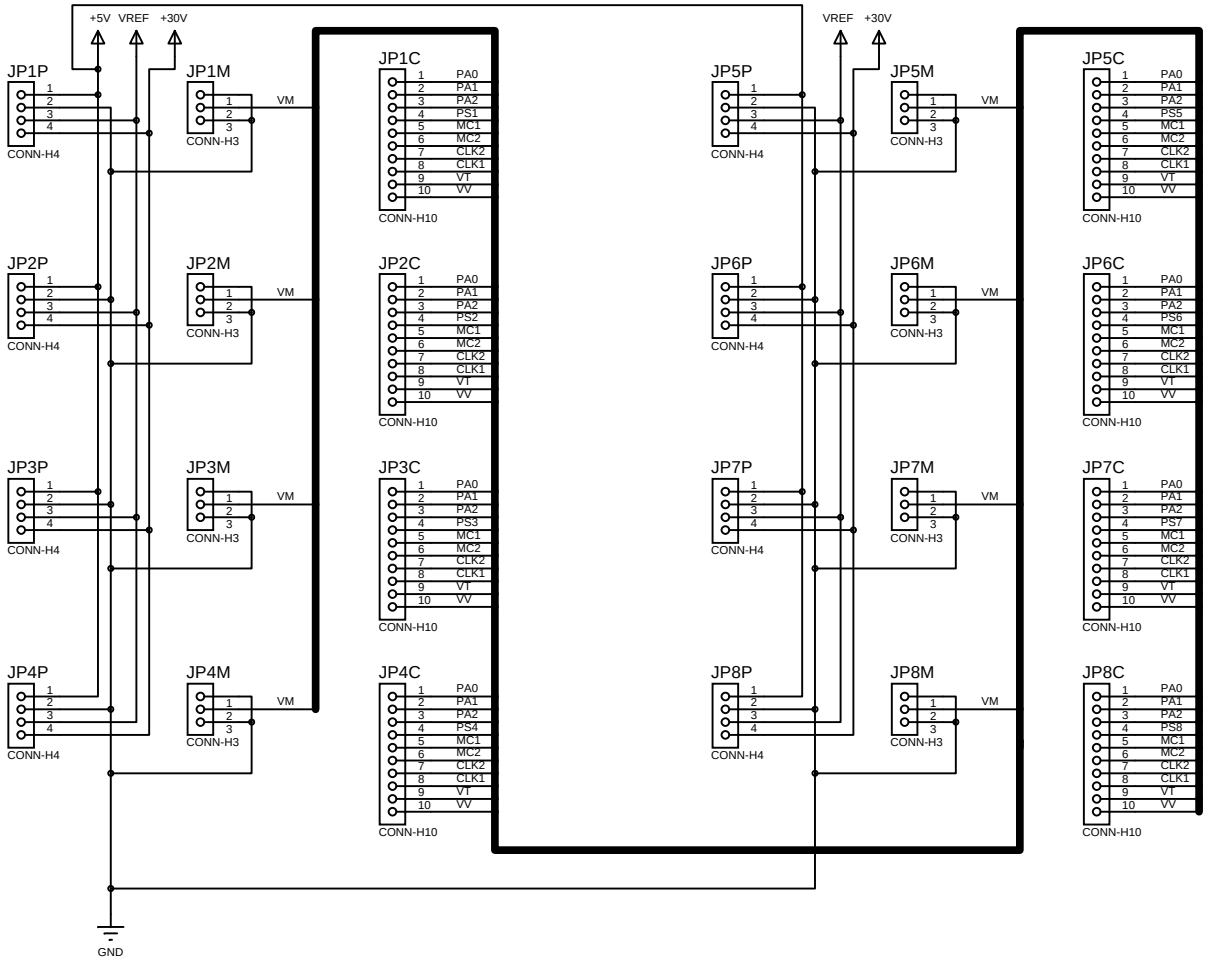
Project
General Purpose Device Programmer


U4
LED8 SEVEN-SEG
LED7 SEVEN-SEG
LED6 SEVEN-SEG
LED5 SEVEN-SEG
LED4 SEVEN-SEG
LED3 SEVEN-SEG
LED2 SEVEN-SEG
LED1 SEVEN-SEG
R11 330
R12 330
R13 330
R14 330
R15 330
R16 330
R17 330
R18 330
Q1 BC109
Q2 BC109
Q3 BC109
Q4 BC109
Q5 BC109
Q6 BC109
Q7 BC109
Q8 BC109
R21 10K
R22 10K
R23 10K
R24 10K
R25 10K
R26 10K
R27 10K
R28 10K
R1 470
R2 470
C3 0.1uF
U?
NE555
C1 1nF
C2 10nF
U1
8279
J1
CONN-H15
U3
138
K1
RJS MPU KEYBOARD
KBDRJS1
Keyboard Interface Design Circa 1985
Author R. J. Spriggs
Redrawn 17 December 2001 (RJS)
+5V
VCC
GND

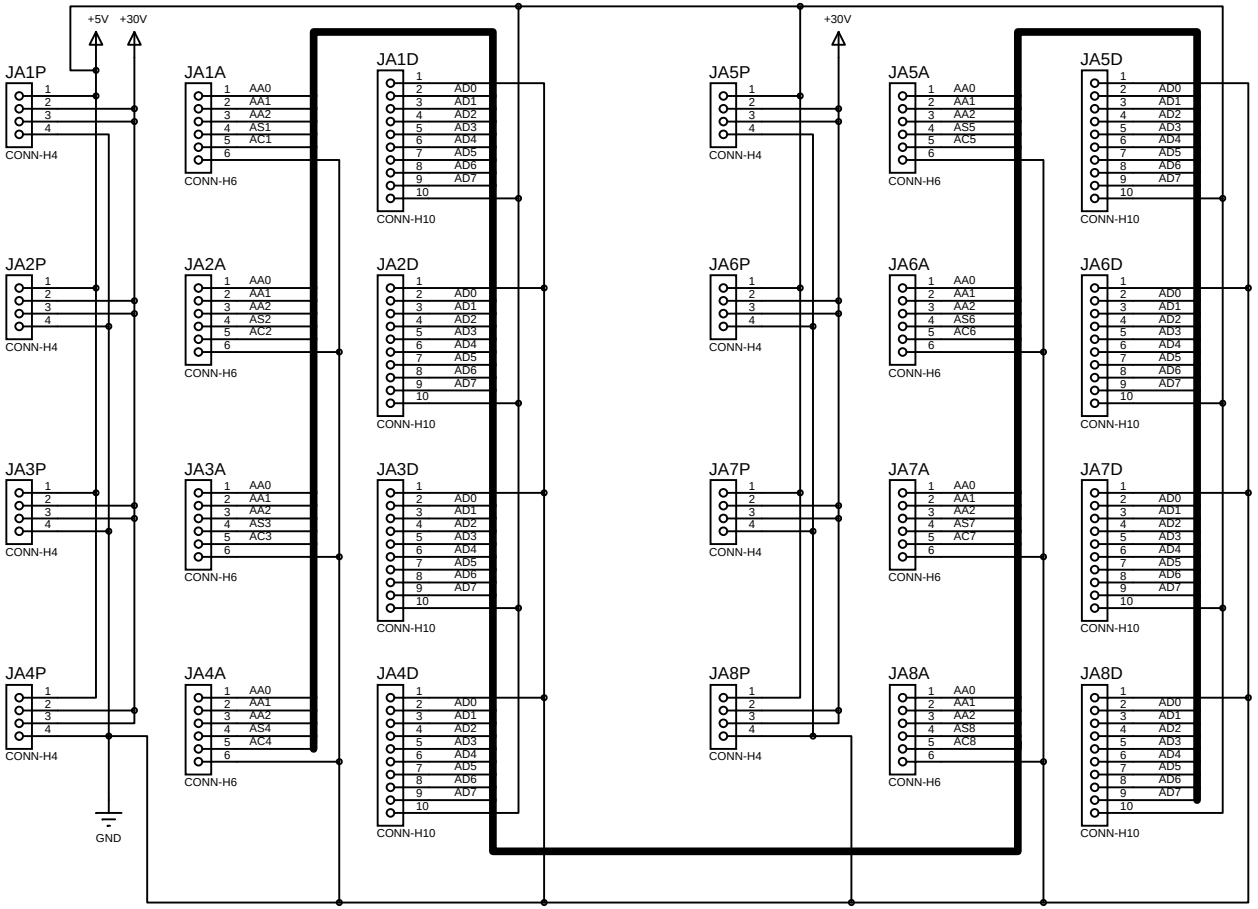
General Programmer Main Interface
Processor Interface and Control Logic
Author R. J. Spriggs 23/Apr/2002 Rev 1.2



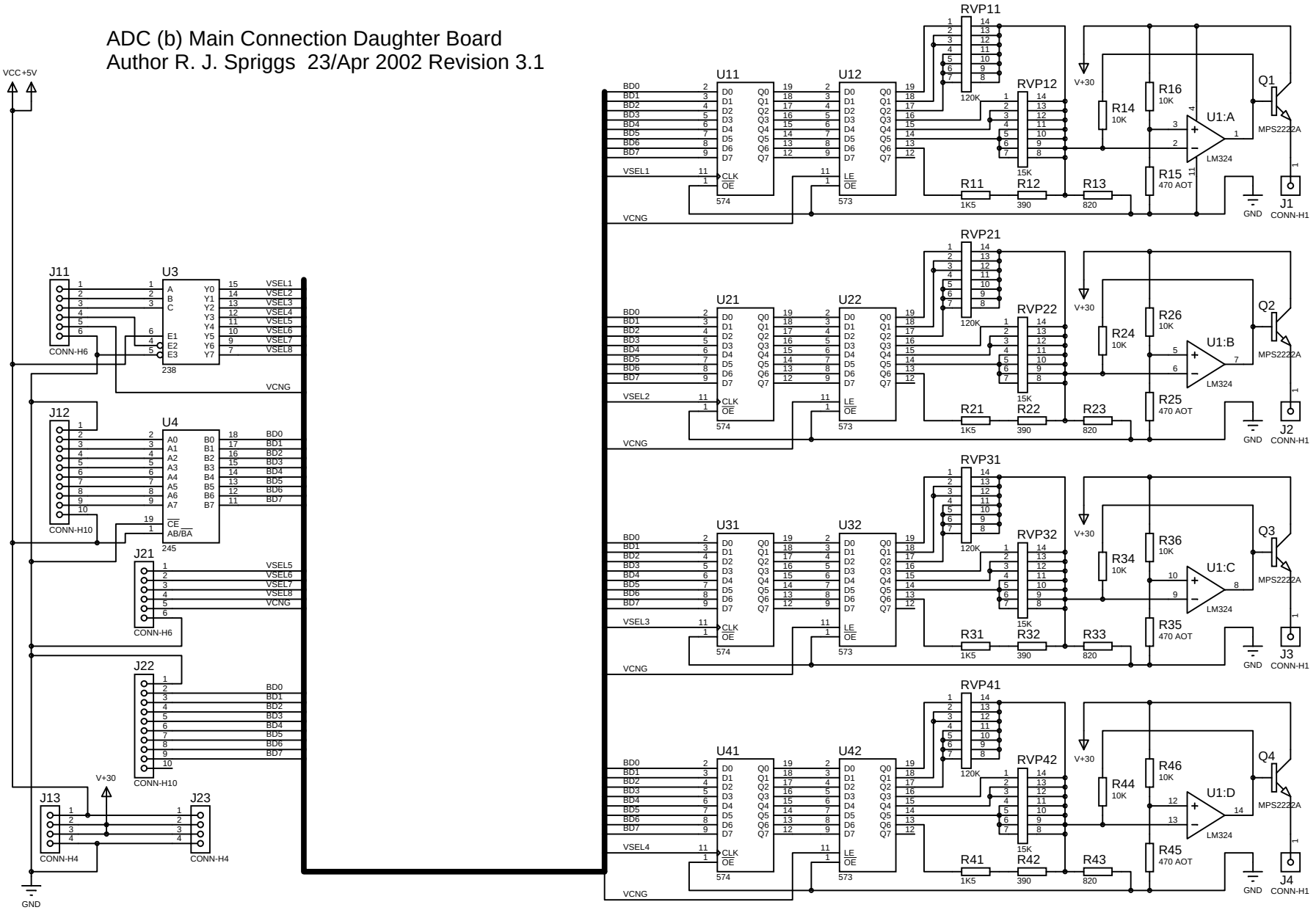
General Programmer Main Interface
Pinboard Connector Interface Section
Author R. J. Spriggs 03/Jan/2002 Revision 1.0



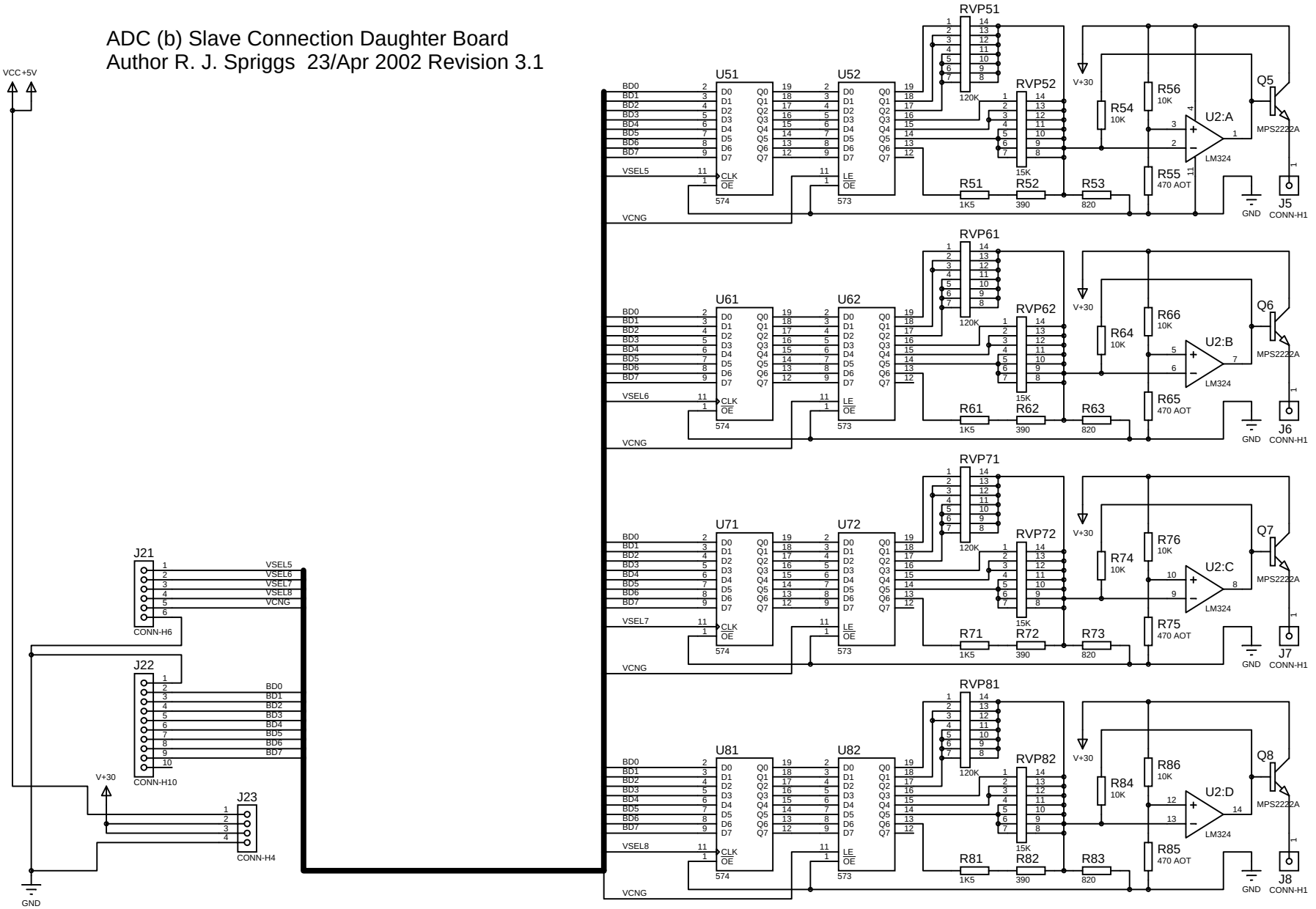
General Programmer Main Interface
ADC Connector Interface Section
Author R. J. Spriggs 02/Jan/2002 Revision 1.0



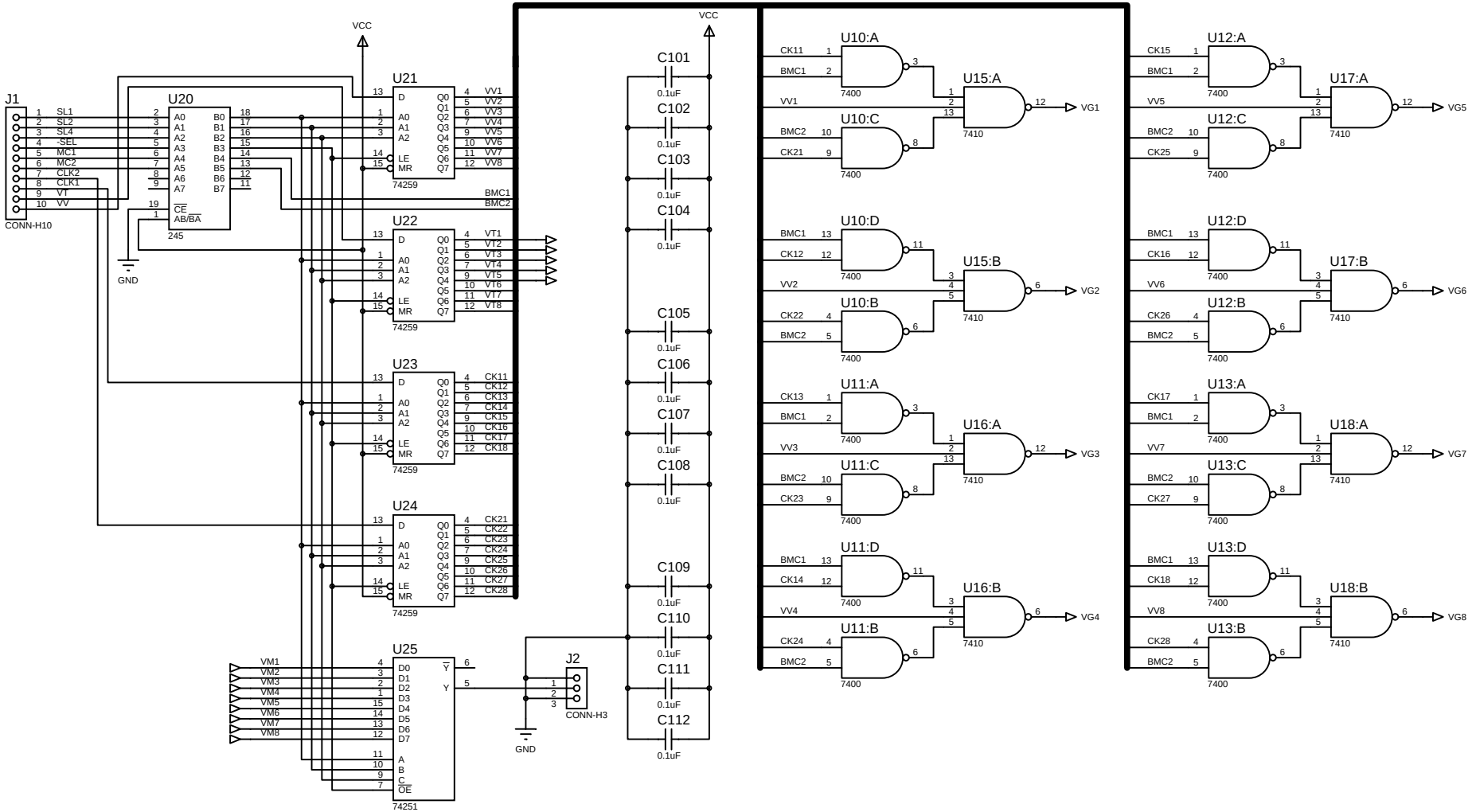
ADC (b) Main Connection Daughter Board
Author R. J. Spriggs 23/Apr 2002 Revision 3.1



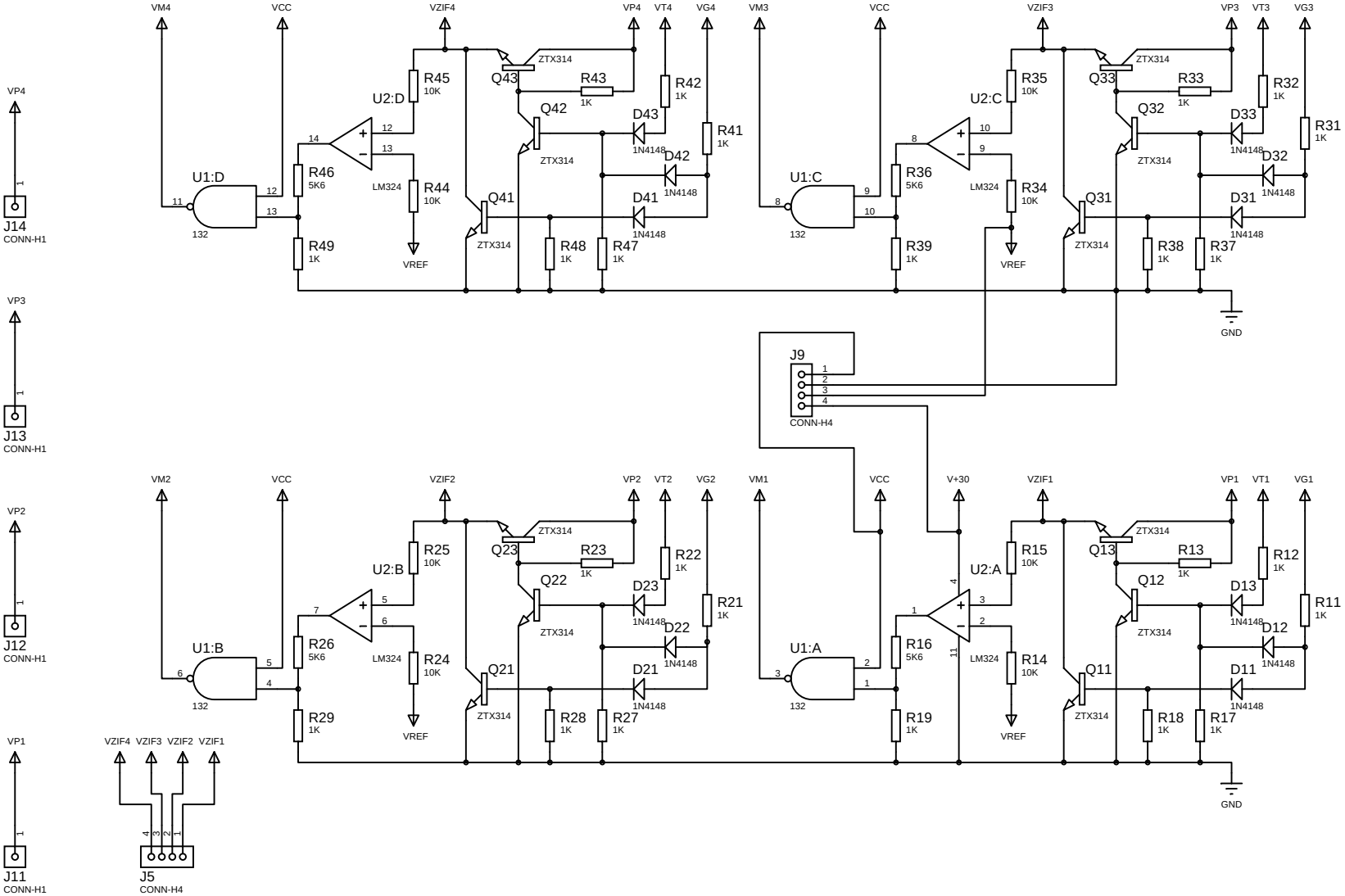
ADC (b) Slave Connection Daughter Board
Author R. J. Spriggs 23/Apr 2002 Revision 3.1



