A Project Report on

Microcontroller Based Control and Monitoring the Radio Telescope Receiver System

By

Nandish K.C.
1KN01EC025

Kishan J.
1KN01EC052

Submitted to

Visveswaraiah Technological University, Belgaum

in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Electronics and Communication

Project work carried out at

Raman Research Institute
Bangalore – 560080

Internal Guide
Mrs. Minnie Alfred
Lecturer, Dept. of E & C
K.N.S.I.T

External Guide
Prof. A. Raghunathan
Radio Astronomy Lab

Department of Electronics and Communication
K. N. S. Institute of Technology
Bangalore-560064

June 04, 2005

Certificate

This is to certify that the project work entitled

"Microcontroller-based Control and Monitoring System for a Radio Telescope Receiver"

was carried out under my guidance by

Nandish K.C. (1KN01EC025)
Kishan J. (1KN01EC052)

students of the VIII semester, Bachelor of Engineering (Electronics and Communication Engineering), KNS Institute of Technology, Bangalore, in partial fulfillment of the requirement for the award of Bachelor's Degree in Electronics and Communication Engineering during the semester March 2005 – May 2005.

A. Raghunathan

A. Raghunathan
Engineer
Radio Astronomy Laboratory

Declaration

Project Title: ***"Microcontroller Based Control and Monitoring the Radio Telescope Receiver System"***

The project dissertation is submitted in partial fulfilment of academic requirements for Bachelor of Engineering in Electronics and Communication. This dissertation is a result our own investigation and no part of it have been submitted for any degree or diploma of any institution previously. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Nandish K. C.

Kishan J.

Acknowledgement

We thank Prof. N.Kumar, Director of Raman Research Institute for giving us an opportunity to work in RRI.

We thank Prof. Kumara Velu, HOD, Dept. of Electronics and Communication, KNSIT for giving us permission to use this opportunity.

We thank Dr. A.K. Khargekar, Principal, KNSIT, for his valuable suggestions.

We thank Dr. N. Udaya Shankar, Incharge, Radio Astronomy Lab, RRI for his constant encouragement.

We express our sincere gratitude and profound thanks to our guide Mr. A Raghunathan, Radio Astronomy Lab, RRI for his guidance.

We are indebted to Mrs. Minnie Alfred, Lecturer, Dept. of E & C, internal guide for helping us each and every moment of the project by providing invaluable suggestions.

We sincerely thank Mr. K.B Raghavendra for giving constant support throughout the project. We thank Mr. Aras and Mr. Somshekhar for their useful suggestions.

We thank Mr. Patil – Librarian for providing us the required reading material needed for our project.

Finally we express out sincere thanks to those who are directly and indirectly helpful in carrying out the project at the Raman Research Institute.

Contents

List of Figures

List of Tables

Abstract

This project deals with controlling and monitoring the signals received at the front end receiver of radio telescope. Some celestial bodies in space emit electromagnetic radiations (radio waves). These radio waves are one million longer than light waves and are weak. The total power received from all observed radio sources at all observatories throughout history is scarcely enough to strike a match. A typical radio signal has a power of only $2x10^{-15}W$.

The 4 - 8 Ghz front end receiver is being built for on going 12 m Radio Telescope project at RRI. In the front end receiver, the received noise buried RF signal is amplified in a Low noise amplifier having noise temperature of $60 - 70^{o}$ K. The amplified RF signal is spectrally divided into 4 filter bands having center frequencies 4.1 Ghz (BW=800 MHz), 5.2 Ghz (BW=1Ghz), 6.68 Ghz (BW=50Mhz) and 8Ghz (BW=800 MHz).

Any one of the 4 filters will be selected and the selected RF band is downconverted to 1st IF of 1.4Ghz(BW=500Mhz) in 1st IF Section. Then it is further downconverted to 60Mhz in the 2nd IF Section. This is the analog system development part of our receiver system.

In this project we are dealing with the digital system to control and monitor the Front end receiver system. Basically our system contains a PC, microcontroller ADuC812, Xilinx CPLD(X19572),Altera EPLD (EPM7064) and analog multiplexers (ADG507A).

Chapter 1 - Introduction to Microcontrollers

1.1 Introduction

Microprocessor is a multipurpose programmable clock driver register based electronic device. It accepts binary data as input, processes the data according to the instructions given and produces the output. It is a general-purpose digital computer that has a central processing unit (CPU). It has to work along with the peripheral devices like input/output devices, memory to function as a complete digital computer.

Following the development of the microprocessor microcontroller was developed incorporating all the necessary devices in it. In principle, a microcontroller chip can function as a computer. It has all the features of a typical microprocessor and in addition it has also memory, both RAM and ROM, parallel I/O, serial I/O, counters and a clock. The primary use of a microcontroller is to (read data, perform limited operations) control the operations of a machine using a fixed program stored in it.

1.2 Salient Features of Microcontroller

Microcontrollers, as the name suggests, are small controllers. They are like single chip computers that are often embedded into other systems to function as processing or controlling unit. A microcontroller may take input from the device it's controlling and controls the device by sending signals to different components in the device. A microcontroller is often small and low cost. The components may be chosen appropriately for the application it's being used for and also see to that its size and cost be minimum as possible.

The key features of microcontrollers include:

- **High Integration of Functionality**

Microcontrollers are sometimes called single chip computers because they have on-chip memory and I/O circuitry and other circuitries that enable them to function as small standalone computers without other supporting circuitry.

- **Field Programmability, Flexibility**

Microcontrollers often use EEPROM or EPROM as their storage device to allow field programmability so they are flexible to use. Once the program is tested to be correct then large quantities of microcontrollers can be programmed to be useful in embedded systems.

- **Easy to Use**

Assembly language is often used in microcontrollers and since they usually follow RISC architecture, the instruction set is small. The development package of microcontrollers often includes an assembler, a simulator, a programmer to burn the chip and a demonstration board. Some packages include a high-level language compiler such as a C compiler and more sophisticated libraries.

1.3 Basic Block Diagram of a Microcontroller

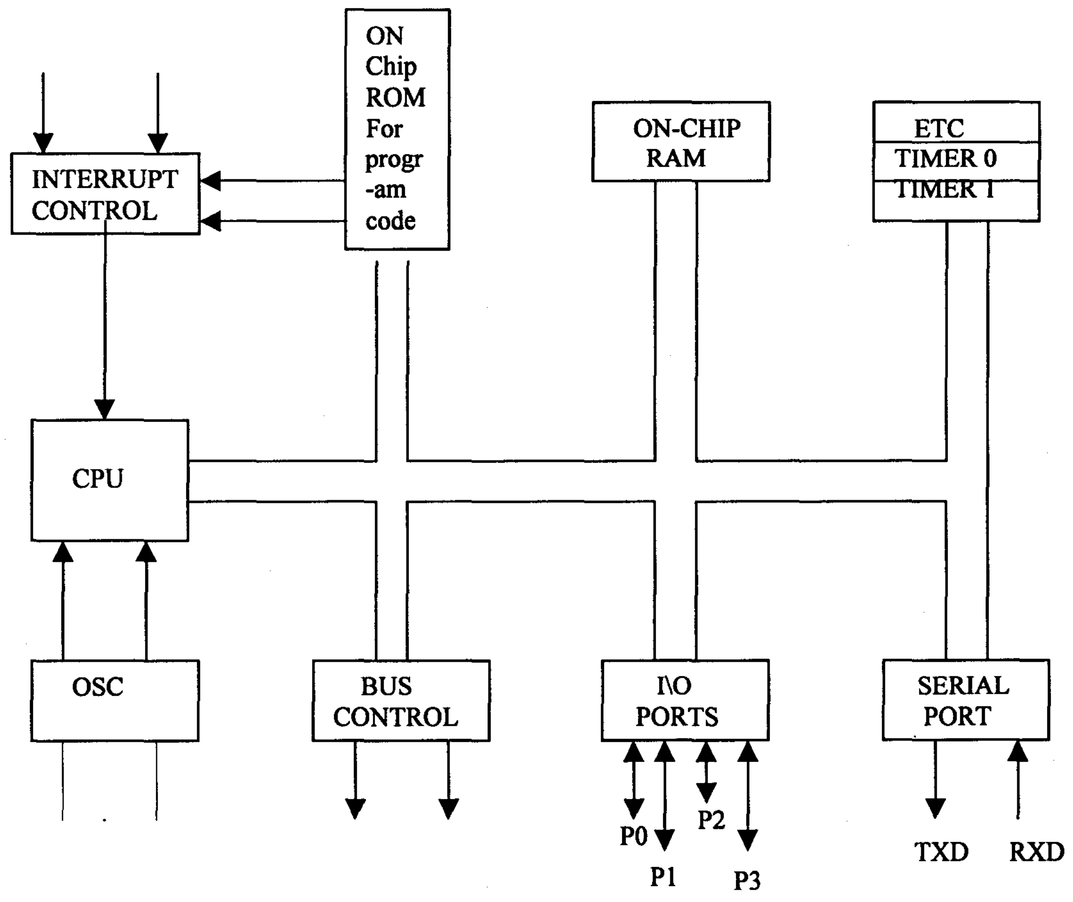


Figure 1-1: Block Diagram of Microcontroller

1.3.1 Memory Uuit

Memory is part of the microcontroller whose function is to store data. It consists of RAM and ROM as shown in the figure above. The function of ROM is to provide information that is fixed and permanent. ROM is also known as non-volatile memory since the data stored in it is permanent even when the power is switched off. Due to this reason it is used in microcontrollers to store the code and some important data. It usually comes in packages of 640 bytes, 1KB, and so on depending upon the microcontroller.

In contrast, RAM is used to store information that is not permanent and can change with time. This is used by the computer for storing temporary data during its operation.

The data stored in RAM is lost when the power is switched off. Hence it is also known as volatile memory. It is mainly used to store data and some variables during the execution of a program.

1.3.2 Central Processing Uuit

The CPU is the heart of the microcontroller. It contains registers, ALU, program counter and an instruction decoder. The registers are used as temporary storage locations for the data, which is being processed by the CPU. It may be a result after the execution of an instruction.

Registers may be 8-bit, 16-bit, 32-bit, 64-bit depending upon the CPU. The ALU section of the CPU is responsible for performing arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT.

The function of the program counter is to point to the address of the next instruction to be executed. As each instruction is being executed, the program counter is incremented to point to the next instruction to be executed.

The instruction decoder is used to interpret the instruction being fetched into the CPU.It decodes the instruction in such a way such that the CPU understands what type of operation is to be done.

1.3.3 Bus

It represents a group of 8, 16, or more wires. There are two types of buses, address and data bus. The address bus is used to access the different memory locations required by the CPU to execute instructions. The data bus is used to move around data between various blocks of the microcontroller as shown in the block diagram.

1.3.4 Timer Unit

The timer unit usually consists of tow timers. They can be used either as timers or as event counters. In timer mode, the timer register is incremented in equal intervals depending upon the internal clock frequency of the microcontroller.

In counter mode, the timer registers will be incremented depending upon the external clock given to the microcontroller. This is done usually when it has been interfaced to some other external device. Timers are also responsible for serial communication of the microcontroller to other external devices such as a PC.

1.3.5 Input Output Unit

The input/output ports are basically the circuits used to connect the microcontroller to the outside world. Each port has a special function register that is made up of latches. Each pin of that port can be addressed at the SFR address. These ports are used to control external devices that have been interfaced to the microcontroller.

1.3.6 Interrupt Controls

During the execution of the program there will be jumps from the main program to particular subroutines depending upon the status of the flags and port pins or either when the particular flags are set or hardware interrupt signals are encountered. Interrupts are often the only way in which real-time programming can be done successfully.

After the interrupt has been handled by the interrupt subroutine, which is placed by the programmer at the interrupt location on program memory, the interrupted program resumes it's operation at the instruction where the interruption had taken place.

1.3.7 Serial Port

The serial port is used to establish communication between the microcontroller and other devices such as a PC. The data is transmitted and received serially through this port. The baud rate for the data transfer through the serial port can be set using the special function registers allocated for serial communication. It usually consists of a RXD and TXD pins which are the receive and transmit pins respectively, and are connected to the serial data network.

1.3.8 Oscillator and Clock

The clock circuitry is the one that generates the clock pulses by which all internal operations are synchronized. Pins XTAL1 and XTAL2 are provided for connecting a resonant network to form an oscillator. It usually consists of a quartz crystal and capacitors. The crystal frequency is usually the basic internal clock frequency of the microcontroller.

1.4 Classification of Microcontrollers

The microcontrollers are classified as 8-bit, 16-bit, 32-bit and 64-bit microcontrollers.

By a 16-bit or a 32-bit microcontroller we mean that the microcontroller can process a 16-bit data at once, or in other words the accumulator of the microcontroller is 16-bit or 32-bit wide. This means that the microcontroller can make operations such as addition, subtraction, etc on a 16-bit data at once. The most common microcntrollers used are 8-bit microcontrollers.

The different types of microcontrollers are:

- **Four-Bit Microcontroller:**

Eight-bit microcontrollers represent a transition zone between the dedicated, high volume, 4-bit microcontrollers and the high-performance, 16 and 32-bit devices. Eight bits has proven to be a very useful word size for most of the computing tasks.

The one-byte word is adequate for many control and monitor applications. Serial ASCII data is also stored in byte sizes, making 8 bits the natural choice for data communication. The 8-bit configuration interfaces easily to that of the data buses, which is also 8-bit wide. Most of the memory used in microcontrollers like EEPROM, EPROM are also 8-bit wide. (Each location is 8-bit wide). Hence an 8-bit microcontroller would be very convenient and efficient to access the memory.

- **Eight-Bit Microcontrollers:**

Eight-bit microcontrollers represent a transition zone between the dedicated, high volume, 4-bit microcontrollers and the high-performance, 16 and 32-bit devices. Eight bits has proven to be a very useful word size for most of the computing tasks. The one-byte word is adequate for many control and monitor applications. Serial ASCII data is also stored in byte sizes, making 8 bits the natural choice for data communication. The 8-bit configuration interfaces easily to that of the data buses, which is also 8-bit wide.

Most of the memory used in microcontrollers like EEPROM, EPROM are also 8-bit wide. (Each location is 8-bit wide). Hence an 8-bit microcontroller would be very convenient and efficient to access the memory.

- **Sixteen-Bit Microcontroller:**

Eight-bit microcontrollers can be used in a variety of applications that involve limited calculations and relatively simple strategies. As the requirement for faster response and more sophisticated calculations grows, the 8-bit device's efficiency become limited. So the solution to this is to increase the clock speed, or to increase the size of the data word.

Sixteen-bit microcontrollers have evolved to solve high-speed control problems. The 16-bit controllers have also been designed to take advantage of high-level programming languages in the expectation that very little assembly language programming will be done when employing these controllers in sophisticated applications.

- **Thirty-two Bit Microcontrollers:**

The 32-bit microcontroller designs target mainly robotics, highly intelligent instrumentation, avionics, image processing, telecommunications, automobiles and other fields of applications that require an operating system for its functioning.

The design emphasis now switches from on-chip features, such as RAM, ROM, timers, and serial ports to high-speed computation features.

1.5 Criteria for Choosing a Microcontroller

1) The main criterion for choosing a microcontroller is that it must meet the task at hand efficiently and must be cost effective. We must also see that whether an 8-bit, 16-bit, or 32-bit microcontroller can besthandle the computing needs of the task most effectively.

2) The second criterion in choosing a microcontroller is to see how easy it is to develop products around it. This includes the availability of an assembler, debugger, a code efficient C compiler, emulator, and technical support.

3) The third criterion in choosing a microcontroller is its availability readily in the market both, at present and in the future.

Chapter 2 – Description of AduC812 Microcontroller Card

A microcontroller card based on AduC812 has been used as part of 4 – 8 GHz Control and Monitoring system. This card is used basically as an interface card between the computer and the receiver system.

The main functions of this card are:

1. It sets all the required parameter of the receiver system.
2. It digitises all the monitoring analog signals of the receiver system.
3. It provides a number of I/O lines for controlling peripheral devices.

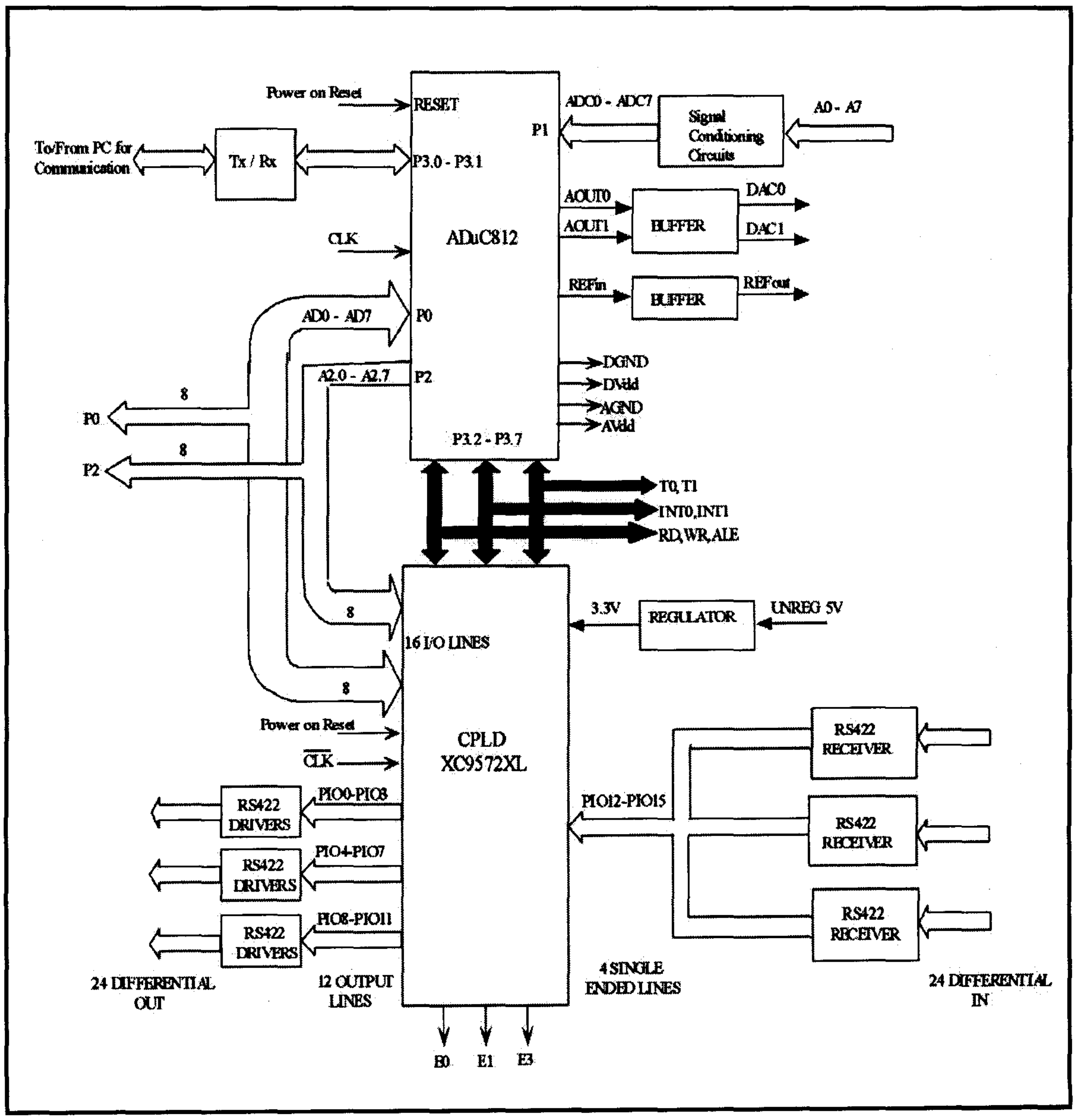


Figure 2-1: Block Diagram of the ADuC812 Card

2.1 Salient Features of Microcontroller Card

- **Serial Communication in the Card**

During transmitting phase of the data ie: from PC to Micro Controller card, the RS232 logic (+ 12V) from PC is converted to the TTL by MAX232 IC and then level translated to RS422 logic (+ 9V differential), having differential mode of signal enables us to communicate with a peripheral device situated at a longer distance from PC. The differential mode is converted back to single (RS232) by the microcontroller card. TxD and RxD are the 2 terminals of port 3 of microcontroller used for serial communication.

- **Clock Oscillator of the Microcontroller**

A crystal is used to generate clock for the microcontroller card. The crystal resonates at 11.0592MHz. A parallel resonant crystal is connected between XTAL1 and XTAL2 of the ADUC812 Micro Controller, as shown in the Figure 2-2.

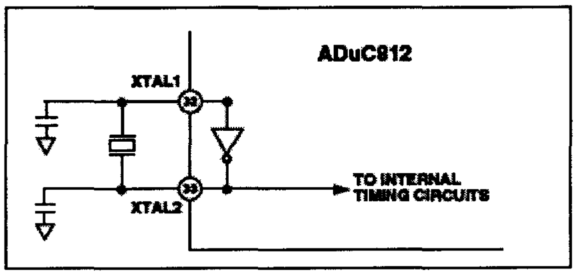


Figure 2-2: External Parallel Resonant Crystal Connections

- **Power On Reset**

External power on reset circuitry is implemented to drive the reset pin of the ADUC812. The circuit will hold reset pin asserted whenever the power supply (DVDD) is below 2.5 V. In the card we configured an active power on reset chip with a manual push-button as an additional reset source.

- **Inputs and Outputs of ADUC812 Micro Controller**

ADUC812 Micro Controller has four ports i.e. P0, P1, P2 and P3 where each port is of 8 bits. The 8 pins of bi-directional P0 (AD0 to AD7), the 8 pins of unidirectional P2 (A2.0 to A2.7), the RD, WR, INT1, INT0, T0 and T1 pins of the bi-directional P3, ALE and PSEN of the Micro Controller are brought out for the user interface. Whenever the code is downloaded into the micro controller, the active low PSEN pin is held low.

- **TTL Signal between ADUC812 and CPLD**

The P0 and P2 port outputs of the Micro Controller are given to the I /O lines of CPLD as well as they are brought out for user interface. The controller signals, timers and the interrupt P3 are routed to CPLD where these port pins are configured as bi-directional.

In addition INT0, INT1, RD, WR, of P3 are also routed through CPLD. The four I / O pins of CPLD (PIO 0 – PIO 3) are configured to receive P2 upper nibble and other 8 I / O pins of CPLD (PIO 4 – PIO 11) are configured as P0 data lines. These port outputs of CPLD are routed through the single ended to differential (RS422 drivers) during transmitting phase. T_EN1, T_EN2 and T_EN3 are the pins of the CPLD, which enables the three RS422 drivers. During receiving phase, 24 bits differential signals are given to RS422 receivers in which they are converted into single 12 bits and are routed to the CPLD. The E1, E2, E3 are the pins of CPLD which enables the RS422 receivers.

