IAP_ECS

Emergency Communication System to be implemented into IAP Satellite System

IAP_ECS Demonstration Platform

Project duration: May 2013 – Jan 2014

Authors:

Mahmoud Zohby

Samir Mourad

January 2014

Content

Abbrevations

ECS   Emergency Communication System

1   Abstract

Emergency Communication System
شبكة اتصالات آمنة للطوارئ و الاسعافات

Last update: 03 December 2013

SCS-SMS after changes to GIS-STD

System Design of v1.0

Advanced
Encryption
Standard
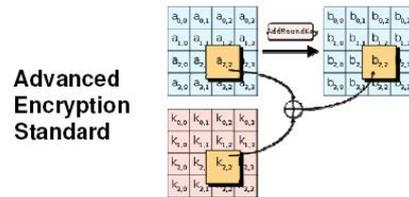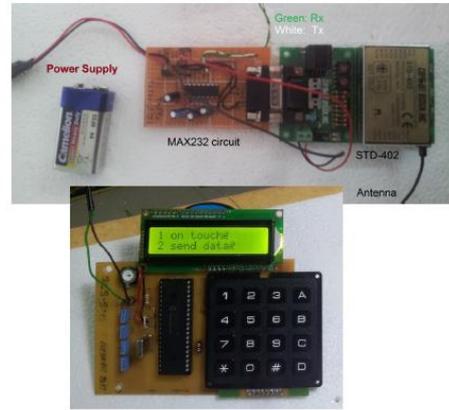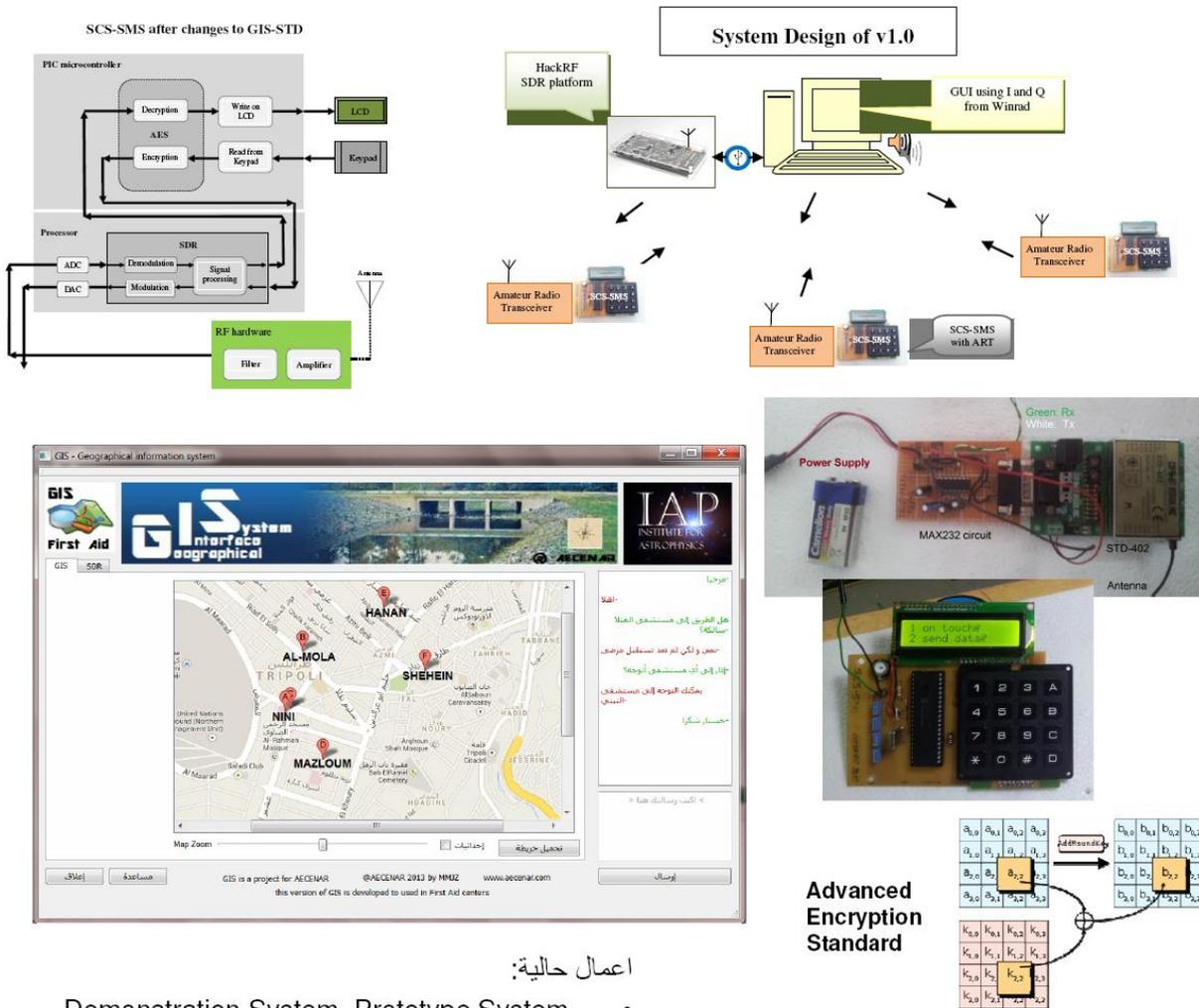
اعمال حالية:

- Demonstration System, Prototype System

الحاجيات لعام 2014:

- 2 اشخاص
- 5.000$ للمواد

2 Project Management

2.1 Project Definition History

First there were developed SMS-SCS and AES. (June – August 2013). Later on there was an investigation about the possibility of using SDR (Software Designed Radio). (September, October).

Later on it was decided to make a demonstration system for an Emergency Communication System. (October). IAP ECS was developed. SMS-SCS and AES were integrated into this system. (October 2013 – January 2014).

2.2 System budget (time and cost) for Demo system

Part	Task	Time (week)	Cost (USD)
Project Plan	1 engineer (Project manager)	3 weeks	E.C.
Client Side	SCS-SMS project hardware (components for 2 items)	2 week	**30**$ x 2
	PIC program development (1 engineer)	3 weeks	E.C.
	RF transceiver for SCS-SMS (components for 3 sides)	2 weeks	**12**$ x 6
Base Station Side	Know how of HDSDR	1/2 week	E.C.
	Know how of WinRad	1/2 week	E.C.
	Know how of HackRF	1 week	E.C.
	VC++ & Qt software tutorials	2 weeks	E.C.
	Qt GUI interface	2 weeks	E.C.
	SDR platform (2 USB stick)	1 week	**40**$ x 2
Overall System	Documentation and report	1 week	E.C.
	Testing system and solving problem	1 week	E.C.
Total:		19 weeks	212 $

One engineer cost **200**$ each week. So, for 19 weeks he costs:	200$ x 19= 3800$
Summation with the hardware cost:	4012$
4000$ in approx. 5 months	

2.3 At 21 Jan still open tasks for IAP ECS Demo System when using (Only Integration)

Event	Time
Using WinRad to receive Radio wave using ran T-stick[+]	1 week
~~Complete the SCS-SMS project 1 of secured communication system~~	~~3 weeks~~
SW for connection SCS-SMS hardware to the RF module, testing	1 week
Take I and Q from WinRad to a file	1 week
Adapting GUI interface to read SMS from file	2 weeks
System testing	1 week