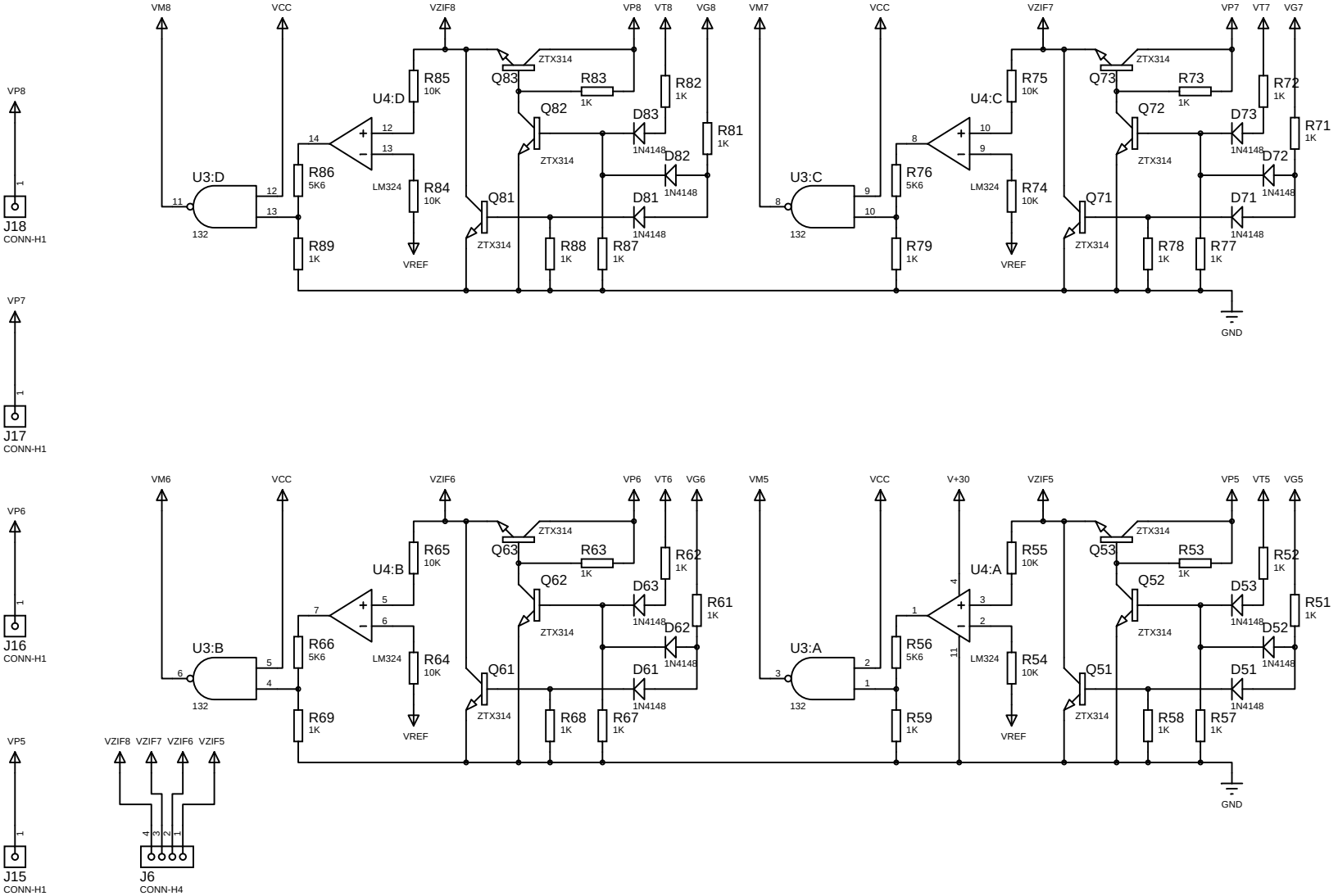
Digital Interface Section
Octal ZIF Pin Basic Control Daughter Board
Author R. J. Spriggs 22/Dec/2001 Rev 2.1



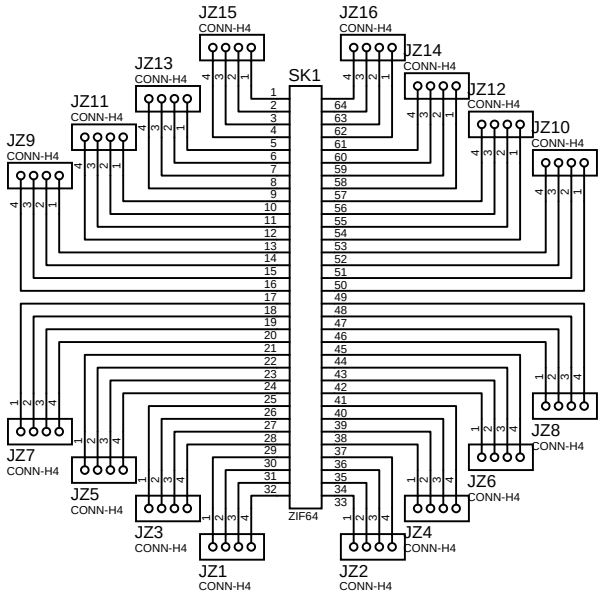
Part One , Quad Section pins 1 to 4
Octal ZIF Pin Basic Control Daughter Board
Author R. J. Spriggs 22/Dec/2001 Rev 2.1



Part Two , Quad Section pins 5 to 8
Octal ZIF Pin Basic Control Daughter Board
Author R. J. Spriggs 22/Dec/2001 Rev 2.1

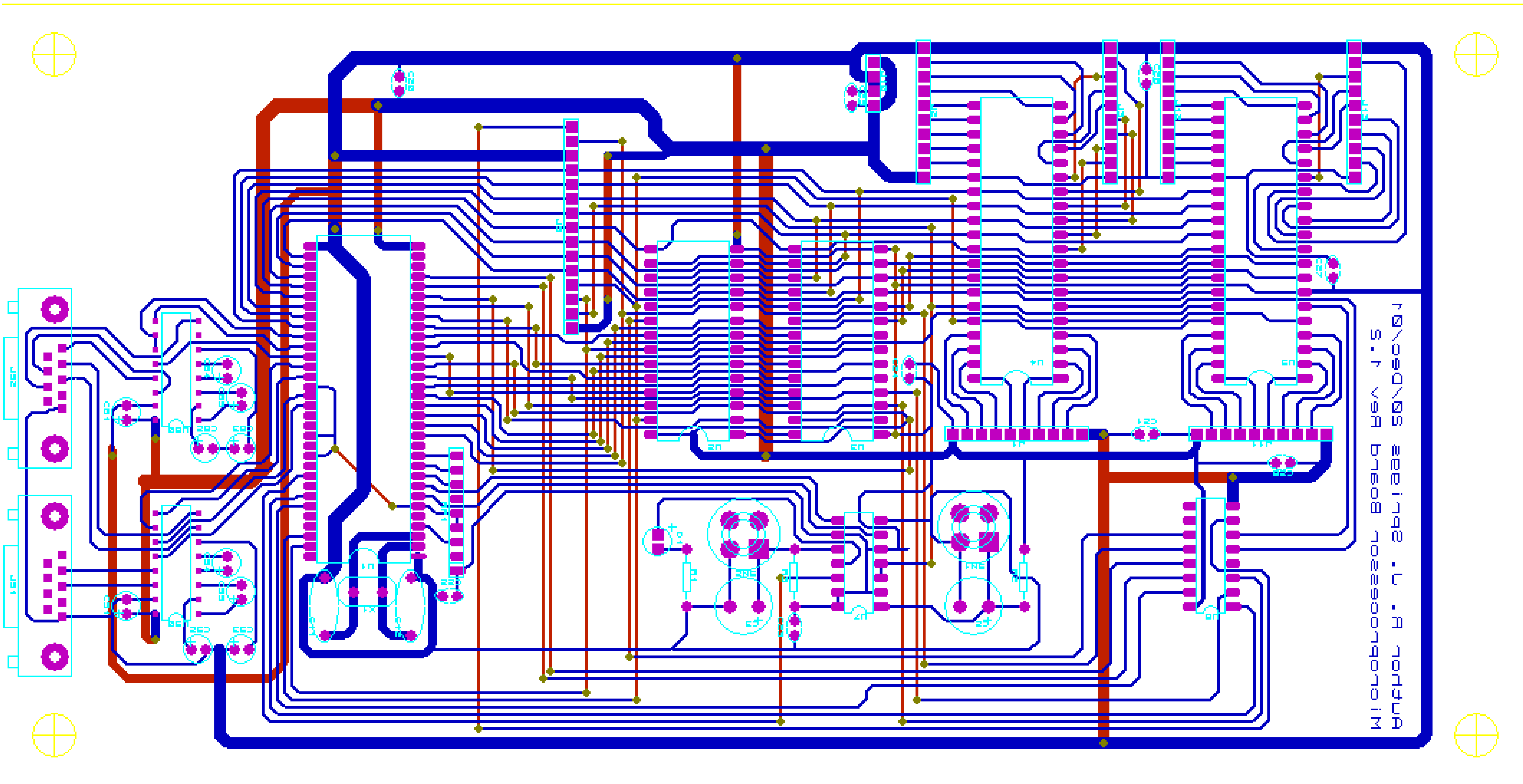


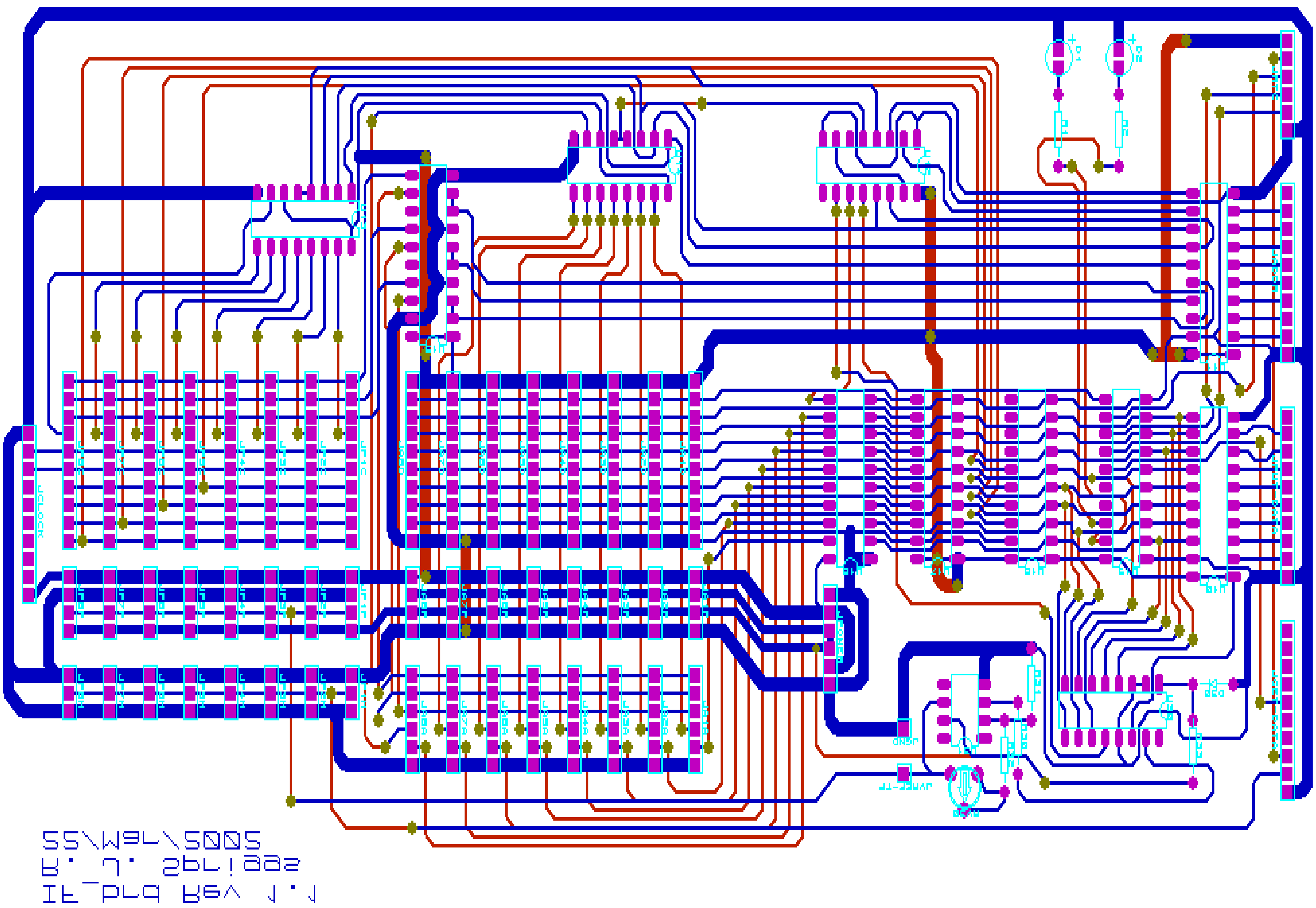
General Programmer Device Interface
ZIF Connector Interface Section
Author R. J. Spriggs 03/Jan/2002 Revision 1.0

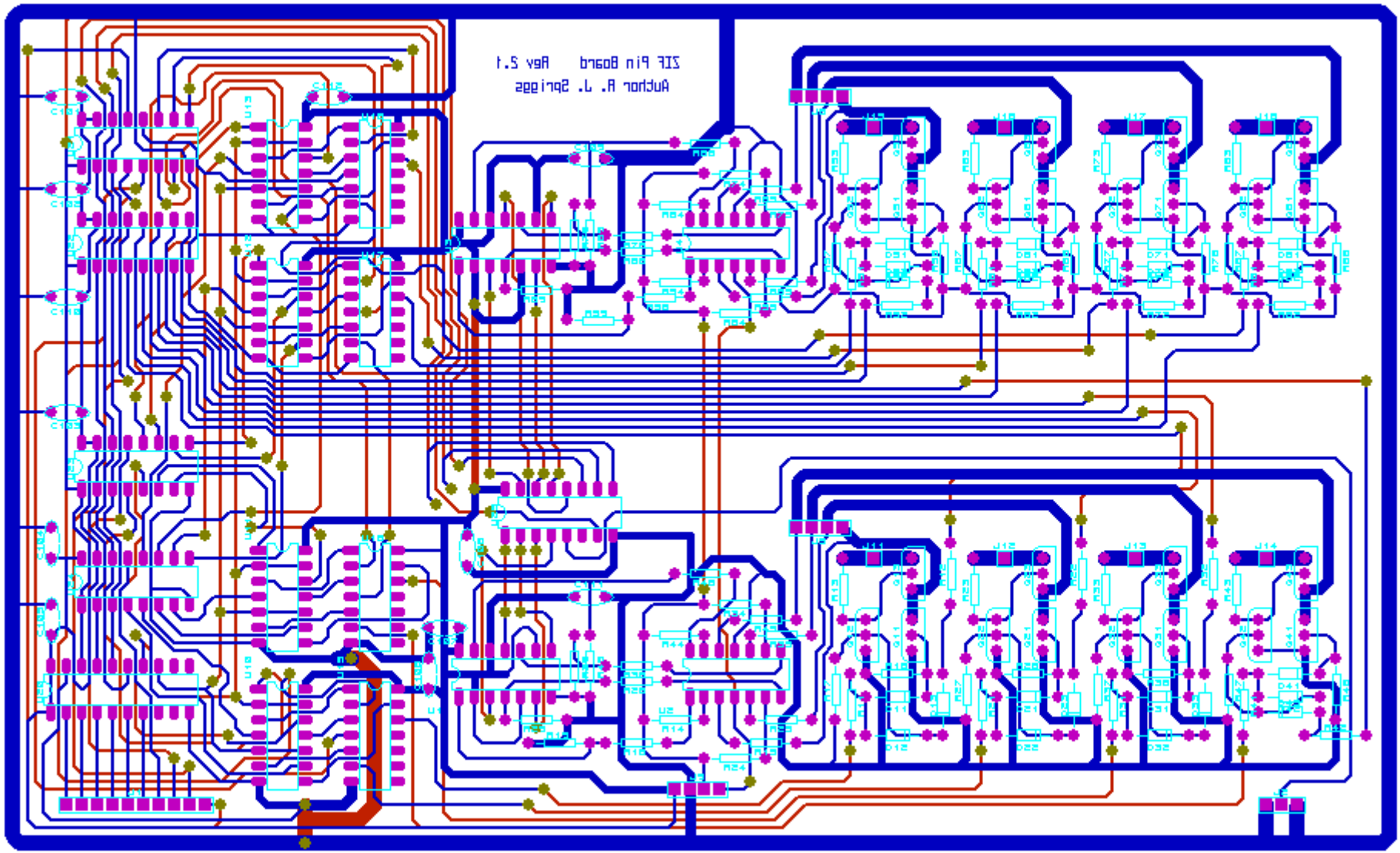


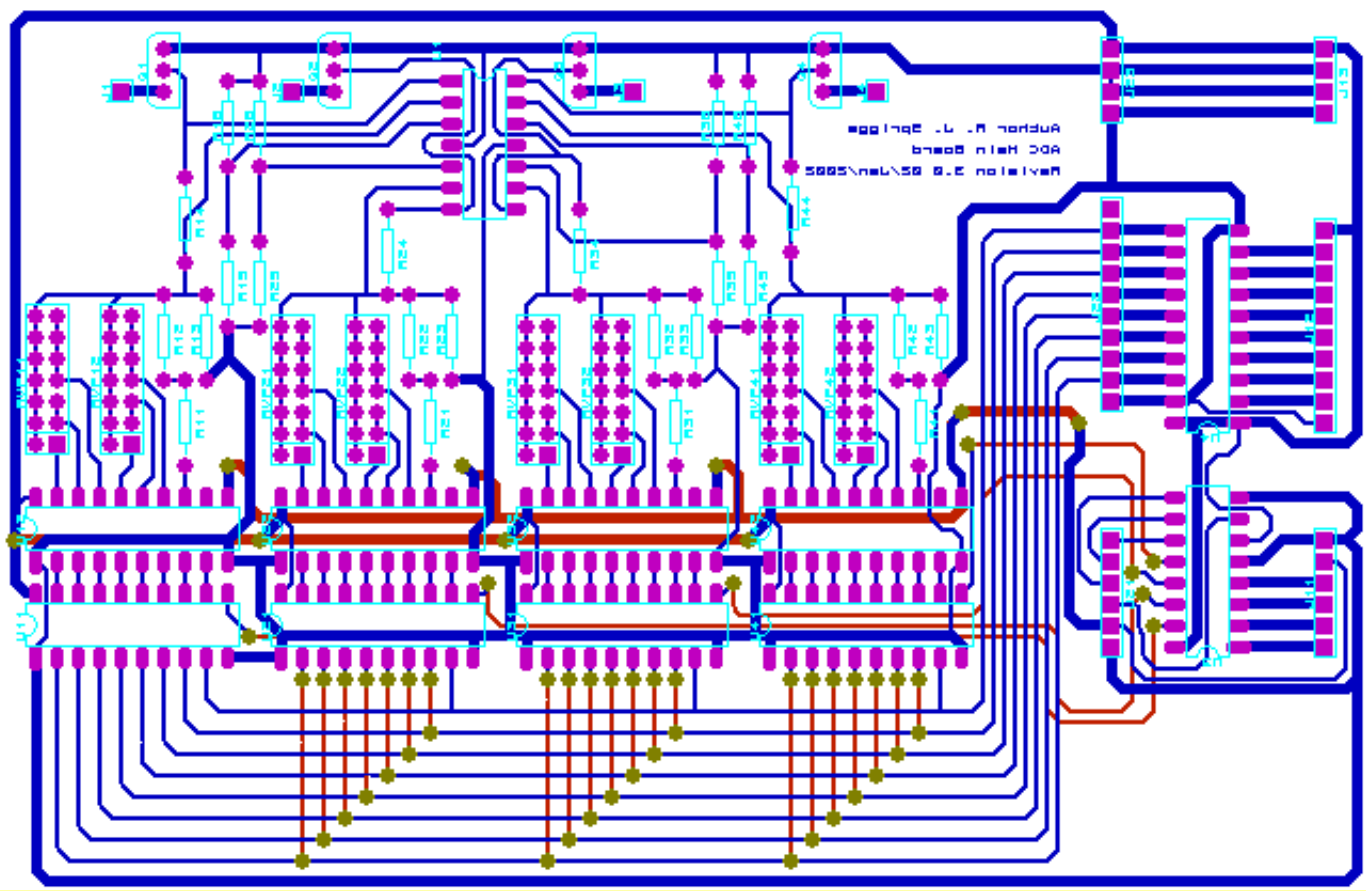
Resistor Division Ratios

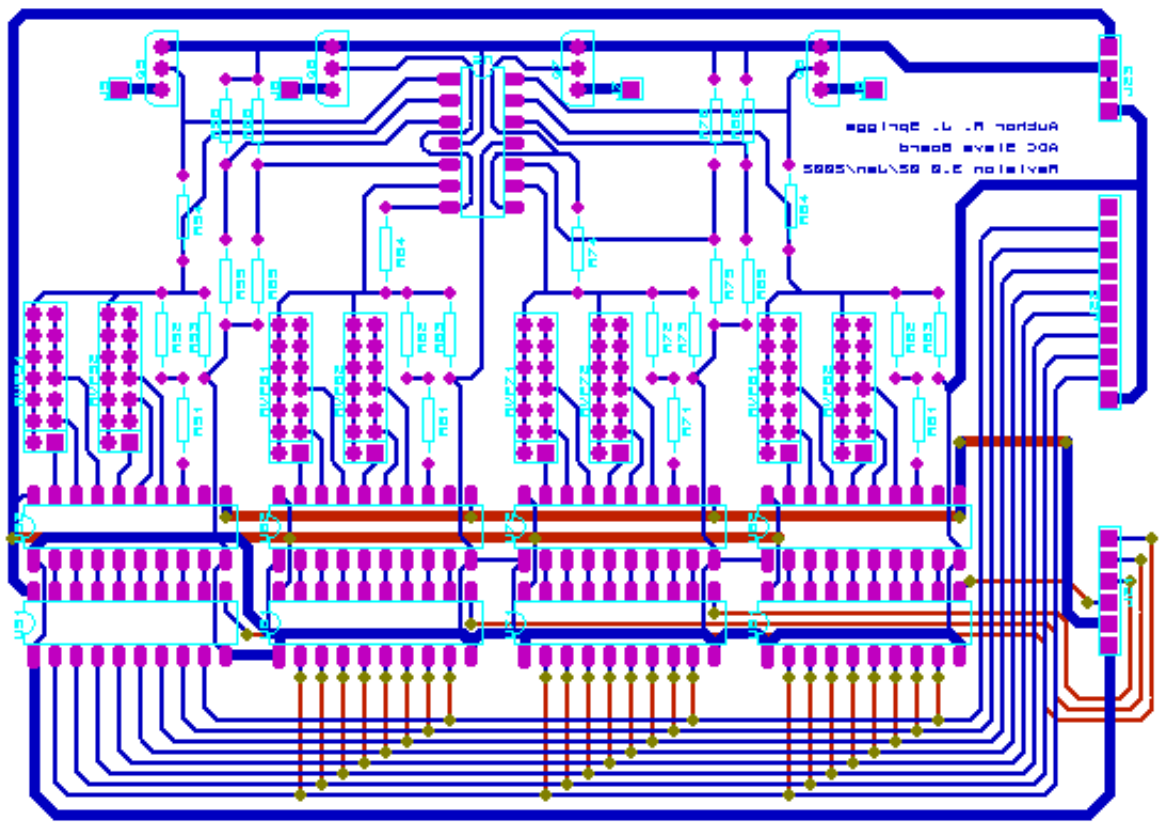
E12	10	12	15	18	22	27	33	39	47	56	68	82	100
Div 2	5	6	7.5	9	11	13.5	16.5	19.5	23.5	28	34	41	
Div 4	2.5	3	3.75	4.5	5.5	6.75	8.25	9.75	11.75	14	17	20.5	
Div 8	1.25	1.5	1.875	2.25	2.75	3.375	4.125	4.875	5.875	7	8.5	10.25	
Scale Error%	4.166667	0	4.166667	2.272727	1.851852	2.272727	5.769231	3.723404	4.910714	2.941176	3.658537	2.5	
E24 Adds	11	13	16	20	24	30	36	43	51	62	75	91	111
Div 2	5.5	6.5	8	10	12	15	18	21.5	25.5	31	37.5	45.5	
Div 4	2.75	3.25	4	5	6	7.5	9	10.75	12.75	15.5	18.75	22.75	
Div 8	1.375	1.625	2	2.5	3	3.75	4.5	5.375	6.375	7.75	9.375	11.375	
Scale Error%	5.769231	1.5625	0	4.166667	0	4.166667	4.651163	5.392157	2.822581	3.333333	3.021978	2.477477	
	Note	The 1.875 Resistor can be built from two preferred Resistors with values of 1500 + 390 = 1K89 giving a close approximation.											
	Note	If any Data Bit drives a Resistor value that is noticeably high then a kink occurs. However this can be removed by											
		ignoring the overlapping values and selecting a preferred subset of the values.											
		i.e. If only 64 values are selected this would still give a resolution of ½ a volt per bit when selecting the											
		best pattens during the calibration process.											
	Note	The Scale Error % compares the division by 8 effect and the next preferred											
		times 10 value from the Resistor Set series of values.											
	Note	The selected Resistors bands would be the 12 and 15 sets giving the values 12, 6, 3, (1.5 or 15, 7.5, 3.75, 1.875).											
		The final circuit values would be 120K, 60K, 30K, 15K, 7K5, 3K75 and 1K89.											
		The last resistor is built from two resistors in series, the other values are created by parallel combinations of the main values.											