- **Analog Inputs.**

The P1 port of ADUC812 Micro Controller is configured as an Analog Input port. It can receive 8 analog inputs, these analog inputs are signal conditioned before giving to ADC as shown in Figure 2-3.

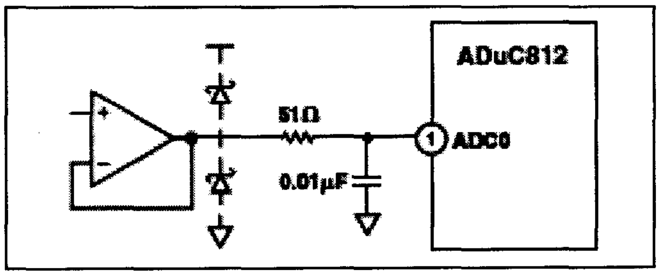


Figure 2-3: Buffering Analog Inputs

- **DAC**

The ADuc812 microcontroller has two Digital to Analog Converters (DAC0 , DAC1)in which both are 12 bit wide.

- **Power Supplies**

The ADuc812's operate under power supply voltage range of 2.7V – 5.25V. Separate analog and digital power supply lines (AVdd and DVdd respectively) allow the analog power line to be relatively free from the noisy digital signals. A separate 3.3 regulated voltage is given to CPLD.

The analog ground and the digital ground are connected together in one place on the card and brought out of the card as a single ground.

2.2 Architecture of AduC812

It is highly integrated 12-bit data acquisition system having a high performance self-calibrating multi-channel ADC, dual DAC and a programmable 8 bit MCU. It has got 8k bytes of Eeprigram memory, 640 bytes of EE data memory and 256 bytes of data SRAM on the chip. For serial I/O operation, UART and SPI are provided .It can be used for DAS and Communication systems, transient capture system etc.

2.2.1 Detailed Description of the uC812

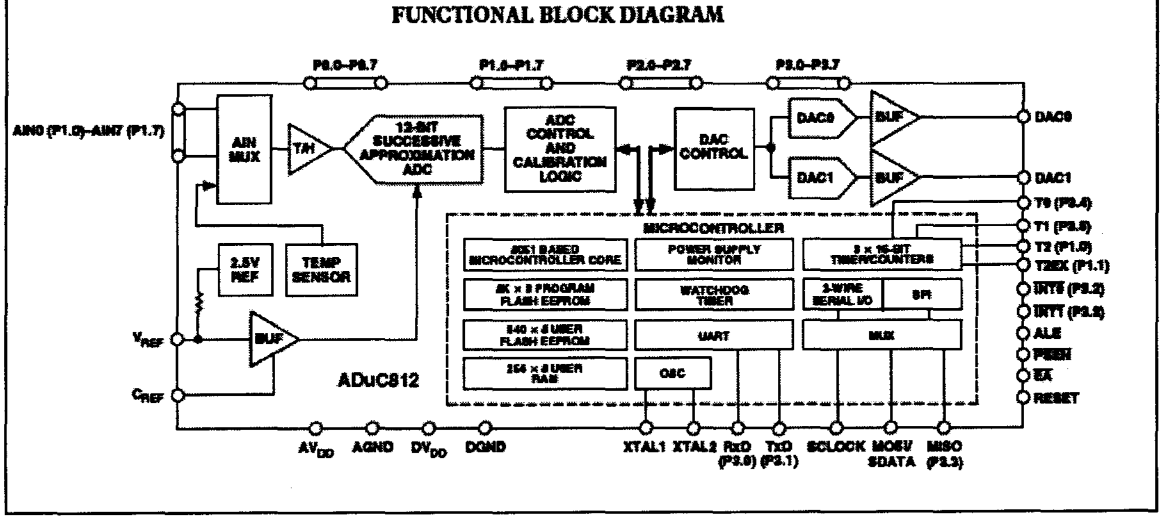


Figure 2-4: Functional Block Diagram of AduC812

The functional block diagram of the microcontroller chip is shown in the figure above. The primary components of the chip are i) 12 bit ADC ii) 2 nos. of DAC and iii) 8 bit MCU.

- **Analog to Digital Converter**

The ADC conversion block incorporates a fast, 8- channel, 12-bit, single supply A\D converter. This block provides the user with multi-channel mux, track/hold, on-chip reference, calibration features and A\D converter.

It incorporates a successive approximation (SAR) architecture involving a charge-sampled input stage. The Figure 2-5 shows the equivalent circuit of the analog input section. Each ADC conversion is divided into two distinct phases as defined by the position of the switches.

During the sampling phase (with SW1 and SW2 in the "track" position) are charge proportional to the voltage on the analog input is developed across the input sampling capacitor.

During the conversion phase (with both switches in the "hold" position) the capacitor DAC is adjusted via internal SAR logic. If the voltage on node A is zero indicating that the sampled charge on the input capacitor is balanced out by the charge being output by the capacitor DAC.

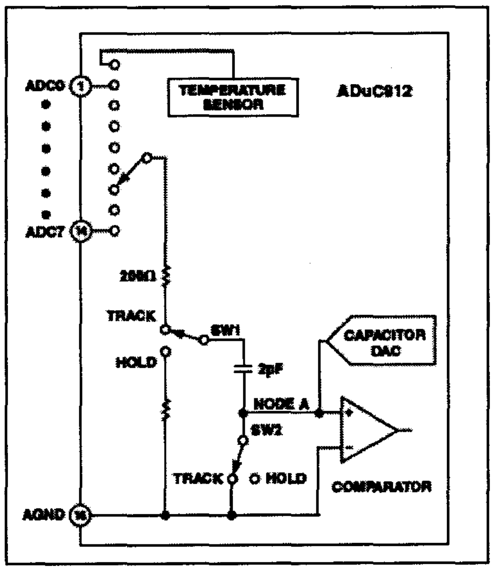


Figure 2-5: Internal ADC Structure

The digital value finally contained in the SAR is then latched out as the result of the ADC conversion. Control of the SAR, and timing of the acquisition and sampling modes,is handled automatically by. Built-in ADC control logic. Acquisition and conversion times are also fully configurable under user control.

Typical Operation

The ADC is configured via the ADCCON1-3 SFRs, the ADC will covert the analog input and Provides an ADC 12- bit result word in the ADCDATAH/L SFRs. The top 4 bits of the ADC 12-bit result word in the ADCDATAH SFR will be written with the channel selection bits to identify the channel result. The format of the ADC 12-bit result word is shown in Figure 2-6.

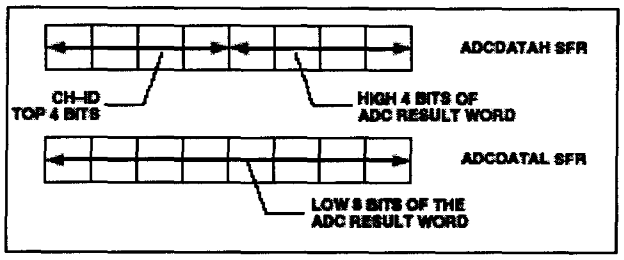

ADCDATAH SFR
CH-ID TOP 4 BITS
HIGH 4 BITS OF ADC RESULT WORD
ADCDATAL SFR
LOW 8 BITS OF THE ADC RESULT WORD

Figure 2-6: ADC Result Format

1. ADCCON 1

The ADCCON1 register controls conversion and acquisition times, hardware conversion modes and power-down modes as detailed below:

SFR Address	EFH
SFR Power-On Default Value	20H

MD1	MD0	CK1	CK0	AQ1	AQ0	T2C	EXC

2. ADCCON 2

The ADCCON2 register controls ADC channel selection and conversion modes as detailed below:

SFR Address	D8H
SFR Power On Default Value	00H

ADCI	DMA	CCONV	SCONV	CS3	CS2	CS1	CS0

3. ADCCON 3

The ADCCON3 register gives user software an indication of ADC busy status.

SFR Address	F5H
SFR Power On Default Value	00H

Here the 8th bit i.e. busy is utilized in which it is a read-only status bit that is set during a valid ADC Conversion or Calibration cycle. Busy is automatically cleared by the core at the end of Conversion or Calibration. Other bits are reserved for the future use.

Table 2-1: ADCCON 1 SFR

Bit	Name	Descriptions
7	MD1	The mode bits (MD1, MD0) select the active operating mode of the ADC as follows:
6	MD0	MD1 MD0 Active Mode 0 0 ADC powered down 0 1 ADC normal mode 1 0 ADC powered down if not executing a conversion cycle 1 1 ADC standby if not executing a conversion cycle
5	CK1	The ADC clock divide bits (CK1, CK0) select the divide ratio for the master clock used to generate the ADC clock.
4	CK0	CK1 CK0 MCLK Divider 0 0 1 0 1 2 1 0 4 1 1 8
3	AQ1	The ADC acquisition select bits (AQ1, AQ0) select the time provided for the input Track/hold amplifier
2	AQ0	AQ1 AQ0 #ADC Clks 0 0 1 0 1 2 1 0 4 1 1 8
.	T2C	The Timer 2 conversion bit (T2C) is set by the user to enable the Timer 2 overflow bit be used as the ADC convert start trigger input.
0	EXC	The external trigger enable bit (EXC) is set by the user to allow the external CONVST pin to be used as the active low convert start input

Table 2-2: ADCCON 2

Bit	Name	Description
7	ADCI	The ADC interrupt bit (ADCI) is set by hardware at the end of a single ADC conversion cycle or at the end of a DMA block conversion. ADCI is cleared by hardware when the PC vectors to the ADC Interrupt Service Routine.
6	DMA	The DMA mode enable bit (DMA) is set by the user to enable a pre configured ADC DMA mode operation. A more detailed description of this mode is given in the ADC DMA Mode section.
5	CCONV	The continuous conversion bit (CCONV) is set by the user to initiate the ADC into a continuous mode of conversion. In this mode, the ADC starts converting based on the timing and channel configuration already set up in the ADCCON SFRs; the ADC automatically starts another conversion once a previous conversion has completed.
4	SCONV	The single conversion bit (SCONV) is set to initiate a single conversion cycle. The SCONV bit is automatically reset to "0" on completion of the single conversion cycle.
3-0	CS3-CS0	The channel selection bits (CS3–0) allow the user to program the ADC channel selection under CS2 software control. When a conversion is initiated, the channel converted will be the one pointed to by CS1 these channel selection bits. CS3 CS2 CS1 CS0 CH# 0 0 0 0 0 0 0 0 1 1 0 0 1 0 2 0 0 1 1 3 0 1 0 0 4 0 1 0 1 5 0 1 1 0 6 0 1 1 1 7 1 0 0 0 Temp Sensor 1 1 1 1 DMA STOP

- **Digital to Analog Converter**

The AduC812 incorporates two 12-bit, voltage output DACs on-chip. D/A converter architecture. Consists of a resistor string DAC followed by an output buffer amplifier, the functional equivalent of which is illustrated in Figure 2-7:

Each has two selectable ranges, 0 V to VREF (the internal band gap 2.5 V reference) and 0 V to AVDD. Each can operate in 12-bit or 8-bit mode. Both DACs share a control register, DACCON, and four data registers, DAC1H/L, DAC0H/L .The 12-bit DAC data should be written into DACxH/L right justified such that DACL contains the lower 8 bits, and the lower nibble of DACH contains the upper 4 bits.

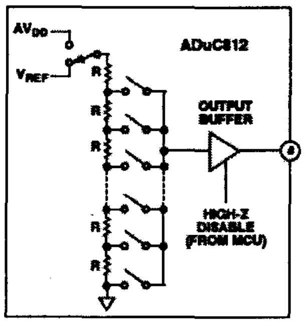


Figure 2-7: Resistor String DAC Functional Equivalent

DACCON- DAC control register

SFR Address	FDH
Power – On Default value	04H
Bit Addressable	No

MODE	RNG1	RNG0	CLR1	CLR0	SYNC	$\overline{\text{PD1}}$	PD0

Table 2-3: DACCON SFR

Bit	Name	Description
7	MODE	The DAC MODE bit sets the overriding operating mode for both DACs. Set to “1” = 8-bit mode (Write eight Bits to DACxL SFR). Set to “0” = 12-bit mode.
6	RNG1	DAC1 Range Select Bit. Set to “1” = DAC1 range 0–VDD. Set to “0” = DAC1 range 0–VREF
5	RNG0	DAC0 Range Select Bit. Set to “1” = DAC0 range 0–VDD. Set to “0” = DAC0 range 0–VREF.
4	CLR1	DAC1 Clear Bit. Set to “0” = DAC1 output forced to 0 V. Set to “1” = DAC1 output normal.
3	CLR0	DAC0 Clear Bit. Set to “0” = DAC1 output forced to 0 V. Set to “1” = DAC1 output normal.
2	SYNC	DAC0/1 Update Synchronization Bit When set to “1” the DAC outputs update as soon as DACxL SFRs are written. The user can simultaneously update both DACs by first updating the DACxL/H SFRs while SYNC is “0.” Both DACs will then update simultaneously when the SYNC bit is set to “1”.
1	PD1	DAC1 Power-Down Bit. Set to “1” = Power-on DAC1. Set to “0” = Power-off DAC1
0	PD0	DAC0 Power-Down Bit. Set to “1” = Power-on DAC0. Set to “0” = Power-off DAC0

- **Microcontroller**

1. **Flash/EE Program Memory**

The AduC812 has 8k bytes of Flash/EE program memory, which is spaced and addressed separately as shown in Figure 2-8.

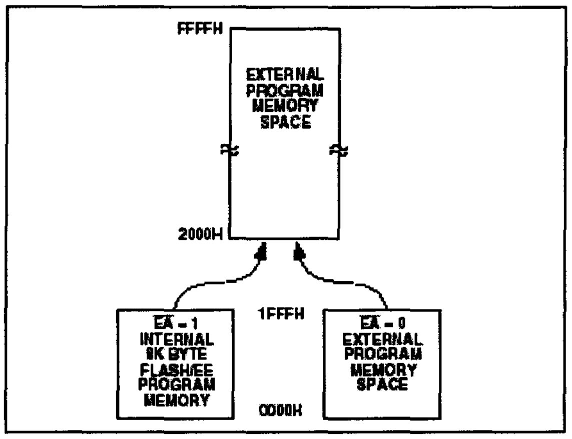

FFFFH
EXTERNAL PROGRAM MEMORY SPACE
2000H
1FFFH
0000H
EA = 1 INTERNAL 8K BYTE FLASH/EE PROGRAM MEMORY
EA = 0 EXTERNAL PROGRAM MEMORY SPACE

Figure 2-8: Program Memory Space Read Only

2. **User Flash EEPROM**

AduC812 incorporates 640x 8 user flash EEPROM which is addressed and spaced separately, as shown in Figure 2-9, this user data flash memory area is accessed indirectly via a group of control registers mapped in the SFR area in the Data memory space.

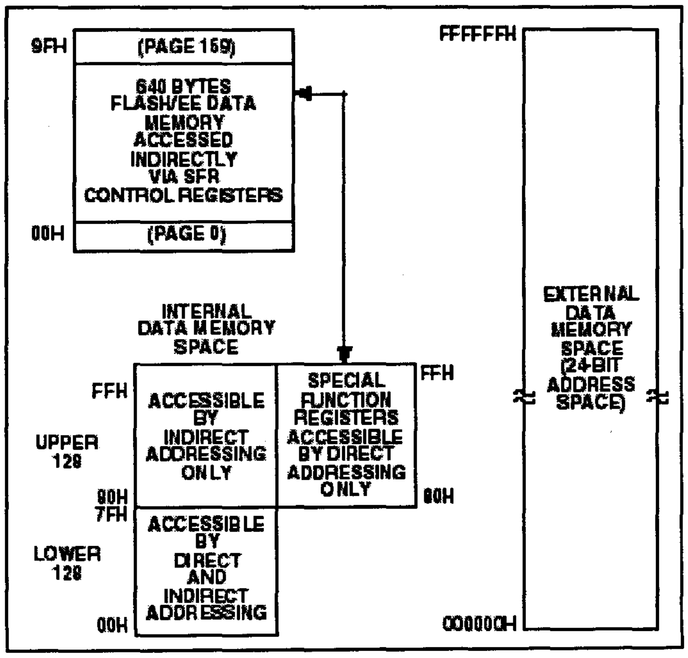

9FH
(PAGE 159)
640 BYTES FLASH/EE DATA MEMORY ACCESSED INDIRECTLY VIA SFR CONTROL REGISTERS
00H
(PAGE 0)
INTERNAL DATA MEMORY SPACE
FFH
UPPER 128
80H
7FH
LOWER 128
00H
ACCESSIBLE BY INDIRECT ADDRESSING ONLY
SPECIAL FUNCTION REGISTERS ACCESSIBLE BY DIRECT ADDRESSING ONLY
ACCESSIBLE BY DIRECT AND INDIRECT ADDRESSING
FFH
80H
FFFFFFH
EXTERNAL DATA MEMORY SPACE (24-BIT ADDRESS SPACE)
000000H

Figure 2-9: Data Memory Space Read/Write

The user Flash/EE data memory array is configured into 160(page 00H to Page 9FH), 4- byte pages. As shown in Figure 2-10.

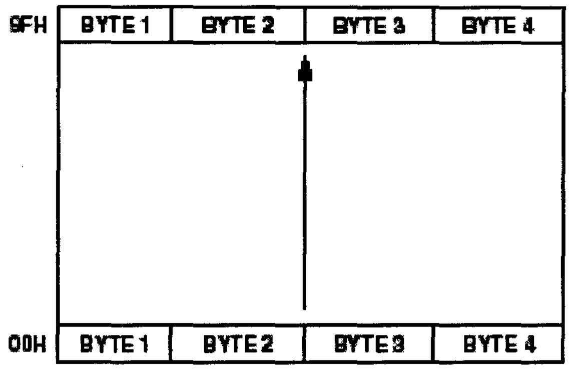


Figure 2-10: User Flash/EE memory configuration

The interface to this memory space is via a group of registers mapped in the SFR space. A group of four data registers (EDATA1-EDATA4) are used to hold the 4-byte page being accessed. EADRL is used to hold the 8-bit address of the page being accessed. Finally, ECON is an 8-bit control register that may be written with one of five Flash/EE memory access commands to trigger various read, write, erase and verify functions. These registers can be summarized as follows:

ECON: SFR Address	B9H
Function	Controls access to 640 bytes flash/EE data space
Default	00H

EADRL: SFR Address	C6H
Function	Holds the Flash/EE Data Page address.
Default	00H

EDATA 1-4: SFR Address	BCH to BFH
Function	Holda the Flash/EEData page address.
Default	EDATA1-4—00H

ECON Flash\EE memory memory control register command modes:

01H – READ COMMAND Results in four bytes being read into EDATA1–4 from memory page address contained in EADRL

02H - PROGRAM COMMAND Results in four bytes (EDATA1–4) being written to memory page address in EADRL. This write command assumes the designated "write" page has been pre-erased.

03H - RESERVED FOR INTERNAL USE 03H should not be written to the ECON SFR.

04H - VERIFY COMMAND allows the user to verify if data in EDATA1–4 is contained in page address designated by EADRL.

05H - ERASE COMMAND Results in an erase of the 4-byte page designated in EADRL.

06H - ERASE-ALL COMMAND Results in erase of the full Flash/EE data memory 160-page (640 bytes) array.

07H _ FFH – Reserved for future use.

A block diagram of the SFR registered interface to the data Flash/EE memory array is shown in Figure 2-11:

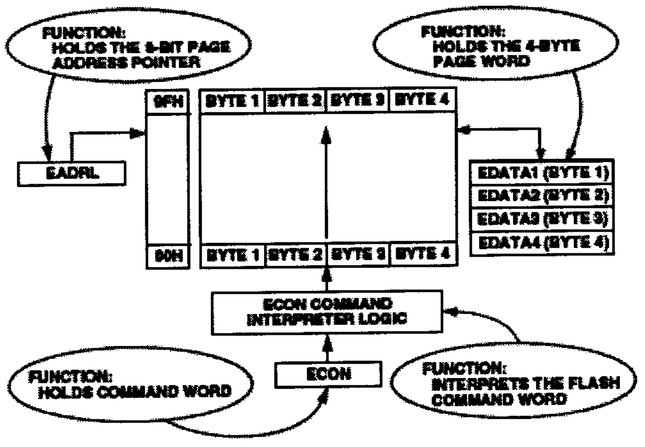


Figure 2-11: User Flash / EE Control and Configuration

3. User RAM

The AduC812 incorporates 256x8 USER RAM. The lower 128 bytes of internal data memory are mapped as shown in Figure2-12. The lowest 32 bytes are grouped into four banks of eight registers addressed as R0 through R7. The next 16 bytes (128 bits) above the register banks form a block of bit addressable memory space at bit addresses 00H through 7FH. The upper 128 bytes are accessible by indirect addressing only; it is from 80H to FFH.

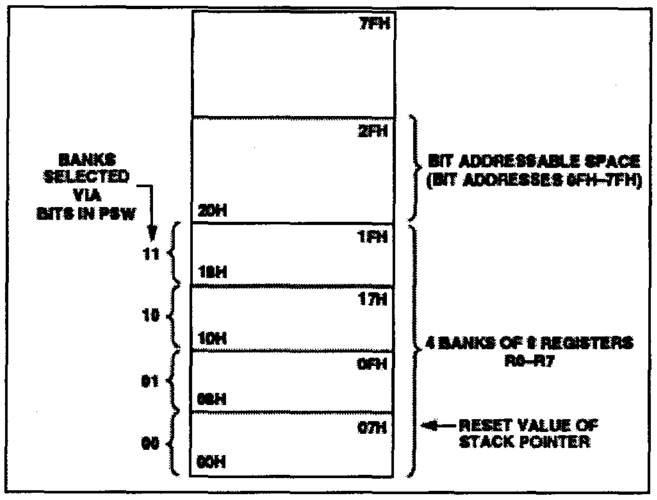

7FH
2FH
20H
1FH
18H
17H
10H
0FH
08H
07H
00H
BANKS SELECTED VIA BITS IN PSW
11
10
01
00
BIT ADDRESSABLE SPACE (BIT ADDRESSES 0FH–7FH)
4 BANKS OF 8 REGISTERS R0–R7
RESET VALUE OF STACK POINTER

Figure 2-12: Lower 128 bytes of internal RAM

4. Watch Dog Timer

The purpose of the watchdog timer is to generate a device reset within a reasonable amount of time if the ADuC812 enters an erroneous state, possibly due to a programming error. The Watchdog function can be disabled by clearing the WDE (Watchdog Enable) bit in the Watchdog Control (WDCON) SFR. When enabled, the watchdog circuit will generate a system reset if the user program fails to set the watchdog timer refresh bits (WDR1, WDR2) within a predetermined amount of time (see PRE2–0 bits in WDCON). The watchdog timer itself is a 16-bit counter. The watchdog timeout interval can be adjusted via the PRE2–0 bits in WDCON. Full Control and Status of the watchdog timer function can be controlled via the watchdog timer control SFR (WDCON).

WDCON – Watchdog Timer control register	
SFR Address	C0H
Power- on Default Value	00H
Bit Addressable	Yes

PRE2	PRE1	PRE0	—	WDR1	WDR2	WDS	WDE

Table 2-4: WDCON SFR

Bit	Name	Description			
7	PRE2	Watchdog Timer Prescale Bits.			
6-5	PRE1-PRE0	PRE2	PRE1	PRE0	Timeout Period (ms)
		0	0	0	16
		0	0	1	32
		0	1	0	64
		0	1	1	128
		1	0	0	256
		1	0	1	512
		1	1	0	1024
		1	1	1	2048
4	---	Reserved			
3-2	WDR1-WDR2	Watchdog Timer Refresh Bits. Set sequentially to refresh the watchdog timer.			
1	WDS	Watchdog Status Bit. Set by the Watchdog Controller to indicate that a watchdog timeout has occurred. Cleared by writing a "0" or by an external hardware reset. It is not cleared by a watchdog reset.			
0	WDE	Watchdog Enable Bit. Set by user to enable the watchdog and clear its counters.			

5. UART Serial Interface

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can begin receiving a second byte before a previously received byte has been read from the receive register. However, if the first byte still has not been read by the time reception of the second byte is complete, the first byte will be lost. The physical interface to the serial data network is via Pins RXD (P3.0) and TXD(P3.1) , while the SFR interface to the UART is comprised of SBUF and SCON, as described below.

SBUF: The serial port receive and transmit registers are both accessed through the SBUF SFR (SFR address = 99H). Writing to SBUF loads the transmit register and reading SBUF accesses a physically separate receive register.

SCON – UART Serial Port Control Register

SFR Address	98H
Power- On Default value	00H
Bit Addressable	Yes

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Table 2-5: SCON SFR

Bit	Name	Description
7	SM0	UART Serial Mode Select Bits.
6	SM1	These bits select the Serial Port operating mode as follows: SM0 SM1 Selected Operating Mode 0 0 Mode 0: Shift Register, fixed baud rate (Core_Clk/2) 0 1 Mode 1: 8-bit UART, variable baud rate 1 0 Mode 2: 9-bit UART, fixed baud rate (Core_Clk/64) or (Core_Clk/32) 1 1 Mode 3: 9-bit UART, variable baud rate
5	SM2	Multiprocessor Communication Enable Bit. Enables multiprocessor communication in Modes 2 and 3. In Mode 0, SM2 should be cleared. In Mode 1, if SM2 is set, RI will not be activated if a valid stop bit was not received. If SM2 is cleared, RI will be set as soon as the byte of data has been received. In Modes 2 or 3, if SM2 is set, RI will not be activated if the received ninth data bit in RB8 is 0. If SM2 is cleared, RI will be set as soon as the byte of data has been received.
4	REN	Serial Port Receive Enable Bit. Set by user software to enable serial port reception. Cleared by user software to disable serial port reception.
3	TB8	Serial Port Transmit (Bit 9). The data loaded into TB8 will be the ninth data bit that will be transmitted in Modes 2 and 3.
2	RB8	Serial Port Receiver Bit 9. The ninth data bit received in Modes 2 and 3 is latched into RB8 For Mode 1; the stop bit is latched into RB8.
1	TI	Serial Port Transmit Interrupt Flag. Set by hardware at the end of the eighth bit in Mode 0, or at the beginning of the stop bit in modes 1,2 ,3 TI must be cleared in software.
0	RI	Serial Port Receive Interrupt Flag. Set by hardware at the end of the eighth bit in Mode 0, or halfway through the stop bit in Modes 1, 2, and 3. RI must be cleared by software.

a. Operating Modes of Serial Port

i. Mode 0 (8-Bit Shift Register Mode)

Mode 0 is selected by clearing both the SM0 and SM1 bits in the SFR SCON. Serial data enters and exits through RxD. TxD outputs the shift clock. Eight data bits are transmitted or received. Transmission is initiated by any instruction that writes to SBUF. The data is shifted out of the RxD line. The eight bits are transmitted with the least significant bit (LSB) first, as shown Figure 2-13.

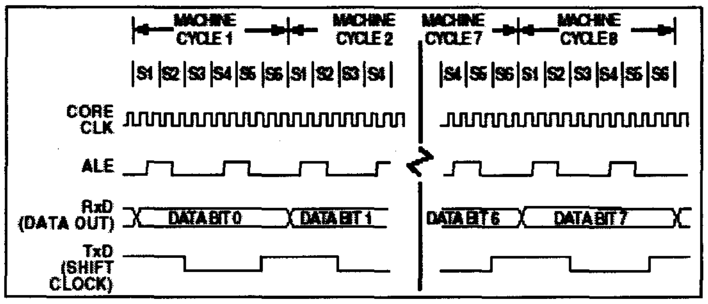


Figure 2-13: UART Serial Port Transmission, Mode 0

Reception is initiated when the receive enable bit (REN) is 1 and the receive interrupt bit (RI) is 0. When RI is cleared, the data is clocked into the RxD line and the clock pulses are output from the TxD line.

ii. Mode 1 (8-Bit UART, Variable Baud Rate)

Mode 1 is selected by clearing SM0 and setting SM1. Each data byte (LSB first) is preceded by a start bit (0) and followed by a stop bit (1). Therefore 10 bits are transmitted on TxD or received on RxD. The baud rate is set by the Timer 1 or Timer 2 overflow rate, or a combination of the two (one for transmission and the other for reception). Transmission is initiated by writing to SBUF. The "write to SBUF" signal also loads a 1 (stop bit) into the ninth bit position of the transmit shift register. The data is output bit by bit until the stop bit appears on TxD and the transmit interrupt flag (TI) is automatically set, as shown in Figure 2-14.

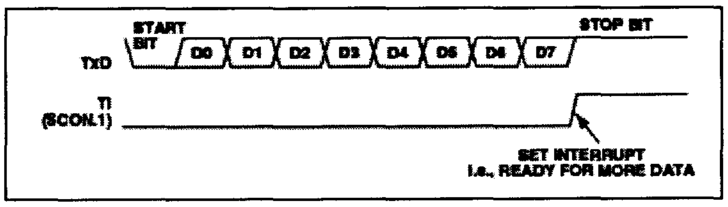


Figure 2-14: UART Serial Port Transmission, Mode 1

iii. Mode 2 (9-Bit UART, Fixed Baud Rate)

Mode 2 is selected by setting SM0 and clearing SM1. In thismode, the UART operates in 9-bit mode with a fixed baud rate. The baud rate is fixed at Core_Clk/64 by default, although by setting the SMOD bit in PCON, the frequency can be doubled to

Core_Clk/32. Eleven bits are transmitted or received, a start bit (0), eight data bits, a programmable ninth bit, and a stop bit (1). The ninth bit is most often used as a parity bit, although it can be used for anything, including a ninth data bit if required. To transmit, the eight data bits must be written into SBUF. The ninth bit must be written to TB8 in SCON. When transmission is initiated, the eight data bits (from SBUF) are loaded onto the transmit shift register (LSB first). The contents of TB8 are loaded into the ninth bit position of the transmit shift register. The transmission will start at the next valid baud rate clock. The TI flag is set as soon as the stop bit appears on TxD. Reception for Mode 2 is similar to that of Mode 1.

The eight bits in the receive shift register are latched into SBUF. The ninth data bit is latched into RB8 in SCON. The Receiver interrupt flag (RI) is set. This will be the case if, and only if, the following conditions are met at the time the final shift pulse is generated: RI = 0, and Either SM2 = 0, or SM2 = 1 and the received stop bit = 1.

iv. Mode 3 (9-Bit UART, Variable Baud Rate)

Mode 3 is selected by setting both SM0 and SM1. In this mode the 8051 UART serial port operates in 9-bit mode with a variable baud rate determined by either Timer 1 or Timer 2. The operation of the 9-bit UART is the same as for Mode 2, but the baud rate can be varied as for Mode 1. In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

b. UART Serial Port Baud Rate Generation

Mode 0 Baud Rate Generation

The baud rate in Mode 0 is fixed:

$$Mode\ 0\ Baud\ Rate = (Core\ Clock\ Frequency/12)$$

Mode 2 Baud Rate Generation

The baud rate in Mode 2 depends on the value of the SMOD bit in the PCON SFR. If SMOD = 0, the baud rate is 1/64 of the core clock. If SMOD = 1, the baud rate is 1/32 of the core clock:

$$Mode\ 2\ Baud\ Rate = \left(2^{SMOD}/64\right) \times (Core\ Clock\ Frequency)$$

Mode 1 and 3 Baud Rate Generation

The baud rates in Modes 1 and 3 are determined by the overflow rate in Timer 1 or Timer 2, or both (one for transmit and the other for receive).

6. Timers/Counters

The ADuC812 has three 16-bit Timer/Counters: Timer 0,Timer 1, and Timer 2. The Timer/Counter hardware has been included on-chip to relieve the processor core of the overhead inherent in implementing timer/counter functionality in software. Each Timer/Counter consists of two 8-bit registers, THx and TLx (x = 0, 1, and 2). All three can be configured to operate either as timers or event counters. In Timer function, the TLx register is incremented every machine.

a. Description of Timer/Counter 0 and 1

Timer/Counter 0 and Timer/counter 1 can be configured via SFRs TMOD, TCON. We are using Timer 1in 8 – bit auto reload mode which is set for the baud rate of 9600 and Timer0 in 8-bit auto reload mode for the LO switching in the system

TMOD

SFR Address	89H
Power-On Default Value	00H
Bit Addressable	No

Gate	C/T	M1	M0	Gate	C/T	M1	M0

Timer/Counter 0 Timer/Counter 1

TCON 1 Control Register

SFR Address	88H
Power-On Default Value	00H
Bit Addressable	Yes

TF1	TR1	TF0	TR0	IE1*	IT1*	IE0*	IT0*

Table 2-6: TMOD

Name	Description
GATE	Timer 1/0 Gating Control. Set by software to enable Timer/Counter 1 only while INT1 pin is high and TR1 control bit is set. Cleared by software to enable Timer 1 whenever TR1 control bit is set.
C/T	Timer 1/0 Timer or Counter Select Bit. Set by software to select counter operation (input from T1 pin). Cleared by software to select timer operation (input from internal system clock).
M1	Timer 1/0 Mode Select Bit 1 (used with M0 Bit).
M0	Timer 1/0 Mode Select Bit 0. M1 M0 0 0 TH1/0 operates as an 8-bit timer/counter. TL1/0 serves as 5-bit prescaler. 0 1 16-Bit Timer/Counter. TH1/0 and TL1/0 are cascaded; there is no prescaler. 1 0 8-Bit Autoreload Timer/Counter. TH1/0 holds a value that is to be reloaded into TL1/0 each time it overflows. 1 1 Timer/Counter 1/0 Stopped.

Table 2-7: TCON

Bit	Name	Description
7	TF1	Timer 1 Overflow flag. Set by hardware on a timer/counter 1 overflow Cleared by hardware when the Program Counter (PC) vectors to the interrupt service routine.
6	TR1	Timer 1 Run Control Bit. Set by user to turn on Timer/Counter 1. Cleared by user to turn off Timer/Counter 1.
5	TF0	Timer 0 Overflow Flag. Set by hardware on a Timer/Counter 0 overflow. Cleared by hardware when the PC vectors to the interrupt service routine.
4	TR0	Timer 0 Run Control Bit. Set by user to turn on Timer/Counter 0. Cleared by user to turn off Timer/Counter 0.
3	IE1	External Interrupt 1 (INT1) Flag. Set by hardware by a falling edge or zero level being applied to external interrupt pin INT1,depending on bit IT1 state.Cleared by hardware when the when the PC vectors to the interrupt service routine only if the interrupt was transition-activated. If level-activated, the external requesting source controls therequest flag, rather than the on-chip hardware.

Table 2-7 continued

2	IT1	External Interrupt 1 (IE1) Trigger Type. Set by software to specify edge-sensitive detection (i.e., 1-to-0 transition). Cleared by software to specify level-sensitive detection (i.e., zero level).
1	IE0	External Interrupt 0 (INT0) Flag. Set by hardware by a falling edge or zero level being applied to external interrupt pin INT0, depending on bit IT0 state. Cleared by hardware when the PC vectors to the interrupt service routine only if the interrupt was transition activated. If level activated, the external requesting source controls the request flag, rather than the on-chip hardware.
0	IT0	External Interrupt 0 (IE0) Trigger Type. Set by software to specify edge-sensitive detection (i.e., 1-to-0 transition). Cleared by software to specify level-sensitive detection (i.e., zero level).

Timer/Counters 0 and 1 Operating Modes 1

i. Mode 0 (8-Bit Shift Register Mode)

It configures an 8-Bit Timer/Counter with a divide by 32 prescaler, Figure 2-15 shows the Mode 0 operation.

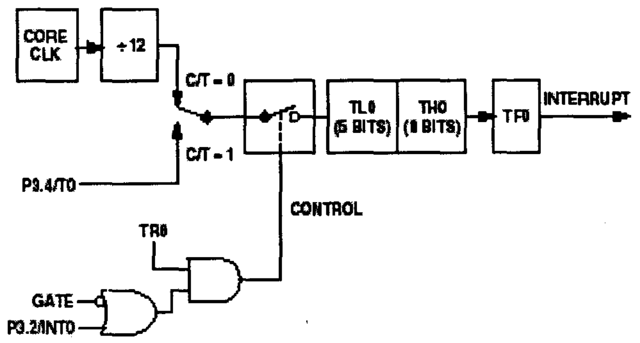


Figure 2-15: Timer/Counter 0, Mode 0

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer overflow flag TF0. The overflow flag, TF0, can then be used to request an interrupt. The counted input is enabled to the timer when TR0 = 1 and either Gate = 0 or INT0 = 1. Setting Gate = 1 allows the timer to be controlled by external input INT0 to facilitate pulse width measurements. TR0 is a control bit in the special function register TCON; Gate is in TMOD. The 13-bit register consists of all eight bits of TH0 and the lower five bits of TL0.

ii. Mode 1 (16 Bit Timer/Counter)

Mode 2 configures the timer register as an 8-bit counter (TL0) with automatic reload, as shown in Figure 2-16. Overflow from TL0 not only sets TF0, but also reloads TL0 with the contents of TH0, which is preset by software. The reload leaves TH0 unchanged.

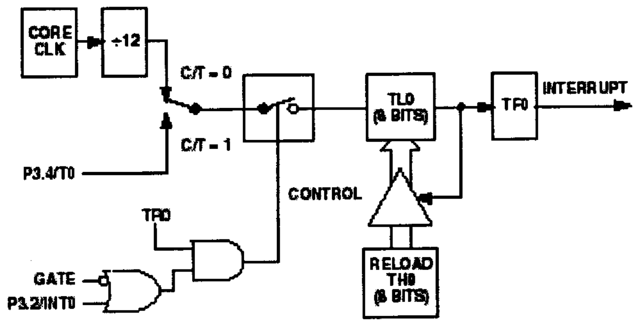


Figure 2-16: Timer/Counter 0, Mode 2

iii. Mode 3 (Two 8 Bit Timer/Counters)

Mode 3 has different effects on Timer 0 and Timer 1. Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0. Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. This configuration is shown in Figure 2-17. TL0 uses the Timer 0 control bits: C/T, Gate, TR0, INT0, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Mode 3 is provided for applications requiring an extra 8-Bit timer/counter and also used by serial interface as a baud rate generator.

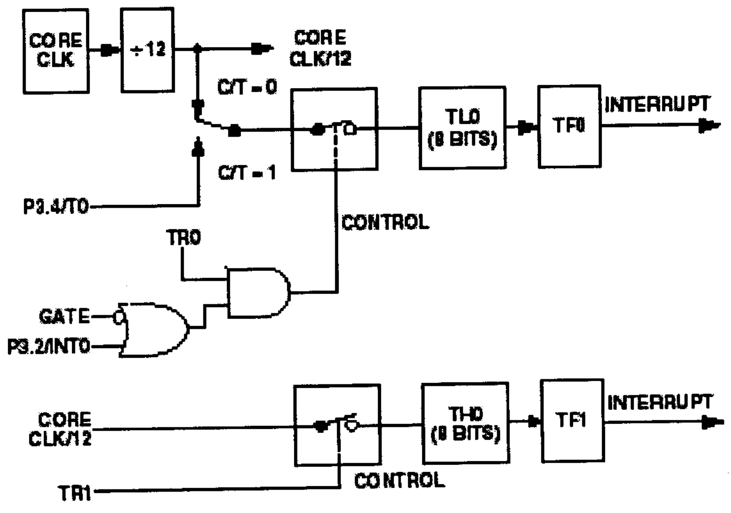


Figure 2-17: Timer/Counter 0, Mode 3

b. Description of Timer/Counter 2

T2CON	Control Register
SFR Address	C8H
Power-On Default Value	00H
Bit Addressable	Yes

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CNT2	CAP2

Table 2-8: T2CON SFR

Bit	**Name**	**Description**
7	TF2	Timer 2 Overflow Flag. Set by hardware on a Timer 2 overflow. TF2 will not be set when either RCLK = 1 or TCLK = 1.Cleared by user software.
6	EXF2	Timer 2 External Flag. Set by hardware when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1.Cleared by user software.
5	RCLK	Receive Clock Enable Bit. Set by user to enable the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3.Cleared by user to enable Timer 1 overflow to be used for the receive clock.
4	TCLK	Transmit Clock Enable Bit. Set by user to enable the serial port to use Timer 2 overflow pulses for its transmit clock in serialport Modes 1 and 3.Cleared by user to enable Timer 1 overflow to be used for the transmit clock.
3	EXEN2	Timer 2 External Enable Flag. Set by user to enable a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. Cleared by user for Timer 2 to ignore events at T2EX.
2	TR2	Timer 2 Start/Stop Control Bit. Set by user to start Timer 2. Cleared by user to stop Timer 2.
1	CNT2	Timer 2 Timer or Counter Function Select Bit. Set by the user to select counter function (input from external T2 pin). Cleared by the user to select timer function (input from on-chip core clock).
0	CAP2	Timer 2 Capture/Reload Select Bit. Set by user to enable captures on negative transitions at T2EX if EXEN2 = 1. Cleared by user to enable autoreloads with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to autoreload on Timer 2 overflow.

Timer/Counters Operation Modes

i. 16 Bit Autoreload Mode

In Autoreload mode, there are two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over, it not only sets TF2 but also causes the Timer 2 registers toreload with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1 then Timer 2 still performs the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2. The Autoreload mode is illustrated in Figure 2-18.

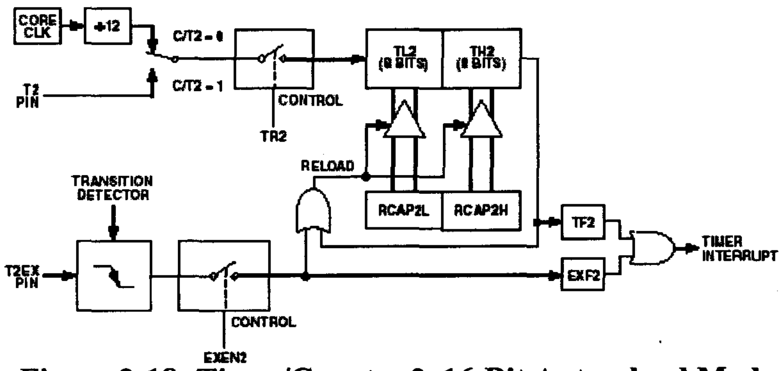


Figure 2-18: Timer/Counter 2, 16-Bit Autoreload Mode

ii. 16 Bit Capture Mode

In the Capture mode, there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter that, upon overflowing, sets bit TF2, the Timer 2 overflow bit, that can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still performs the above, but a l-to-0 transition on external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. The capture mode is illustrated in Figure 2-19. The baud rate generator mode is selected by RCLK = 1 and / or TCLK = 1

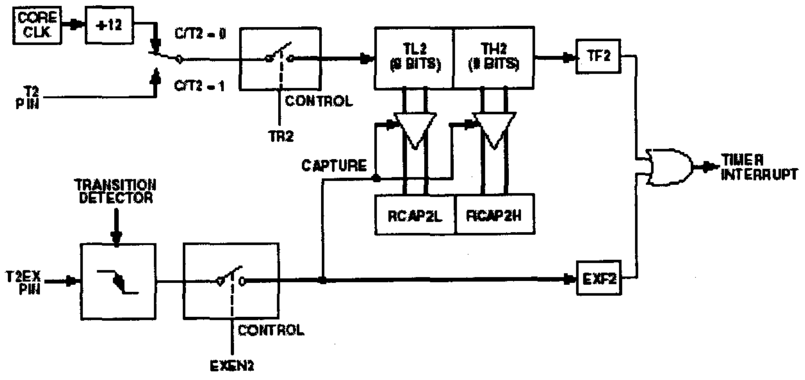


Figure 2-19: Timer/Counter 2, 16-Bit Capture Mode

7. Serial Peripheral Interface

The ADuC812 integrates a complete hardware serial peripheral interface. The SPI port can be configured for Master or Slave operation and typically consists of 4 pins namely,

a. MISO (Master In Slave Out Data I/O Pin)

The MISO (master in, slave out) pin is configured as an input line in master mode and an output line in slave mode. The MISO line on the master (data in) should be connected to the MISO line in the slave device (data out). The data is transferred as byte wide (8-bit) serial data, MSB first.

b. MOSI (Master Out Slave In Data I/O Pin)

The MOSI (master out, slave in) pin is configured as an output line in master mode and an input line in slave mode. The MOSI line on the master (data out) should be connected to the MOSI line in the slave device (data in). The data is transferred as byte wide (8-bit) serial data, MSB first.

c. SLOCK (Serial Clock I/O Pin)

The master serial clock (SCLOCK) is used to synchronize the data being transmitted and received through the MOSI and MISO data lines. A single data bit is transmitted and received in each SCLOCK period. Therefore, a byte is transmitted/received after eight SCLOCK periods. The SCLOCK pin is configured as an output in master mode and as an input in slave mode. In master mode, the bit rate, polarity, and phase of the clock are controlled by the CPOL, CPHA, SPR0, and SPR1 bits in the SPICON SFR (see Table XI). In slave mode, the SPICON register will have to be configured with the phase and polarity (CPHA and CPOL) of the expected input clock.

d. $\overline{\text{SS}}$ (Slave Select Input Pin)

The Slave Select (SS) input pin is shared with the ADC5 input. To configure this pin as a digital input, the bit must be cleared, data lines. A single data bit is transmitted and received in each SCLOCK period. Therefore, a byte is transmitted/received after eight SCLOCK periods. The SCLOCK pin is configured as an output in master mode and as an input in slave mode. In master mode, the bit rate, polarity, and phase of the clock are controlled by the CPOL, CPHA, SPR0, and SPR1 bits in the SPICON SFR (see Table XI). In slave mode, the SPICON register will have to be configured with the phase and polarity (CPHA and CPOL) of the expected input clock. In both master and slave modes. This line is active low. Data is only received or transmitted in slave mode when the SS pin is low, allowing the ADuC812 to be used in single master, multi-slave SPI configurations. If CPHA = 1, then the SS input may be permanently pulled low. With CPHA = 0, the SS input must be driven low before the first bit in a byte wide transmission or reception, and return high again after the last bit in that byte wide transmission or reception. In SPI Slave mode, the logic level on the external SS pin can be read via the SPR0 bit in the SPICON SFR. The following SFR registers are used to control the SPI interface.