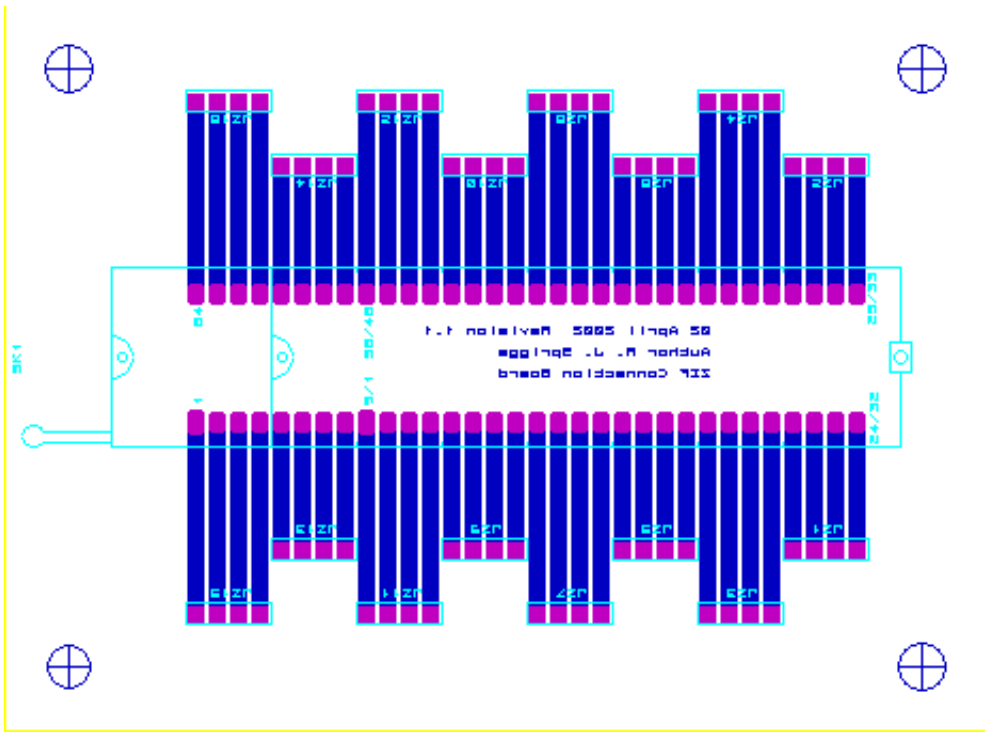

Microprocessor Board Rev 1.2




ZIF Pin Board Rev 5.1
Author R. L. Sprigge






02 April 2005 Revision 1.1
Author M. J. Sprigge
ZIF Connection Board

APPENDIX
Sub-Section

This section contains the following items:

Documentation, Demonstration and Test Programmes.

Description :

This section contains documentation and programmes used by the development system and consists of the following items :-

Title of Pages	Number of Pages
Documentation Files.	
General Purpose Programmer Test Procedures. Filename TESTPROC.DOC	6
Test Programmes.	
The Compiled Test Programme listing BOARDTST.LST	14
Demonstration Programmes.	
Z80TST.ASM a Zilog Z80 programme that uses an 8255 device and compatible with both the General Purpose Simulator and the Z80 based Flight Boards.	3
8031TST.ASM an Intel 8032 processor programme that uses an 8255 device and compatible with both the General Purpose Simulator and the 8032 based Flight Boards.	3
PIC16TST.ASM a MicroChip 16C84 processor programme that just tests out some of the instruction set. Can be run using the General Purpose Simulator.	3

Construction and Test Procedures for the General Purpose Device Programmer.

Common Construction and Board Test Procedures.

1. Visual inspection for shorts between pads ,vias and tracks. (Clear Problems)
2. Meter Check for shorts adjacent pads ,vias and tracks. (Clear Problems)
3. Populate vias, sockets, connectors and any passives.
4. Meter Check for shorts adjacent pads ,vias and tracks. (Clear Problems)
5. Check Power rails for correct distribution. (Resolve problems)
6. Apply power in controlled manner. (Check for and resolve problems)
7. Populate active components (As Appropriate {see Specific test procedures}).
8. Apply power in controlled manner. (Check for, and resolve problems)
9. Power Down and prepare for board specific tests.

Test Programmes for Device programmer.

Code Name	Start Address	Comment and Remarks
ZIF01	1000H	Basic Board Driver & Continuity Test
ZIF02	1200H	Test Functionality of U21,U22,U23,U24
ZIF03	1400H	Full Board test
ADC01	1800H	Basic Board Driver & Continuity Test
ADC02	1900H	Full Board test
IF01	2000H	Bare board checks
IF02	2200H	U12, U13, U14, U15, U17, U18 Checks
IF03	2300H	U16 & U19 Check Operations
IF04	2400H	Up, Stop, Down, Stop Ramps from DAC
IF05	2500H	DAC = Zero/Quarter/Half and Max Volt
IF06	2600H	DAC Single Bit Steps

Construction and Test Procedures for the General Purpose Device Programmer.

The Processor Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.
If EPROM Emulator is being used it will be neccessary to change processor XTAL to 6.144MHz .
Test equipment required is a DVM ,Dual beam oscilliscope and Eprom Emulator.

1. Install U7 and apply +5V to Power board.
2. Verify that U7 Pin 3 = Logic 0 and Pins 6,8 = Logic 1. Press Reset Key verify Pins 3,6 Toggle. Press NMI Key verify Pin 8 Toggles.
3. Using 10K resistor Link Pin 12 to Pin 14 then Pin 12 to Pin 7 verify the Halt LED operates.
4. Power Down and prepare for next test if operations OK.

1. Install U1, U6 and EPROM/Emulator in U3 then apply +5V to Power board.
2. Run Test programme Z80TST01
3. Verify that after Pressing Reset Key the Halt lamp comes on after about 2 seconds.
4. Power Down and prepare for next test if operations OK.

1. Install U4 and EPROM/Emulator in U3 then apply +5V to Power board.
2. Run Test programme Z80TST02
3. Verify that counting patten appears on J1, J2 and J3.
4. Power Down and prepare for next test if operations OK.

1. Apply +5V to Power board. Run Test programme HEX
2. Verify that Full EPROM checks OK by attaching LED board to Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Install U3, apply +5V to Power board, Run Test programme Z80TST03
2. Verify that Full RAM checks OK by attaching LED board to Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Attach Keyboard/Display to J5 apply +5V to Power board, Run Test programme Z80TST04
2. Verify that Display checks OK, Press KEYS and monitor code appear on Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Attach Serial Line Monitor set to 9600 Baud, 8 Bits, 1 Stop Bit, No Parity to either J51 or J52 apply +5V to Power board, Run Test programme Z80TST06 in Output mode.
2. Verify that Monitor Display shows lines of “ABCDEFGHIJKL”.
3. Run Test programme Z80TST06 in Duplex Mode verify input echo mode operational.
4. Power Down and prepare for next test if operations OK.
5. All Test complete, Board now ready for service.

Construction and Test Procedures for the General Purpose Device Programmer.

Octal ZIF Pin Basic Control Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect ZIF Board to Processor Board using (Test lead 10/2*5 Port B LS-5 , Port C MS-5)
2. Run Test programme ZIF01 in Image BOARDTST.
3. Verify that counting patten appears on U20-Pins 2,3,4,5,6,7 and pulses are also present on Pin,13 on U21, U22, U23 and U24.
4. Power Down and prepare for next test if operations OK.

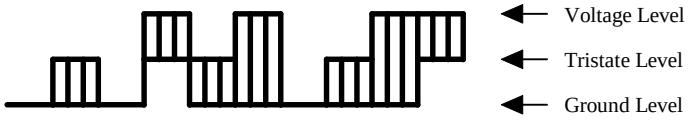
1. Install U20 and apply +5V to Power board.
2. Run Test programme ZIF01 in Image BOARDTST.
3. Verify that counting patten appears on U20-Pins 2,3,4,5,6,7 + 18,17,16,15,14,13 and pulses are also present on Pin,13 on U21, U22, U23 and U24.
4. Power Down and prepare for next test if operations OK.

1. Install U21 , U22 , U23 , U24 and apply +5V to Power board.
2. Run Test programme ZIF02 in Image BOARDTST.
3. Verify that walking 1's patten appears on U21,U22,U23,U24-Pins 4,5,6,7,8,9,10,11,12.
4. Power Down and prepare for next test if operations OK.

1. Install U10 , U11 , U12 , U13 and apply +5V to Power board.
2. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U10,U11,U12,U13-Pins 3,6,8,11.
4. Power Down and prepare for next test if operations OK.

1. Install U15 , U16 , U17 , U18 and apply +5V to Power board.
2. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U15,U16,U17,U18-Pins 6,12.
4. Power Down and prepare for next test if operations OK.

1. Apply Voltage to (Programme Voltage) VP pins and apply +5V to Power board.
2. Attach ZIF pin Conditioner to ZIF Pins. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that ZIF pins give a similar style presentation to those shown in diagram.
4. Power Down and prepare for next test if operations OK.



1. Install U1,U2,U3,U4 and apply +5V,+25V to Power board.
2. Set Vref to +3V.Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U2,U4-Pins 1,7,8,14.
4. Verify that Negative going pulse group appear on U1,U3-Pins 3,6,8,11.
5. Power Down and prepare for next test if operations OK.
6. All Test complete, Board now ready for service.

Construction and Test Procedures for the General Purpose Device Programmer.

ADC Main Daughter Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect ADC Board to Processor Board using (Test leads Port B 10/10 {Note the Special Cable Interceptor must be used to remove +VCC from Pin 10} & Port C 10/6)
2. Run Test programme ADC01 in Image BOARDTST.(**All** ICs but U3,U4 must be removed)
3. Verify that counting patten appears on U4-Pins 2,3,4,5,6,7,8,9 and U3-Pins 1,2,3,4 and a pulse appears on U12,U13,U14,U15 Pin 11.
4. Power Down and prepare for next test if operations OK.

1. Install U3 , U4 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that counting patten appears on {U4-Pins 2,3,4,5,6,7,8,9 + 18,17,16,15,14,13,12,11} {U11, U21, U31 & U41-Pins 2,3,4,5,6,7,8,9 } also U3-Pins 1,2,3,4 and a walking patten on. U3-Pins 7,9,10,11,12,13,14,15.
4. Power Down and prepare for next test if operations OK.

1. Install U11, U21, U31, U41 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that count patten appears on {U11, U21, U31, U41-Pins 19,18,17,16,15,14,13,12} {U12, U22, U32 & U42-Pins 2,3,4,5,6,7,8,9 }.
4. Power Down and prepare for next test if operations OK.

1. Install U12, U22, U32, U42 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that count patten appears on {U12, U22, U32, U42-Pins 19,18,17,16,15,14,13,12}.
4. Power Down and prepare for next test if operations OK.

1. Install U1 and apply +5V and +25V to Power board.
2. Run Test programme ADC02 in Image BOARDTST.
3. Verify that analogue Ramp patten appears on U1-Pins 1,7,8,14 ,13,12. and on Connectors J1, J2, J3, and J4.
4. Power Down and prepare for next test if operations OK.
5. All Test complete, Board now ready for service.

Construction and Test Procedures for the General Purpose Device Programmer.

Interface Connection Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect power to board +5V and +30 Verify power appears on correct IC Pins.
2. Verify voltage across D30 is 5.6 Volts. Using a 470Ω resistor use it to link U19,20 and U19,19 RED LED should switch on also U19,20 and U19,18 and GREEN LED should switch on.
3. Power Down and prepare for next test if operations OK.

1. Connect IF Board to Processor Board using (Normal leads 3 * 10/10 PORT_A=ADDR , PORT_B=PUT_DATA and PORT_C = GET_DATA)
2. Run Test programme IF01 in Image BOARDTST.(Only run this test if **All** ICs are removed)
3. Verify that counting patten appears on U10 & U11-Pins 2,3,4,5,6,7,8,9 and pulses appears on U19 Pin 15 also JRTC Pin 2 and on Pin 2 on all JP⊗M Connectors (⊗ is 1 to 8).

1. Install U10, U11 and apply +5V to Power board via JPOWER.
2. Run Test programme IF02 in Image BOARDTST.
3. Verify that count patten appears on {U10 & U11-Pins 18,17,16,15,14,13,12,11 } {U13 & U14-Pins 1,2,3 } {U12-Pins 2,3,13,14 } {U16, U17, U18 & U19-Pins 2,3,4,5,6,7,8,9 } {U15-Pins 2,3,5,6,8,9 } {U30-Pins 4,5,6,7,8,,9,10,11 }.
4. Power Down and prepare for next test if operations OK.

1. Install U12, U13, U14, U15, U17, U18 and apply +5V to Power board via JPOWER.
2. Run Test programme IF02 in Image BOARDTST.
3. Verify that count patten appears on {U17 & U18-Pins 18,17,16,15,14,13,12,11 } and pulses appears on {U15-Pins 18,17,15,14,12,11} {U13 & U14-Pins 7,9,10,11,12,13,14,15} {U12-Pins 4,5,7,10,11,12 } {U30-Pin 12 }.
4. Power Down and prepare for next test if operations OK.

1. Install U16, U19 and apply +5V to Power board via JPOWER.
2. Run Test programme IF03 in Image BOARDTST.
3. Verify that count patten appears on {U16 & U19-Pins 19,18,17,16,15,14,13,12 }.
4. Power Down and prepare for next test if operations OK.

1. Install U30, U31, Set RV30 to Mid Travel and apply +5V & +30V to Power board via JPOWER.
2. Check Voltage across D30 is 5.6Volts.
3. Run Test programme IF04 in Image BOARDTST.
4. Verify that a Ramp waveform appears on Test point JVREF-TP. Adjust RV30 till Maximum height ramp equal 25.5 Volts and Minimum Voltage is typically less that 0.2 Volts. Change R32 for more appropriate value if adjustment is at end of travel of RV30.
5. Run Test programme IF05 in Image BOARDTST if using DVM to calibrate.
6. Power Down and prepare for next test if operations OK.
7. All Test complete, Board now ready for service.

Construction and Test Procedures for the General Purpose Device Programmer.

Important Pre Instalation Instructions.

If external clock module is **NOT** going to be installed prior to use then wire wrap JCLOCK pins 1,2 and 3 together. Failure to complete this activity WILL make programming algorithms invalid.

Document revision status.