SPI Control	
SPICON	Register
SFR	Address F8H
Power-On Default Value	OOH
Bit Addressable	Yes

ISPI	WCOL	SPE	SPIM	CPOL	CPHA	SPR1	SPR0

Table 2-9: SPICON SFR

Bit	Name	Description
7	ISPI	SPI Interrupt Bit. Set by MicroConverter at the end of each SPI transfer. Cleared directly by user code or indirectly by reading the SPIDAT SFR.
6	WCOL	Write Collision Error Bit. Set by MicroConverter if SPIDAT is written to while an SPI transfer is in progress. Cleared by user code.
5	SPE	SPI Interface Enable Bit. Set by user to enable the SPI interface. Cleared by user to enable I2C interface.
4	SPIM	SPI Master/Slave Mode Select Bit. Set by user to enable Master mode operation (SCLOCK is an output). Cleared by user to enable Slave mode operation (SCLOCK is an input).

Table 2-9 continued

3	CPOL	Clock Polarity Select Bit. Set by user if SCLOCK idles high. Cleared by user if SCLOCK idles low.
2	CPHA	Clock Phase Select Bit. Set by user if leading SCLOCK edge is to transmit data. Cleared by user if trailing SCLOCK edge is to transmit data.
1	SPR1	SPI Bit Rate Select Bits.
0	SPR0	These bits select the SCLOCK rate (bit rate) in Master mode as follows: SPR1 SPR0 Selected Bit Rate 0 0 fOSC/4 0 1 fOSC/8 1 0 fOSC/32 1 1 fOSC/64 In SPI Slave mode, i.e., SPIM = 0, the logic level on the external SS pin can be readvia the SPR0 bit.

Using the SPI Interface

Depending upon the configuration of the Bits the SPICON SFR, the ADuC812 SPI interface will transmit or receive data in two possible modes

i. SPI Interface – Master Mode

In master mode a byte transmission or reception is initiated by a write to SPIDAT. Eight clock periods are generated via the SCLOCK pin and the SPIDAT byte being transmitted via MOSI. With each SCLOCK period a data bit is also sampled via MISO. After eight clocks, the transmitted byte will have been completely transmitted and the input byte will be waiting in the input shift register. The ISPI flag will be set automatically and an interrupt will occur if enabled. The value in the shift register will be latched into SPIDAT.

ii. SPI Interface – Master Mode

In slave mode, a data bit is transmitted via MISO and a data bit is received via MOSI through each input SCLOCK period. After eight clocks, the transmitted byte will have been completely transmitted and the input byte will be waiting in the input shift register. The ISPI flag will be set automatically and an interrupt will occur if enabled. The value in the shift register will be latched into SPIDAT only when the transmission/reception of a byte has been completed. The end of transmission occurs after the eighth clock has been received if CPHA = 1, or when SS returns high if CPHA = 0.

8. Power Supply Monitor

The function of the Power Supply Monitor is to monitor both supplies (AVdd and DVdd) on the ADuC 812. It will indicate when either power supply drops below one of five user selectable voltage trip points from 2.63 V to 4.63 V. For correct operation of the Power Supply Monitor function, AVDD must be equal to or greater than 2.7 V. The Power Supply Monitor function is controlled via the PSMCON SFR.

Power Supply Monitor	
PSMCON	Control Register
SFR	Address DFH
Power-On Default Value	DCH
Bit Addressable	No

—	CMP	PSMI	TP2	TP1	TP0	PSF	PSMEN

Table 2-10:PSMCON SFR

Bit	Name	Description
7	—	Not Used.
6	CMP	AVDD and DVDD Comparator Bit. This is a read-only bit and directly reflects the state of the AVDD and DVDD comparators. Read "1" indicates that both the AVDD and DVDD supplies are above their selected trip points. Read "0" indicates that either the AVDD or DVDD supply is below its selected trip point.
5	PSMI	Power Supply Monitor Interrupt Bit. This bit will be set high by the MicroConverter if CMP is low, indicating low analog or digital supply. The PSMI bit can be used to interrupt the processor. Once CMPD and/or CMP return (and remain) high, a 256 ms counter is started. When this counter times out, the PSMI interrupt is cleared. PSMI can also be written by the user. However, if either comparator output is low, it is not possible for the user to clear PSMI.
4	TP2	V_{DD} Trip Point Selection Bits.
3	TP1	

Table 2-10 continued

2	TP0	These bits select the AVDD and DVDD trip point voltage as follows:			
		TP2	TP1	TP0	Selected DVDD Trip Point (V)
		0	0	0	4.63
		0	0	1	4.37
		0	1	0	3.08
		0	1	1	2.93
		1	0	0	2.63
1	PSF	AV_{DD}/DV_{DD} Fault Indicator. Read "1" indicates that the AV_{DD} supply caused the fault condition. Read "0" indicates that the DV_{DD} supply caused the fault condition.			
0	PSMEN	Power Supply Monitor Enable Bit. Set to "1" by the user to enable the Power Supply Monitor Circuit. Cleared to "0" by the user to disable the Power Supply Monitor Circuit.			

9. I/O Ports of AduC812

The ADuC812 uses four input/output ports to exchange data with external devices. In addition to performing general-purpose I/O, some ports are capable of external memory operations; others are multiplexed with an alternate function for the peripheral features on the device.

Port 0 is an 8-bit, open-drain, bi directional I/O port that is directly controlled via the P0 SFR (SFR address = 80H). Port 0 pins that have 1s written to them via the Port 0 SFR will be configured as open drain and will therefore float. In that state, Port 0 pins can be used as high impedance inputs. An external pull-up resistor will be required on Port 0 outputs to force a valid logic high level externally. Port 0 is also the multiplexed low order address and data bus during accesses to external program or data memory. In this application, it uses strong internal pull-ups when emitting 1s.

Port 1 is also an 8-bit port directly controlled via the P1 SFR (SFR address = 90H). Port 1 is an input only port. Port 1 digital output capability is not supported on this device. Port 1 pins can be configured as digital inputs or analog inputs. By (power-on) default these pins are configured as analog inputs, i.e., "1" written in the corresponding Port 1 register bit. To configure any of these pins as digital inputs, the user should write a "0" to these port bits to configure the corresponding pin as a high impedance digital input.

Port 2 is a bidirectional port with internal pull-up resistors directly controlled via the P2 SFR (SFR address = A0H). Port 2 pins that have 1s written to them are pulled high by the internal pull-up resistors and, in that state, can be used as inputs. As inputs, Port 2 pins being pulled externally low will source current because of the internal pull-up resistors. We are using Port 2 for latching the latches and buffers in EPLD of control and monitor card.

Port 3 is a bidirectional port with internal pull-ups directly controlled via the P3 SFR (SFR address = B0H). Port 3 pins that have 1s written to them are pulled high by the internal pull-ups and, in that state, can be used as inputs. As inputs, Port 3 pins being pulled externally low will source current because of the internal pull-ups. Port 3 pins also have various secondary functions described in Table 1. We are using only RxD(UART Input Pin) and TxD(UART Output Pin) for serial data transfer from this port.

Table 2-11: Port 3 - Alternate Pin Functions

Pin	Alternate Function
P3.0	RxD (UART Input Pin) (or Serial Data I/O in Mode 0)
P3.1	TxD (UART Output Pin) (or Serial Clock Output in Mode 0)
P3.2	$\overline{\text{INT0}}$ (External Interrupt 0)
P3.3	$\overline{\text{INT1}}$ (External Interrupt 1)
P3.4	T0 (Timer/Counter 0 External Input)
P3.5	T1 (Timer/Counter 1 External Input)
P3.6	$\overline{\text{WR}}$ (External Data Memory Write Strobe)
P3.7	$\overline{\text{RD}}$ (External Data Memory Read Strobe)

10. Other Special Function Registers

a. I^{2}C Compatible Interface

The ADuC812 supports a 2-wire serial interface mode that is I2C compatible. The I2C compatible interface shares its pins with the on-chip SPI interface and therefore the user can only enable one or the other interface at any given time.

I2CCON	I2C Control Register
SFR Address	E8H
Power-On Default Value	00H
Bit Addressable	Yes

MDO	MDE	MCO	MDI	I2CM	I2CRS	I2CTX	I2CI

Table 2-12: I2CCON SFR

Bit	Name	Description
7	MDO	I2C Software Master Data Output Bit (Master Mode Only). This data bit is used to implement a master I2C transmitter interface in software. Data written to this bit will be output on the SDATA pin if the data output enable (MDE) bit is set.
6	MDE	I2C Software Master Data Output Enable Bit (Master Mode Only). Set by the user to enable the SDATA pin as an output (Tx). Cleared by the user to enable SDATA pin as an input (Rx).
5	MCO	I2C Software Master Data Output Bit (Master Mode Only). This data bit is used to implement a master I2C transmitter interface in software. Data written to this bit will be output on the SCLOCK pin.
4	MDI	I2C Software Master Data Input Bit (Master Mode Only). This data bit is used to implement a master I2C receiver interface in software. Data on the SDATA pin is latched into this bit on SCLOCK if the Data Output Enable (MDE) = 0.
3	I2CM	I2C Master/Slave Mode Bit. Set by user to enable I2C software master mode. Cleared by user to enable I2C hardware slave mode.
2	I2CRS	I2C Reset Bit (Slave Mode Only). Set by user to reset the I2C interface. Cleared by user for normal I2C operation.
1	I2CTX	I2C Direction Transfer Bit (Slave Mode Only). Set by the MicroConverter if the interface is transmitting. Cleared by the MicroConverter if the interface is receiving.
0	I2CI	I2C Interrupt Bit (Slave Mode Only). Set by the MicroConverter after a byte has been transmitted or received. Cleared by user software.

b. Interrupt System

The ADuC812 provides a total of nine interrupt sources with two priority levels. The control and configuration of the interrupt system is carried out through three interrupt related SFRs.

IE	Interrupt Enable Register
IP	Interrupt Priority Register
IE2	Secondary Interrupt Enable Register

IE	Interrupt Enable Register
SFR Address	A8H
Power-On Default Value	00H
Bit Addressable	Yes

EA	EADC	ET2	ES	ET1	EX1	ET0	EX0

Table 2-13: Interrupt Enable Register

Bit	Name	Description
7	EA	Written by user to enable "1" or disable "0" all interrupt sources.
6	EADC	Written by user to enable "1" or disable "0" ADC interrupt.
5	ET2	Written by user to enable "1" or disable "0" Timer 2 interrupt.
4	ES	Written by user to enable "1" or disable "0" UART serial port interrupt.
3	ET1	Written by user to enable "1" or disable "0" Timer 1 interrupt.
2	EX1	Written by user to enable "1" or disable "0" External Interrupt 1.
1	ET0	Written by user to enable "1" or disable "0" Timer 0 interrupt.
0	EX0	Written by user to enable "1" or disable "0" External Interrupt 0.

IP	Interrupt Priority Register
SFR Address	B8H
Power-On Default Value	00H
Bit Addressable	Yes

PSI	PADC	PT2	PS	PT1	PX1	PT0	PX0

Table 2-14: Interrupt Priority Register

Bit	Name	Description
7	PSI	Written by user to select I2C/SPI priority ("1" = High; "0" = Low).
6	PADC	Written by user to select ADC interrupt priority ("1" = High; "0" = Low).
5	PT2	Written by user to select Timer 2 interrupt priority ("1" = High; "0" = Low).
4	PS	Written by user to select UART serial port interrupt priority ("1" = High; "0" = Low).
3	PT1	Written by user to select Timer 1 interrupt priority ("1" = High; "0" = Low).
2	PX1	Written by user to select External Interrupt 1 priority ("1" = High; "0" = Low).
1	PT0	Written by user to select Timer 0 interrupt priority ("1" = High; "0" = Low).
0	PX0	Written by user to select External Interrupt 0 priority ("1" = High; "0" = Low).

IE2 Enable	Secondary Interrupt Register
SFR Address	A9H
Power-On Default Value	00H
Bit Addressable	No

—	—	—	—	—	—	EPSMI	ESI

Table 2-15: Secondary Interrupt Register

Bit	Name	Description
7	—	Reserved for future use.
6	—	Reserved for future use.
5	—	Reserved for future use.
4	—	Reserved for future use.
3	—	Reserved for future use.
2	—	Reserved for future use.
1	EPSM1	Written by user to Enable "1" or Disable "0" power supply monitor interrupt.
0	ESI	Written by user to Enable "1" or Disable "0" I2C/SPI serial port interrupt.

11. Quick Start Development System

The QuickStart Development System is a full featured, low cost development tool suite supporting the ADuC812. The system consists of the following PC based (Windows® compatible) hardware and software development tools.

Hardware:	ADuC812 Evaluation Board, Plug-In Power Supply and Serial Port Cable
Code Development:	8051 Assembler
Code Functionality:	Windows Based Simulator
In-Circuit Code Download:	Serial Downloader
In-Circuit Debugger:	Serial Port Debugger
Miscellaneous Other:	CD-ROM Documentation and Two Additional Prototype Devices

Figure shows the typical components of a QuickStart Development System. A brief description of some of the software tools components in the QuickStart Development System is given in the following sections. to the MicroConverter functionality and integrates many standard debug features including multiple breakpoints, single stepping, and code execution trace capability. This tool can be used both as a tutorial guide to the part as well as an efficient way

Download—In-Circuit Serial Downloader

The Serial Downloader is a Windows application that allows the user to serially download an assembled program (Intel Hex format file) to the on-chip program FLASH memory via the serial COM1 port on a standard PC. Application Note uC004 detailing this serial download protocol is available at www.analog.com/ microconverter.

Figure 2-20: Components of the Quick Start Development System

a. DeBug—In-Circuit Debugger

The Debugger is a Windows application that allows the user to debug code execution on silicon using the MicroConverter UART serial port. The debugger provides access to all on-chip peripherals during a typical debug session as well as single-step and breakpoint code execution control.

b. ADSIM—Windows Simulator

The Simulator is a Windows application that fully simulates all the MicroConverter functionality including ADC and DAC peripherals. The simulator provides an easy-to-use, intuitive interface to the MicroConverter functionality and integrates many standard debug features including multiple breakpoints, single stepping, and code execution trace capability. This tool can be used both as a tutorial guide to the part as well as an efficient way to prove code functionality before moving to a hardware platform. The QuickStart development tool suite software is freely available at the Analog Devices MicroConverter website, www.analog.com/ microconverter.

Figure 2-21: Typical Debug Session

Chapter 3 – Control and Monitoring System for 4-8 GHz Front End Receiver System

Control and Monitoring System is used to set various parameters of the Front End Receiver before using it for the observations. In addition the set of parameters are also monitored and stored in the computer (Refer Figure 3-1)

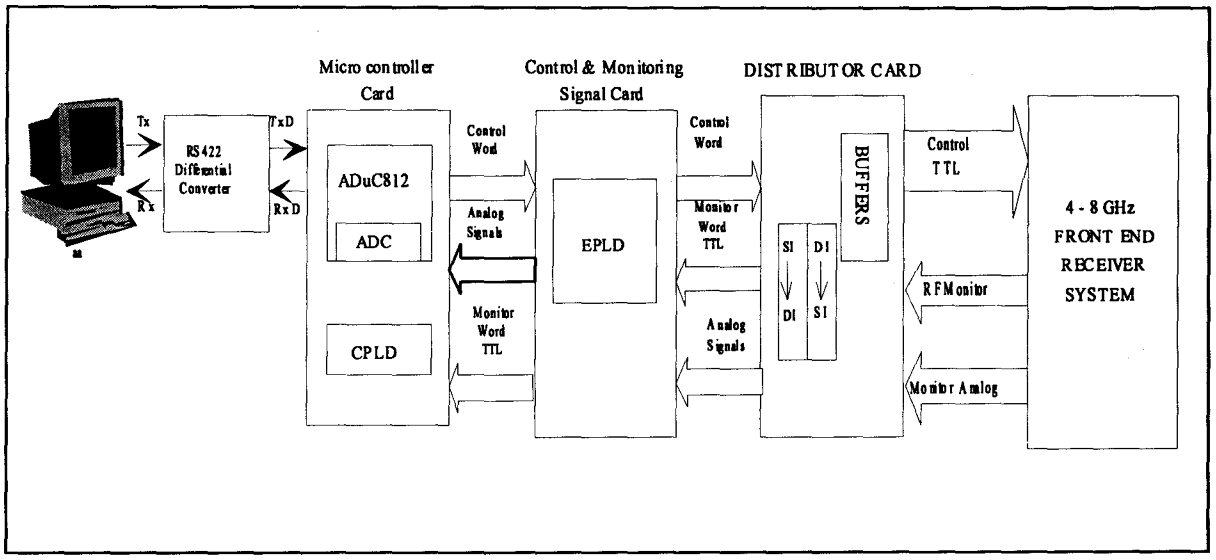


Figure 3-1: Block Diagram

3.1 General Description of Control and Monitoring System

The 4-8GHz Front End Receiver Control and Monitoring system consists of

a. PC for serial communication with the receiver system.

b. MICROCONTROLLER CARD storing the required parameters

c. CONTROL AND MONITORING CARD for latching and buffering the TTL control words.

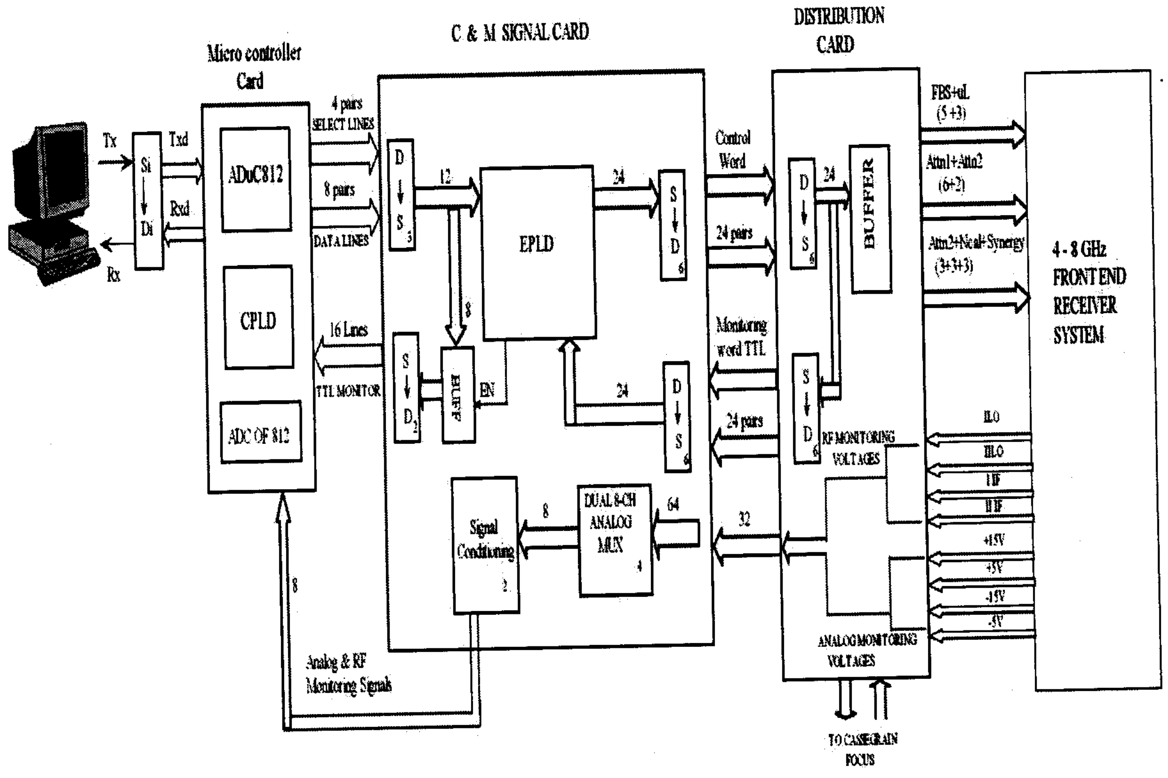
d. DISTRIBUTION CARD for distributing the control words to various RF modules.

The control words for setting the various parameters are sent serially from PC to the Micro controller card. These parameters are processed by Micro controller and arerouted to the CPLD in the Microcontroller card and are sent serially to control and monitoring card, which consists of an EPLD for latching and buffering purposes. The control words from the control and monitoring card are sent to the receiver system through the Distribution card.

For health monitoring of the system, various analog signals are monitored and stored in the computer after proper signal conditioning.

3.2 Description of Control and Monitoring System in Detail

The various units of the Control and Monitoring System are explained below in detail. The detailed architecture of the Control and Monitoring system is shown in Figure 3-2



Control Signals

Frequency Band Signal

First LO (uL)

Variable Attn 1

Variable Attn 2

Second LO (Synergy)

Noise Cal Duty Cycle

Monitoring Signals

Frequency Band Signal

First LO (Lock Detect)

Variable Attn 1

Variable Attn 2

Second LO (Lock Detect)

Noise Cal Duty Cycle

No. of Analog Monitoring Signals

RF Power Levels of Ist IF, IInd IF, Ist LO, IInd LO

Figure 3-2: Detailed Block Diagram of Front-end Controlling Monitoring System

3.2.1 PC Serial Port

PCs normally have 2 serial ports (COM1, COM2) in which both COM ports have RS232-type connectors. Many PCs use one of each of the DB-25 and DB-9 RS232 connectors, we are using DB-9 RS232 connector of the COM1 port of our PC.