Initial Preparation.	12th/Feb/2002
ZIF03 Test programme update.	15th/Feb/2002
Include actual used Processor Test Procedure.	17th/Feb/2002
Include Additional Tests for OCTAL ZIF Board.	18th/Feb/2002
Include Initial ADC Board Tests.	18th/Feb/2002
Corrections to ADC Test procedure	1st/Mar/2002
Include Initial Interface Board Tests.	9th/Apr/2002
Include Table of Test Programme Start Address	10th/Apr/2002
Procedure Corrections for ADC boards testing.	11th/Apr/2002
Updates to Processor Serial Test programmes.	23th/Apr/2002
Updated to Interface Board Checks	25th/Apr/2002
.	..th/.../2002
.	..th/.../2002

```
*  Printer is FILE.TMP  File Being Processed is BOARDTST.LST                    On 06-May-2002   At 16:32:37       *

      1    Programme Title is BOARDTST.ASM GENERAL BOARD TEST CODE PROGRAMMES
      2
      3                                          1           .Title BOARDTST.ASM General Board Test Code Programmes
      4                                          2           .SBTTL  AUTHOR R. J. SPRIGGS    283 Belle Vue Road Bournemouth
      5                                          3           .SBTTL  Initial Preparation     12/Feb/2002     Last Update 11/04/02
      6                                          4
      7                                          5   ;
      8                                          6   ;       General Note
      9                                          7   ;       These programmes are started via ZMON which is linked to this
     10                                          8   ;       application.
     11                                          9   ;
     12                                         10   ;       To start a specific application use a Monitor GOTO .ORG address
     13                                         11   ;       of section you wish to process.
     14                                         12   ;
     15                                         13
     16                                         14           .ASSEMBLER Z80+Extras
     17                                         15
     18                                         16   ;           ZIF BOARD Application Settings and Information.
     19                                         17   ;           ==============================================
     20                                         18   ;
     21                                         19   ;       Connect ZIF Board to processor board using a 10/2*5 cable to the
     22                                         20   ;       Primary parallel ports B and C (Port B = LS 5 Bits Port C = MS bits)
     23                                         21   ;
     24                                         22   ;       BIT allocation
     25                                         23   ;
     26                                         24   ;       Port B      Name        Port Bit    ZIF_BRD     Port B
     27                                         25   ;                   Ground                              Pin 1
     28                                         26   ;                   SL1         Bit 0       Pin 1       Pin 2
     29                                         27   ;                   SL2         Bit 1       Pin 2       Pin 3
     30                                         28   ;                   SL4         Bit 2       Pin 3       Pin 4
     31                                         29   ;                   -SEL        Bit 3       Pin 4       Pin 5
     32                                         30   ;                   MC1         Bit 4       Pin 5       Pin 6
     33                                         31   ;                   TBA         Bit 5                   Pin 7
     34                                         32   ;                   TBA         Bit 6                   Pin 8
     35                                         33   ;                   TBA         Bit 7                   Pin 9
     36                                         34   ;                   VCC                                 Pin 10
     37                                         35   ;
     38                                         36   ;       Port C      Name        Port Bit    ZIF_BRD     Port C
     39                                         37   ;                   Ground                              Pin 1
     40                                         38   ;                   MC2         Bit 0       Pin 6       Pin 2
     41                                         39   ;                   CLK1        Bit 1       Pin 7       Pin 3
     42                                         40   ;                   CLK2        Bit 2       Pin 8       Pin 4
     43                                         41   ;                   VT          Bit 3       Pin 9       Pin 5
     44                                         42   ;                   VV          Bit 4       Pin 10      Pin 6
     45                                         43   ;                   TBA         Bit 5                   Pin 7
     46                                         44   ;                   TBA         Bit 6                   Pin 8
     47                                         45   ;                   TBA         Bit 7                   Pin 9
     48                                         46   ;                   VCC                                 Pin 10
     49                                         47
     50                                         48
     51                                         49   ;           ADC Main  BOARD Application Settings and Information.
     52                                         50   ;           ================================================
     53                                         51   ;
     54                                         52   ;       Connect ADC Main  Board to processor board using a 10/10 cable
     55                                         53   ;       to the Primary parallel ports B and J12.
     56                                         54   ;       Connect ADC Main  Board to processor board using a 10/6 cable
     57                                         55   ;       to the Primary parallel ports C and J11.
     58                                         56   ;
     59                                         57   ;       BIT allocation
     60                                         58   ;
     61                                         59   ;       Port B      Name        Port Bit    ADC         Port C
     62                                         60   ;                   Ground                  Pin 1       Pin 1
     63                                         61   ;                   Data0       Bit 0       Pin 2       Pin 2
     64                                         62   ;                   Data1       Bit 1       Pin 3       Pin 3
     65                                         63   ;                   Data2       Bit 2       Pin 4       Pin 4
     66                                         64   ;                   Data3       Bit 3       Pin 5       Pin 5
     67                                         65   ;                   Data4       Bit 4       Pin 6       Pin 6
     68                                         66   ;                   Data5       Bit 5       Pin 7       Pin 7
     69                                         67   ;                   Data6       Bit 6       Pin 8       Pin 8
     70                                         68   ;                   Data7       Bit 7       Pin 9       Pin 9
     71                                         69   ;                   VCC                     N/C         Pin 10
     72                                         70   ;
     73                                         71   ;       Port C      Name        Port Bit    ACD         Port C
     74                                         72   ;                   Ground                  Pin 6       Pin 1
     75                                         73   ;                   Chan1       Bit 0       Pin 1       Pin 2
     76                                         74   ;                   Chan2       Bit 1       Pin 2       Pin 3
     77                                         75   ;                   Chan4       Bit 2       Pin 3       Pin 4
     78                                         76   ;                   -Enable     Bit 3       Pin 4       Pin 5
     79                                         77   ;                   Change      Bit 4       Pin 5       Pin 6
     80                                         78   ;                   TBA         Bit 5                   Pin 7
     81                                         79   ;                   TBA         Bit 6                   Pin 8
     82                                         80   ;                   TBA         Bit 7                   Pin 9
     83                                         81   ;                   VCC                     N/C         Pin 10
     84                                         82   ;
     85                                         83
     86                                         84
     87                                         85   ;           InterFace Main  BOARD Application Settings and Information.
     88                                         86   ;           ==========================================================
     89                                         87   ;
     90                                         88   ;       Connect IF Main  Board to processor board using a 3*10/10 cable
     91                                         89   ;       to the Primary parallel Port A and JADDR , Ports B and JPUT-DATA
     92                                         90   ;       and Port C and JGET-DATA.
     93                                         91   ;
     94                                         92   ;       BIT allocation
     95                                         93   ;
     96                                         94   ;       Port A                                      (Normally Output)
     97                                         95   ;       Port A      Name        Port Bit    IF_BRD      Port A
     98                                         96   ;                   Ground                  Pin 1       Pin 1
     99                                         97   ;                   Address0    Bit 0       Pin 2       Pin 2
    100                                         98   ;                   Address1    Bit 1       Pin 3       Pin 3
    101                                         99   ;                   Address2    Bit 2       Pin 4       Pin 4
    102                                        100   ;                   Address3    Bit 3       Pin 5       Pin 5
    103                                        101   ;                   Address4    Bit 4       Pin 6       Pin 6
    104                                        102   ;                   Address5    Bit 5       Pin 7       Pin 7
    105                                        103   ;                   Address6    Bit 6       Pin 8       Pin 8
    106                                        104   ;                   Address7    Bit 7       Pin 9       Pin 9
```

```
                                                ;                   VCC                     N/C         Pin 10
                                                ;
                                                ;       Port B                                          (Normally Output)
                                                ;       Port B      Name        Port Bit    IF_BRD      Port B
                                                ;                   Ground                  Pin 1       Pin 1
                                                ;                   Data0       Bit 0       Pin 2       Pin 2
                                                ;                   Data1       Bit 1       Pin 3       Pin 3
                                                ;                   Data2       Bit 2       Pin 4       Pin 4
                                                ;                   Data3       Bit 3       Pin 5       Pin 5
                                                ;                   Data4       Bit 4       Pin 6       Pin 6
                                                ;                   Data5       Bit 5       Pin 7       Pin 7
                                                ;                   Data6       Bit 6       Pin 8       Pin 8
                                                ;                   Data7       Bit 7       Pin 9       Pin 9
                                                ;                   VCC                     N/C         Pin 10
                                                ;
                                                ;       Port C                                          (Normally Input)
                                                ;       Port B      Name        Port Bit    IF_BRD      Port C
                                                ;                   Ground                  Pin 1       Pin 1
                                                ;                   VM          Bit 0       Pin 2       Pin 2
                                                ;                   SCI         Bit 1       Pin 3       Pin 3
                                                ;                   TBA         Bit 2       Pin 4       Pin 4
                                                ;                   TBA         Bit 3       Pin 5       Pin 5
                                                ;                   Test Line   Bit 4       Pin 6       Pin 6
                                                ;                   TBA         Bit 5       Pin 7       Pin 7
                                                ;                   TBA         Bit 6       Pin 8       Pin 8
                                                ;                   TBA         Bit 7       Pin 9       Pin 9
                                                ;                   VCC                     N/C         Pin 10
                                                ;
                                                ;
                                                ;
                                                ;
                                                ;              PORT DEFINITIONS SECTION
                                                ;
                                                ;
                                                PORTA  = 40H            ;A few Variant PORT assignment Methods
                                                PORTB  = 41H
                                                PORTC  = PORTB + 1
                                                CONTRL = PORTA + 3
                                                ;
                                                ;       ============================================================
                                                ;       $X = 00000000%          ; Preset Definition Word
                                                ;       ============================================================
                                                ;       $X = $X + 10000000%     ; Control Word is a  BIT Set  Definition
                                                ;       $X = $X + 10000000%     ; Control Word is an I/O Mode Definition
                                                ;       ============================================================
                                                ;       $X = $X + 00000000%     ; Group 0      is Mode 0
                                                ;       $X = $X + 00100000%     ; Group 0      is Mode 1
                                                ;       $X = $X + 01000000%     ; Group 0      is Mode 2
                                                ;       ============================================================
                                                ;       $X = $X + 00000000%     ; Group 1      is Mode 0
                                                ;       $X = $X + 00000001%     ; Group 1      is Mode 1
                                                ;       ============================================================
                                                ;       $X = $X + 00010000%     ; Port A       is Input
                                                ;       $X = $X + 00000000%     ; Port A       is Output
                                                ;       ============================================================
                                                ;       $X = $X + 00010000%     ; Port A       is Input
                                                ;       $X = $X + 00000000%     ; Port A       is Output
                                                ;       ============================================================
                                                ;       $X = $X + 00000010%     ; Port B       is Input
                                                ;       $X = $X + 00000000%     ; Port B       is Output
                                                ;       ============================================================
                                                ;       $X = $X + 00000001%     ; Port C Low   is Input
                                                ;       $X = $X + 00000000%     ; Port C Low   is Output
                                                ;       ============================================================
                                                ;       $X = $X + 00001000%     ; Port C High  is Input
                                                ;       $X = $X + 00000000%     ; Port C High  is Output
                                                ;       ============================================================
                                                
                                                
                                                
                                                        .PAGE
                                                        .SBTTL  .       Application ZIF01 SECTION
                                                
                        AA=1000                 .ORG 1000
                                                
                                                ZIF01:
1000    3E 80                                   LD A,#$X                ;PIO Control Port Settings Assignment
1002    D3 43                                   out (Contrl),A          ;Configure 8255
                                                
1004    3E 00                                   LD A,#0
1006    D3 40                                   out (PortA),A           ;Write Binary Patten to 8255
1008    D3 41                                   out (PortB),A           ;Write Binary Patten to 8255
100A    D3 42                                   out (PortC),A           ;Write Binary Patten to 8255
                                                
100C    01 0000                                 LD BC,#0                ;PRESET COUNTERS
                                                                        ;REPEAT FOREVER
100F    78                              100$:   LD A,B                  ; HOLD FAST CHANGE BITS
1010    D3 41                                   out (PortB),A           ; Write Binary Patten to 8255
                                                
1012    79                                      LD A,C                  ; HOLD SLOW CHANGE BITS
1013    D3 42                                   out (PortC),A           ; Write Binary Patten to 8255
                                                
1015    04                                      INC B                   ; UPDATE FAST CHANGE BITS
1016    3E 1F                                   LD A,#1F                ; SET UP LIMIT MASK
1018    A0                                      AND B                   ; LIMIT RANGE
1019    47                                      LD B,A                  ; KEEP UPDATED VALUE
101A    C2 0F10                                 JP NZ,100$              ; IF SLOW UPDATE REQUIRED
                                                                        ; THEN
101D    0C                                      INC C                   ;  UPDATE SLOW CHANGE BITS
101E    3E 1F                                   LD A,#1F                ;  SET UP LIMIT MASK
1020    A1                                      AND C                   ;  LIMIT RANGE
1021    4F                                      LD C,A                  ;  KEEP UPDATED VALUE
                                                                        ; ENDIF
1022    C3 0F10                                 JP 100$                 ;END REPEAT
                                                
                                                
                                                
                                                        .PAGE
                                                        .SBTTL  .       Application ZIF02 SECTION
                                                
```

```
                        AA=1200
1200    3E 80           LD A,#$X                ;PIO Control Port Settings Assignment
1202    D3 43           out (Contrl),A          ;Configure 8255

1204    3E 00           LD A,#0
1206    D3 40           out (PortA),A           ;Write Binary Patten to 8255
1208    D3 41           out (PortB),A           ;Write Binary Patten to 8255
120A    D3 42           out (PortC),A           ;Write Binary Patten to 8255


120C    06 08           LD B,#008.              ;REPEAT 8 TIMES
120E    0E 00           LD C,#00                ;START BIT TO PROCESS

                                                ;REPEAT FOR EACH BIT
1210    3E 00           LD A,#00000000%         ; -SEL  IS   ACTIVE (BIT 3)
                                                ;  MC1  IS INACTIVE (BIT 4)
1212    B1              OR A,C                  ;COMBINE THE VALUES
1213    D3 41           out (PortB),A           ;Write Binary Patten to 8255 (BOTTOM 5 BITS)

1215    3E 1E           LD A,#00011110%         ;  MC2  IS INACTIVE (BIT 0)
                                                ;  CLK1 IS   ACTIVE (BIT 1)
                                                ;  CLK2 IS   ACTIVE (BIT 2)
                                                ;  VT   IS   ACTIVE (BIT 3)
                                                ;  VV   IS   ACTIVE (BIT 4)
1217    D3 42           out (PortC),A           ;Write Binary Patten to 8255 (TOP    5 BITS)

1219    11 FF0F         LD DE,#0FFF             ;DELAY VALUE
121C    1B              DEC DE
121D    7A              LD A,D                  ;LOOP TILL COUNT EXPIRED
121E    B3              OR E
121F    C2 1C12         JP NZ,250$

1222    3E 00           LD A,#00000000%         ; -SEL  IS   ACTIVE (BIT 3)
                                                ;  MC1  IS INACTIVE (BIT 4)
1224    B1              OR A,C                  ;COMBINE THE VALUES
1225    D3 41           out (PortB),A           ;Write Binary Patten to 8255 (BOTTOM 5 BITS)

1227    3E 00           LD A,#00000000%         ;  MC2  IS INACTIVE (BIT 0)
                                                ;  CLK1 IS INACTIVE (BIT 1)
                                                ;  CLK2 IS INACTIVE (BIT 2)
                                                ;  VT   IS INACTIVE (BIT 3)
                                                ;  VV   IS INACTIVE (BIT 4)
1229    D3 42           out (PortC),A           ;Write Binary Patten to 8255 (TOP    5 BITS)

122B    11 FF0F         LD DE,#0FFF             ;DELAY VALUE
122E    1B              DEC DE
122F    7A              LD A,D                  ;LOOP TILL COUNT EXPIRED
1230    B3              OR E
1231    C2 2E12         JP NZ,260$

1234    0C              INC C                   ;SWITCH TO NEXT BIT FOR TEST
1235    10 D9           DJNZ 210$               ;REPEAT ALL BITS

1237    C3 0C12         GOTO 200$               ;REPEAT TEST AGAIN


                       .PAGE
                       .SBTTL  .       Application ZIF03 SECTION

                        AA=1400         .ORG 1400
                        ;
                        ;               DEFINITIONS OF REGISTER USAGE
                        ;
                                                ;L Register Bits
                        FASTCLK = 0             ;Fast clock Bit in Counter  (Master Clock 1)
                        SLOWCLK = 2             ;Slow clock Bit in Counter  (Master Clock 2)
                        V.CLK1  = 4             ;Pin Modfied by Clock 1
                        V.CLK2  = 5             ;Pin Modfied by Clock 2
                        V.Tri   = 7             ;Pin Tristate Control

                                                ;H Register Bits
                        V.CNTRL = 2             ;Pin Voltage  Control
                        TERMSEQ = 00001111%     ;Sequence terminator Mask

                                                ;Control Register Bits
                        B.MC1   = 4             ;Master Clock 1 Bit (Fastest)   (Port B)
                        B.MC2   = 0             ;Master Clock 2 Bit (Slower)    (Port C)
                        B.SC1   = 1             ;Select Clock 1 Bit             (Port C)
                        B.SC2   = 2             ;Select Clock 2 Bit             (Port C)
                        B.TRI   = 3             ;Tristate       Bit             (Port C)
                        B.VOLT  = 4             ;Voltage        Bit             (Port C)


1400    3E 80           ZIF03:
1402    D3 43           LD A,#$X                ;PIO Control Port Settings Assignment
                        out (Contrl),A          ;Configure 8255

1404    3E 00           LD A,#0
1406    D3 40           out (PortA),A           ;Write Binary Patten to 8255
1408    D3 41           out (PortB),A           ;Write Binary Patten to 8255
140A    D3 42           out (PortC),A           ;Write Binary Patten to 8255


140C    06 08           LD B,#008.              ;REPEAT 8 TIMES
140E    0E 00           LD C,#00                ;START BIT TO PROCESS

1410    21 0000         LD HL,#0                ;INITIALISE CONTROL SIGNALS

                                                ;REPEAT FOR EACH BIT
1413    3E 00           320$:                   ;START CONDITION
                        LD A,#00000000%         ; -SEL  IS   ACTIVE (BIT 3)

1415    CB 45           BIT FASTCLK,L
1417    20 02           JR NZ,321$
1419    CB E7           SET B.MC1,A

141B    B1              321$:
141C    D3 41           OR A,C                  ;COMBINE IN ZIP PIN CHANNEL
                        out (PortB),A           ;Write Binary Patten to 8255 (BOTTOM 5 BITS)
```

```
                                                 
                                                 ;        LD DE,#0FFF             ;DELAY VALUE
                                                 ;350$:   DEC DE
                                                 ;        LD A,D                  ;LOOP TILL COUNT EXPIRED
                                                 ;        OR E
                                                 ;        JP NZ,350$
                                                 
141E    3E 00                                    LD A,#00000000%         ; All Functions Inactive
                                                 
1420    CB 55                                    BIT SLOWCLK,L
1422    20 02                                    JR NZ,351$
1424    CB C7                                    SET B.MC2,A
                                                 351$:
1426    CB 54                                    BIT V.CNTRL,H
1428    20 02                                    JR NZ,352$
142A    CB E7                                    SET B.VOLT,A
                                                 352$:
142C    CB 7D                                    BIT V.TRI,L
142E    20 02                                    JR NZ,353$
1430    CB DF                                    SET B.TRI,A
                                                 353$:
1432    CB 65                                    BIT V.CLK1,L
1434    20 02                                    JR NZ,354$
1436    CB CF                                    SET B.SC1,A
                                                 354$:
1438    CB 6D                                    BIT V.CLK2,L
143A    20 02                                    JR NZ,355$
143C    CB D7                                    SET B.SC2,A
                                                 355$:
                                                 
143E    D3 42                                    out (PortC),A           ;Write Binary Patten to 8255 (TOP    5 BITS)
                                                 
1440    23                                       INC HL                  ;UPDATE TRIGGER SIGNALS
1441    3E 0F                                    LD A,#TERMSEQ           ;Sequence terminator Mask
1443    A4                                       AND A,H                 ;LOOP TILL TEST SEQUENCE COMPLETE
1444    B5                                       OR L
1445    C2 1314                                  JP NZ,320$
                                                 
                                                 ;        LD DE,#0FFF             ;DELAY VALUE
                                                 ;370$:   DEC DE
                                                 ;        LD A,D                  ;LOOP TILL COUNT EXPIRED
                                                 ;        OR E
                                                 ;        JP NZ,370$
                                                 
1448    0C                                       INC C                   ;SWITCH TO NEXT BIT FOR TEST
1449    10 C5                                    DJNZ 310$               ;REPEAT ALL BITS
                                                 
144B    C3 0C14                                  GOTO 300$               ;REPEAT TEST AGAIN
                                                 
                                                 
                                                 
                                                .PAGE
                                                .SBTTL  .       Application ADC SECTION
                                                 
        AA=1800                                  .ORG 1800
                                                 
                                                 ;       Just use this to check all bits change on all I/P lines
                                                 
                                                 ADC01:
1800    3E 80                                    LD A,#$X                ;PIO Control Port Settings Assignment
1802    D3 43                                    out (Contrl),A          ;Configure 8255
                                                 
1804    3E 00                                    LD A,#0
1806    D3 40                                    out (PortA),A           ;Write Binary Patten to 8255
1808    D3 41                                    out (PortB),A           ;Write Binary Patten to 8255
180A    D3 42                                    out (PortC),A           ;Write Binary Patten to 8255
                                                 
180C    01 0000                                  LD BC,#0                ;PRESET COUNTERS
                                                                         ;REPEAT FOREVER
180F    78                                       LD A,B                  ; HOLD SLOW BITS
1810    D3 41                                    out (PortB),A           ; Write Binary Patten to 8255
                                                 
1812    79                                       LD A,C                  ; HOLD FAST CHANGE BITS
1813    D3 42                                    out (PortC),A           ; Write Binary Patten to 8255
1815    03                                       INC BC                  ; UPDATE BIT Patten
1816    C3 0F18                                  JP 500$                 ;END REPEAT
                                                 
                                                 
                                                 
        AA=1900                                  .ORG 1900
                                                 ;
                                                 ;               DEFINITIONS OF REGISTER USAGE
                                                 ;
                                                                         ;E Register
                                                 ;       Bits 0,1,2 Used to select channel
                                                 ;       Bit  3     Used to select VCNG
                                                 
                                                                         ;B Register Bits
                                                 ;       All bit are Data Patten
                                                 
                                                 
                                                 ADC02:
1900    3E 80                                    LD A,#$X                ;PIO Control Port Settings Assignment
1902    D3 43                                    out (Contrl),A          ;Configure 8255
                                                 
1904    3E 00                                    LD A,#0
1906    D3 40                                    out (PortA),A           ;Write Binary Patten to 8255
1908    D3 41                                    out (PortB),A           ;Write Binary Patten to 8255
190A    D3 42                                    out (PortC),A           ;Write Binary Patten to 8255
                                                 
                                                                         ;REPEAT FOREVER
190C    06 00                                    LD B,#0                 ;Restart the Whole Process
190E    48                                       LD C,B                  ;Remember the Counter settings
                                                 
190F    06 10                                    LD B,#16.               ;Restart the Channel Loop
1911    58                                       LD E,B                  ;Remember the Control control settings
1912    3E 07                                    LD A,#7                 ;Extraction Mask for channel number
1914    A3                                       AND A,E                 ;Get the Channel to Process
1915    CB 5B                                    BIT 3,E                 ; IF   Imediate Data transfer reqd
1917    20 02                                    JR NZ,585$              ; THEN
```

```
1919    CB E7                   SET 4,A                 ;      Select that feature

191B    CB 9F           585$:   RES 3,A                 ;Always Ensure the Board is being Selected
191D    D3 42                   out (PortC),A           ;Write Binary Patten to 8255

191F    79                      LD A,C                  ;Hold the Data Patten
1920    D3 41                   out (PortB),A           ;Write Binary Patten to 8255

1922    43                      LD B,E                  ;Remember Previous channel settings
1923    10 EC                   DJNZ 580$               ;Process all Channels

1925    41                      LD B,C                  ;Remember Previous counter settings
1926    10 E6                   DJNZ 560$               ;Process all Data Pattens

1928    C3 0C19                 GOTO 550$               ;Restart everything again
```

```
                               .PAGE
                               .SBTTL  .       Application Interface Board SECTION

;       ============================================================
                $Y = 00000000%          ; Preset Definition Word
;       ============================================================
;               $Y = $Y + 10000000%     ; Control Word is a  BIT Set  Definition
                $Y = $Y + 10000000%     ; Control Word is an I/O Mode Definition
;       ============================================================
                $Y = $Y + 00000000%     ; Group 0      is Mode 0
;               $Y = $Y + 00100000%     ; Group 0      is Mode 1
;               $Y = $Y + 01000000%     ; Group 0      is Mode 2
;       ============================================================
                $Y = $Y + 00000000%     ; Group 1      is Mode 0
;               $Y = $Y + 00000001%     ; Group 1      is Mode 1
;       ============================================================
;               $Y = $Y + 00010000%     ; Port A       is Input
                $Y = $Y + 00000000%     ; Port A       is Output
;       ============================================================
;               $Y = $Y + 00010000%     ; Port A       is Input
                $Y = $Y + 00000000%     ; Port A       is Output
;       ============================================================
;               $Y = $Y + 00000010%     ; Port B       is Input
                $Y = $Y + 00000000%     ; Port B       is Output
;       ============================================================
;               $Y = $Y + 00000001%     ; Port C Low   is Input
                $Y = $Y + 00000000%     ; Port C Low   is Output
;       ============================================================
                $Y = $Y + 00001000%     ; Port C High  is Input
;               $Y = $Y + 00000000%     ; Port C High  is Output
;       ============================================================
```

```
        AA=2000                 .ORG 2000
                        ;
                        ;       PORT A      =       BOARD ADDRESS
                        ;       PORT B      =       BOARD DATA
                        ;       PORT C      =       BOARD CONTROL INFORMATION REPLY
                        ;
                        IF01:
2000    3E 80                   LD A,#$X                ;PIO Control Port Settings Assignment
2002    D3 43                   out (Contrl),A          ;Configure 8255

2004    06 00                   LD B,#0                 ;REPEAT FOREVER
2006    78              2100$:  LD A,B                  ; HOLD COUNT
2007    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
2009    D3 41                   out (PortB),A           ; Write Binary Patten to 8255
200B    D3 42                   out (PortC),A           ; Write Binary Patten to 8255
200D    04                      INC B                   ; UPDATE CHANGE BITS
200E    C3 0620                 JP 2100$                 ;END REPEAT


        AA=2200                 .ORG 2200
                        IF02:
2200    3E 88                   LD A,#$Y                ;PIO Control Port Settings Assignment
2202    D3 43                   out (Contrl),A          ;Configure 8255

2204    06 00                   LD B,#0                 ;REPEAT FOREVER
2206    78              2200$:  LD A,B                  ; HOLD COUNT
2207    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
2209    D3 41                   out (PortB),A           ; Write Binary Patten to 8255
220B    04                      INC B                   ; UPDATE CHANGE BITS
220C    C3 0622                 JP 2200$                 ;END REPEAT


        AA=2300                 .ORG 2300
                        IF03:
2300    3E 88                   LD A,#$Y                ;PIO Control Port Settings Assignment
2302    D3 43                   out (Contrl),A          ;Configure 8255

                        2300$:

2304    06 00                   LD B,#0                 ;REPEAT FOR ALL PATTENS
2306    78              2310$:  LD A,B                  ; HOLD COUNT
2307    D3 41                   out (PortB),A           ; Write Binary Patten to 8255

2309    3E D0                   LD A,#0D0H              ;Select U16
230B    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
                                ;Strobe Data on Rising edge
230D    3E FF                   LD A,#0FFH              ;DeSelect U16
230F    D3 40                   out (PortA),A           ; Write Binary Patten to 8255

2311    3E E0                   LD A,#0E0H              ;Select U19
2313    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
                                ;Strobe Data on Rising edge
2315    3E FF                   LD A,#0FFH              ;DeSelect U19
2317    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
```

```
                                                          INC B                   ; UPDATE CHANGE BITS
2319    04                                                JP NZ,2310$             ;END REPEAT
231A    C2 0623
                                                          JP 2300$                ;START AGAIN
231D    C3 0423
                                                          .ORG 2400
        AA=2400
                                                  IF04:
                                                          LD A,#$Y                ;PIO Control Port Settings Assignment
2400    3E 88                                             out (Contrl),A          ;Configure 8255
2402    D3 43
                                                  2400$:
2404    3E C0                                             LD A,#0C0H              ;Select U30
2406    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255

2408    06 00                                             LD B,#0                 ;REPEAT FOR ALL PATTENS
240A    78                                        2410$:  LD A,B                  ; HOLD COUNT
240B    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255
240D    04                                                INC B                   ; UPDATE CHANGE BITS
240E    C2 0A24                                           JP NZ,2410$             ;END REPEAT

2411    06 00                                             LD B,#0                 ;REPEAT FOR DELAY PERIOD
2413    78                                        2420$:  LD A,B                  ; HOLD COUNT
2414    04                                                INC B                   ; UPDATE CHANGE BITS
2415    C2 1324                                           JP NZ,2420$             ;END REPEAT

2418    06 00                                             LD B,#0                 ;REPEAT FOR ALL PATTENS
241A    05                                        2450$:  DEC B                   ; UPDATE CHANGE BITS
241B    78                                                LD A,B                  ; HOLD COUNT
241C    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255
241E    A7                                                AND A                   ; Test patten value
241F    C2 1A24                                           JP NZ,2450$             ;END REPEAT

2422    06 00                                             LD B,#0                 ;REPEAT FOR DELAY PERIOD
2424    78                                        2460$:  LD A,B                  ; HOLD COUNT
2425    04                                                INC B                   ; UPDATE CHANGE BITS
2426    C2 2424                                           JP NZ,2460$             ;END REPEAT

2429    C3 0424                                           JP 2400$                ;START AGAIN

        AA=2500                                           .ORG 2500

                                                  IF05:
2500    3E 88                                             LD A,#$Y                ;PIO Control Port Settings Assignment
2502    D3 43                                             out (Contrl),A          ;Configure 8255
2504    3E FF                                             LD A,#0FFH              ;DeSelect Everything
2506    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255

                                                  2500$:
2508    3E 00                                             LD A,#000H              ;Select 0V Reference
250A    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255

250C    3E C0                                             LD A,#0C0H              ;Select U30
250E    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255
2510    3E FF                                             LD A,#0FFH              ;DeSelect U30
2512    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255


2514    3E 00                                             LD A,#000H              ;Select RED LED OFF, GREEN LED OFF
2516    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255

2518    3E E0                                             LD A,#0E0H              ;Select U19
251A    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255
                                                          ;Strobe Data on Rising edge
251C    3E FF                                             LD A,#0FFH              ;DeSelect U19
251E    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255

2520    01 0000                                           LD BC,#0                ;Long Delay period
2523    0B                                        2510$:  DEC BC                  ;Reduce Delay Counter
2524    00                                                NOP                     ;Extra Delay
2525    00                                                NOP                     ;Extra Delay
2526    00                                                NOP                     ;Extra Delay
2527    00                                                NOP                     ;Extra Delay
2528    78                                                LD A,B                  ;Hold High Byte
2529    B1                                                OR C                    ;Merge Low Byte
252A    C2 2325                                           JP NZ,2510$             ;Wait till delay exausted

                                                  ;       ========================================================

252D    3E 3F                                             LD A,#063.              ;Select Quarter Max Volt Reference
252F    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255

2531    3E C0                                             LD A,#0C0H              ;Select U30
2533    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255
2535    3E FF                                             LD A,#0FFH              ;DeSelect U30
2537    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255


2539    3E 02                                             LD A,#002H              ;Select RED LED OFF, GREEN LED ON
253B    D3 41                                             out (PortB),A           ; Write Binary Patten to 8255

253D    3E E0                                             LD A,#0E0H              ;Select U19
253F    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255
                                                          ;Strobe Data on Rising edge
2541    3E FF                                             LD A,#0FFH              ;DeSelect U19
2543    D3 40                                             out (PortA),A           ; Write Binary Patten to 8255

2545    01 0000                                           LD BC,#0                ;Long Delay period
2548    0B                                        2520$:  DEC BC                  ;Reduce Delay Counter
2549    00                                                NOP                     ;Extra Delay
254A    00                                                NOP                     ;Extra Delay
254B    00                                                NOP                     ;Extra Delay
254C    00                                                NOP                     ;Extra Delay
254D    78                                                LD A,B                  ;Hold High Byte
254E    B1                                                OR C                    ;Merge Low Byte
```

```
254F    C2 4825                 JP NZ,2520$             ;Wait till delay exausted


;       ========================================================

2552    3E 7F                   LD A,#127.              ;Select Half Max Volt Reference

2554    3E C0                   LD A,#0C0H              ;Select 0V Reference
2556    D3 41                   out (PortB),A           ; Write Binary Patten to 8255

2558    3E C0                   LD A,#0C0H              ;Select U30
255A    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
255C    3E FF                   LD A,#0FFH              ;DeSelect U30
255E    D3 40                   out (PortA),A           ; Write Binary Patten to 8255

2560    3E 01                   LD A,#001H              ;Select RED LED ON , GREEN LED OFF
2562    D3 41                   out (PortB),A           ; Write Binary Patten to 8255

2564    3E E0                   LD A,#0E0H              ;Select U19
2566    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
                                                        ;Strobe Data on Rising edge
2568    3E FF                   LD A,#0FFH              ;DeSelect U19
256A    D3 40                   out (PortA),A           ; Write Binary Patten to 8255

256C    01 0000                 LD BC,#0                ;Long Delay period
256F    0B                      DEC BC                  ;Reduce Delay Counter
2570    00                      NOP                     ;Extra Delay
2571    00                      NOP                     ;Extra Delay
2572    00                      NOP                     ;Extra Delay
2573    00                      NOP                     ;Extra Delay
2574    78                      LD A,B                  ;Hold High Byte
2575    B1                      OR C                    ;Merge Low Byte
2576    C2 6F25                 JP NZ,2530$             ;Wait till delay exausted


;       ========================================================

2579    3E FF                   LD A,#0FFH              ;Select Max Volt Reference
257B    D3 41                   out (PortB),A           ; Write Binary Patten to 8255

257D    3E C0                   LD A,#0C0H              ;Select U30
257F    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
2581    3E FF                   LD A,#0FFH              ;DeSelect U30
2583    D3 40                   out (PortA),A           ; Write Binary Patten to 8255


2585    3E 03                   LD A,#003H              ;Select RED LED ON , GREEN LED ON
2587    D3 41                   out (PortB),A           ; Write Binary Patten to 8255

2589    3E E0                   LD A,#0E0H              ;Select U19
258B    D3 40                   out (PortA),A           ; Write Binary Patten to 8255
                                                        ;Strobe Data on Rising edge
258D    3E FF                   LD A,#255.              ;DeSelect U19
258F    D3 40                   out (PortA),A           ; Write Binary Patten to 8255

2591    01 0000                 LD BC,#0                ;Long Delay period
2594    0B                      DEC BC                  ;Reduce Delay Counter
2595    00                      NOP                     ;Extra Delay
2596    00                      NOP                     ;Extra Delay
2597    00                      NOP                     ;Extra Delay
2598    00                      NOP                     ;Extra Delay
2599    78                      LD A,B                  ;Hold High Byte
259A    B1                      OR C                    ;Merge Low Byte
259B    C2 9425                 JP NZ,2540$             ;Wait till delay exausted

259E    C3 0825                 JP 2500$                ;START AGAIN




        AA=2600                 .ORG 2600

                        IF06:
2600    3E 88                   LD A,#$Y                ;PIO Control Port Settings Assignment
2602    D3 43                   out (Contrl),A          ;Configure 8255
2604    3E FF                   LD A,#0FFH              ;DeSelect Everything
2606    D3 40                   out (PortA),A           ; Write Binary Patten to 8255

                        2600$:
2608    3E 00                   LD A,#000H              ;Select 0V Reference
260A    CD 3826                 CALL 2650$              ;Set voltage

260D    3E 01                   LD A,#001H              ;Select Reference 1
260F    CD 3826                 CALL 2650$              ;Set voltage

2612    3E 02                   LD A,#002H              ;Select Reference 2
2614    CD 3826                 CALL 2650$              ;Set voltage

2617    3E 04                   LD A,#004H              ;Select Reference 3
2619    CD 3826                 CALL 2650$              ;Set voltage

261C    3E 08                   LD A,#008H              ;Select Reference 4
261E    CD 3826                 CALL 2650$              ;Set voltage

2621    3E 10                   LD A,#010H              ;Select Reference 5
2623    CD 3826                 CALL 2650$              ;Set voltage

2626    3E 20                   LD A,#020H              ;Select Reference 6
2628    CD 3826                 CALL 2650$              ;Set voltage

262B    3E 40                   LD A,#040H              ;Select Reference 7
262D    CD 3826                 CALL 2650$              ;Set voltage

2630    3E 80                   LD A,#080H              ;Select Reference 8
2632    CD 3826                 CALL 2650$              ;Set voltage

2635    C3 0826                 GOTO 2600$              ;Start again
```

```
                                                    
                                                    
2638    D3 41                                       
                                                    
263A    3E C0                                       
263C    D3 40                                       
263E    D3 40                                       
2640    3E FF                                       
2642    D3 40                                       
                                                    
2644    01 0000                                     
2647    0B                                          
2648    00                                          
2649    00                                          
264A    00                                          
264B    00                                          
264C    78                                          
264D    B1                                          
264E    C2 4726                                     
2651    C9                                          
                                                    
                                                    
                                                    
                                                    
                                                    
                                                    
                                                    
                                                    
Selected Assembler is Z80+EXTRAS
Selected Library   is Z80.ALB
Library Identity   is <Z80 23/10/01 V1.06>
Number of Lines     = 788
Number of Symbols   = 263
Number of Warnings  = 0
Number of Errors    = 0
Elapsed Time        = 17 Secs
Processed on 25-Apr-2002 at 13:56:47
                                                    
                                                    
```

```
                                                    
                                                    
2650$:                                              
        out (PortB),A           ; Write Binary Patten to 8255
                                                    
        LD A,#0C0H              ;Select U30         
        out (PortA),A           ; Write Binary Patten to 8255
        out (PortA),A           ; Write Binary Patten to 8255
        LD A,#0FFH              ;DeSelect U30       
        out (PortA),A           ; Write Binary Patten to 8255
                                                    
        LD BC,#0                ;Long Delay period  
2660$:  DEC BC                  ;Reduce Delay Counter
        NOP                     ;Extra Delay        
        NOP                     ;Extra Delay        
        NOP                     ;Extra Delay        
        NOP                     ;Extra Delay        
        LD A,B                  ;Hold High Byte     
        OR C                    ;Merge Low Byte     
        JP NZ,2660$             ;Wait till delay exausted
        RET                     ;Back to caller     
                                                    
                                                    
        .END    ;of programme                       
                                                    
```

```
      1           .Title Z80TST.ASM Binary count using 8255 Device with LEDs on Port B
      2
      3
      4           .ASSEMBLER Z80+EXTRAS
      5
      6           .ORG 0
      7   ;
      8   ;                       Definitions Section
      9   ;
     10
     11           PORTA  = 80H            ;A few Variant PORT assignment Methods
     12           PORTB  = 81H
     13           PORTC  = PORTB + 1
     14           CONTRL = PORTA + 3
     15
     16           ld SP,#StkAddr
     17
     18   ;       ================================================================
     19           $X = 00000000%          ; Preset Definition Word
     20   ;       ================================================================
     21   ;       $X = $X + 10000000%     ; Control Word is a  BIT Set  Definition
     22           $X = $X + 10000000%     ; Control Word is an I/O Mode Definition
     23   ;       ================================================================
     24           $X = $X + 00000000%     ; Group 0      is Mode 0
     25   ;       $X = $X + 00100000%     ; Group 0      is Mode 1
     26   ;       $X = $X + 01000000%     ; Group 0      is Mode 2
     27   ;       ================================================================
     28           $X = $X + 00000000%     ; Group 1      is Mode 0
     29   ;       $X = $X + 00000001%     ; Group 1      is Mode 1
     30   ;       ================================================================
     31   ;       $X = $X + 00010000%     ; Port A       is Input
     32           $X = $X + 00000000%     ; Port A       is Output
     33   ;       ================================================================
     34   ;       $X = $X + 00000010%     ; Port B       is Input
     35           $X = $X + 00000000%     ; Port B       is Output
     36   ;       ================================================================
     37           $X = $X + 00000001%     ; Port C Low   is Input
     38   ;       $X = $X + 00000000%     ; Port C Low   is Output
     39   ;       ================================================================
     40           $X = $X + 00001000%     ; Port C High  is Input
     41   ;       $X = $X + 00000000%     ; Port C High  is Output
     42   ;       ================================================================
     43
     44   ;       LD A,#89H               ;See Above for 8255 Mode BIT settings
     45           LD A,#$X                ;Prefered/Alternative Assignment Method
     46           ld a,#257.
     47
     48           out (Contrl),A          ;Configure 8255
     49
     50           xor a                   ;Clear Acumulator (A good Starting value)
     51
     52   500$:
     53           out (PortB),A           ;Write Binary Patten to LEDs via 8255
     54           inc a                   ;Add 1 to Display Patten
     55           ld bc,250.              ;Delay time 250msecs
     56   ;       Call $mSec              ;Sleep for 1/4 Second
     57           Jp  500$                ;Loop forever
     58
     59
     60          .PAGE
     61          .SBTTL  HALT PROGRAME FOR A NUMBER OF MILLI-SECONDS
     62
     63          .GAP 20h
     64
     65   $MSEC:
     66
     67   ;
     68   ;      DESCRIPTION
     69   ;      THIS ROUTINE WILL CREATE A DELAY OF 1 TO 64K MILLI SECONDS
     70   ;      THE TIME OF THE CALL TO THIS ROUTINE AND ITS RETURN IS TAKEN
     71   ;      INTO ACCOUNT AS THE TIMING PERIOD.
     72   ;
     73   ;      ENTRY   CONDITIONS
     74   ;      BC      =       NUMBER OF MILISECONDS TO DELAY
     75   ;
     76   ;      EXIT    CONDITIONS
     77   ;      NONE
     78   ;
     79   ;      AUTHOR:                 R. J. SPRIGGS
     80   ;
     81   ;      CALL $MSEC      ;17T    ;TIME TO CALL THIS ROUTINE
     82           PUSH AF         ;11T    ;PROTECT ALL REGISTERS
     83           PUSH BC         ;11T
     84           PUSH DE         ;11T
     85
     86           LD DE,#78.      ;10T    ;TIMMING PERIOD COUNTER IF ONLY ONE PASS
     87                           ;60T    ;TOTAL FOR SECTOR 'A'
     88
     89   10$:    NOP             ;4T     ;TIMMING PAD
     90                           ;4T     ;TOTAL FOR SECTOR 'B'
     91
     92   20$:    DEC DE          ;6T     ;REDUCE PERIOD COUNTER
     93           LD A,D          ;4T     ;HOLD HIGH BYTE
     94           OR E            ;4T     ;CHECKSUM WITH LOW BYTE
     95           JP NZ,20$       ;10T    ;LOOP TILL TIMING COUNT EXAUSTED
     96                           ;24T    ;TOTAL FOR SECTOR 'C'
     97
     98           DEC BC          ;6T     ;REDUCE REPEAT COUNTER
     99           LD A,B          ;4T     ;HOLD HIGH BYTE
    100           OR C            ;4T     ;CHECKSUM WITH LOW BYTE
    101           JP Z,30$        ;10T    ;EXIT IF REPEAT COUNT EXAUSTED
    102                           ;24T    ;TOTAL FOR SECTOR 'D'
    103
    104           LD DE,#81.      ;10T    ;INTERNAL RESET FOR PERIOD COUNTER
    105           NOP             ;4T     ;TIMMING PAD
    106           NOP             ;4T     ;TIMMING PAD
```

```
          JP 20$          ;10T    ;ENTER DELAY LOOP AGAIN
                          ;28T    ;TOTAL FOR SECTOR 'E'

30$:      POP DE          ;10T    ;RESTORE ALL USED REGISTERS
          POP BC          ;10T
          POP AF          ;10T
          RET             ;10T    ;BACK TO CALLER
                          ;40T    ;TOTAL FOR SECTOR 'F'

;
;               TIMING CALCULATIONS
;
;       WHERE :-
;                               2T STATES = 1 MICRO SECOND
;               A       =       60T STATES
;               B       =        4T STATES
;               C       =       24T STATES
;               D       =       24T STATES
;               E       =       28T STATES
;               F       =       40T STATES
;
;       ASSUME 'BC' = 1         TIME =  A+B+78C+D+F
;                                       60+4+1872+24+40  = 2000
;
;       ASSUME 'BC' = N         TIME =  A+B+78C+D+F +(N-1)(E+B+81C+D)
;                                       60+4+1872+24+40  = 2000 +
;                                      (N-1)(4 +28 +1944 +24) = (N-1)(2000)
;


           .ORG 1800
           .reserve 30              ;Reserve space for Stack
Stkaddr:                            ;Note Stack operates top of memory down


       .end   ; of programme
```

```
    1        .TITLE 8031TEST.ASM Knight Rider 8255 Port Programme LEDs on Port B
    2    ;
    3    ;                              ;Special Function Registers
    4        ACC     = 0E0H             ;Acumulator                              Bit Add
    5        B       = 0F0H             ;B Register                              Bit Add
    6        PSW     = 0D0H             ;Programme Status Word                   Bit Add
    7        SP      = 81H              ;Stack Pointer
    8    ;   DPTR                       ;Data Pointer 2 Bytes
    9        DPL     = 82H              ;Low  Byte
   10        DPH     = 83H              ;High Byte
   11        P0      = 80H              ;Port 0                                  Bit Add
   12        P1      = 90H              ;Port 1                                  Bit Add
   13        P2      = 0A0H             ;Port 2                                  Bit Add
   14        P3      = 0B0H             ;Port 3                                  Bit Add
   15        IP      = 0B8H             ;Interupt Priority Control               Bit Add
   16        IE      = 0A8H             ;Interupt Enable   Control               Bit Add
   17        TMOD    = 89H              ;Timer/Counter Mode Control
   18        TCON    = 88H              ;Timer/Counter      Control              Bit Add
   19        T2CON   = 0C8H             ;Timer/Counter 2    Control  8052 only   Bit Add
   20        TH0     = 8CH              ;Timer/Counter 0 High Byte
   21        TL0     = 8AH              ;Timer/Counter 0 Low  Byte
   22        TH1     = 8DH              ;Timer/Counter 0 High Byte
   23        TL1     = 8BH              ;Timer/Counter 0 Low  Byte
   24        TL2     = 0CCH             ;Timer/Counter 0 Low  Byte    8052 only
   25        RCAP2H  = 0CBH             ;T/C 2 Capture Reg High Byte  8052 only
   26        RCAP2L  = 0CAH             ;T/C 2 Capture Reg Low  Byte  8052 only
   27        SCON    = 98H              ;Serial Control                          Bit Add
   28        SBUF    = 99H              ;Serial Data Buffer
   29        PCON    = 87H              ;Power Control
   30
   31    ;   Definitions Section
   32
   33        PORTA  = 0FF40H            ;A few Variant PORT assignment Methods
   34        PORTB  = 0FF41H
   35        PORTC  = PORTB + 1
   36        CONTRL = PORTA + 3
   37
   38
   39        .ORG 0                     ;Start of system
   40
   41        LJMP BEGIN                 ;GOTO where programme starts
   42
   43
   44        .ORG 8100H                 ;Address set to Match Flight Board
   45
   46    BEGIN:
   47        MOV SP,#5FH                ;Set up Stack Pointer Address
   48
   49        MOV DPTR,#CONTRL           ;Point at 8255 Control Word
   50
   51    ;   ============================================================
   52        $X = 00000000%             ; Preset Definition Word
   53    ;   ============================================================
   54    ;   $X = $X + 10000000%        ; Control Word is a  BIT Set  Definition
   55        $X = $X + 10000000%        ; Control Word is an I/O Mode Definition
   56    ;   ============================================================
   57        $X = $X + 00000000%        ; Group 0      is Mode 0
   58    ;   $X = $X + 00100000%        ; Group 0      is Mode 1
   59    ;   $X = $X + 01000000%        ; Group 0      is Mode 2
   60    ;   ============================================================
   61        $X = $X + 00000000%        ; Group 1      is Mode 0
   62    ;   $X = $X + 00000001%        ; Group 1      is Mode 1
   63    ;   ============================================================
   64    ;   $X = $X + 00010000%        ; Port A       is Input
   65        $X = $X + 00000000%        ; Port A       is Output
   66    ;   ============================================================
   67    ;   $X = $X + 00000010%        ; Port B       is Input
   68        $X = $X + 00000000%        ; Port B       is Output
   69    ;   ============================================================
   70        $X = $X + 00000001%        ; Port C Low   is Input
   71    ;   $X = $X + 00000000%        ; Port C Low   is Output
   72    ;   ============================================================
   73        $X = $X + 00001000%        ; Port C High  is Input
   74    ;   $X = $X + 00000000%        ; Port C High  is Output
   75    ;   ============================================================
   76        MOV A,#$X                  ;Alternative Assignment Method
   77
   78    ;    MOV A,#89H                 ; See Above for 8255 Mode BIT settings
   79        MOVX @DPTR,A               ;Configure 8255
   80
   81        MOV DPTR,#CONTRL
   82        MOV A,#89H
   83        MOVX @DPTR,A
   84
   85        MOV DPTR,#PORTB
   86
   87        MOV R7,#1           ; Master Display Patten
   88        MOV R6,#1           ; Go Right Direction Flag
   89
   90    LOOP:
   91        MOV A,R7            ; Light Display Patten
   92        MOVX @DPTR,A        ; Display Patten
   93        CJNE R6,#1,RIGHT
   94
   95        RL A                ; Move Patten Left
   96        MOV R7,A            ; Hold Display Patten
   97        CJNE R7,#80,wait    ; If end Reached
   98        MOV R6,#0           ; Set Direction Right
   99        SJMP WAIT
  100
  101    RIGHT:
  102        RR A                ; Move Patten Right
  103        MOV R7,A            ; Hold Display Patten
  104        CJNE R7,#01,wait    ; If end Reached
  105        MOV R6,#1           ; Set Direction Left
  106
```

```
WAIT:
;   ACALL DELAY         ;Delay Removed to give quick Simulator Demo

    SJMP LOOP               ;Loop Forever


DELAY:              ; Sleep for 3 * 1/10 Secs Period
    MOV R2,#3
DELAY1:             ; Sleep for 1/10 Secs Delay
    MOV R1,#200.
DELAY3:             ; Sleep for 1/2  msec Delay
    MOV R0,#249.
    NOP
DELAY5:
    DJNZ R0,DELAY5
    DJNZ R1,DELAY3
    DJNZ R2,DELAY1
    RET


    .END                    ;Source Complete

; Nothing is processed after the .END Directive

;
; Assembler variations and Notes
;
;       1.      NO EQU directive    So use SYMBOL = <Expression> structure
;       2.      ORG is prefixed with a dot ie. .ORG <Address>
;       3.      Default RADIX is 16 (Can be altered using .RADIX directive)
;       4.      Note SYMBOL and LABEL names MAX 6 Characters
;       5.      Note #?? implies a Constant
;       6.      Note $?? implies a Label or Symbol
;       7.      Note #$?? implies a Symbolic Constant
;       8.      Currently no Generic JMP order hence must use Specifics
;               ie. SJMP , LJMP , AJMP
```

```
                .TITLE  RJS/MICROCHIP PIC 14 Bit Instructions TEST CROSS-ASSEMBLER
                .SBTTL  AUTHOR R. J. SPRIGGS    283 Belle Vue Road Bournemouth
                .SBTTL  TEST PROGRAMME


                .ORG    0000H           ;START ADDRESS OF PROGRAMME

        ;
        ;
        ;               FULL MICROCHIP INSTRUCTION SET TEST FOR 16C84
        ;
        ;       f       = Register file 0 -> 7F
        ;       W       = Working Register (Acumulator)
        ;       b       = Bit Address within an 8-bit file register <0 - 7>
        ;       k       = Literal field constant data or label
        ;           Where k can be :-
        ;
        ;           #7  = 7  bit Constant         <0 - 003FH>
        ;           #8  = 8  bit Constant         <0 - 00FFH>
        ;           #11 = 11 bit Constant         <0 - 07FFH>
        ;           n   = 11 bit address          <0 - 07FFH>
        ;
        ;       x       = Dont care location (Defaults to 0 recommended form)
        ;       d       = Destination select    =0 store result in W
        ;                                       =1 store result in register file
        ;
                $FileA  = $File2-$File1
                $FileB  = $File3-$File2
                $POS    = 20.
                $NEG    = -33.
        ;
        ;                               ;Special Function Registers
                INDF    = 000H          ;Contents of FSR to address data memory
                TMR0    = 001H          ;8 Bit real time clock/counter
                PCL     = 002H          ;Low order 8 bits of PC
                STATUS  = 003H          ;Status Register

                BIT.Z   = 002h          ;Zero Bit
                BIT.C   = 000h          ;Carry Bit
        ;
        ;       Special Definitions
                Dest.W  = 0             ;Destination = W = Accumulator
                Dest.f  = 1             ;Destination = File Register


                MOVLW   5               ;MOVLW  k       Where k = #8    W=5
                ADDLW   2               ;ADDLW  k       Where k = #8    W=7
                SUBLW   4               ;SUBLW  k       Where k = #8    W=3
                ANDLW   2               ;ANDLW  k       Where k = #8    W=2
                IORLW   0A8H            ;IORLW  k       Where k = #8    W=AA
                XORLW   0AH             ;XORLW  k       Where k = #8    W=0A
                CLRW                    ;CLRW                           W=0
                CALL    200$            ;CALL   k       Where k = #11
                CLRF    $File2          ;CLRF   f       Where f = #7    f=0
                DECF    $file2,Dest.f   ;DECF   f,d     Where f = #7    f=-1
                COMF    $file1,Dest.f   ;COMF   f,d     Where f = #7    f=0
                INCF    $file1,Dest.F   ;INCF   f,d     Where f = #7    f=1
                MOVF    $file2,Dest.W   ;MOVWF  f,d     Where f = #7    W=1
                ADDWF   $file1,Dest.f   ;ADDWF  f,d     Where f = #7    f=2
                BSF     $file1,2        ;BSF    f,b     Where f = #7    f=6
                BCF     $file1,1        ;BCF    f,b     Where f = #7    f=4
                IORWF   $file1,Dest.f   ;IORWF  f,d     Where f = #7    f=5
                ANDWF   $file1,Dest.f   ;ANDWF  f,d     Where f = #7    f=1
                SUBWF   $file1,Dest.w   ;SUBWF  f,d     Where f = #7    W=0
                MOVWF   $file1          ;MOVWF  f       Where f = #7    f=0
                GOTO    100$            ;GOTO   k       Where k = #11
                NOP                     ;NOP

                .ORG    00FFH           ;WORST ADDRESS FOR A SLEEP ORDER TO SIT
        100$:   SLEEP                   ;SLEEP                          PC=PC-1
                NOP                     ;NOP


        200$:
                CALL    300$            ;CALL   k       Where k = #11
                RETURN                  ;RETURN

        300$:
                CALL    400$            ;CALL   k       Where k = #11
                RETLW   55.             ;RETLW  k       Where k = #8    W=37

        400$:
                RETFIE                  ;RETFIE


        ; *     ADDLW $POS              ;ADDLW  k       Where k = #8
        ; *     ADDWF $file1,Dest.W     ;ADDWF  f,d     Where f = #7
        ;
        ; *     ANDLW $NEG              ;ANDLW  k       Where k = #8
        ; *     ANDWF $file2,Dest.f     ;ANDWF  f,d     Where f = #7
        ;
        ; *     BCF   STATUS,BIT.C      ;BCF    f,b     Where f = #7
        ; *     BSF   STATUS,BIT.z      ;BSF    f,b     Where f = #7
        ;       BTFSC STATUS,BIT.C      ;BTFSC  f,b     Where f = #7
        ;       BTFSS STATUS,BIT.Z      ;BTFSS  f,b     Where f = #7
        ;
        ; *     CALL    LABEL           ;CALL   k       Where k = #11
        ;
        ; *     CLRF    $File2          ;CLRF   f       Where f = #7
        ; *     CLRW                    ;CLRW
        ;       CLRWDT                  ;CLRWDT
        ;
        ; *     COMF    $file1,Dest.W   ;COMF   f,d     Where f = #7
        ;
        ; *     DECF    $file3,Dest.f   ;DECF   f,d     Where f = #7
        ;       DECFSZ  $file2,Dest.W   ;DECFSZ f,d     Where f = #7
```

```
107     ;
108     ; *     GOTO    LABEL           ;GOTO   k       Where k = #11
109     ;
110     ; *     INCF    $file1,Dest.W   ;INCF   f,d     Where f = #7
111     ;       INCFSZ  $file2,Dest.f   ;INCFSZ f,d     Where f = #7
112     ;
113     ; *     IORLW   $POS            ;IORLW  k       Where k = #8
114     ; *     IORWF   $file2,Dest.f   ;IORWF  f,d     Where f = #7
115     ;
116     ;LABEL:
117     ; *     MOVLW   $NEG            ;MOVLW  k       Where k = #8
118     ; *     MOVF    $file2,Dest.f   ;MOVWF  f,d     Where f = #7
119     ; *     MOVWF   $file1          ;MOVWF  f       Where f = #7
120     ;
121     ; *     NOP                     ;NOP
122     ;
123     ; *     RETFIE                  ;RETFIE
124     ; *     RETLW   55.             ;RETLW  k       Where k = #8
125     ; *     RETURN                  ;RETURN
126     ;
127     ;       RLF     $file2,Dest.w   ;RLF    f,d     Where f = #7
128     ;       RRF     $file2,Dest.f   ;RRF    f,d     Where f = #7
129     ;
130     ; *     SLEEP                   ;SLEEP
131     ;
132     ; *     SUBLW   $POS            ;SUBLW  k       Where k = #8
133     ; *     SUBWF   $file2,Dest.w   ;SUBWF  f,d     Where f = #7
134     ;
135     ;       SWAPF   $file2,Dest.f   ;SWAPF  f,d     Where f = #7
136     ;
137     ; *     XORLW   $NEG            ;XORLW  k       Where k = #8
138     ;       XORWF   $file2,Dest.W   ;XORWF  f,d     Where f = #7
139     ;
140
141             .REGION REGISTERS
142             .ORG    0000H           ;START ADDRESS OF FILE DATA
143
144             .reserve 20             ;Note this area is not presetable
145     $FILE1:                         ;So preset check assembler ONLY
146             .EVEN
147     $FILE2: .RESERVE 11.
148             .EVEN
149     $FILE3: .reserve 2
150
151             .END
```