Table 3-1: IBM PC DB-9 Signals

PIN	Description
1	Data Carrier Detect (DCD)
2	Received Data (RxD)
3	Transmitted data (TxD)
4	Data Terminal Ready (DTR)
5	Signal Ground (GND)
6	Data set Ready (DSR)
7	Request To Send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

Description of COM1 Port pins:

a) DTR (Data terminal Ready): When the PC COM port is turned on, after going through a self-test, it sends out signal Data Terminal Ready to indicate that it is ready for communication. If there is something wrong in with the COM port, this signal will not get activated. It is an active low output pin for the PC COM port.

b) DSR (Data set Ready): When data communication equipment (DCE) peripheral device is turned on, it asserts Data Set Ready to indicate that it is ready to communicate, thus it an output from the Data Communication Equipment and input to PC.It is an active low signal.

c) RTS (request to send): When the PC has a byte to transmit, it asserts Request To Send to signal the Data Communication Equipment that it has a byte of data to transmit. It is an active low output from the PC and an input to the Data Communication Equipment.

d) CTS (Clear to send): In response to Request To Send signal, when the Data Communication Equipment has room for storing data, it sends out signal Clear To Send signal to the PC to indicate that it can receive the data.

e) DCD (Data carrier detect): The Data Communication Equipment asserts signal Data Carrier Detect to inform the PC that the valid carrier has been detected and that contact between self and the other Data Communication Equipment is established. Therefore Data Carrier Detect is an output from the Data Communication Equipment and input to PC.

f) RI (ring indicator): Of the 6 hand shaking signals this not commonly used. This is used when modem is connected to PC.

The simplest connection between a PC and microcontroller requires a minimum of three pins- TXD, RXD and GND as shown in diagram below (Refer Figure 3-3):

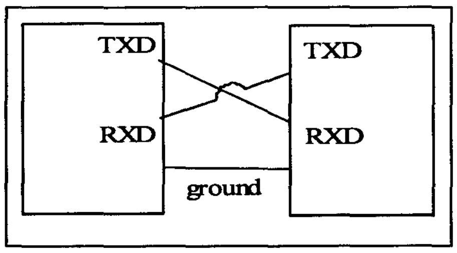


Figure 3-3: Connection Between PC and Microcontroller

3.2.2 RS232 – RS422 Differential Converter Card Details

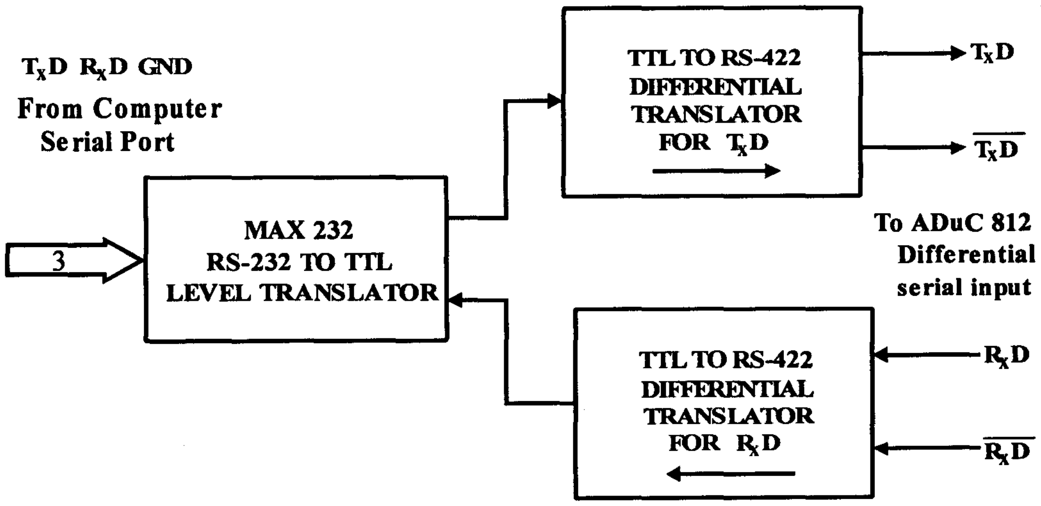

T_XD R_XD GND
From Computer Serial Port
3
MAX 232 RS-232 TO TTL LEVEL TRANSLATOR
TTL TO RS-422 DIFFERENTIAL TRANSLATOR FOR T_XD
T_XD
$\overline{T_XD}$
To ADuC 812 Differential serial input
TTL TO RS-422 DIFFERENTIAL TRANSLATOR FOR R_XD
R_XD
$\overline{R_XD}$

Figure 3-4: Block Diagram of RS232 to Differential Signal Converter

This card is essentially used to drive the computer serial port RS-232 signals for long distances. The signals of the COM1 serial port (TxD- Transmit, RxD - Receive and ground) are brought out through a D-type DB-9 connector. As the RS-232 signal levels

(–3V to -25V for logic '1' and +3V to +25V for logic '0') are translated to TTL levels using MAX-232 chip.

MAX-232 level translator includes two receivers and two transmitters in the same package, so that it serves the purpose of using the same chip for both transmit line as well as receive line. The TTL transmit and receive single ended signals are made o differential using a RS-422 differential transceiver chip as shown in the Figure 3-4 above. Two such transceivers are used one to transmit and another to receive. The outputs of transceivers are connected microcontroller for serial communications.

3.2.3 ADuC812 Microcontroller Card

While setting the parameters, the control words are sent in Differential mode to micrcontroller card in which they are translated to single ended TTL signals and are processed as described in CHAPTER 2. The processed TTL signals are brought out in differential mode and given to the CONTROL & MONITORING CARD.

During the monitoring phase, the control words coming differentially from the control and monitoring card are converted back to single ended TTL signals and given to CPLD, then to microcontroller. Apart from control words of the parameters, analog signals from the control and monitoring card are given separately as analog inputs of the AduC812 for analog to digital conversion .The digital data is then displayed on the PC.

3.2.4 Control and Monitoring Card

The control and monitoring card of the Rx system consists mainly of ALTERA EPLD, MULTIPLEXERS and SIGNAL CONDITIONING CIRCUITS. ALTERA EPLD (Erasable programmable logic device) can be programmed to have any desired logic circuit. In our C & M card of RX system EPLD is programmed as a 3: 8 decoder, three latches and three buffers as shown in the Block diagram. (Refer Figure 3-5).

The parameters used to configure the 4-8GHz receiver system are FBS (Frequency. Band Selection), Microlambda (LO1), Attenuation1, Attenuation 2, Synergy and Noise cal. (Refer Table 3-2)

So the Table 3-2 indicates that about 25 bits are to be sent to the Front-end receiver system for various parameters settings. Since only 24 bits are available, one of this is multiplexed for sending Noisecal and Synergy control bit .The 4 select lines i.e. the upper nibble of port2 of microcotroller are given to 3: 8 Decoder of EPLD for selecting three latches and three buffers and the port0 lines of microcontroller are used as data lines to transmit the data byte to the latch where the decoder has selected as shown in Figure 3-5 below.

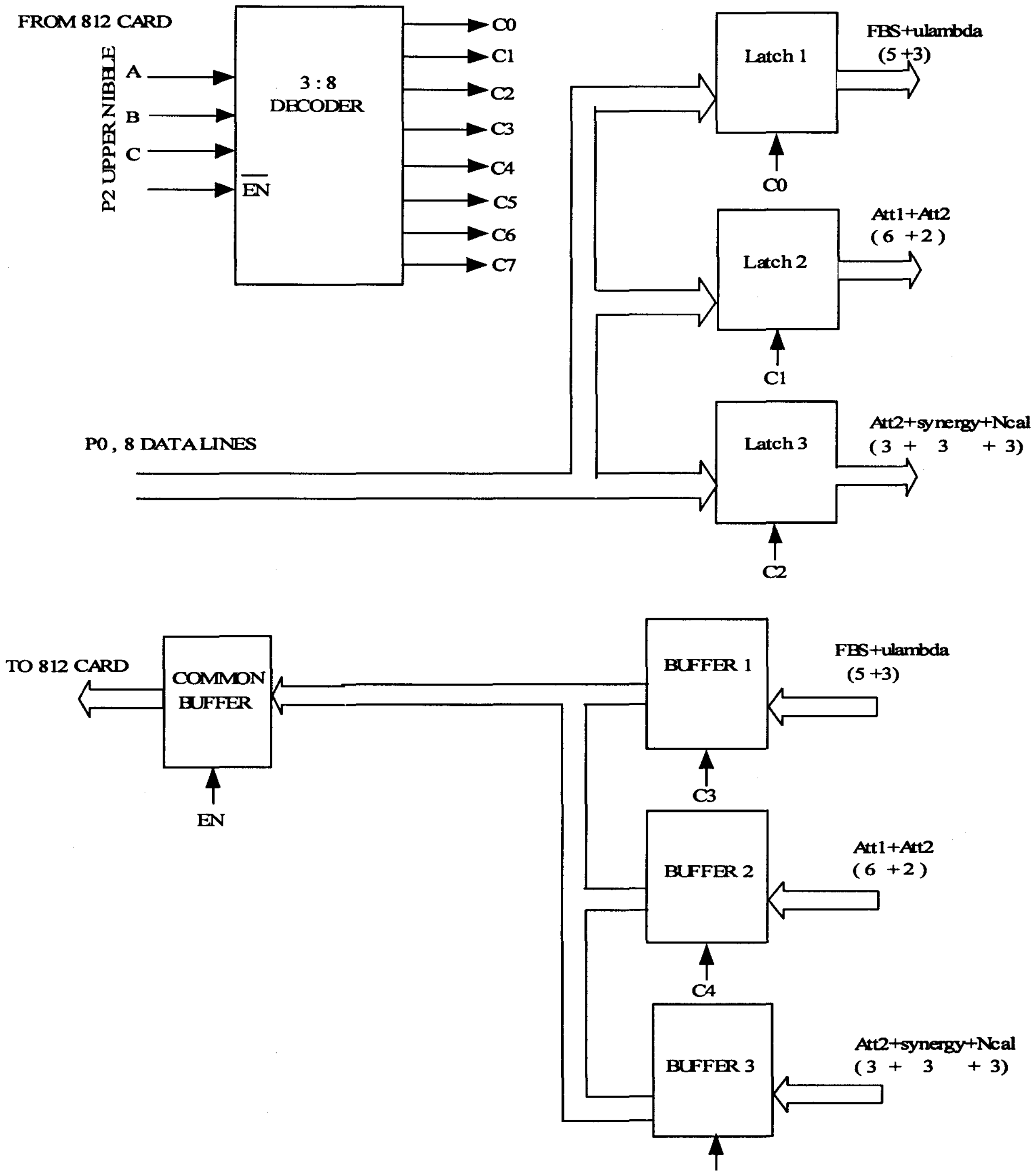


Figure 3-5 : Internal Diagram of EPLD

Table 3-2: Parameters and their Bits

PARAMETERS	NO OF BITS
1) FBS	5
2) First LO (uL)	3
3) Variable Att 1	6
4) Variable Att 2	5
5) Second LO (Synergy)	3
6) Noise cal duty cycle	3

When the Select lines are 000, latch 1 is selected and a byte of data is sent to latch1 where Frequency Band Selection (5 bits) and MicroLambda(3 bits)control bits are latched . Similarly when the latch 2 is selected Attenuation1(6 bits) Attenuation2(2 bits)control bits are latched and when latch3 is selected Attenuation2(3 bits), Synergy(3 bits),Noise cal duty cycle(3 bits) control bits are latched, totally 3 bytes of data are latched on to latches in EPLD. Then all the parameters are converted from single ended to differential and sent to the distribution card.

During monitoring phase, the decoder selects the first buffer and the byte of data (FBS+uLambda (5+3)) is stored in the Buffer coming from the distribution card. Similarly second byte is stored (Attn1+Attn2(6+2)) in the buffer2 and the third byte (Attn2+Synergy+Noisecal(3+3+3)) are stored in the buffer3 . These three bytes of control words of parameters are then transferred to the common buffer, which is enabled by the EPLD and are sent differentially to the microcontroller card and back to PC through Differential converter card. The analog signals coming from distribution card are given to Dual - 8 channel analog multiplexers .The 8 analog signals at the output of the multiplexer signal conditioned to limit voltages to Vref (2.5V). The signal conditioned output signals are inturn connected analog input of the microcontroller,these signals are digitized sequentially and stored in the computer.

Table 3-3 : State Table of 3:8 Decoder

A	B	C	Selected Lines
0	0	0	C0
0	0	1	C1
0	1	0	C2

Table 3.3 continued

0	1	1	C3
1	0	0	C4
1	0	1	C5
1	1	0	C6
1	1	1	C7

3.2.5 Distribution Card

In this distribution card all the control words sent differentially from the control and monitoring card are converted to single ended. These signals are buffered and sent to the receiver system during the controlling phase. Whereas in monitoring phase all the single ended control words are converted back to differential and are sent to control and monitoring card as shown in the Figure 3-2. The TTL and analog monitoring signals and analog monitoring signals from receiver are given to this card and are sent to Control and Monitoring card for further processing as described above. The TTL and analog monitoring signals are given in the table (Refer Table 3-4 and Table 3-5)

Table 3-4: Monitoring TTL Signals

Name of the parameter	No of Bits
1) FBS	5
2) I LO (Lock Detect)	1
3) Variable Att1	6
4) Variable Att2	5
5) II LO lock detect	1
6) NCAL duty cycle	3

Table 3-5: Analog RF Power Levels

Analog voltages	No of bits
+ 15V	6
+5V	5
-5V	4
-15V	1

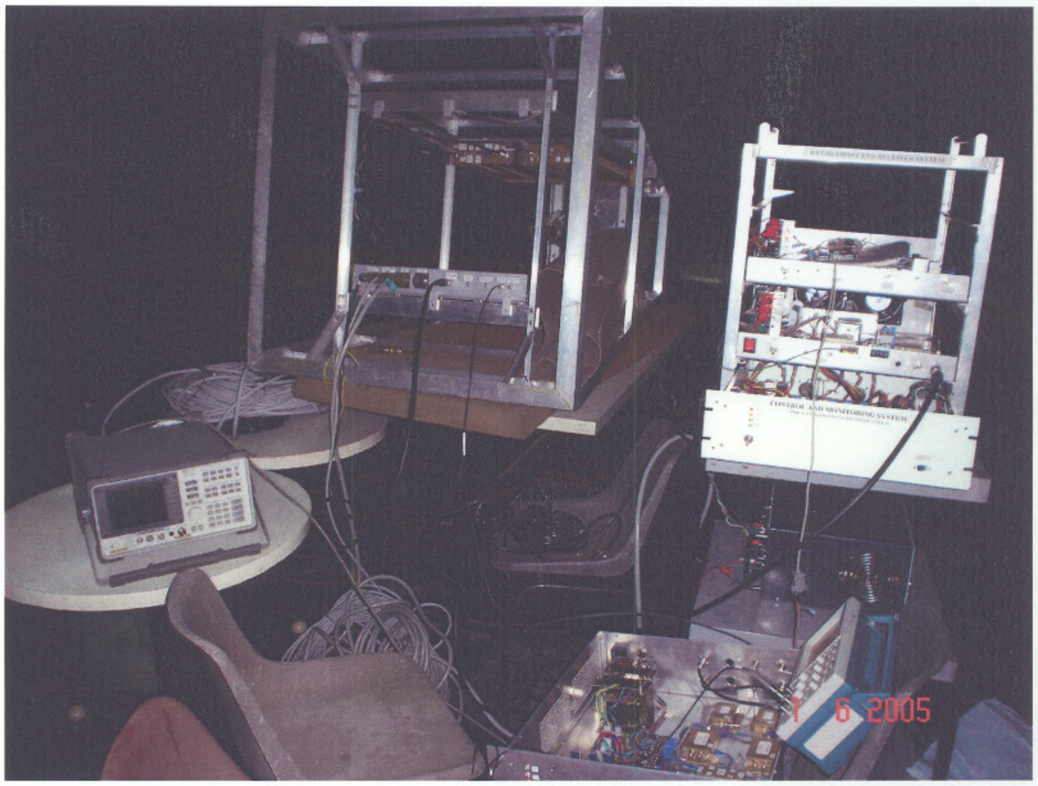


Figure 3-6: Set up of the Control and Monitoring System

3.3 Objectives of the Project

Individual Addressing:

In the earlier system the control words were sent sequentially to the Front end Receiver system through the microcontroller card. Under this mode of operation, if any change had to be made with settings, the entire program had to be run. This was time consuming and uncalled format. Therefore there is a necessity of modifying the code, which can individually address each parameter and do the changes. This is taken as the first objective of the project.

Frequency Switching:

In the earlier system the frequency of the 1st Local Oscillator was generated to down convert the RF signal. There was no provision to change the frequency by a small amount repeatedly. This was required for spectral line observation of the sky. Implementing Switching of the Local Oscillator frequency is considered as the II objective of the project.

Noise Cal Ramping :

Earlier in this system, only a particular value of the Noise cal was selected at a time and injected to the receiver system for calibration purposes. Injecting different Noise cal values into the receiver system in sequence would help us to check the linearity of the system. Implementing this is taken as III objective of the project.

Chapter 4 – Measurements and Readings

The software written for the control and monitoring system has been modified to implement the objectives described in the previous chapter. The results obtained after suitable modifications of the code are given below.

4.1 Individual Addressing

After incorporating suitable change in the code, it was observed that the parameter could be set individually within short span of time without resetting all parameters unnecessarily. Please refer Appendix for the modified code.

4.2 Frequency Switching

The technique of frequency switching is normally adopted whenever radio observation made for spectral line studies. The main purpose of frequency switching is to calibrate the band pass of the observed band particularly when the spectral line being observed is very weak.

Frequency switching has been implemented by loading the control words corresponding to 2 frequencies at a regular interval of time in to µLamda. The 2 frequencies generated by µLamda have a frequency separation of 1Mhz between them.

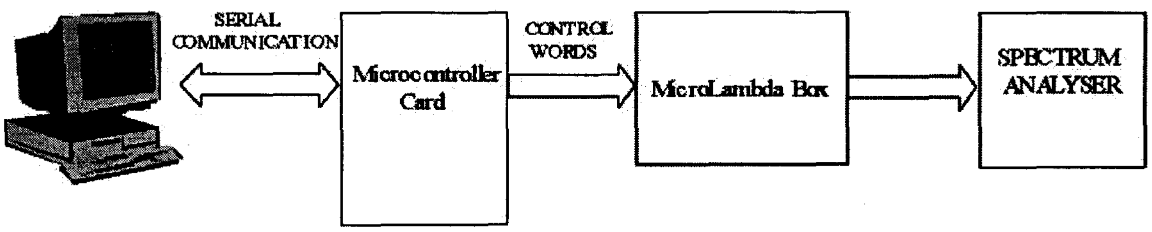


Figure 4-1: Block Diagram of Frequency Switching Set-up

The control words µLamda from the computer are sent serially to the microcontroller card in which they are properly set for switching. These control words are then given to the microcontroller box, which mainly consists of digital PLL. The output of the microcontroller box is given to the spectrum analyser to view the switching frequencies. The spectrum analyser is made for measuring the frequency contents of a complex signal; basically it gives the frequency spectrum of any arbitrary signal. The Figure 4-2 shows the 2 frequencies at the μLamda switching at the difference of 1MHz after loading the control words into the μLamda box. Frequency switching is done at the rate of 1 sec.

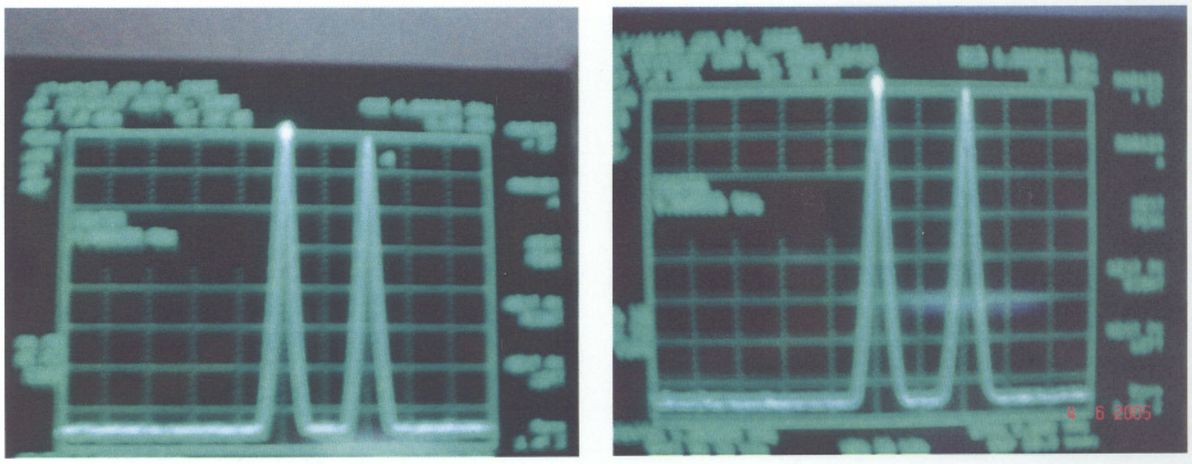


Figure 4-2: Frequency Switching Signal Displayed on Spectrum Analyser

4.2 Noise Cal Ramping

Noise Cal is well-calibrated signal injected into the receiver system for calibrating the gain and measuring the signal collected by the antenna. The magnitude noise cal being injected depends upon the radio astronomical source being observes by antenna. For fainter source we normally use very low value of noise cal signal and for stronger sources we use larger value of noise cal so the magnitude of noise cal in the present 4-8 GHz front-end receiver system is adjusted by varying the duty cycle of the signal which in turn controls the injection of noise cal.

The most important property of receiver system is its dynamic range. The dynamic range ensures the operation of receiver system in linear range. To make sure that receiver system being operated in linear range one may have to inject known amount of noise into it and find out corresponding out put.

If the system is operating in linear range the change in the receiver output must be equal to change in the receiver input. We have implemented this by injecting different noise cal signals into receiver sequentially. By measuring the receiver output and correlating with receiver input one will be able to find out whether the system is functioning in its linear range or not.