APPENDIX
Sub-Section

This section contains the following item:

Application Specific Language Source code for PIC16

Description :

This is the Application Specific Language Source code for the General Purpose Cross Assembler. This Source is specifically targeted for the PIC16C84 processor, however, should be suitable for the whole of the PIC16xxxx series processors.

Title of Pages	Number of Pages
Documentation Files.	
The Cross Assembler Application Specific Language code for the PIC16C84. Filename PIC16.LDS	3

```
    1   ;
    2   ; Filename is PIC16.LDS    Author R. J. Spriggs on 20/08/01     Updated 03/09/01
    3   ;
    4   ;              .... Library for General Purpose Cross Assembler
    5   ;
    6   ;   Mod Record
    7   ;   V1.01       Initial Release                                          24/08/01
    8   ;   V1.02       ???LW Orders Range checking +255,-128                    03/09/01
    9   ;
   10   ;*************************************************************
   11   ;             SINGLE PARAMETER ORDERS
   12   ;*************************************************************
   13   IF_COUNT 1
   14   BEGIN
   15
   16       META IF_PARM 1=CLRW       : BEGIN : DO "01 03"    : EXIT_HEX : END
   17       META IF_PARM 1=CLRWDT     : BEGIN : DO "00 64"    : EXIT_HEX : END
   18       META IF_PARM 1=NOP        : BEGIN : DO "00 00"    : EXIT_HEX : END
   19       META IF_PARM 1=RETFIE     : BEGIN : DO "00 09"    : EXIT_HEX : END
   20       META IF_PARM 1=RETURN     : BEGIN : DO "00 08"    : EXIT_HEX : END
   21       META IF_PARM 1=SLEEP      : BEGIN : DO "00 63"    : EXIT_HEX : END
   22
   23       IF_PARM 1=~~ID_MAIN
   24         META BEGIN : EXIT_CODE "PIC16 03/09/01 V1.02" : END
   25
   26       EXIT_ERROR Unknown Opcode
   27       RESTART LIBRARY_END
   28   END
   29
   30   ;*************************************************************
   31   ;             TWO    PARAMETER ORDERS
   32   ;*************************************************************
   33   IF_COUNT 2
   34   BEGIN
   35
   36       IF_PARM 1=ADDLW
   37           META BEGIN : DO "03EH",2="{+255\-128}"    : EXIT : END
   38
   39       IF_PARM 1=ANDLW
   40           META BEGIN : DO "039H",2="{+255\-128}"    : EXIT : END
   41
   42       IF_PARM 1=CALL
   43           META BEGIN : DO "( "2="/512.+20H ){+40\+32}",2="/2&0FFH" : EXIT : END
   44
   45       IF_PARM 1=CLRF
   46           META BEGIN : DO "001H","080H+"2="{+255\+128}"  : EXIT : END
   47
   48       IF_PARM 1=GOTO
   49           META BEGIN : DO "( "2="/512.+28H ){+47\+40}",2="/2&0FFH" : EXIT : END
   50
   51       IF_PARM 1=IORLW
   52           META BEGIN : DO "038H",2="{+255\-128}"        : EXIT : END
   53
   54       IF_PARM 1=MOVLW
   55           META BEGIN : DO "030H",2="{+255\-128}"        : EXIT : END
   56
   57       IF_PARM 1=MOVWF
   58           META BEGIN : DO "000H","080H+"2="{+255\+128}"  : EXIT : END
   59
   60       IF_PARM 1=RETLW
   61           META BEGIN : DO "034H",2="{+255\-128}"        : EXIT : END
   62
   63       IF_PARM 1=SUBLW
   64           META BEGIN : DO "03CH",2="{+255\-128}"        : EXIT : END
   65
   66       IF_PARM 1=XORLW
   67           META BEGIN : DO "03AH",2="{+255\-128}"        : EXIT : END
   68
   69       IF_PARM 1=~~INS_WIDTH
   70       BEGIN
   71         IF_PARM 2=ISPACE
   72           META BEGIN : EXIT_CODE 2 : END
   73           RESTART LIBRARY_END
   74       END
   75   ;
   76       EXIT_ERROR Unknown Opcode
   77       RESTART LIBRARY_END
   78   END
   79
   80   ;*************************************************************
   81   ;
   82   ;             THREE  PARAMETER ORDERS
   83   ;*************************************************************
   84   IF_COUNT 3
   85   BEGIN
   86
   87
   88       IF_PARM 1=ADDWF
   89           META BEGIN : DO "007H",3="*128.+"2="{+255\+0}" : EXIT : END
   90
   91       IF_PARM 1=ANDWF
   92           META BEGIN : DO "005H",3="*128.+"2="{+255\+0}" : EXIT : END
   93
   94       IF_PARM 1=COMF
   95           META BEGIN : DO "009H",3="*128.+"2="{+255\+0}" : EXIT : END
   96
   97       IF_PARM 1=DECF
   98           META BEGIN : DO "003H",3="*128.+"2="{+255\+0}" : EXIT : END
   99
  100       IF_PARM 1=DECFSZ
  101           META BEGIN : DO "00BH",3="*128.+"2="{+255\+0}" : EXIT : END
  102
  103       IF_PARM 1=INCF
  104           META BEGIN : DO "00AH",3="*128.+"2="{+255\+0}" : EXIT : END
  105
  106       IF_PARM 1=INCFSZ
```

```
            META BEGIN : DO "00FH",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=IORWF
            META BEGIN : DO "004H",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=MOVF
            META BEGIN : DO "008H",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=RLF
            META BEGIN : DO "00DH",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=RRF
            META BEGIN : DO "00CH",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=SUBWF
            META BEGIN : DO "002H",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=SWAPF
            META BEGIN : DO "00EH",3="*128.+"2="{+255\+0}" : EXIT : END

        IF_PARM 1=XORWF
            META BEGIN : DO "006H",3="*128.+"2="{+255\+0}" : EXIT : END

    ;       BIT OPERATIONS

        IF_PARM 1=BCF
        BEGIN
            META DO "( "3="/2+10H ){+19\+16}",3="&01H*128.+"2="{+255\+0}" : EXIT
        END

        IF_PARM 1=BSF
        BEGIN
            META DO "( "3="/2+14H ){+23\+20}",3="&01H*128.+"2="{+255\+0}" : EXIT
        END

        IF_PARM 1=BTFSC
        BEGIN
            META DO "( "3="/2+18H ){+27\+24}",3="&01H*128.+"2="{+255\+0}" : EXIT
        END

        IF_PARM 1=BTFSS
        BEGIN
            META DO "( "3="/2+1CH ){+31\+28}",3="&01H*128.+"2="{+255\+0}" : EXIT
        END
    ;
    EXIT_ERROR Unknown Opcode
    END

    RESTART LIBRARY_END
    EXIT_ERROR Unknown Opcode
    ;
    ;*****************************************************************
    ;               PROCEDURES SECTION
    ;*****************************************************************
    ;
    ;
    ;*****************************************************************
    ;       Library Linking Section
    ;*****************************************************************
    ;
    BEGIN LIBRARY_END
    ;
    ; This is the Library Extention Linking Point   (A dummy access point)
    ;
    END LIBRARY_END
    ;
    ;*****************************************************************
    ;       End of Library
    ;*****************************************************************
    ;
```

APPENDIX
Sub-Section

This section contains the following item:

Application Specific Language Source code for PIC16SIM

Description :

This is the Application Specific Language Source code for the General Purpose Simulator. This Source is specifically targeted for the PIC16C84 processor, however, it should be suitable for simulation of the basic code for some of the other processors in PIC16xxxx series range. Currently NO device specific implementation code has been included in this source, however, the access handles are available.

Title of Pages	Number of Pages
Documentation Files.	
The Simulator Application Specific Language code for the PIC16C84. Filename PIC16SIM.LDS	26

```
    1    ;
    2    ; Filename is Pic16SIM.LDS  Author R. J. Spriggs on 28/08/01  Updated 08/09/01
    3    ;
    4    ; START DESIGN                                    V1.01       RJS   01/09/01
    5    ; Modification  Registers accessed via procedure  V1.02       RJS   08/09/01
    6    ; Modification  More Instructions Added           V1.03       RJS   16/09/01
    7    ; Modification  More Instructions Added           V1.04       RJS   17/09/01
    8    ; Modification  More Instructions Added           V1.05       RJS   18/09/01
    9    ; Modification  General Source Tidy               V1.06       RJS   19/09/01
   10    ; Modification                                    V1.10       RJS   ../../01
   11    ;
   12    ;              Pic16 Library for General Purpose Simulator
   13    ;
   14    ;*************************************************************
   15    ;             SINGLE PARAMETER ORDERS
   16    ;*************************************************************
   17
   18    IF_PARM 1=~~ID_MAIN
   19    BEGIN
   20       EXIT_CODE Pic16SIM 19/09/01 V1.06 Seq 0067
   21    END
   22
   23    IF_PARM 1=PROCESSOR
   24    BEGIN
   25    ;    IF_FOUND 2,DEBUG
   26    ;    BEGIN
   27    ;        TRACE "DEBUG PARAMETER LOCATED"
   28    ;    END
   29
   30        ;Regular Initialisation operations
   31            ;Register "I"     Reset I=Inc/Dec Value = 1
   32            ;Register "T"     Reset Time State Transitions Counter
   33        DO TS00IS01
   34
   35                               ;Always the First Operation
   36        PERFORM GET_OP_CODE
   37                               ;H = High Op_Code   PC=PC+1 Q=PCH   P=PCL
   38                               ;                           H=(PCH) L=(PCL)
   39
   40
   41
   42    ;       *****************************************************************
   43    ;       *   Instruction Set Crude hacking method
   44    ;       *****************************************************************
   45
   46        DO 0S30H&
   47        IF_REG 0=00
   48        BEGIN
   49          IF_REG H=00
   50          BEGIN
   51    ;
   52    ;               This should be one of the following operations
   53    ;               MOVWF   NOP     CLRWDT  RETFIE  RETURN  SLEEP
   54    ;
   55            DO 0S9FL&
   56            IF_REG 0=00
   57            BEGIN
   58                META IF_FOUND 2,DECOMP : BEGIN : DO 'NOP' : END
   59                RESTART GENERAL_EXIT
   60            END
   61
   62            DO 0S80L&
   63            IF_REG 0=80
   64            BEGIN
   65                DO 0S7FL&BP
   66                META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVWF  $'BH : END
   67                CACHE_SEL 3
   68                CACHE_LOAD 02
   69                CACHE_R A
   70                PERFORM PUT_REGFILE
   71                RESTART GENERAL_EXIT
   72            END
   73
   74            IF_REG L=64
   75            BEGIN
   76                META IF_FOUND 2,DECOMP : BEGIN : DO 'CLRWDT' : END
   77                RESTART GENERAL_EXIT
   78            END
   79
   80            IF_REG L=09
   81            BEGIN
   82                META IF_FOUND 2,DECOMP : BEGIN : DO 'RETFIE' : END
   83                PERFORM POP
   84                ;   MORE CODE STILL REQUIRED HERE
   85                RESTART GENERAL_EXIT
   86            END
   87
   88            IF_REG L=08
   89            BEGIN
   90                META IF_FOUND 2,DECOMP : BEGIN : DO 'RETURN' : END
   91                PERFORM POP
   92                RESTART GENERAL_EXIT
   93            END
   94
   95            IF_REG L=63
   96            BEGIN
   97                META IF_FOUND 2,DECOMP : BEGIN : DO 'SLEEP' : END
   98                ; PC = PC - 1
   99                DO PGI-PP
  100                META IF_REG_LOWER P,00 : BEGIN : DO 0SFFP&PPQGI-QP0S1FQ&QP : END
  101                RESTART GENERAL_EXIT
  102            END
  103
  104         END
  105
  106         IF_REG_LOWER H=08
```

```
        BEGIN
          IF_REG H=01
          BEGIN
;
;               This should be one of the following operations
;               CLRF    CLRW
;
              IF_REG L=03
              BEGIN
                  META IF_FOUND 2,DECOMP : BEGIN : DO 'CLRW' : END
                  ;   Get contents of status register into R8  and Mask Flag Bits
                  META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
                  ;   Zero RA , Set ZERO Bit , Put contents of RA into W register
                  META DO AS00 0S048!8P : SEL 3 : CL 02 : WR A
                  RESTART FLAG_PROCESS
              END

              DO 0S7FL&BP0S80L&
              IF_REG 0=80
              BEGIN
                  META IF_FOUND 2,DECOMP : BEGIN : DO 'CLRF  'BH : END
                  ;   Get contents of status register into R8  and Mask Flag Bits
                  META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
                  ;   Zero RA , Set ZERO Bit , Put contents of RA into f register
                  DO AS00 0S048!8P
                  PERFORM PUT_REGFILE
                  RESTART FLAG_PROCESS
              END
          END

          IF_REG H=02
          BEGIN
;
;               SUBWF   Operation
;
              DO 0S7FL&BP0S80L&LP
              META IF_FOUND 2,DECOMP : BEGIN : DO 'SUBWF' : END

              ;   Get contents of f register into RA
              PERFORM GET_REGFILE

              ;   Get contents of W register into RC
              META SEL 3 : CL 02 : RD C

              ;   Process the Calculation leaving Result in RA
              DO CG2PAG5P
              PERFORM GENERAL_8_BIT_SUB
              DO 2GAP
              RESTART END_WF_OPERATION
          END

          IF_REG H=03
          BEGIN
;
;               DECF    Operation
;
              DO 0S7FL&BP0S80L&LP
              META IF_FOUND 2,DECOMP : BEGIN : DO 'DECF ' : END

              ;   Get contents of status register into R8  and Mask Flag Bits
              META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

              ;   Get contents of f register into RA
              PERFORM GET_REGFILE

              ;   Process the Calculation leaving Result in RA
              DO AGI-AP0SFFA&AP
              META IF_REG A=00 : BEGIN : DO 0S048!8P : END
              RESTART END_WF_OPERATION
          END

          IF_REG H=04
          BEGIN
;
;               IORWF   Operation
;
              DO 0S7FL&BP0S80L&LP
              META IF_FOUND 2,DECOMP : BEGIN : DO 'IORWF' : END

              ;   Get contents of status register into R8  and Mask Flag Bits
              META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

              ;   Get contents of f register into RA
              PERFORM GET_REGFILE

              ;   Get contents of W register into RC
              META SEL 3 : CL 02 : RD C

              ;   Process the Calculation leaving Result in RA
              DO CGA!AP
              META IF_REG A=00 : BEGIN : DO 0S048!8P : END
              RESTART END_WF_OPERATION
          END

          IF_REG H=05
          BEGIN
;
;               ANDWF   Operation
;
              DO 0S7FL&BP0S80L&LP
              META IF_FOUND 2,DECOMP : BEGIN : DO 'ANDWF' : END

              ;   Get contents of status register into R8  and Mask Flag Bits
              META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

              ;   Get contents of f register into RA
              PERFORM GET_REGFILE

              ;   Get contents of W register into RC
              META SEL 3 : CL 02 : RD C
```

```
217
218               ;   Process the Calculation leaving Result in RA
219               DO CGA&AP
220               META IF_REG A=00 : BEGIN : DO 0S048!8P : END
221               RESTART END_WF_OPERATION
222           END
223
224           IF_REG H=06
225           BEGIN
226     ;
227     ;               XORWF   Operation
228     ;
229               DO 0S7FL&BP0S80L&LP
230               META IF_FOUND 2,DECOMP : BEGIN : DO 'XORWF' : END
231
232               ;   Get contents of status register into R8  and Mask Flag Bits
233               META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
234
235               ;   Get contents of f register into RA
236               PERFORM GET_REGFILE
237
238               ;   Get contents of W register into RC
239               META SEL 3 : CL 02 : RD C
240
241               ;   Process the Calculation leaving Result in RA
242               DO CGAEAP
243               META IF_REG A=00 : BEGIN : DO 0S048!8P : END
244               RESTART END_WF_OPERATION
245           END
246
247           IF_REG H=07
248           BEGIN
249     ;
250     ;               ADDWF   Operation
251     ;
252               DO 0S7FL&BP0S80L&LP
253               META IF_FOUND 2,DECOMP : BEGIN : DO 'ADDWF' : END
254
255               ;   Get contents of f register into RA
256               PERFORM GET_REGFILE
257
258               ;   Get contents of W register into RC
259               META SEL 3 : CL 02 : RD C
260
261               ;   Process the Calculation leaving Result in RA
262               DO CG2PAG5P
263               PERFORM GENERAL_8_BIT_ADD
264               DO 2GAP
265               RESTART END_WF_OPERATION
266           END
267
268           RESTART GENERAL_FAIL
269         END
270
271         IF_REG_HIGHER H=07
272         BEGIN
273
274           IF_REG H=08
275           BEGIN
276     ;
277     ;               MOVF    Operation
278     ;
279               DO 0S7FL&BP0S80L&LP
280               META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVF ' : END
281
282               ;   Get contents of status register into R8  and Mask Flag Bits
283               META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
284
285               ;   Get contents of f register into RA
286               PERFORM GET_REGFILE
287
288               ;   Process the Calculation leaving Result in RA
289               META IF_REG A=00 : BEGIN : DO 0S048!8P : END
290               RESTART END_WF_OPERATION
291           END
292
293           IF_REG H=09
294           BEGIN
295     ;
296     ;               COMF    Operation
297     ;
298               DO 0S7FL&BP0S80L&LP
299               META IF_FOUND 2,DECOMP : BEGIN : DO 'COMF ' : END
300
301               ;   Get contents of status register into R8  and Mask Flag Bits
302               META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
303
304               ;   Get contents of f register into RA
305               PERFORM GET_REGFILE
306
307               ;   Process the Calculation leaving Result in RA
308               DO AG0IAP
309               META IF_REG A=00 : BEGIN : DO 0S048!8P : END
310               RESTART END_WF_OPERATION
311           END
312
313           IF_REG H=0A
314           BEGIN
315     ;
316     ;               INCF    Operation
317     ;
318               DO 0S7FL&BP0S80L&LP
319               META IF_FOUND 2,DECOMP : BEGIN : DO 'INCF ' : END
320
321               ;   Get contents of status register into R8  and Mask Flag Bits
322               META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
323
324               ;   Get contents of f register into RA
325               PERFORM GET_REGFILE
326
```

```
327             ;   Process the Calculation leaving Result in RA
328             DO AGI+AP0SFFA&AP
329             META IF_REG A=00 : BEGIN : DO 0S048!8P : END
330             RESTART END_WF_OPERATION
331         END
332
333         IF_REG H=0B
334         BEGIN
335 ;
336 ;               DECFSZ  Operation
337 ;
338             DO TGI+TP 0S7FL&BP0S80L&LP
339             META IF_FOUND 2,DECOMP : BEGIN : DO 'DECFSZ' : END
340
341             ;   Get contents of f register into RA
342             PERFORM GET_REGFILE
343
344             ;   Process the Calculation leaving Result in RA
345             DO AGI-AP0SFFA&AP
346
347             ;   PC = PC + 1 WHEN RESULT = ZERO
348             IF_REG A=00
349             BEGIN
350                 DO PGI+PP
351                 IF_REG_HIGHER P,FF
352                 META BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
353             END
354             PERFORM END_W&F_OPERATION
355             RESTART GENERAL_EXIT
356         END
357
358         IF_REG H=0C
359         BEGIN
360 ;
361 ;               RRF     Operation
362 ;
363             DO 0S7FL&BP0S80L&LP
364             META IF_FOUND 2,DECOMP : BEGIN : DO 'RRF  ' : END
365
366             ;   Get contents of status register into R8  and Mask Flag Bits
367             ;   and the C Bit in register RC
368             META SEL 2 : CL 03 : RD 8 : DO 0S018&CP0S068&8P
369
370             ;   Get contents of f register into RA
371             PERFORM GET_REGFILE
372
373             ;   Process the Calculation leaving Result in RA
374             DO 0S01A&8!8P AG0RAP 0S7FA&AP
375             META IF_REG C=01 : BEGIN : DO 0S80A!AP : END
376             RESTART END_WF_OPERATION
377         END
378
379         IF_REG H=0D
380         BEGIN
381 ;
382 ;               RLF     Operation
383 ;
384             DO 0S7FL&BP0S80L&LP
385             META IF_FOUND 2,DECOMP : BEGIN : DO 'RLF  ' : END
386
387             ;   Get contents of status register into R8  and Mask Flag Bits
388             ;   and the C Bit in register RC
389             META SEL 2 : CL 03 : RD 8 : DO 0S018&CP0S068&8P
390
391             ;   Get contents of f register into RA
392             PERFORM GET_REGFILE
393
394             ;   Process the Calculation leaving Result in RA
395             DO 0S80A&
396             META IF_REG 0=80 : BEGIN : DO 0S018!8P : END
397             DO 0S80A&0RAP 0SFEA&C!AP
398             RESTART END_WF_OPERATION
399         END
400
401         IF_REG H=0E
402         BEGIN
403 ;
404 ;               SWAPF   Operation
405 ;
406             DO 0S7FL&BP0S80L&LP
407             META IF_FOUND 2,DECOMP : BEGIN : DO 'SWAPF' : END
408
409             ;   Get contents of status register into R8  and Mask Flag Bits
410             META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
411
412             ;   Get contents of f register into RA
413             PERFORM GET_REGFILE
414
415             ;   Process the Calculation leaving Result in RA
416             DO 0S0FA&2P0SF0A& 1S10 1/5P 2G1*5!AP 0SFFA&AP
417
418             PERFORM END_W&F_OPERATION
419             RESTART GENERAL_EXIT
420         END
421
422         IF_REG H=0F
423         BEGIN
424 ;
425 ;               INCFSZ  Operation
426 ;
427             DO TGI+TP 0S7FL&BP0S80L&LP
428             META IF_FOUND 2,DECOMP : BEGIN : DO 'INCFSZ' : END
429
430             ;   Get contents of f register into RA
431             PERFORM GET_REGFILE
432
433             ;   Process the Calculation leaving Result in RA
434             DO AGI+AP0SFFA&AP
435
436             ;   PC = PC + 1 WHEN RESULT = ZERO
```

```
437                 IF_REG A=00
438                 BEGIN
439                     DO PGI+PP
440                     IF_REG_HIGHER P,FF
441                     META BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
442                 END
443                 PERFORM END_W&F_OPERATION
444                 RESTART GENERAL_EXIT
445             END
446           END
447
448           RESTART GENERAL_FAIL
449         END
450
451         IF_REG 0=10
452         BEGIN
453             DO 0S3CH&
454             IF_REG 0=10
455             BEGIN
456     ;
457     ;               BCF     Operation
458     ;
459                 ; Register N = Bit , B = f
460                 DO 0S7FL&BP 0S03H&NPN+NP 0S80L&
461                 META IF_REG 0=80 : BEGIN : DO 0S01N+NP : END
462                 META IF_FOUND 2,DECOMP : BEGIN : DO 'BCF  'BH','NH : END
463
464                 ;   Get contents of f register into RA
465                 PERFORM GET_REGFILE
466
467                 ;   Process the Calculation leaving Result in RA
468                 REG2BIT A
469                 BIT_CLR N
470                 BIT2REG A
471                 PERFORM PUT_REGFILE
472                 RESTART GENERAL_EXIT
473             END
474
475             IF_REG 0=14
476             BEGIN
477     ;
478     ;               BSF     Operation
479     ;
480                 ; Register N = Bit , B = f
481                 DO 0S7FL&BP 0S03H&NPN+NP 0S80L&
482                 META IF_REG 0=80 : BEGIN : DO 0S01N+NP : END
483                 META IF_FOUND 2,DECOMP : BEGIN : DO 'BSF  'BH','NH : END
484
485                 ;   Get contents of f register into RA
486                 PERFORM GET_REGFILE
487
488                 ;   Process the Calculation leaving Result in RA
489                 REG2BIT A
490                 BIT_SET N
491                 BIT2REG A
492                 PERFORM PUT_REGFILE
493                 RESTART GENERAL_EXIT
494             END
495
496             IF_REG 0=18
497             BEGIN
498     ;
499     ;               BTFSC   Operation
500     ;
501                 ; Register N = Bit , B = f
502                 DO TGI+TP 0S7FL&BP 0S03H&NPN+NP 0S80L&
503                 META IF_REG 0=80 : BEGIN : DO 0S01N+NP : END
504                 META IF_FOUND 2,DECOMP : BEGIN : DO 'BTFSC  'BH','NH : END
505
506                 ;   Get contents of f register into RA
507                 PERFORM GET_REGFILE
508
509                 ;   Process the Calculation leaving Result in RA
510                 BIT_LOAD 00000000
511                 BIT_SET N
512                 BIT2REG 0
513                 DO N&
514                 ;   PC = PC + 1 WHEN RESULT = ZERO
515                 IF_REG 0=00
516                 BEGIN
517                     DO PGI+PP
518                     IF_REG_HIGHER P,FF
519                     META BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
520                 END
521                 RESTART GENERAL_EXIT
522             END
523
524             IF_REG 0=1C
525             BEGIN
526     ;
527     ;               BTFSS   Operation
528     ;
529                 ; Register N = Bit , B = f
530                 DO TGI+TP 0S7FL&BP 0S03H&NPN+NP 0S80L&
531                 META IF_REG 0=80 : BEGIN : DO 0S01N+NP : END
532                 META IF_FOUND 2,DECOMP : BEGIN : DO 'BTFSS  'BH','NH : END
533
534                 ;   Get contents of f register into RA
535                 PERFORM GET_REGFILE
536
537                 ;   Process the Calculation leaving Result in RA
538                 BIT_LOAD 00000000
539                 BIT_SET N
540                 BIT2REG 0
541                 DO N&
542                 ;   PC = PC + 1 WHEN RESULT = ZERO
543                 IF_REG_NOT 0=00
544                 BEGIN
545                     DO PGI+PP
546                     IF_REG_HIGHER P,FF
```

```
                META BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
            END
            RESTART GENERAL_EXIT
        END
      RESTART GENERAL_FAIL
    END

    IF_REG 0=20
    BEGIN
;
;               This should be one of the following operations
;               CALL    GOTO
;

        ;   Get contents of PCLATH register into RA
        META SEL 2 : CL 0A : RD A : DO 0S18A&AP
        DO 0S07H&A!AP0S08H&
        IF_REG 0=00
        BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'CALL  'AHLH : END
            PERFORM PUSH
        END
        IF_REG 0=08
        BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'GOTO  'AHLH : END
        END
        DO AGQPLGPPTGI+TP
        RESTART GENERAL_EXIT
    END

    IF_REG 0=30
    BEGIN
;
;               This should be one of the following operations
;               ADDLW   ANDLW   IORLW   XORLW   RETLW   MOVLW   SUBLW
;

      DO 0S3EH&
      IF_REG 0=3E
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'ADDLW  'LH : END
            ;   Get contents of W register into R2 & R5 = Literal
            META SEL 3 : CL 02 : RD 2 : DO LG5P
            PERFORM GENERAL_8_BIT_ADD
            META DO 2GAP : CACHE_W A
            RESTART FLAG_PROCESS
      END

      IF_REG 0=3C
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'SUBLW  'LH : END
            ;   Get contents of W register into R2 & R5 = Literal
            META SEL 3 : CL 02 : RD 2 : DO LG5P
            PERFORM GENERAL_8_BIT_SUB
            META DO 2GAP : CACHE_W A
            RESTART FLAG_PROCESS
      END


      DO 0S3CH&
      IF_REG 0=30
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVLW  'LH : END
            ;   Put contents of Literal into W register
            META SEL 3 : CL 02 : WR L
            RESTART GENERAL_EXIT
      END

      IF_REG 0=34
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'RETLW  'LH : END
            ;   Put contents of Literal into W register
            META SEL 3 : CL 02 : WR L
            DO TGI+TP
            PERFORM POP
            RESTART GENERAL_EXIT
      END

      IF_REG H=38
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'IORLW  'LH : END
            ;   Get contents of status register into R8  and Mask Flag Bits
            META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
            ;   Get contents of W register into RA
            META SEL 3 : CL 02 : RD A
            DO LGA!AP
            META IF_REG A=00 : BEGIN : DO 0S048!8P : END
            CACHE_W A
            RESTART FLAG_PROCESS
      END

      IF_REG H=39
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'ANDLW  'LH : END
            ;   Get contents of status register into R8  and Mask Flag Bits
            META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
            ;   Get contents of W register into RA
            META SEL 3 : CL 02 : RD A
            DO LGA&AP
            META IF_REG A=00 : BEGIN : DO 0S048!8P : END
            CACHE_W A
            RESTART FLAG_PROCESS
      END

      IF_REG H=3A
      BEGIN
            META IF_FOUND 2,DECOMP : BEGIN : DO 'XORLW  'LH : END
            ;   Get contents of status register into R8  and Mask Flag Bits
            META SEL 2 : CL 03 : RD 8 : DO 0S038&8P
            ;   Get contents of W register into RA
```

```
657                 META SEL 3 : CL 02 : RD A
658                 DO LGAEAP
659                 META IF_REG A=00 : BEGIN : DO 0S048!8P : END
660                 CACHE_W A
661                 RESTART FLAG_PROCESS
662           END
663
664           RESTART GENERAL_FAIL
665       END
666
667
668       RESTART GENERAL_EXIT
669
670
671   ;       ****************************************************************
672   ;       *   .    SPECIAL CASE COMMON EXIT ROUTINES
673   ;       ****************************************************************
674
675       RESTART GENERAL_FAIL
676
677
678   ;       ****************************************************************
679   ;       *   GENERAL EXIT for end of WF Update Processes
680   ;       ****************************************************************
681
682   ;   Register A = Result             Register 8 = Flag Word
683   ;   Register L = f or W indicator
684
685       BEGIN END_WF_OPERATION
686           PERFORM END_W&F_OPERATION
687           RESTART FLAG_PROCESS
688       END   END_WF_OPERATION
689
690
691   ;       ****************************************************************
692   ;       *   GENERAL EXIT with ALU Update Process
693   ;       ****************************************************************
694
695
696   ;   Register 2 = Acumulator         Register 8 = Flag Word
697
698       BEGIN ALU_PROCESS
699           CACHE_SEL 1
700           CACHE_LOAD 00
701           CACHE_W 2
702   ;   Next Instruction Not Needed as required Exit Code follows Immediately
703   ;       RESTART FLAG_PROCESS
704       END ALU_PROCESS
705
706
707
708   ;       ****************************************************************
709   ;       *   GENERAL EXIT with FLAG Update only Process
710   ;       ****************************************************************
711
712
713   ;   Register 8 = Flag Word
714
715       BEGIN FLAG_PROCESS
716           CACHE_SEL 2
717           CACHE_LOAD 03
718           CACHE_R A
719           DO 0S078&8P0SF8A&8!
720           CACHE_W 0
721   ;   Next Instruction Not Needed as required Exit Code follows Immediately
722   ;       RESTART GENERAL_EXIT
723       END FLAG_PROCESS
724
725
726   ;       ****************************************************************
727   ;       *   GENERAL EXIT
728   ;       ****************************************************************
729
730       BEGIN GENERAL_EXIT
731           ; THIS IS THE COMMON EXIT PATH FOR MOST CODE
732           PERFORM RETURN_PC
733           META IF_FOUND 2,DECOMP : BEGIN : TRACE_M : END
734           EXIT_REG T,OK
735       END GENERAL_EXIT
736
737
738
739   ;       ****************************************************************
740   ;       *   GENERAL instruction fail
741   ;       ****************************************************************
742
743       BEGIN GENERAL_FAIL
744           ; THIS IS THE COMMON FAIL PATH FOR MOST BAD CODE
745           PERFORM RETURN_PC
746           META IF_FOUND 2,DECOMP : BEGIN : TRACE_M : END
747           EXIT_REG T,UnKnown Instruction
748       END GENERAL_FAIL
749   END
750
751   ;       ****************************************************************
752   ;       *   MANUAL COMMAND TEST SECTION
753   ;       ****************************************************************
754
755   IF_PARM 1=CONTROL
756   BEGIN
757       IF_PARM 2=TESTER
758       BEGIN
759           TRACE *********** TESTER **************
760   ;       DO AS01NS00
761   ;       PERFORM PUT_BIT
762           DO 2S155S10
763           PERFORM GENERAL_8_BIT_ADD
764           DO 2S175SC2
765           PERFORM GENERAL_8_BIT_ADD
766           DO 2S025S01
```

```
            PERFORM GENERAL_8_BIT_SUB
            DO 2S025S02
            PERFORM GENERAL_8_BIT_SUB
            DO 2S025S03
            PERFORM GENERAL_8_BIT_SUB
        END