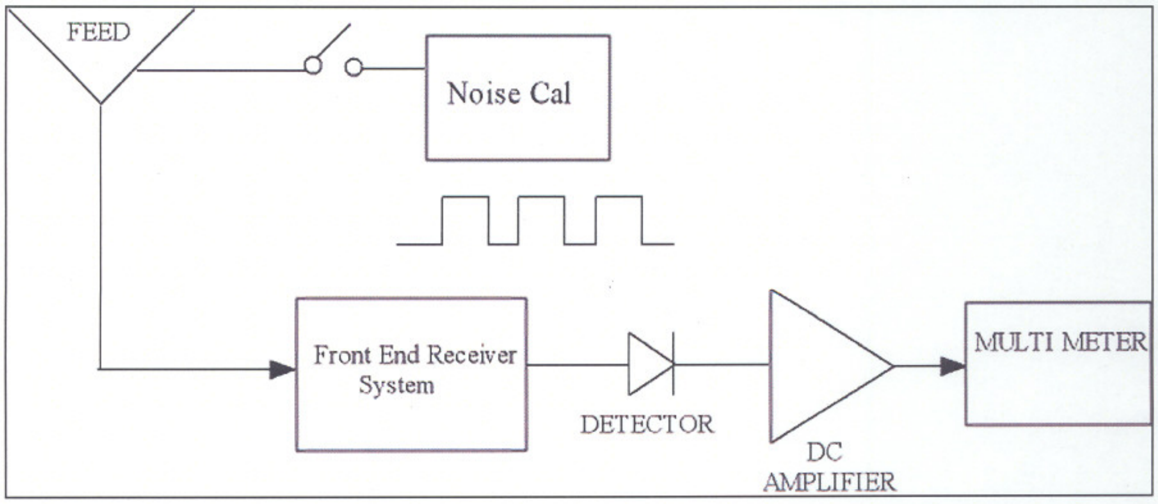


Figure 4-3: Block Diagram for Measuring Noise Cal

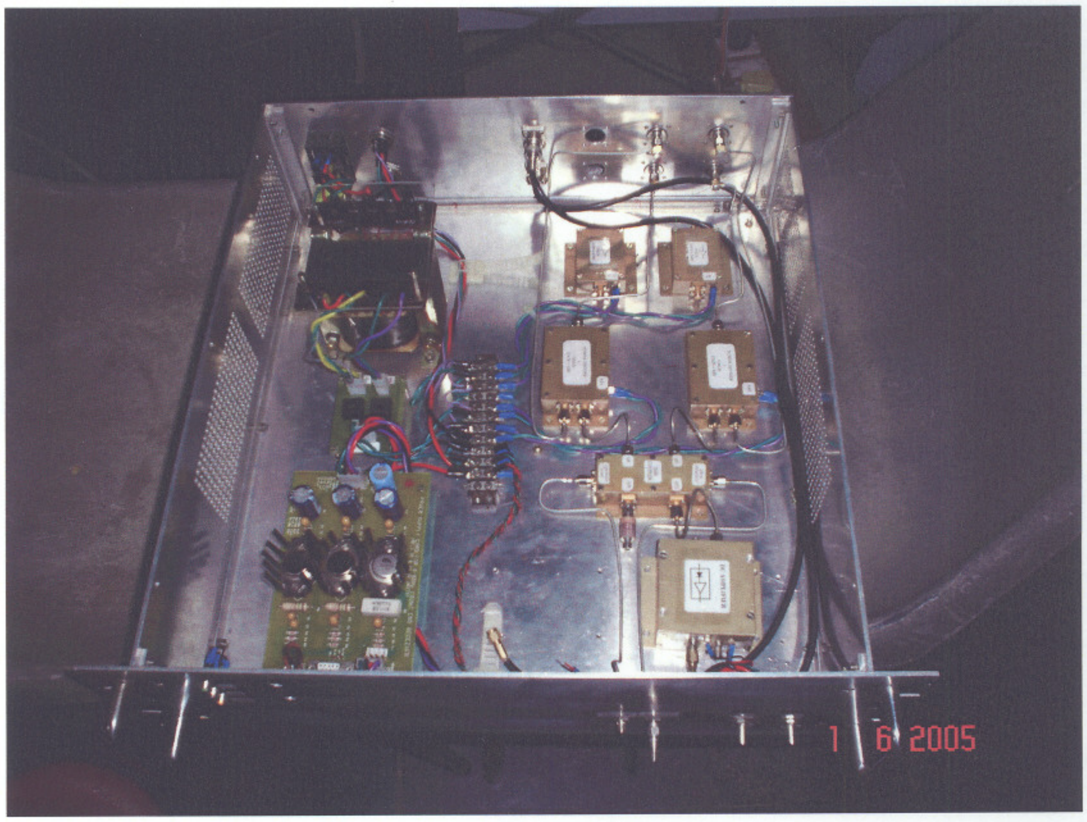


Figure 4-4: The Noise Cal Set-up

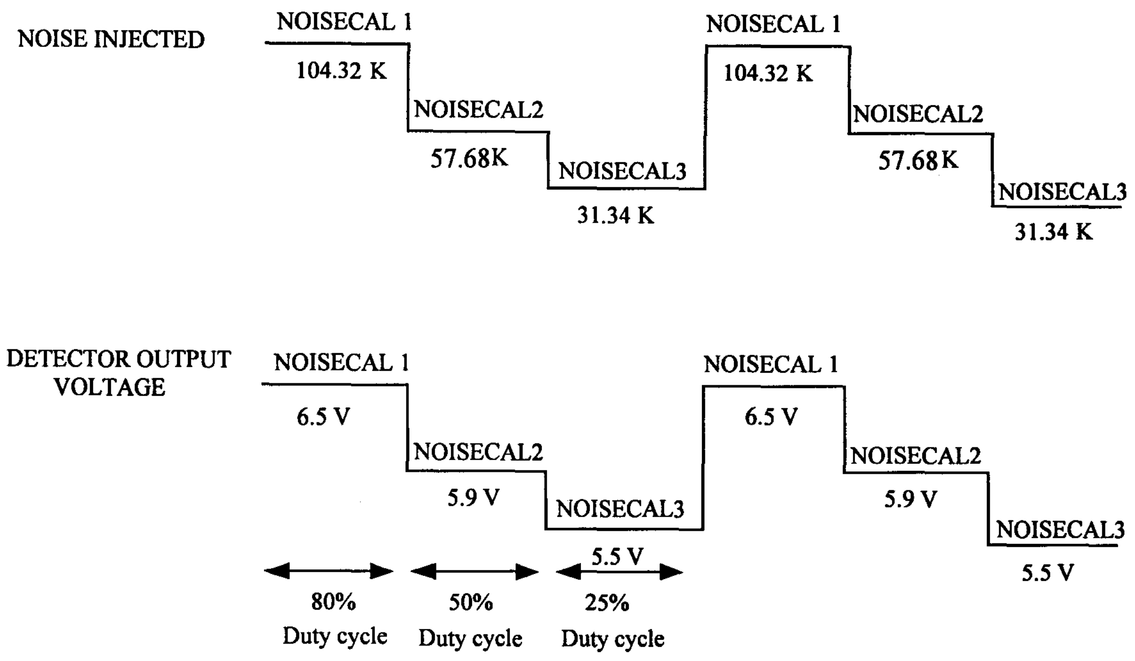


Figure 4-5: Noise Cal Ramp Output at the Multimeter

Calculations for checking the linearity of receiver system :

Percentage decrease in temperature	Percentage decrease in voltage
Noisecal1=(Tsky+Tr+Tlna)+Tcal1	V1=6.5V
=(410)k+104.32K= 514.32K	V2=5.9V
Noisecal2=(Tsky+Tr+Tlna)+Tcal2	V3=5.5V
=(410)K+57.68k= 467.68K	
Noisecal3=(Tsky+Tr+Tlna)+Tcal3	
=(410)K+28.84K=438.84K	
(Noisecal1-Noisecal2)/Noisecal1*100	(V1-V2)/V1*100
= 9.06%	= 9.23%
(Noisecal1-Noisecal3)/Noisecal1*100	(V1-V3)/V1*100
= 14.67%	= 15.28%

From the above calculations as the percentage of decrease in temperature is almost equal to the percentage of decrease in voltage. So we conclude that the receiver system is almost linear.

Chapter 5 - Conclusion

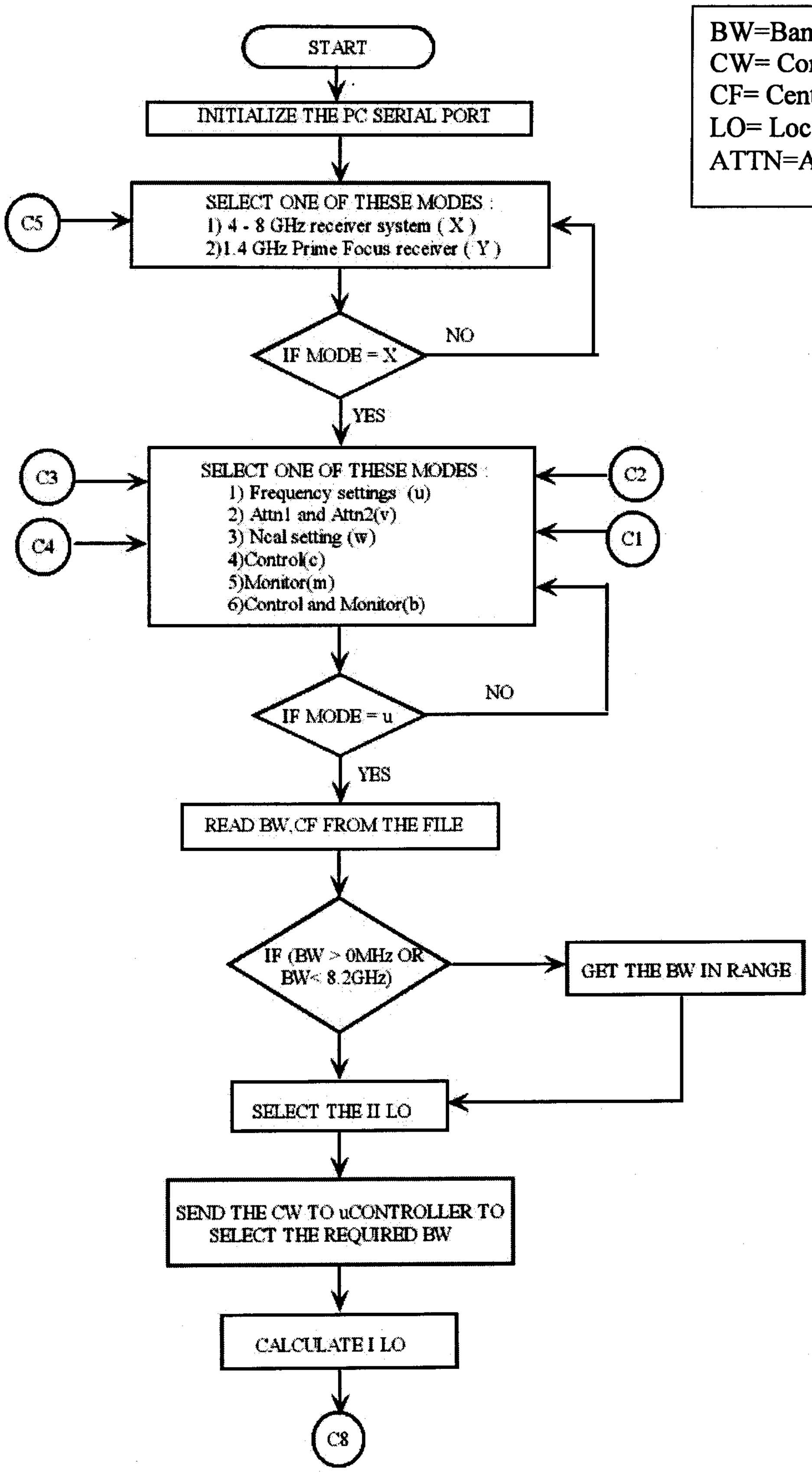
The software written for control and monitoring parameters of 4-8 GHz front-end receiver system has been modified and tested successfully to meet various objectives like individual settings of the parameters, local oscillator frequency switching and injection of various noise cal values. Part of software has been tested to be working in integrated environment.

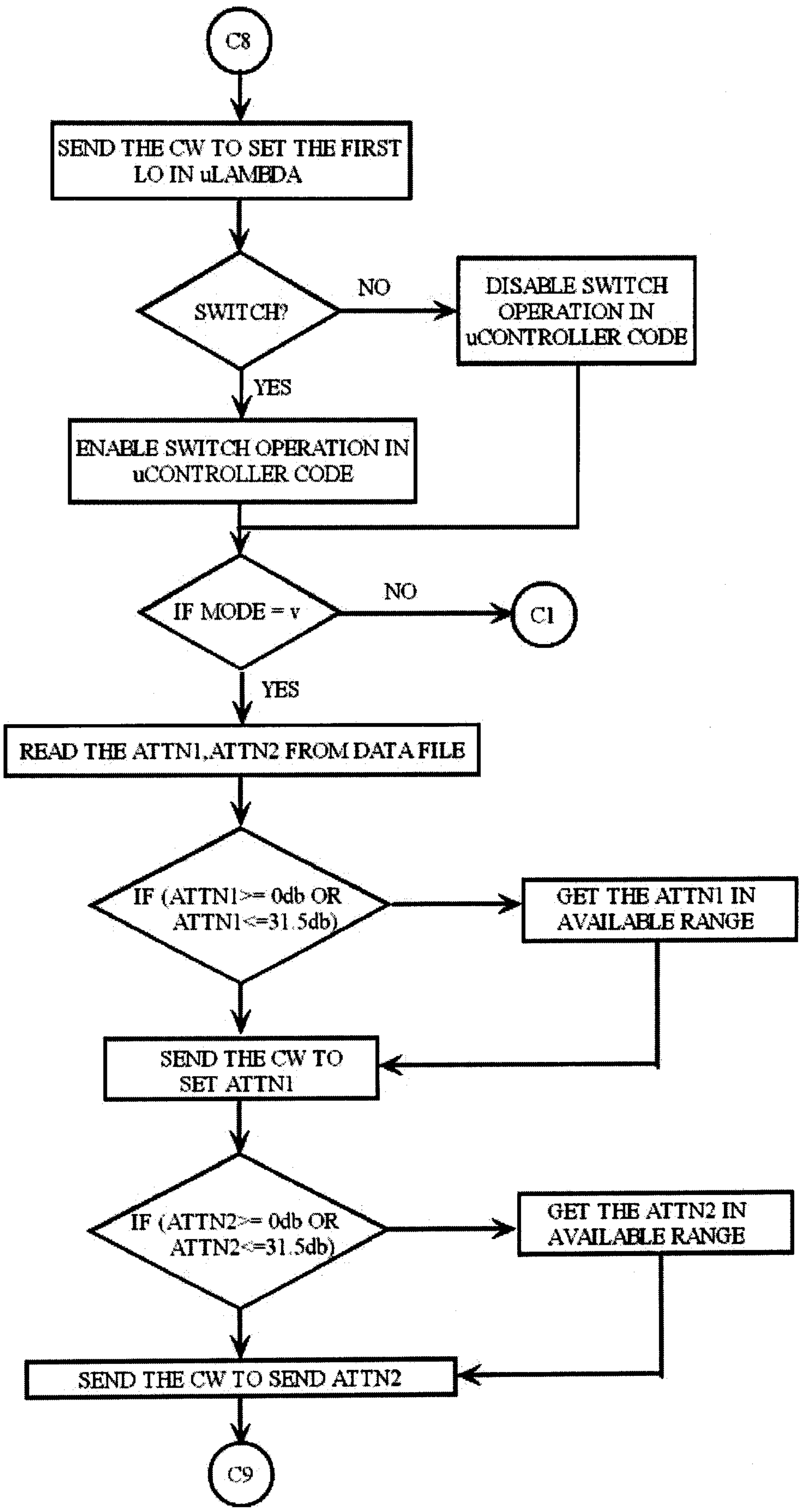
Scope exists further to incorporate many new features in the software modified for characterising the 4-8 GHz front-end receiver system.

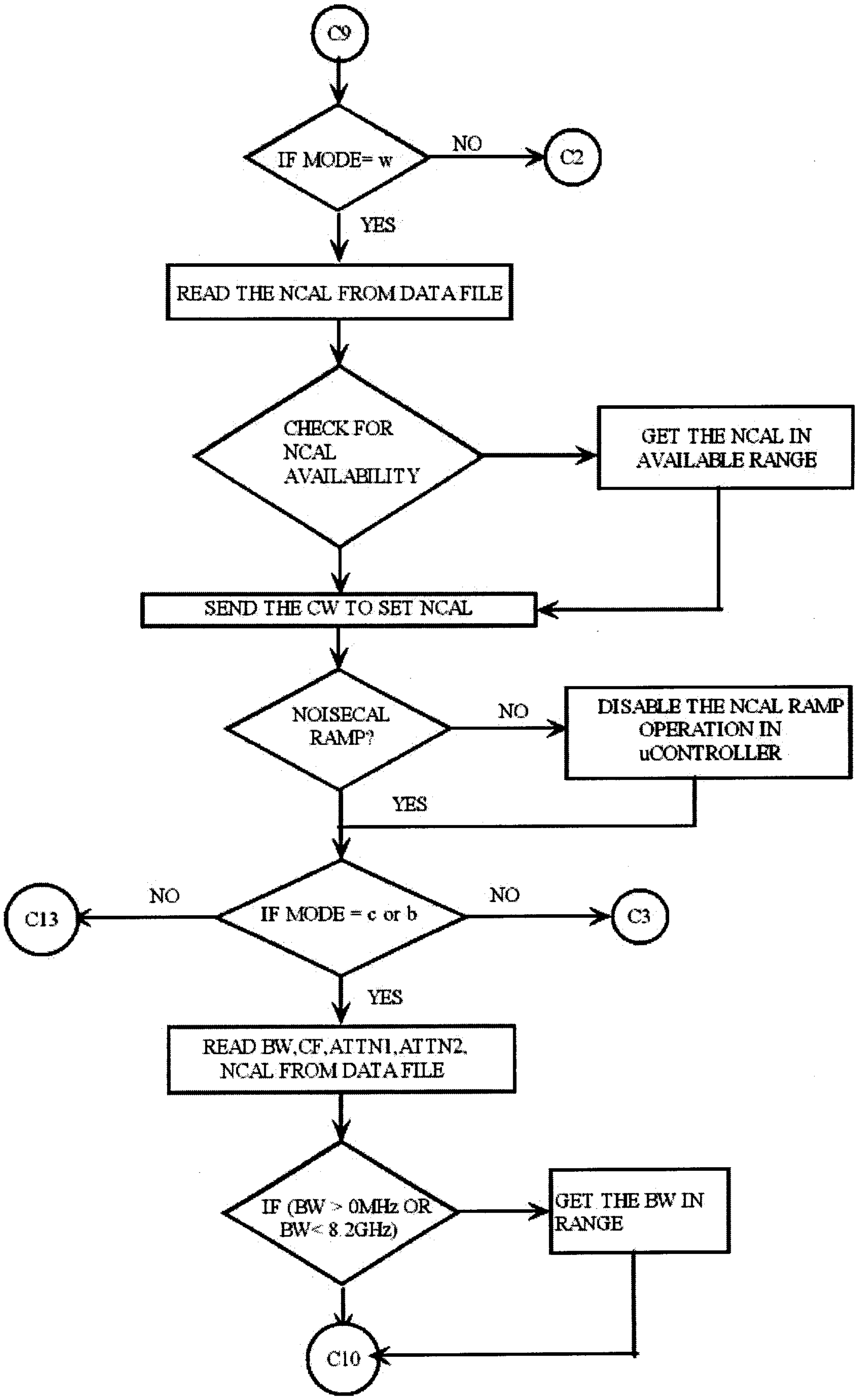
Appendix

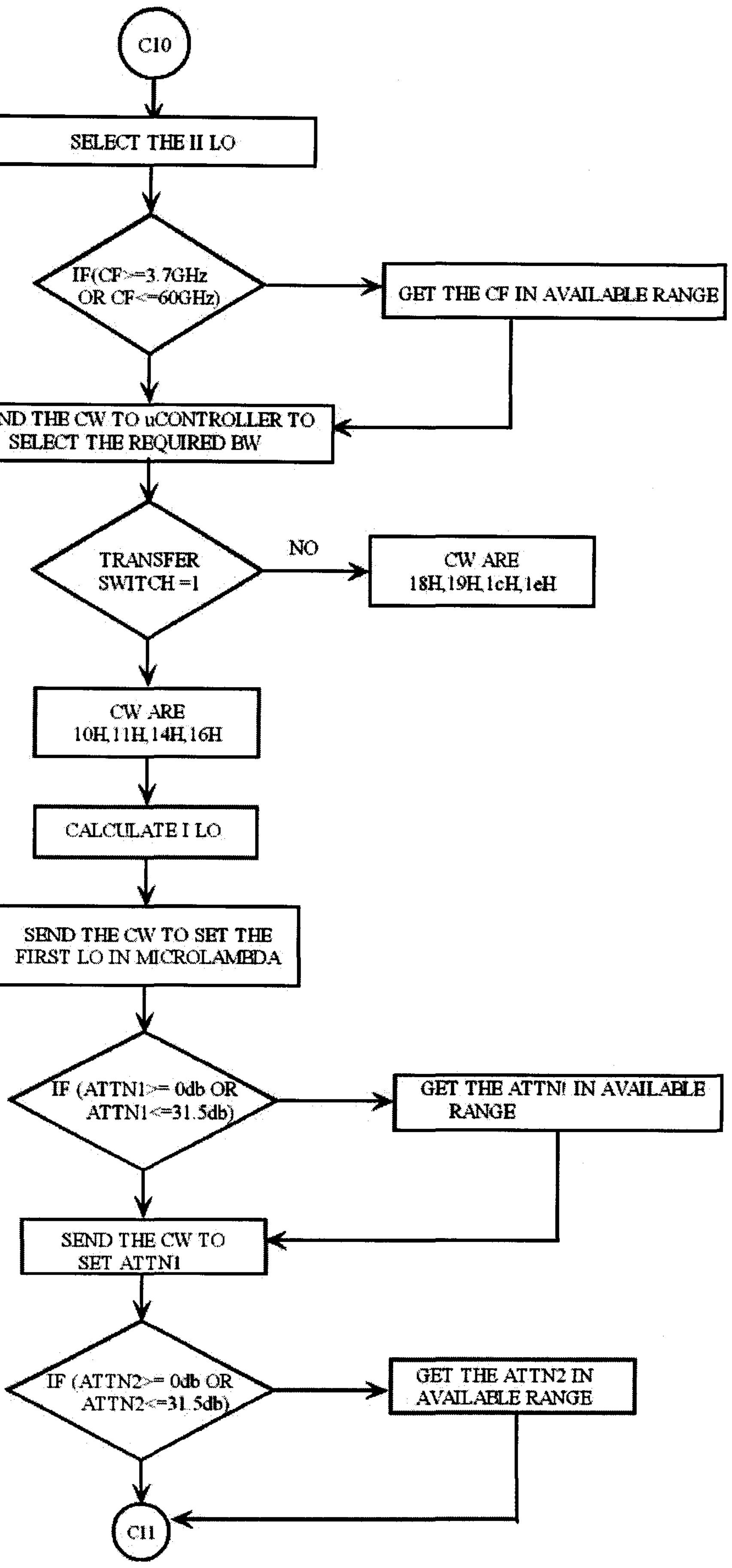
Flow Chart for the C code of control and monitoring system:

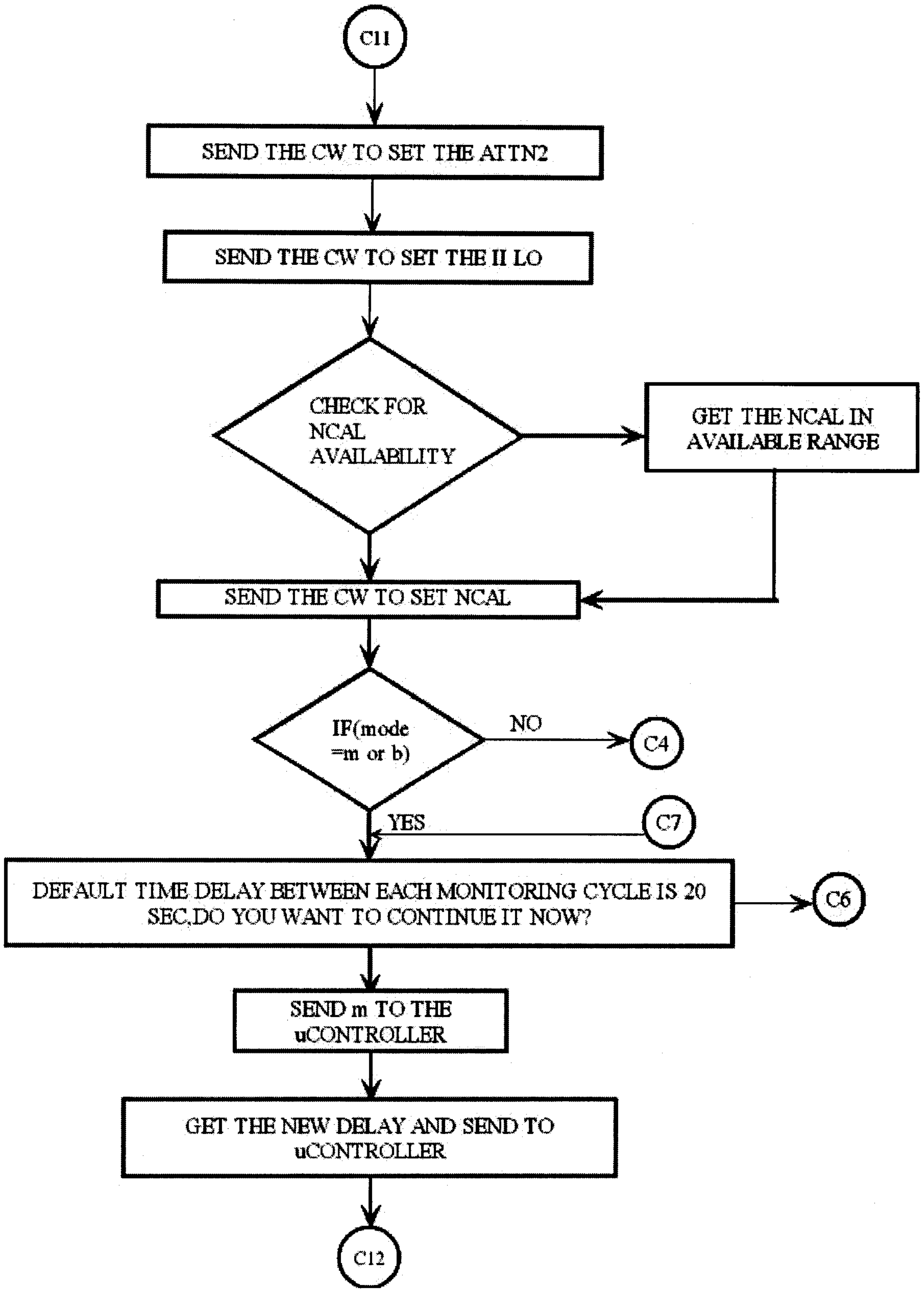
BW=Bandwidth
CW= Control Word
CF= Centre Frequency
LO= Local Oscillator
ATTN=Attenuation

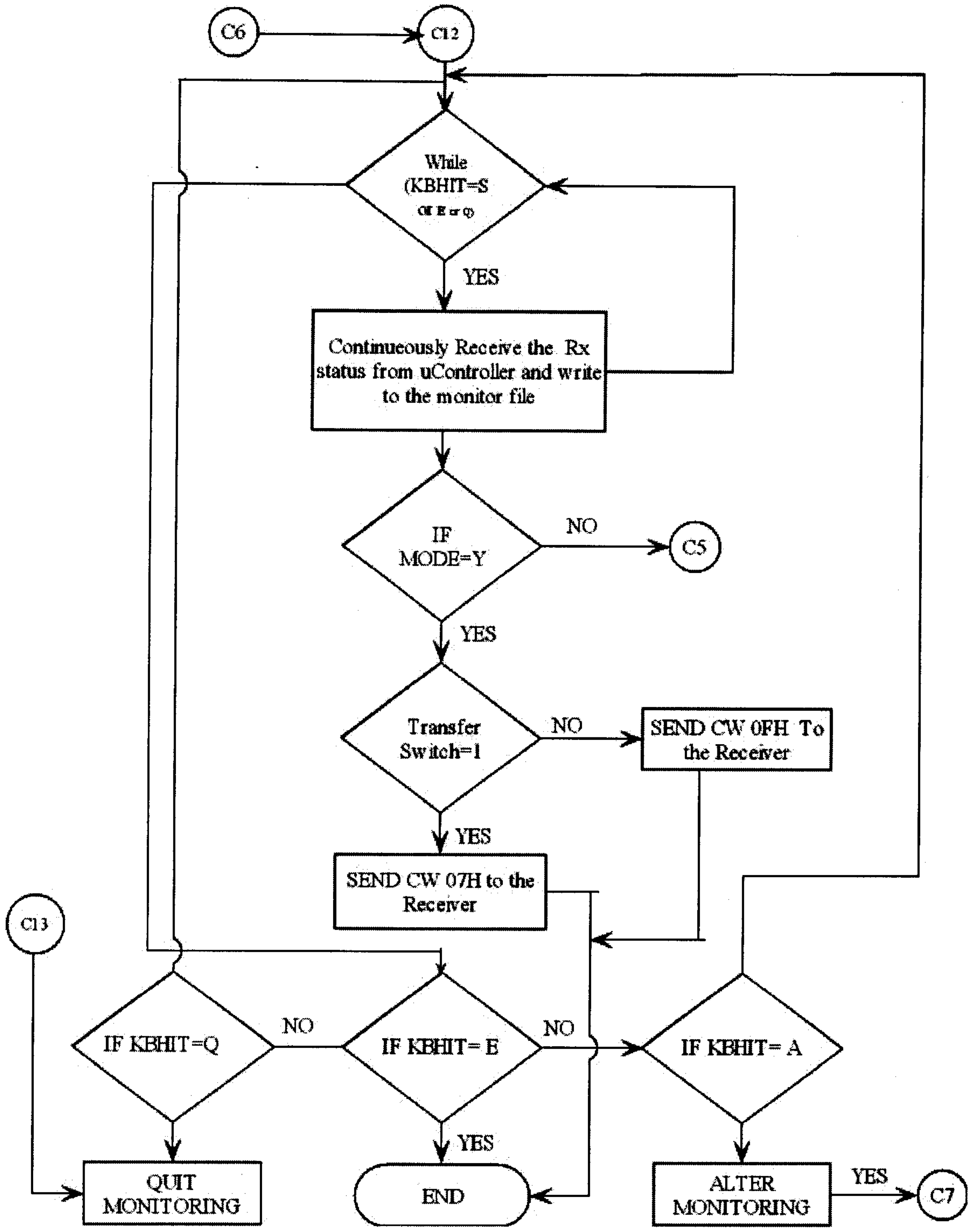

START
INITIALIZE THE PC SERIAL PORT
C5
SELECT ONE OF THESE MODES :
1) 4 - 8 GHz receiver system (X)
2)1.4 GHz Prime Focus receiver (Y)
IF MODE = X
NO
YES
C3
C4
C2
C1
SELECT ONE OF THESE MODES :
1) Frequency settings (u)
2) Attn1 and Attn2(v)
3) Ncal setting (w)
4)Control(c)
5)Monitor(m)
6)Control and Monitor(b)
IF MODE = u
NO
YES
READ BW,CF FROM THE FILE
IF (BW > 0MHz OR BW< 8.2GHz)
GET THE BW IN RANGE
SELECT THE II LO
SEND THE CW TO uCONTROLLER TO SELECT THE REQUIRED BW
CALCULATE I LO
C8


C8
SEND THE CW TO SET THE FIRST LO IN uLAMBDA
SWITCH?
NO
DISABLE SWITCH OPERATION IN uCONTROLLER CODE
YES
ENABLE SWITCH OPERATION IN uCONTROLLER CODE
IF MODE = v
NO
C1
YES
READ THE ATTN1,ATTN2 FROM DATA FILE
IF (ATTN1>= 0db OR ATTN1<=31.5db)
GET THE ATTN1 IN AVAILABLE RANGE
SEND THE CW TO SET ATTN1
IF (ATTN2>= 0db OR ATTN2<=31.5db)
GET THE ATTN2 IN AVAILABLE RANGE
SEND THE CW TO SEND ATTN2
C9


C9
IF MODE= w
NO
C2
YES
READ THE NCAL FROM DATA FILE
CHECK FOR NCAL AVAILABILITY
GET THE NCAL IN AVAILABLE RANGE
SEND THE CW TO SET NCAL
NOISECAL RAMP?
NO
DISABLE THE NCAL RAMP OPERATION IN uCONTROLLER
YES
C13
NO
IF MODE = c or b
NO
C3
YES
READ BW,CF,ATTN1,ATTN2, NCAL FROM DATA FILE
IF (BW > 0MHz OR BW< 8.2GHz)
GET THE BW IN RANGE
C10


C10
SELECT THE II LO
IF(CF>=3.7GHz OR CF<=60GHz)
GET THE CF IN AVAILABLE RANGE
SEND THE CW TO uCONTROLLER TO SELECT THE REQUIRED BW
TRANSFER SWITCH =1
NO
CW ARE 18H,19H,1cH,1eH
CW ARE 10H,11H,14H,16H
CALCULATE I LO
SEND THE CW TO SET THE FIRST LO IN MICROLAMBDA
IF (ATTN1>= 0db OR ATTN1<=31.5db)
GET THE ATTN1 IN AVAILABLE RANGE
SEND THE CW TO SET ATTN1
IF (ATTN2>= 0db OR ATTN2<=31.5db)
GET THE ATTN2 IN AVAILABLE RANGE
C11


C11
SEND THE CW TO SET THE ATTN2
SEND THE CW TO SET THE II LO
CHECK FOR NCAL AVAILABILITY
GET THE NCAL IN AVAILABLE RANGE
SEND THE CW TO SET NCAL
IF(mode =m or b)
NO
C4
YES
C7
DEFAULT TIME DELAY BETWEEN EACH MONITORING CYCLE IS 20 SEC,DO YOU WANT TO CONTINUE IT NOW?
C6
SEND m TO THE uCONTROLLER
GET THE NEW DELAY AND SEND TO uCONTROLLER
C12


C6
C12
While
(KBHIT=S
(or E or Q)
YES
Continueously Receive the Rx status from uController and write to the monitor file
IF MODE=Y
NO
C5
YES
Transfer Switch=1
NO
SEND CW 0FH To the Receiver
YES
SEND CW 07H to the Receiver
C13
IF KBHIT=Q
NO
IF KBHIT= E
NO
IF KBHIT= A
YES
QUIT MONITORING
END
ALTER MONITORING
YES
C7

C code for control and monitoring the receiver system on Linux

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<termios.h>
#include<ctype.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>

#define BAUDRATE B9600
#define device "/dev/ttyS0"   /* PATHNAME OF SERIAL PORT COM1*/
/*
*infile - file pointer of input file home/siva/data.txt used to set the
Rx.
*outfile - file pointer of output file used to monitor the Rx. status
*outfile1 - file pointer of a file used for date & Time setting in the
o/p
file
fo - Center frequency of RF band
flos - Second LO frequency
flo1 - First LO frequency
fif - First IF frequency
sif - Second IF frequency
a1 - attenuation 1
a2 - attenuation 2
*/

FILE *infile,*outfile,*outfile1;
int
fd,i,flos,t,flo1,fif,sif=160,a2,str,s=0,r=0,nc,m=0,p,e,f,sw,a=0,ns,cs,c
s1;
struct termios initerm;
unsigned char rbuf,wbuf='U',dat,*sdata;
char *outputfile,*buf,*d1,d2,mode,d,g,n,*st,y,x;
int step_size=250,prescaler=32,a_ncount,b_ncount,rcount,fref=10,k; //
for
second LO synergy settings
float fo,a1,pf;
int
no_lines_skip=8,no_lines_skip1=9,no_lines_skip2=10,no_lines_skip3=11,no
_lines_skip4=13,no_lines_skip5=14;
char h;

int set_serialport();
int serial_write(unsigned char id);
int serial_write1(unsigned char data);
int serial_read();
int sw_microlambda(unsigned char id,char *dt);
int fo_set(float fo,int flos);
int ncal_set(int nc);
void synergy(int flo2);
int kbhit();
int fo_set1(float fo,int flos);
int fo_set2(float fo,int flos);
```

```
main()
{
st=malloc(sizeof(char));
outputfile=malloc(sizeof(char));
buf=malloc(sizeof(char));
d1=malloc(sizeof(char));
sdata=malloc(sizeof(char));
set_serialport();
system("clear");
printf("Enter 'X' for selection of the 4-8GHz Rx,\nEnter 'Y' for
selection
of the 1.4GHZ prime focus Rx :  \n");
mode=getchar();
getchar();

if((mode=='X'))             /*start of 4-8 GHz Rx system operation*/
{
s=0;
m=0;
while(m!=1)
{

  do
  {
  s=0;
  printf("\nEnter 'u' For Frequency Settings Alone,\nEnter 'v' For
Attn1 and
Attn2 Settings Alone,\nEnter 'w' For Ncal Settings alone\n");
  printf("\nEnter 'c' for sending all the Control Words to RX. only,
\nEnter
'm' for Monitoring the RX.status only,\nEnter 'b' for both sending
Control
words to Rx & Monitoring the RX.status\n" );
  mode=getchar();
  getchar();      /* for enter*/
  //printf("%c\n",mode);
  //if(mode=='c'||(mode=='m'||mode=='b'))
       s=1;
  //else    puts("Please.....");
  //}while(s==0);
  if((mode=='u'))                            /*frequency settings
individually*/
  {

   infile=fopen("data.txt","r");
   // skip first no of lines  //
   for(i=0;i<no_lines_skip;i++)
   while((fgetc(infile))!='\n');

   fscanf(infile,"%f", &fo);
   while((fgetc(infile))!='\n'); // Skip the rest of the line
   printf("\nCenter frequency of RF band is %f GHz\n",fo);


   fscanf(infile,"%d",&sw);
   while((fgetc(infile))!='\n');  //skip the rest of the line//
   printf("LO Switching %d  \n",sw);



   fclose(infile);
```

```
    flos=1290;
    fif=flos+sif;  /*fif=1450MHz*/

    /*frequency band selection*/
    if(fo>=3.7 & fo<=8.2)
    {

          fo_set1(fo,flos);                /*function for individual
                                                 frequency settings*/
    }


    else

    {
     do
     {
       s=0;
       printf("\nRF frequency %f GHz is in out of band",fo);
       printf("\nRF bands available \n center
frequency(GHz)\tbandwidth(MHz)\n");
       printf("enter the RF frequency % with the range\n");
       scanf("%f",&fo);
       getchar();
       if(fo>=3.7 & fo<=8.2)
       s=1;
      } while(s==0);

    printf("center frequency is %f GHz\n",fo);
    fo_set1(fo,flos);
    s=0;
    }

   /* setting  of lo*/



   sprintf(d1,"%d",flo1);
   serial_write('u');
   sw_microlambda('F',d1);
   sw_microlambda('.',"0");



    infile=fopen("data.txt","r");
    // skip first no of lines  //
    for(i=0;i<no_lines_skip1;i++)
    while((fgetc(infile))!='\n');


    fscanf(infile,"%d",&sw);
    while((fgetc(infile))!='\n');  //skip the rest of the line//
   // printf("switching %d  \n",sw);
    fclose(infile);

    serial_write1(sw);
    if(sw==1)                          /* enabling switching of I LO*/
    {
      printf("\n LO is being switched at an offset of 1Mhz \n");
    }
     /*ERROR HANDLING*/
     puts("\nPlease wait......\n");
    //read(fd,&rbuf,1);
    //printf("%c\n",rbuf);
     p=0;
     while (p==0)
     {
       //printf("%c\n",rbuf);
      switch(rbuf)
      {
            case  'C' :{      printf("Error in sending control
words.Pls. check the
First character sent('C')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                      }
            case  'F' :{      printf("Error in sending RF Band select
word.Pls. check
the id sent('F')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                      }
            case  'u' :{      printf("Error in sending First LO control
words.Pls.
check the id sent('u')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                      }
            case  'L' :{      printf("Problem in Resetting First LO
ulambda\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                      }

            case  'O' :{      puts("Successfully Control words were
sent to
uController\n");

                              p=1;
                              break;
                     }
            default   :{
                              read(fd,&rbuf,1);
                     }
      }
    }

    if(mode=='c')
    {
      serial_write('c');
    }


    }  /*end of u operation*/

     else if((mode=='v'))          /* Setting of Attenuation1 and
Attenuation2 individually*/
```

```
        {

          infile=fopen("data.txt","r");
           // skip first no of lines  //
           for(i=0;i<no_lines_skip3;i++)
           while((fgetc(infile))!='\n');

           fscanf(infile,"%f", &a1);
           while((fgetc(infile))!='\n'); // Skip the rest of the line
           printf("\n attenuation1 is %f db\n",a1);

           fscanf(infile,"%d",&a2);
           while((fgetc(infile))!='\n'); //skip the rest of the line
           printf("\n attenuation2 is %d db\n",a2);

           fclose(infile);
         serial_write('J');
         sleep(1);
     /* setting of attenuation in Zsat-31R5*/
     if(a1>=0 & a1<=31.5)
     {
        t=a1/0.5;
     // printf("%d\n",t);
        serial_write('Z');
        sleep(1);
      serial_write1(t);
     // printf("\t");
     }
     else
     {
       do
           {
           printf("\n attenuation1 %g db is not available",a1);
           printf("\n attenuation available is 0_ 31.5 db step size=0.5
db\n");
           printf("enter the required attenuation\n");
           scanf("%f",a1);
           getchar();
           t=a1/0.5;
           if(a1>=0 & a1<= 31.5)
              s=1;
         } while (s==0);
           t=a1/0.5;
           printf("attenuation1 is %f db\n",a1);
           serial_write('Z');
           serial_write1(t);
           //printf("\t");
           s=0;
      }

   /*setting attenuation in HMC307Q*/

   if(a2>=0 & a2<=31)
   {
     sleep(1);
     serial_write('H');
     sleep(1);
     serial_write1(a2);
     //printf("\t");
   }
     else
     {
        do
      {
          printf("\n attenuation2 %g db is not available",a2);
          printf("\n attenuation available is 0- 31 db step size=1
db\n");
          printf("enter the required attenuation\n");
          scanf("%f",a2);
          getchar();
            if(a2>=0 & a2<=31)
          s=1;
        }   while(s==0);
            printf("attenuation2 is %d db\n",a2);
            serial_write('H');
            serial_write1(a2);
            //printf("\t");
          s=0;
     }
      puts("\nPlease wait.......\n");
      read(fd,&rbuf,1);
     // printf("%c\n",rbuf);
      p=0;
     while (p==0)
     {
     //     printf("%c\n",rbuf);
      switch(rbuf)
      {

            case  'Z' :{      printf("Error in sending Attenator1
control word.Pls.
check the id sent('Z')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                       }
            case  'H' :{      printf("Error in sending Attenator2
control word.Pls.
check the id sent('H')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                       }


            case  'O' :{      puts("Successfully Control words were
sent to
uController\n");
                              p=1;
                              break;
                      }
            default   :{
                              read(fd,&rbuf,1);
                      }
      }
    }

    if(mode=='c')
    {
      serial_write('c');
    }

  } /* end of v operation*/
  else if((mode=='w'))              /*Setting of Noise cal
individually*/
  {


     infile=fopen("data.txt","r");
    // skip first no of lines  //
     for(i=0;i<no_lines_skip4;i++)
     while((fgetc(infile))!='\n');

     fscanf(infile,"%d",&nc);
     while((fgetc(infile))!='\n');  //skip the rest of the line//
     printf("noisecal %d  \n",nc);

     fclose(infile);
     serial_write('K');

      /*setting of NOISECAL*/

     if(nc==0||nc==25||nc==40||nc==50||nc==75||nc==90||nc==100)
     {
        ncal_set(nc);
     }
     else
     {
         do
       {
           printf("%d percentage duty cycle is not available\n");
           printf("duty cycles available (%) 0,25,40,50,75,90,100\n");
           printf("enter the duty cycle in%\n");
           scanf("%d",&nc);
           getchar();
           if(nc==0 || nc==25 || nc==40 ||nc==50 ||nc==75
||nc==90||nc==100)
           s=1;
       }while(s==0);
       ncal_set(nc);
       s=0;
     }

       infile=fopen("data.txt","r");
       // skip first no of lines  //
       for(i=0;i<no_lines_skip5;i++)
       while((fgetc(infile))!='\n');


       fscanf(infile,"%d",&ns);
       while((fgetc(infile))!='\n');  //skip the rest of the line//
       printf("ncal ramp %d  \n",ns);
       fclose(infile);

       serial_write1(ns);
       if(ns==1)               /*Enabling Noisecal ramping*/
       {
         printf("\n Noisecal is being ramped...... \n");
       }