    END


    ;       ***************************************************************
    ;       *   PROCEDURES SECTION
    ;       ***************************************************************

    ;PROCEDURE XXX
    ;    ; Entry Conditions
    ;    ;       Register 0 is
    ;    ;
    ;    ; Exit  Conditions
    ;    ;       ...
    ;    ;
    ;END_PROCEDURE XXX


    PROCEDURE GET_REGFILE
        ; Entry Conditions
        ;       Register B is Register to access 0 -> 7F
        ;
        ; Exit  Conditions
        ;       Register A is Contents of Accessed Register
        ;
        ;   Note    <STATUS> Register will hold High bit/s as appropriate to
        ;           adapt the value supplied in B Register
        ;           The full address is caculated into Register pair B & C
        CACHE_SEL 2
        CACHE_LOAD 03
        ;Register S = <STATUS>  Bit 7 = IRP , Bit 6 & 5 = RP1 & RP0
        CACHE_R S

        META DO 0S7FCS00B&BP0S40S& : IF_REG 0=40 : BEGIN : DO CS01 : END
        META DO 0S20S&             : IF_REG 0=20 : BEGIN : DO 0S80B!BP : END

        META DO 0S7FB& : IF_REG 0=00
        BEGIN
        ;   This is an Indirect Operation Memory Access process
            DO 0S80S&
            META IF_REG 0=80 : BEGIN : DO CS01 : END
            CACHE_LOAD 04
            CACHE_R B
            DO 0S7FB&
            ; IF INDIRECTION REGISTER RESELECTED AGAIN WRITE = NOP , READ = 00H
            META IF_REG 0=00 : BEGIN : DO AS00 : RETURN : END
            DO 0S7FB&
        END

        BIT{2REG B
        CACHE_BITLOAD
        CACHE_R A
    END_PROCEDURE GET_REGFILE


    PROCEDURE PUT_REGFILE
        ; Entry Conditions
        ;       Register A is New Contents of Accessed Register
        ;       Register B is Register to access 0 -> 7F
        ;
        ; Exit  Conditions
        ;
        ;   Note    <STATUS> Register will hold High bit/s as appropriate to
        ;           adapt the value supplied in B Register
        ;           The full address is caculated into Register pair B & C
        CACHE_SEL 2
        CACHE_LOAD 03
        ;Register S = <STATUS>  Bit 7 = IRP , Bit 6 & 5 = RP1 & RP0
        CACHE_R S

        META DO 0S7FCS00B&BP0S40S&
        IF_REG 0=40
        BEGIN
            CACHE_SEL 1
            CACHE_LOAD 08
            CACHE_R 0
            DO CS01C&CP
            CACHE_SEL 2
        END
        META DO 0S20S&             : IF_REG 0=20 : BEGIN : DO 0S80B!BP : END

        META DO 0S7FB& : IF_REG 0=00
        BEGIN
        ;   This is an Indirect Operation Memory Access process
            DO 0S80S&
            META IF_REG 0=80 : BEGIN : DO CS01 : END
            CACHE_LOAD 04
            CACHE_R B
            DO 0S7FB&
            ; IF INDIRECTION REGISTER RESELECTED AGAIN WRITE = NOP , READ = 00H
            META IF_REG 0=00 : BEGIN : DO AS00 : RETURN : END
            DO 0S7FB&
        END

        IF_REG 0=02
        BEGIN
        ;   This is a PC Low Byte write process hence Load High Byte with PCLATH
            CACHE_LOAD 0A
            CACHE_R Q
            DO 0S1FQ&QP
        END

        IF_REG B=03
        BEGIN
```

```
        ;   This is a status register process
            DO 0SE7A&AP
            CACHE_LOAD 03
            CACHE_W A
            RETURN
        END

        BIT{2REG B
        CACHE_BITLOAD
        CACHE_W A
    END_PROCEDURE PUT_REGFILE


    PROCEDURE GENERAL_8_BIT_ADD
    ; Entry Conditions
    ;   R2 = BYTE 1         R5 = BYTE 2
    ;
    ; Exit  Conditions
    ;   R8 = FLAG WORD  (Status Bits)
    ;   R2 = BYTE 1 + BYTE 2
    ;
    ;   FLAG WORD BIT ALLOCATION USAGE (PIC)
    ;   Bit     7   6   5   4   3   2   1   0
    ;   Use     *   *   *   *   *   Z   DC  C
    ;
    ;   REGISTER USAGE
    ;                               R1 = SIGN CARRY IN      R2 = HIGH CHUNK 1
    ;   R3 = LOW CHUNK 1                                    R5 = HIGH CHUNK 2
    ;   R6 = LOW CHUNK 2            R7 = CARRY IN           R8 = SIGN BIT
    ;
        DO 1S0F 5G1&6P 2G1&3P 1SF0 5G1&5P 2G1&2P 1S00 8S00
    ; TRACE ******************************
    ; TRACE BEFORE 8 BIT ADD
    ; TRACE_R 2,H BYTE 1
    ; TRACE_R 3,L BYTE 1
    ; TRACE_R 5,H BYTE 2
    ; TRACE_R 6,L BYTE 2

    ; Calculate Sum of Low LS 4 Bits Section , Set Sign Carry IN as reqd
    ; R7 = R3+R6 {ie. Sum Value of LS 4 Bits}  ... R1 = Sign Carry OUT
        DO 3G6+7P 1S101&1P
        DO 0S0F7&7P

    ; Sign Bit Calculation Section
    ; R1 = Sign Bit Carry IN , R2 & R5 = Sign Bits
    ; R2 = (Carry R1)+R2+R5 {ie. Sign Value of Calc}    ... R8 <- Sign Carry OUT
    ;
        DO 2G5+1+2P

    ;   META IF_REG_HIGHER 2,F0 : BEGIN : DO 8S01     : TRACE SIGN CARRY OUT : END
        META IF_REG_HIGHER 2,F0 : BEGIN : DO 8S01     : END
    ;;  META IF_REG_HIGHER 1,10 : BEGIN : DO 0S028!8P : TRACE HALF CARRY OUT : END
        META IF_REG_HIGHER 1,10 : BEGIN : DO 0S028!8P : END

    ; R2 = (Sign + LS Chunk)
        DO 0SFF2&7+2P

    ;;  META IF_REG 2=00        : BEGIN : DO 0S048!8P : TRACE RESULT IS ZERO : END
        META IF_REG 2=00        : BEGIN : DO 0S048!8P : END

    ; TRACE_R 2,RESULT BYTE
    ; TRACE_R 8,FLAGS  WORD

    END_PROCEDURE GENERAL_8_BIT_ADD



    PROCEDURE GENERAL_8_BIT_SUB
    ;   ENTRY CONDITIONS
    ;   R2 = BYTE 1         R5 = BYTE 2
    ;
    ;   EXIT  CONDITIONS
    ;   R8 = FLAG WORD
    ;   R2 = BYTE 1 - BYTE 2
    ;
    ;   FLAG WORD BIT ALLOCATION USAGE (PIC)
    ;   Bit     7   6   5   4   3   2   1   0
    ;   Use     *   *   *   *   *   Z   DC  C

    ;   FLAG WORD BIT ALLOCATION USAGE
    ;   Bit     7   6   5   4   3   2   1   0
    ;   Use     CY              OVR     PAR
    ;
    ;   REGISTER USAGE
    ;   R0 = ...                    R1 = SIGN CARRY IN      R2 = HIGH CHUNK 1
    ;   R3 = LOW CHUNK 1                                    R5 = HIGH CHUNK 2
    ;   R6 = LOW CHUNK 2            R7 = CARRY IN           R8 = SIGN BIT
    ;
    ; TRACE ******************************
    ; TRACE BEFORE 8 BIT SUB
    ; TRACE_R 5,BYTE 2
        DO 0S005-5P
        PERFORM GENERAL_8_BIT_ADD

    END_PROCEDURE GENERAL_8_BIT_SUB


    PROCEDURE GET_OP_CODE
        ; Set Time State Transitions Counter + 1T
        DO 0S01T+TX
        ; Select Internal Area
        CACHE_SEL 3
        ; Get Address of the PIC PC & Access Locally
        CACHE_LOAD 00
        CACHE_R Q
        CACHE_LOAD 01
        CACHE_R P

        ; Get ISPACE Cache Pointer NOTE PC Address Doubled as 2 Byte per OPcode
        CACHE_SEL 1
```

```
        BIT{2REG P
        CACHE_BITLOAD
        CACHE_BITADD
        CACHE_R H
        ;Strip Binary to 14 Bits No matter what is loaded in ISPACE
        DO 0S3FH&HP
        CACHE_ADD I
        CACHE_R L

    ;   IF_FOUND 2,DEBUG
    ;   BEGIN
    ;       ; P,Q = PC , H,L=OPCODE
    ;       TRACE_R Q  "PC HIGH ="
    ;       TRACE_R P  "PC LOW  ="
    ;       TRACE_R H  "(PCH)    ="
    ;       TRACE_R L  "(PCL)    ="
    ;   END

        META IF_FOUND 2,PC : BEGIN : DO 'PC='QHPH'  (PC)='HHLH'  ' : END

        ; PC = PC + 1
        DO PGI+PP
        META IF_REG_HIGHER P,FF : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
    END_PROCEDURE GET_OP_CODE


    PROCEDURE RETURN_PC
    ;   ENTRY CONDITIONS
    ;   RQ = PCH        RP = PCL        Location 07 = Address High Byte Mask
    ;
    ;   EXIT  CONDITIONS
    ;   PC Stored in Internals Area
    ;
        CACHE_SEL 3
        CACHE_LOAD 07
        CACHE_R 0
        DO Q&QP
        CACHE_LOAD 00
        CACHE_W Q
        CACHE_LOAD 01
        CACHE_W P
    END_PROCEDURE RETURN_PC


    PROCEDURE PUSH
        ; Entry Conditions
        ;       Register P is PCH
        ;       Register Q is PCL
        ;
        ; Exit  Conditions
        ;       P + Q placed on Stack and Stack Pointer Adjusted
        ;
        ;   Initially locate SP Address , Limit Range , Write PCH,PCL on stack
        CACHE_SEL 3
        CACHE_LOAD 06
        CACHE_R 1
        DO 0S0F1&1P
        CACHE_LOAD 10
        CACHE_ADD 1
        CACHE_W P
        CACHE_ADD I
        CACHE_W Q
        ;   Update SP Address ie SP=SP+2 , Limit Range , Write SP back
        DO 0S021+1P0S0F1&1P
        CACHE_LOAD 06
        CACHE_W 1
    END_PROCEDURE PUSH


    PROCEDURE POP
        ; Entry Conditions
        ;       P + Q Removed from Stack and Stack Pointer Adjusted
        ;
        ; Exit  Conditions
        ;       Register P is PCH
        ;       Register Q is PCL
        ;
        ;   Initially locate SP Address , Update SP Address ie SP=SP-2
        ;   Limit Range , Write updated SP back , Read PCH,PCL from stack
        CACHE_SEL 3
        CACHE_LOAD 06
        CACHE_R 1
        DO 1GI-I-1P0S0F1&1P
        CACHE_W 1
        CACHE_LOAD 10
        CACHE_ADD 1
        CACHE_R P
        CACHE_ADD I
        CACHE_R Q
    END_PROCEDURE POP


    PROCEDURE END_W&F_OPERATION
        ; Entry Conditions
        ;       Register A = Result
        ;       Register L = f or W indicator
        ;
        ; Exit  Conditions
        ;       Decomp Processed and Location Written to
        ;
        META IF_FOUND 2,DECOMP : BEGIN : DO '  'BH
            META IF_REG L=00 : BEGIN : DO ',0' : END
            META IF_REG L=80 : BEGIN : DO ',1' : END : END
        META IF_REG L=80 : BEGIN : PERFORM PUT_REGFILE : END
        META IF_REG L=00 : BEGIN : SEL 3 : CL 02 : WR A : END
    END_PROCEDURE END_W&F_OPERATION



    ;       ****************************************************************
```

```
;       ****************************************************************
;       *                     CONTROL COMMANDS CODE
;       ****************************************************************
;       ****************************************************************

IF_PARM 1=CONTROL
BEGIN
    TRACE Control Command Located

    IF_PARM 2=INTERUPT
    BEGIN
        TRACE INTERUPT Command Located
    END

    EXIT_CODE OK
END


;       ****************************************************************
;       ****************************************************************
;       *                           DEVICE CODE
;       ****************************************************************
;       ****************************************************************

IF_PARM 1=DEVICE
BEGIN
    IF_PARM 2=FLASHER
    BEGIN
        DEVICE       R02,C20,'S BELOW'
        DEVICE_REG S R03,C20    ;Write S to Screen

        IF_REG S=0
        BEGIN
            DO SS01
            DEVICE R07,C30,<"Flash ON ">
            EXIT_CODE OK
        END

        IF_REG S=1
        BEGIN
            DO SS02
            DEVICE R07,C30,<'Flash OFF'>
            EXIT_CODE OK
        END

        IF_REG S=2
        BEGIN
            DO SS03
            DEVICE R07,C30,"Flash ON "
            EXIT_CODE OK
        END

        IF_REG S=3
        BEGIN
            DO SS00
            DEVICE R07,C30,'Flash OFF'
            EXIT_CODE OK
        END
        EXIT_CODE FAIL
    END
    EXIT_CODE FAIL
END


;       ****************************************************************
;       *                   PROCESSOR INITIALISEE
;       ****************************************************************
;
;       ****************************************
;
;       WARNING DO NOT ADD COMMENTS TO END OF A LINE
;
;       ****************************************
;

IF_PARM 1=PROCESSOR_RESET
BEGIN
    CACHE_MAP 1 ISPACE
    CACHE_MAP 2 REGISTERS
    CACHE_MAP 3 INTERNALS

                            ;Select a Specific Cache Map (INTERNALS AREA)
    CACHE_SEL 3
                            ;Reset Time State Counter
    DO TS00
                            ;Clear PCH=0 & PCL=0
    CACHE_LOAD 00
    CACHE_W T
    CACHE_LOAD 01
    CACHE_W T
                            ;Clear Stack Pointer & its contents
    CACHE_LOAD 06
    CACHE_W T
    CACHE_LOAD 10
    CACHE_W T
    CACHE_LOAD 11
    CACHE_W T
                            ;Initialise PC High Mask (Default = 1K)
    DO TS03
    META IF_SIMULATOR 16C84 : BEGIN : DO TS03 : END
    CACHE_LOAD 07
    CACHE_W T

                            ;Initialise FSR's High Max Mask (Default = 2 Banks)
    DO TS00
    META IF_SIMULATOR 16C84 : BEGIN : DO TS00 : END
    CACHE_LOAD 08
    CACHE_W T

                            ;Select a Specific Cache Map (REGISTERS AREA)
```

```
        CACHE_SEL 2
                                ;Reset STATUS
        DO TS18
        CACHE_LOAD 03
        CACHE_W T
                                ;Reset PCLATH , INTCON
        DO TS00
        CACHE_LOAD 0A
        CACHE_W T
        CACHE_LOAD 0B
        CACHE_W T
                                ;Reset OPTION , TRISA , TRISB
        DO TSFF
        CACHE_LOAD 81
        CACHE_W T
        CACHE_LOAD 85
        CACHE_W T
        CACHE_LOAD 86
        CACHE_W T
                                ;Reset EECON1 , TRISA , TRISB
        DO TS00
        CACHE_LOAD 88
        CACHE_W T

        EXIT_REG T,OK
    END


    ;       ********************************************************************
    ;       ********************************************************************
    ;       *                       DEVICE INITIALISE
    ;       ********************************************************************
    ;       ********************************************************************

    IF_PARM 1=DEVICE_RESET
    BEGIN
        IF_PARM 2=FLASHER
        BEGIN
            DO SS00
            DEVICE R99
            DEVICE R01,C01,'*'
            DEVICE R15,C01,'*'
            DEVICE R01,C79,'*'
            DEVICE R15,C79,'*'
            CACHE_MAP_REV 7 I/O
            EXIT_CODE OK
        END
        EXIT_CODE FAIL
    END

    ;       ********************************************************************
    ;       ********************************************************************
    ;       *                       SYSTEM INITIALISE
    ;       ********************************************************************
    ;       ********************************************************************

    ;
    ; INITIALIZATION PARAMETERS FOR SIMLATOR
    ;
    IF_PARM 1=~~LIB_CHECK
    BEGIN
        META IF_SIMULATOR 16C84    : BEGIN : EXIT_CODE OK : END
        EXIT_CODE Failed
    END


    ;
    ; INITIALIZATION PARAMETERS FOR PROCESSOR SPEED
    ;
    IF_PARM 1=~~SPEED_CHECK
    BEGIN
        IF_SIMULATOR 16C84
        BEGIN
            META IF_PARM 2=10MHZ  : BEGIN : EXIT_CODE 400 : END
            META IF_PARM 2=5MHZ   : BEGIN : EXIT_CODE 800 : END
            META IF_PARM 2=4MHZ   : BEGIN : EXIT_CODE 1000 : END
        END
        EXIT_CODE 2000
    END


    ;       ********************************************************************
    ;       ********************************************************************
    ;       *                       HARDWARE INITIALISE
    ;       ********************************************************************
    ;       ********************************************************************


    IF_PARM 1=~~CONFIG
    BEGIN

      META IF_PARM 2=00 : BEGIN : EXIT_CODE .TYPE INTERNALS : END
      META IF_PARM 2=01 : BEGIN : EXIT_CODE .PHYSICAL RO# 40 INTERNALS : END
      META IF_PARM 2=02 : BEGIN : EXIT_CODE .PHYSICAL END : END


      META IF_PARM 2=03 : BEGIN
        META EXIT_CODE .MODIFY 01 00 00 X RW% <PCH> : END
      META IF_PARM 2=04 : BEGIN
        META EXIT_CODE .MODIFY 01 01 01 X RW% <PCL> : END
      META IF_PARM 2=05 : BEGIN : IF_SIMULATOR TEST : BEGIN
        META EXIT_CODE .MODIFY 01 02 02 RW RW% <W>        : END
        META EXIT_CODE .MODIFY 01 02 02 X  RW% <W>        : END

      META IF_PARM 2=06 : BEGIN
        META EXIT_CODE .MODIFY 01 03 03 X RW% <Bus Byte> : END
      META IF_PARM 2=07 : BEGIN
        META EXIT_CODE .MODIFY 01 04 04 X RW% <Addr  Hi> : END
      META IF_PARM 2=09 : BEGIN
        META EXIT_CODE .MODIFY 01 05 05 X RW% <Addr  Lo> : END
```

```
1317      META IF_PARM 2=0A : BEGIN
1318        META EXIT_CODE .MODIFY 01 06 06 X RW% <Stack Pointer> : END
1319      META IF_PARM 2=0B : BEGIN
1320        META EXIT_CODE .MODIFY 01 07 07 X RW% <PC_High_Mask> : END
1321      META IF_PARM 2=0C : BEGIN
1322        META EXIT_CODE .MODIFY 01 08 08 X RW% <FSR_Bank_Mask> : END
1323      META IF_PARM 2=0D : BEGIN
1324        META EXIT_CODE .MODIFY 01 09 0F X UD- <Undefined> : END
1325
1326
1327      META IF_PARM 2=0E : BEGIN
1328        META EXIT_CODE .MODIFY 01 10 10 X RW% <Stack0.0>  : END
1329      META IF_PARM 2=0F : BEGIN
1330        META EXIT_CODE .MODIFY 01 11 11 X RW% <Stack0.1>  : END
1331      META IF_PARM 2=10 : BEGIN
1332        META EXIT_CODE .MODIFY 01 12 12 X RW% <Stack1.0>  : END
1333      META IF_PARM 2=11 : BEGIN
1334        META EXIT_CODE .MODIFY 01 13 13 X RW% <Stack1.1>  : END
1335      META IF_PARM 2=12 : BEGIN
1336        META EXIT_CODE .MODIFY 01 14 14 X RW% <Stack2.0>  : END
1337      META IF_PARM 2=13 : BEGIN
1338        META EXIT_CODE .MODIFY 01 15 15 X RW% <Stack2.1>  : END
1339      META IF_PARM 2=14 : BEGIN
1340        META EXIT_CODE .MODIFY 01 16 16 X RW% <Stack3.0>  : END
1341      META IF_PARM 2=15 : BEGIN
1342        META EXIT_CODE .MODIFY 01 17 17 X RW% <Stack3.1>  : END
1343      META IF_PARM 2=16 : BEGIN
1344        META EXIT_CODE .MODIFY 01 18 18 X RW% <Stack4.0>  : END
1345      META IF_PARM 2=17 : BEGIN
1346        META EXIT_CODE .MODIFY 01 19 19 X RW% <Stack4.1>  : END
1347      META IF_PARM 2=18 : BEGIN
1348        META EXIT_CODE .MODIFY 01 1A 1A X RW% <Stack5.0>  : END
1349      META IF_PARM 2=19 : BEGIN
1350        META EXIT_CODE .MODIFY 01 1B 1B X RW% <Stack5.1>  : END
1351      META IF_PARM 2=1A : BEGIN
1352        META EXIT_CODE .MODIFY 01 1C 1C X RW% <Stack6.0>  : END
1353      META IF_PARM 2=1B : BEGIN
1354        META EXIT_CODE .MODIFY 01 1D 1D X RW% <Stack6.1>  : END
1355      META IF_PARM 2=1C : BEGIN
1356        META EXIT_CODE .MODIFY 01 1E 1E X RW% <Stack7.0>  : END
1357      META IF_PARM 2=1D : BEGIN
1358        META EXIT_CODE .MODIFY 01 1F 1F X RW% <Stack7.1>  : END
1359      META IF_PARM 2=1E : BEGIN
1360        META EXIT_CODE .MODIFY 01 20 2F X RW% <TBA>       : END
1361
1362      META IF_PARM 2=1F : BEGIN : EXIT_CODE .MODIFY END : END
1363      META IF_PARM 2=20 : BEGIN
1364        META EXIT_CODE .MAPPED 01 00 00 40  <Internals Map 1> : END
1365      META IF_PARM 2=21 : BEGIN : EXIT_CODE .MAPPED END : END
1366
1367
1368      META IF_PARM 2=22 : BEGIN : EXIT_CODE .TYPE REGISTERS : END
1369      META IF_PARM 2=23 : BEGIN : EXIT_CODE .PHYSICAL RW# 200 REGISTERS : END
1370      META IF_PARM 2=24 : BEGIN : EXIT_CODE .XPHYSICAL INTERNALS 01 : END
1371      META IF_PARM 2=25 : BEGIN : EXIT_CODE .PHYSICAL END : END
1372
1373    ;           PAGE 0 SET Address 000 - 07FH
1374
1375      META IF_PARM 2=26 : BEGIN : EXIT_CODE .MODIFY 01 00 00 X RW% <Ind Addr> : END
1376      META IF_PARM 2=27 : BEGIN : EXIT_CODE .MODIFY 01 01 01 X RW% <TMR0    > : END
1377    ; META IF_PARM 2=27 : BEGIN : EXIT_CODE .MODIFY 01 02 02 X RW% <PCL     > : END
1378      META IF_PARM 2=28 : BEGIN : IF_SIMULATOR TEST : BEGIN
1379        META EXIT_CODE .MODIFY 01 03 03 RW RW% <STATUS  > : END
1380        META EXIT_CODE .MODIFY 01 03 03 X  RW% <STATUS  > : END
1381      META IF_PARM 2=29 : BEGIN : EXIT_CODE .MODIFY 01 04 04 X RW% <FSR     > : END
1382      META IF_PARM 2=2A : BEGIN : EXIT_CODE .MODIFY 01 05 05 X RW% <PORTA   > : END
1383      META IF_PARM 2=2B : BEGIN : EXIT_CODE .MODIFY 01 06 06 X RW% <PORTB   > : END
1384      META IF_PARM 2=2C : BEGIN : EXIT_CODE .MODIFY 01 07 07 X UD% <NOT Imp > : END
1385      META IF_PARM 2=2D : BEGIN : EXIT_CODE .MODIFY 01 08 08 X RW% <EEDATA  > : END
1386      META IF_PARM 2=2E : BEGIN : EXIT_CODE .MODIFY 01 09 09 X RW% <EEADR   > : END
1387      META IF_PARM 2=2F : BEGIN : EXIT_CODE .MODIFY 01 0A 0A X RW% <PCLATH  > : END
1388      META IF_PARM 2=30 : BEGIN : EXIT_CODE .MODIFY 01 0B 0B X RW% <INITCON > : END
1389      META IF_PARM 2=31 : BEGIN : IF_SIMULATOR TEST : BEGIN
1390        META EXIT_CODE .MODIFY 01 0C 2F RW RW% <GPR SRAM> : END
1391        META EXIT_CODE .MODIFY 01 0C 2F X  RW% <GPR SRAM> : END
1392      META IF_PARM 2=32 : BEGIN : EXIT_CODE .MODIFY 01 30 7F X UD% <NOT Imp > : END
1393
1394    ;           PAGE 1 SET Address 080 - 0FFH
1395
1396      META IF_PARM 2=33 : BEGIN : EXIT_CODE .MODIFY 01 81 81 X RW% <OPTION  > : END
1397      META IF_PARM 2=34 : BEGIN : EXIT_CODE .MODIFY 01 85 85 X RW% <TRISA   > : END
1398      META IF_PARM 2=35 : BEGIN : EXIT_CODE .MODIFY 01 86 86 X RW% <TRISB   > : END
1399      META IF_PARM 2=36 : BEGIN : EXIT_CODE .MODIFY 01 88 88 X RW% <EECON1  > : END
1400      META IF_PARM 2=37 : BEGIN : EXIT_CODE .MODIFY 01 89 89 X RW% <EECON2  > : END
1401      META IF_PARM 2=38 : BEGIN : EXIT_CODE .MODIFY 01 B0 FF X UD% <NOT Imp > : END
1402
1403      META IF_PARM 2=39 : BEGIN : EXIT_CODE .MODIFY END : END
1404      META IF_PARM 2=3A : BEGIN
1405      META EXIT_CODE .MAPPED 01  00  00 002  <Register Bank 0> : END
1406      META IF_PARM 2=3B : BEGIN
1407      META EXIT_CODE .MAPPED 02  01  02 001  <Re_Map   PCL   > : END
1408      META IF_PARM 2=3C : BEGIN
1409      META EXIT_CODE .MAPPED 01  03  03 07D  <Register Bank 0> : END
1410      META IF_PARM 2=3D : BEGIN
1411      META EXIT_CODE .MAPPED 01  00  80 001  <Re-Map   Bank 1> : END
1412      META IF_PARM 2=3E : BEGIN
1413      META EXIT_CODE .MAPPED 01  81  81 001  <OPTION>          : END
1414
1415      META IF_PARM 2=3F : BEGIN
1416      META EXIT_CODE .MAPPED 02  01  82 001  <Re_Map   PCL   > : END
1417      META IF_PARM 2=40 : BEGIN
1418      META EXIT_CODE .MAPPED 01  02  83 002  <Re-Map   Bank 1> : END
1419      META IF_PARM 2=41 : BEGIN
1420      META EXIT_CODE .MAPPED 01  85  85 002  <TRISA TRISB>     : END
1421      META IF_PARM 2=42 : BEGIN
1422      META EXIT_CODE .MAPPED 01  07  87 001  <Re-Map   Bank 1> : END
1423      META IF_PARM 2=43 : BEGIN
1424      META EXIT_CODE .MAPPED 01  88  88 002  <EECON1 & 2       : END
1425      META IF_PARM 2=44 : BEGIN
1426      META EXIT_CODE .MAPPED 01  0A  8A 075  <Re-Map   Bank 1> : END
```

```
1427      META IF_PARM 2=45 : BEGIN
1428      META EXIT_CODE .MAPPED 01 100 100 07F  <Register Banks 2,3> : END
1429      META IF_PARM 2=46 : BEGIN : EXIT_CODE .MAPPED END : END
1430
1431      META IF_PARM 2=47 : BEGIN : EXIT_CODE .TYPE VECTORS : END
1432      META IF_PARM 2=48 : BEGIN : EXIT_CODE .PHYSICAL RO# 10 VECTORS : END
1433      META IF_PARM 2=49 : BEGIN : EXIT_CODE .PHYSICAL END : END
1434      META IF_PARM 2=4A : BEGIN
1435        META EXIT_CODE .MODIFY 01 02 06 X UD- <Undefined> : END
1436      META IF_PARM 2=4B : BEGIN : EXIT_CODE .MODIFY END : END
1437      META IF_PARM 2=4C : BEGIN
1438        META EXIT_CODE .MAPPED 01 00 00 10  <Vector Map 1> : END
1439      META IF_PARM 2=4D : BEGIN : EXIT_CODE .MAPPED END : END
1440
1441      META IF_PARM 2=4E : BEGIN : EXIT_CODE .TYPE ISPACE 2 : END
1442      META IF_PARM 2=4F : BEGIN : EXIT_CODE .PHYSICAL RO# 1000 ISPACE : END
1443      META IF_PARM 2=50 : BEGIN : EXIT_CODE .PHYSICAL END : END
1444
1445      META IF_PARM 2=51 : BEGIN
1446        META EXIT_CODE .MAPPED 01 00 00 1000 <14 Bit Instructions> : END
1447      META IF_PARM 2=52 : BEGIN : EXIT_CODE .MAPPED END : END
1448
1449      META IF_PARM 2=53 : BEGIN : EXIT_CODE .TYPE END_PROCESSOR_CONFIG : END
1450    ; Exit Code is always the parameter number of last config statement in HEX
1451      META IF_PARM 2=COUNT : BEGIN : EXIT_CODE 53 : END
1452    END
1453
1454
1455
1456    IF_PARM 1=~~DEVICE_CONFIG
1457    BEGIN
1458
1459      META IF_PARM 2=00 : BEGIN : EXIT_CODE .TYPE EXTERNAL : END
1460      META IF_PARM 2=01 : BEGIN : EXIT_CODE .PHYSICAL RO# 20 EXTERNAL : END
1461      META IF_PARM 2=02 : BEGIN : EXIT_CODE .PHYSICAL END : END
1462      META IF_PARM 2=03 : BEGIN
1463        META EXIT_CODE .MODIFY 01 00 00 X RW- <Display 1 Segments> : END
1464      META IF_PARM 2=04 : BEGIN
1465        META EXIT_CODE .MODIFY 01 01 01 X RW- <Display 2 Segments> : END
1466      META IF_PARM 2=05 : BEGIN
1467        META EXIT_CODE .MODIFY 01 02 02 X RW- <Display 3 Segments> : END
1468      META IF_PARM 2=06 : BEGIN
1469        META EXIT_CODE .MODIFY 01 03 03 X RW- <Display 4 Segments> : END
1470      META IF_PARM 2=07 : BEGIN
1471        META EXIT_CODE .MODIFY 01 04 04 X RW- <Display 1 Bright Level> : END
1472      META IF_PARM 2=08 : BEGIN
1473        META EXIT_CODE .MODIFY 01 05 05 X RW- <Display 2 Bright Level> : END
1474      META IF_PARM 2=09 : BEGIN
1475        META EXIT_CODE .MODIFY 01 06 06 X RW- <Display 3 Bright Level> : END
1476      META IF_PARM 2=0A : BEGIN
1477        META EXIT_CODE .MODIFY 01 07 07 X RW- <Display 4 Bright Level> : END
1478      META IF_PARM 2=0B : BEGIN
1479        META EXIT_CODE .MODIFY 01 08 1E X RO% <Unused> : END
1480      META IF_PARM 2=0C : BEGIN
1481        META EXIT_CODE .MAPPED 01 00 00 20  <4 * 7 Segment Map> : END
1482
1483      META IF_PARM 2=0D : BEGIN : EXIT_CODE .MAPPED END : END
1484      META IF_PARM 2=0E : BEGIN : EXIT_CODE .TYPE END_DEVICE_CONFIG : END
1485
1486      META IF_PARM 2=COUNT : BEGIN : EXIT_CODE 0E : END
1487    END
1488
1489    RESTART LIBRARY_END
1490    ;
1491    ;
1492    ;*************************************************************
1493    ;       Library Linking Section
1494    ;*************************************************************
1495    ;
1496    BEGIN LIBRARY_END
1497    ;
1498    ; This is the Library Extention Linking Point   (A dummy access point)
1499    ;
1500    END LIBRARY_END
1501    ;
1502    ;*************************************************************
1503    ;       End of Library
1504    ;*************************************************************
1505    ;
```

Document End

This document was created on 23/August/2011 as reproduction of the original report created by May 2002 and submitted as part of
The Applied Computing and Electronic BEng Hons) Course
held at Bournemouth University.

Some entries may have adjusted dates due to this documents recent production and also slightly differing page lengths as a result of using alternative development software. for print production.