      /*ERROR HANDLING*/
      puts("\nPlease wait.......\n");
      read(fd,&rbuf,1);
    //printf("%c\n",rbuf);
      p=0;
      while (p==0)
      {
      //printf("%c\n",rbuf);
      switch(rbuf)
      {

            case  'N' :{      printf("Error in sending Noise cal
control word.Pls.
check the id sent('N')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                       }

            case  'O' :{      puts("Successfully Control words were
sent to
uController\n");

                              p=1;
                              break;
                      }
            default   :{
                              read(fd,&rbuf,1);
                      }
      }
    }

    if(mode=='c')
    {
      serial_write('c');
    }

  }   /*end of w operation*/


else if((mode=='c'||mode=='b'))
  {
   printf("Required parameters to set Control words are available in
'home/siva/data.txt'\n");
   printf("If You want to change as per your requirement \n");
   printf("Please do it now then enter 'g'\n");
   d=getchar();
   //printf("%c\n",d);
   getchar();
   if(d=='g')
   {
      infile=fopen("data.txt","r");
      do
      {

            fscanf(infile,"%s",buf);
            //printf("%s\n",buf);
            str=strcmp(buf,"start");
      }       while(str);
             //fscanf(infile,"%d",&str);
             //printf("%d\n",str);

      do
      {
            d2=fgetc(infile);
      }while(d2!='\n');
      fscanf(infile,"%f",&fo);
      printf("Center frequency of RF band is %f GHz\n",fo);
```

```
        do
        {
                d2=fgetc(infile);
           }while(d2!='\n');
          fscanf(infile,"%d",&sw);
            printf("switching %d  \n",sw);

        do
        {
               d2=fgetc(infile);
          }while(d2!='\n');
            fscanf(infile,"%d",&cs);
            printf("channel swapping %d  \n",cs);

        do
        {
              d2=fgetc(infile);
        }while(d2!='\n');
        fscanf(infile,"%f",&a1);
        printf("Attenuation1 is %f dB\n",a1);

        do
        {
              d2=fgetc(infile);
        }while(d2!='\n');
        fscanf(infile,"%d",&a2);
        printf("Attenuation2 is %d dB\n",a2);

        do
        {
              d2=fgetc(infile);
        }while(d2!='\n');
        fscanf(infile,"%d",&nc);
        printf("Noise cal duty cycle is %d%\n",nc);



        fclose(infile);
        //serial_write('I');

        flos=1290;              //1290 Second LO frequency in MHZ
        fif=flos+sif;
        //flo1=ceilf(fo*1000.0000)-fif;

        /*FREQUENCY BAND SELECTION*/

        if(fo>=3.7 & fo<=8.2)
        {
              if( cs==0)
            {
                fo_set(fo,flos);
            }
           else if(cs==1)
            {
              fo_set2(fo,flos);            /*Function when the transfer
switch is
enabled*/
            }
        }
        else
        {
              do
            {
                  s=0;
                  printf("\nRF FREQUENCY  %f GHz IS IN OUT OF 
BAND",fo);
                  printf("\nRF Bands available\n Center 
frequency(GHz)\t Bandwidth(MHz)\n
4.1 \t\t\t 800 \n 5.2 \t\t\t 1000 \n 6.68 \t\t\t 50 \n 7.8 \t\t\t 800 
\n");
                  printf("Enter the RF frequency with in this 
range\n");
                  scanf("%f",&fo);
                  getchar();
                  if(fo>=3.7 & fo<=8.2)
                        s=1;
            }while(s==0);
            printf("Center frequency is %f GHz\n",fo);
            fo_set(fo,flos);
            s=0;
      }

      /* SETTING FIRST LO IN MICROLAMBDA*/
      sprintf(d1,"%d",flo1);
      serial_write('u');
      sw_microlambda('F',d1);
      sw_microlambda('.',"0");
      //printf(" \t");
      //printf("%g\n",fo)



      /* SETTING ATTENUATION IN ZSAT-31R5*/
      if(a1>=0 & a1<=31.5)
      {
            t=a1/0.5;
            //printf("%d\n",t);
            serial_write('Z');
            serial_write1(t);
            //printf("\t");
      }
      else
      {
            do
            {
                  printf("\nATTENUATION1 %g dB IS NOT AVAILABLE",a1);
                  printf("\nAttenuation available is 0 - 31.5 dB Step 
size= 0.5 dB \n");
                  printf("Enter the required Attenuation\n");
                  scanf("%f",&a1);
                  getchar();
                  t=a1/0.5;
                  if(a1>=0 & a1<=31.5)
                        s=1;
            }while(s==0);
            t=a1/0.5;
            printf("Attenuation1 is %f dB\n",a1);
            serial_write('Z');
            serial_write1(t);
            //printf("\t");
            s=0;

      }
      /*SETTING ATTENUATION IN HMC307Q*/
      if(a2>=0 & a2<=31)
      {
            serial_write('H');
            serial_write1(a2);
            //printf("\t");
      }
      else
      {
            do
            {
                  printf("\nATTENUATION2 %d dB IS NOT AVAILABE",a2);
                  printf("\nAttenuation available- Min = 1 dB Max = 31
dB in 1 dB Step
\n");

                  printf("Enter the required Attenuation\n");
                  scanf("%d",&a2);
                  getchar();
                  if(a2>=0 & a2<=31)
                        s=1;
            }while(s==0);
            printf("Attenuation2 is %d dB\n",a2);
            serial_write('H');
            serial_write1(a2);
            //printf("\t");
            s=0;
      }

      /* SETTING SECOND LO IN SYNERGY*/
      synergy(flos);
      //printf("\t");

      /* SETTING NOISE CAL*/
      if(nc==0||nc==25||nc==40 ||nc==50 ||nc==75 ||nc==90||nc==100)
      {
            ncal_set(nc);
      }
      else
      {
            do
            {
                  printf("%d Percentage duty cycle is not
avilable\n",nc);
                  printf("Duty cycles available(%)
0,25,40,50,75,90,100\n");
                  printf("Enter the duty cycle in%\n");
                  scanf("%d",&nc);
                  getchar();
                  if(nc==0|| nc==25 ||nc==40 ||nc==50 ||nc==75
||nc==90||nc==100)
                        s=1;
            }while(s==0);
            ncal_set(nc);
            s=0;
      }
    } /*End of first if loop (g)*/
    /*ERROR HANDLING*/
    puts("\nPlease wait\n");
    //read(fd,&rbuf,1);
    //printf("%c\n",rbuf);
     p=0;
     while (p==0)
     {
      //printf("%c\n",rbuf);
      switch(rbuf)
      {
            case  'C' :{      printf("Error in sending control
words.Pls. check the
First character sent('C')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'F' :{      printf("Error in sending RF Band select
word.Pls. check
the id sent('F')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'u' :{      printf("Error in sending First LO control
words.Pls.
check the id sent('u')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'L' :{      printf("Problem in Resetting First LO
ulambda\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'Z' :{      printf("Error in sending Attenator1
control word.Pls.
check the id sent('Z')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'H' :{      printf("Error in sending Attenator2
control word.Pls.
check the id sent('H')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'S' :{      printf("Error in sending Second LO
control words.Pls.
check the id sent('S')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
                              exit(0);
                        }
            case  'N' :{      printf("Error in sending Noise cal
control word.Pls.
check the id sent('N')\n");
                              printf("Then try by sending control words
again\n");
                              p=1;
```

```
                                        exit(0);
                               }
               case  'M'  :{            printf("Error in sending Monitoring
control word.Pls.
check the id sent('M')\n");
                                        printf("Try by monitoring part alone\n");
                                        p=1;
                                        exit(0);
                               }
               case  'O'  :{            puts("Successfully Control words were
sent to
uController\n");
                                        p=1;
                                        break;
                              }
               default    :{
                                        read(fd,&rbuf,1);
                              }
        }
     }
     if(mode=='c')
     {
       //serial_write('c');
     }

  }/*End of if loop(c||b)*/



  m=1;
  n='0';
  s=0;
  if(mode=='m'||mode=='b')
  {
      printf("\nMonitoring the Receiver Status........\n\n");
      puts("Enter the file name with extension '.txt' to write the
monitored data
\n");
      scanf("%s",outputfile);
      getchar();
      //printf("outputfile %s\n",outputfile);
      outfile=fopen(outputfile,"w+");
      outfile1=fopen("amonitor1.txt","r+");
      system("date +%c> amonitor1.txt");
      fseek(outfile1,0,SEEK_SET);
      while(s!=4)
      {
            fscanf(outfile1,"%s",sdata);
            fprintf(outfile,"%s  ",sdata);
            s++;
      }
      s=0;
      fclose(outfile1);
      fprintf(outfile,"\nTime      Start\t  FBS(GHz)\tLD-LO1      LD-LO2
ATTN1(dB) ATTN2(dB)  NCAL(%)    LO2\t\tLO1\t\tII IF\t\tI
IF\t\t15V(CFH)\t5V(CFH)\t\t-(-5V)(CFH)\t15V(Ref)\t5V(LO)\t      -(-
5V)(II
IF)\t15V(II IF)\t5V(II IF)      ");
      fprintf(outfile,"-(-5V)(I IF)\t15V(I IF)\t5V(I
IF)\t-(-5V)(RF)\t15V(RF)      \t5V(RF)\t\t-(-
5V(NC))\t15V(LO)\t\t5V(NC)\t\t-(-5V(NC))\t15V(NC)\t\t5V(NC)\t\t-
15V(CFH)\tStop\n");
      fclose(outfile);
      printf("\nMonitored data will be stored in
'/home/siva/%s'\n\n",outputfile);

  }
     /*To monitor the receiver status*/
  while(mode=='m'||mode=='b')
  {
      printf("The minimum time delay between each monitoring cycle is 5
seconds,maximum is 1275 seconds (21.25 minutes)\n");
      m=0;
      printf("Do you want to change it now?\n");
      s=0;
      do
      {
            printf("Enter(y/n)\n");
            do
            {
                  g=getchar();
                  getchar();
            }while(g=='\n');
            if(g=='y'||g=='n')
            s=1;
            else  puts("Please.....");
      }while(s==0);
      s=0;
      t=1;
      if(g=='y')
      {
            serial_write('M');
            puts("Delay = t * 5 seconds t(min)=1,t(max)=255 t=0 is
equal to 256
(Maximum delay) \n");
            do
            {
                  s=0;
                  puts("Enter t (integer)\n");
                  scanf("%d",&t);
                  getchar();
                  if(t>=0 && t<=255)
                  s=1;
                  else  puts("Please.....");
            }while(s==0);
            s=0;
            if (t==0)
            printf("Now delay  is maximum = 1280 Seconds\n");
            else
            printf("Now delay = %d seconds\n",t*5);
            serial_write1(t);
      }
      else if(g=='n')
      {
            serial_write('M');
            printf("Now delay = %d seconds\n",t*5);
            serial_write1(t);
      }
      //outfile=fopen(outputfile,"a");
      puts("\nPress 'a' to alter the delay between each monitoring\n");
      puts("Press 'q' to Quit monitoring\n");
      puts("Press 'e' to Exit the programme\n");
      r=0;
      s=0;
      while(s!=1)
      {
            if(n != 'a'|| n!='q'|| n!='e')
                  serial_read();
            e=0;
            while(e==0)
            {
                  n='0';
                  //sleep(.9);
                  n=kbhit();
                  if(n == 'a'|| n=='q'|| n=='e')
                        e=1;
                  r++;
                  if(r==(t*5))
                        s=1;
                  e=e||s;
                  s=0;
            }
            r=0;

            if(n=='a')
            {
                  s=1;
                  puts("\nMonitoring is stopped & the delay between
each monitoring is set
to 5 seconds\nyou can open a new file for monitoring & enter the new
delay
between each monitoring\n");
                  //puts("Please wait for 2 seconds\n");
                  serial_write('E');
                  printf("Do you want to open a new file for
monitoring?\n");
                  do
                  {
                        s=0;
                        printf("Enter(y/n)\n");
                        do
                        {
                              g=getchar();
                              getchar();
                        }while(g=='\n');
                        if(g=='y'||g=='n')
                        s=1;

                  }while(s==0);
                  if(g=='y')
                  {
                        puts("\nEnter the file name with extension
'.txt' to write the monitored
data \n");
                        scanf("%s",outputfile);
                        getchar();
                        printf("Monitored data will be stored in
'/home/siva/%s'\n\n",outputfile);
                        outfile=fopen(outputfile,"w+");
                        outfile1=fopen("amonitor1.txt","r+");
                        system("date +%c> amonitor1.txt");
                        fseek(outfile1,0,SEEK_SET);
                        while(r!=4)
                        {
                              fscanf(outfile1,"%s",sdata);
                              fprintf(outfile,"%s  ",sdata);
                              r++;
                        }
                        r=0;
                        fclose(outfile1);
                        fprintf(outfile,"\nTime       Start\t
FBS(GHz)\tLD-LO1     LD-LO2
ATTN1(dB) ATTN2(dB)  NCAL(dB)    LO2\t\tLO1\t\tII IF\t\tI
IF\t\t15V(CFH)\t5V(CFH)\t\t-(-5V)(CFH)\t15V(Ref)\t5V(LO)\t      -(-
5V)(II
IF)\t15V(II IF)\t5V(II IF)     ");
                        fprintf(outfile,"-(-5V)(I IF)\t15V(I IF)\t5V(I
IF)\t-(-5V)(RF)\t15V(RF)      \t5V(RF)\t\t-(-
5V(NC))\t15V(LO)\t\t5V(NC)\t\t-(-5V(NC))\t15V(NC)\t\t5V(NC)\t\t-
15V(CFH)\tStop\n");
                        fclose(outfile);
                  }
            }
            if(n=='q')
            {
                  s=1;
                  m=0;
                  mode='q';
                  puts("\nquitting the monitoring loop....\n");
                  puts("Please wait for 2 seconds\n");
                  serial_write('E');
            }
            if(n=='e')
            {
                  s=1;
                  m=1;
                  mode='e';
                  puts("\nExiting ....\n");
                  serial_write('E');
            }

      }
   } /*end of while m||b*/
    printf("\n do u want to continue=? \n");
    h=getchar();
    getchar();
  } while(h=='y');

  } /* end of first while*/
} /* end  of X operation */
    else if((mode=='Y'))

      {
        infile=fopen("data1.txt","r");
      do
      {

            fscanf(infile,"%s",buf);
            //printf("%s\n",buf);
            str=strcmp(buf,"start");
      }       while(str);
             //fscanf(infile,"%d",&str);
             //printf("%d\n",str);
       do
       {
          d2=fgetc(infile);
         }while(d2!='\n');
         fscanf(infile,"%d",&cs1);
        // printf("channel swapping %d  \n",cs1);
       fclose(infile);

      if(cs1==0)
```

```
        {
           dat=0x0f;     /*  0  0  0  0  1  1  1  1 */
                         /*  0              F       */
           printf("\n channel unswapped \n");
         sleep(1);
           serial_write1(dat);
           printf("\n Primary Focus Receiver is selected and control word
is
0X0F\n");
          }

           else if(cs1==1)
         {


             dat=0x07;         /*  0  0  0  0  0  1  1  1*/
                               /*      0        7      */
             printf("\n channel swapped \n");
           sleep(1);
             serial_write1(dat);
             printf("\n Primary Focus Receiver is selected and control
word is
0X07\n");
           }
      }
} /* end of main*/

int set_serialport()
{
      int f,c1,w,r1;

      fd = open(device,O_RDWR |O_NOCTTY |O_NDELAY);/*O_RDWR -open a
decice for
both reading & writing,O_NOCTTY - device  will not become
                                          the process's controling
terminal O_NDELAY - Neither the open nor
any  subsequent  opera-
                                                tions on the file
descriptor which is returned will
cause the calling process to wait*/
      //printf("%d\n",fd);
      tcgetattr(fd, &initerm);                 /*This function gets the
parameters
associated with the file/device referred to by the handle fd and stores
                                          them in the termios structure
initerm */
      cfmakeraw(&initerm);
            initerm.c_cflag |= BAUDRATE;       /*c_cflag - Control flag
constant
BAUDRATE=9600 bits/second*/
            initerm.c_cflag |= CS8;            /*CS8 - 8 bit no parity
1 stop bit */
            initerm.c_iflag |= ICRNL;          /*c_iflag - input flag
constant ICRNL -
Translate carriage return to newline on input */
            c1=tcsetattr(fd,TCSANOW,&initerm);   /*This function sets
termios
structure for the device open on the handle fd from the
                                          structure termios initerm
TCSANOW - the change occurs
immediately.*/
      f=fcntl(fd,F_SETFL,0);                   /*This function performs
the operation
```

```
specified by F_SETFL on the file open on handle fd.
                                                  F_SETFL - Sets the open mode
and status flags associated with
the handle fd     */
      //w=write(fd,&wbuf,1);
      //r1=read(fd,&rbuf,1);
      //printf("%d,%d,%d,%c,%d,%c\n",f,c1,w,wbuf,r1,rbuf);
      return 0;
}

int serial_write(unsigned char id)
{
      int i,n;
      unsigned char rdata;

      printf("%c\n",id);
      write(fd,&id,1);                  /*Write a byte in to a
file/device open on the
handle fd*/
      //read(fd,&rdata,1);
      //printf("%c\n",rdata);
      //printf("\t");
      return 0;
}

int serial_write1(unsigned char data)
{
      int i;
      unsigned char rdata;

      printf("%x\n",data);
      write(fd,&data,1);
      //read(fd,&rdata,1);
      //printf("%x\n",rdata);
      sleep(1);
      //printf("\t");
      return 0;
}

int serial_read()
{
      int i,j;
      float adata,scalefactor=1.6;
      unsigned char rdata,a,b,c,fbs,ncal;

            //rdat=malloc(sizeof(char));
            //b=malloc(sizeof(char));
            outfile=fopen(outputfile,"a+");
            outfile1=fopen("amonitor1.txt","r+");
            system("date +%T > amonitor1.txt");
            while(rdata!=204)
            {
            r=read(fd,&rdata,1);          /*read a byte from a
file/device open on the handle
fd*/
            //printf("%d\n",rdata);
            }
            fseek(outfile1,0,SEEK_SET);
            fscanf(outfile1,"%s",sdata);
            fprintf(outfile,"%s  ",sdata);
            fclose(outfile1);

            fprintf(outfile," %d\t",rdata);
            /*TTL SIGNALS MONITORING*/
            r=read(fd,&rdata,1);
            //printf("%d\n",rdata);
            a=rdata/16;
            //printf("a=%d\n",a);
            r=read(fd,&rdata,1);
            fbs=(a+((rdata/16)%2)*16);
            //printf("%X\n",fbs);
            switch(fbs)
            {

                  case  0x1c :{     fprintf(outfile," 3.7 - 4.5
");
                              break;
                             }

                  case  0x1b :{     fprintf(outfile," 4.7 - 5.7
");
                              break;
                             }
                  case  0x1e :{     fprintf(outfile," 6.675 - 6.725
");
                              break;
                             }
                  case  0x1f :{     fprintf(outfile," 7.4 - 8.2
");
                              break;
                             }
            }
            if((rdata/16)%2)
                  rdata=(rdata/16)-9;
            else
                  rdata=(rdata/16)-8;
            fprintf(outfile,"%X          ",(rdata/2)%2);
            fprintf(outfile,"%X       ",rdata/4);
            r=read(fd,&rdata,1);
            a=rdata/16;
            r=read(fd,&rdata,1);
            b=rdata/16;

      fprintf(outfile,"%2.1f\t\t",(a+((((b%2)*1)+(((b/2)%2)*2))*16))/2.
0);
            r=read(fd,&rdata,1);
            a=rdata/16;
            fprintf(outfile,"%d\t
",((b/4)+(a*4)));//((a%2)*1)+(((a/2)%2)*2)+(((a/4)%2)*4)*4));
            r=read(fd,&rdata,1);
            ncal=rdata/16;
            switch(ncal)
            {

                  case  0 :{  fprintf(outfile,"0 \t    ");
                              break;
                           }

                  case  1 :{  fprintf(outfile,"25\t    ");
                              break;
                           }
                  case  2 :{  fprintf(outfile,"40\t    ");
                              break;
                           }
                  case  3 :{  fprintf(outfile,"50\t    ");
```

```
                                break;
                            }
                case  4 :{  fprintf(outfile,"75\t   ");
                            break;
                            }
                case  5 :{  fprintf(outfile,"90\t   ");
                            break;
                            }
                case  6 :{  fprintf(outfile,"100\t  ");
                            break;
                            }
            }
            //printf("a=%X\n",ncal);
            //fprintf(outfile,"     %d     ",ncal);
            /*ANALOG SIGNALS MONITORING*/
            /*1.POWER LEVELS OF RF SIGNALS MONITORING*/
            for (i=1;i<=4;i++)
            {
                  r=read(fd,&rdata,1);
                  //printf("%f\n",rdata*(.01));
                  fprintf(outfile,"%f\t",rdata*(.01));
            }
            /*2.POWER SUPPLY MONITORING*/
            for (i=1;i<=21;i++)
            {
                  r=read(fd,&rdata,1);
                  //printf("%f\n",rdata*(.1));
                  fprintf(outfile,"%f\t",rdata*(.1));
            }
            r=read(fd,&rdata,1);
            fprintf(outfile,"%d\t",rdata);
            //time=system("date +%r");
            //system("date +%r >> amonitor.txt");
            fprintf(outfile,"\n");
            //system("stat amonitor.txt >> amonitor.txt");
            fclose(outfile);
      return 0;
}

int fo_set(float fo,int flos)
{
/* 8 BIT WORD D7 D6 D5 D4 D3 D2 D1 D0 , D4=0 -> 0.5 - 1.7 GHz, D5=0 ->
2 - 4
GHz
      D2,D1,D0 -> 4 - 8 GHz */


          if(fo>=3.7 && fo<=4.5)                          /*D7 D6 D5 D4 D3
D2 D1 D0*/
         {
            flo1=ceilf(fo*1000.000) + fif;
            //printf("%d,%g\n",flo1,flos);
            dat=0x18;                       /*0  0  0  1  1  0  0  0 */
         }                                  /*    1        8 (HEX)*/
        else if(fo>4.5 && fo<4.7)
         {
            printf("No RF Band is covering 4.5 - 4.7 GHz\n");
            exit(0);
         }
         else if(fo>= 4.70001 && fo<=5.7)
         {
            flo1=ceilf(fo*1000.000)-fif;
            dat=0x19;                       /*0  0  0  1  1  0  0  1 */
```