Planned time: approximately **6 weeks**

3 Basics

3.1 Communication Basics[1]
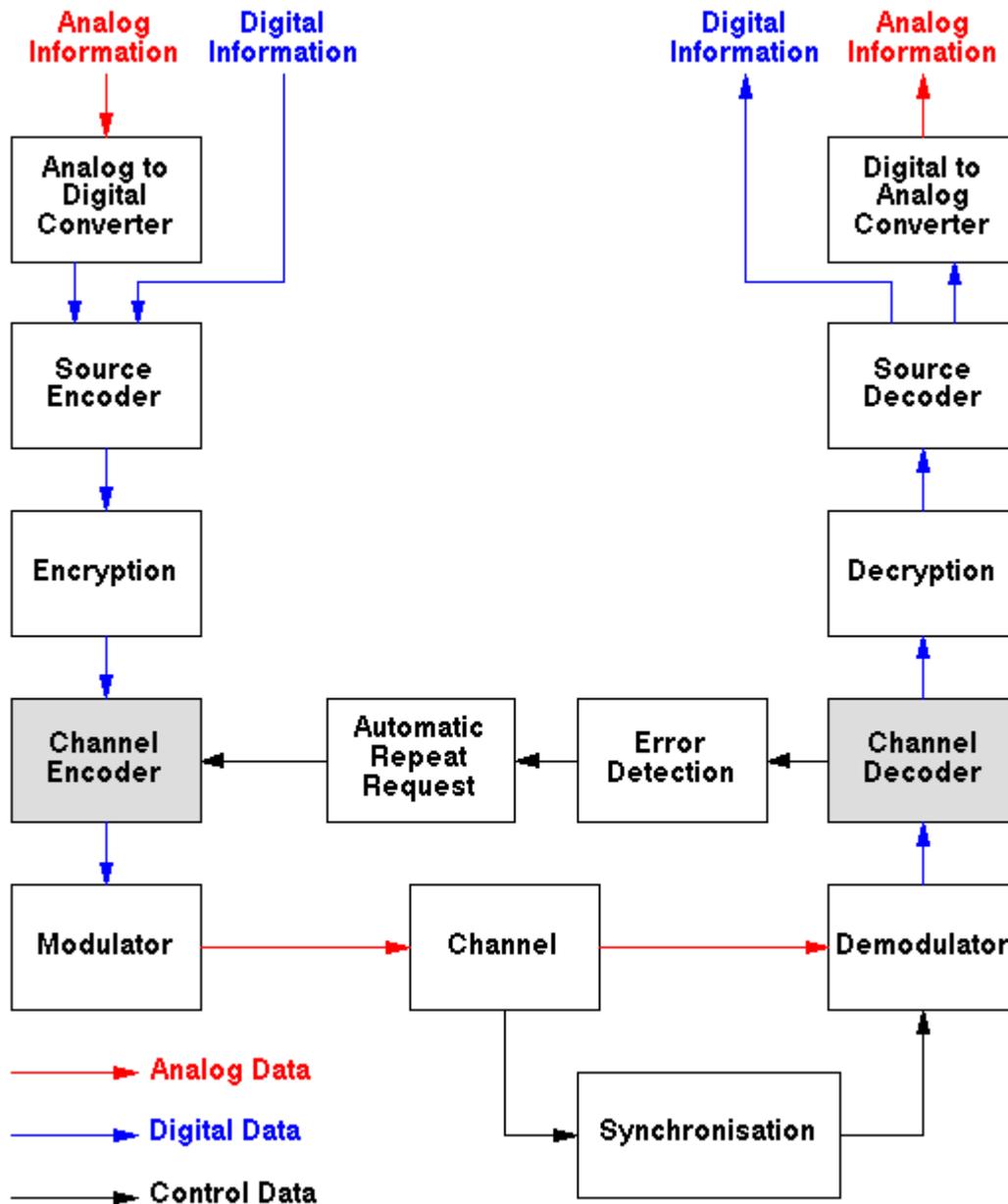
Die Aufgabe der Nachrichtentechnik besteht darin, Informationen von einem Sender zu einem Empfänger zu befördern. Die Nachrichtentechnik kann grob in zwei große Gebiete geteilt werden, in die

• Übertragungstechnik und in die

• Vermittlungstechnik.

[1] Many is taken from: Prof. Dr.-Ing. Gerhard P. Fettweis, Technische Universität Dresden, Fakultät Elektrotechnik, Skript zur Vorlesung Einführung in die Nachrichtentechnik, Sommersemester 2012

Basics

Beispiele für nachrichtentechnische Anwendungen sind:

- Hörrundfunk und Fernsehen
    – analog: AM-Radio (Mittelwelle), FM-Radio (UKW),
    – digital: DAB (digital audio broadcasting), DVB (digital video broadcasting),

- Telefon
    – Festnetz,
    – Mobilfunk.

Nachrichtenübertragungssysteme

Man kann Nachrichtenübertragungssysteme durch das in Abb. 2.1 dargestellte Modell beschreiben.
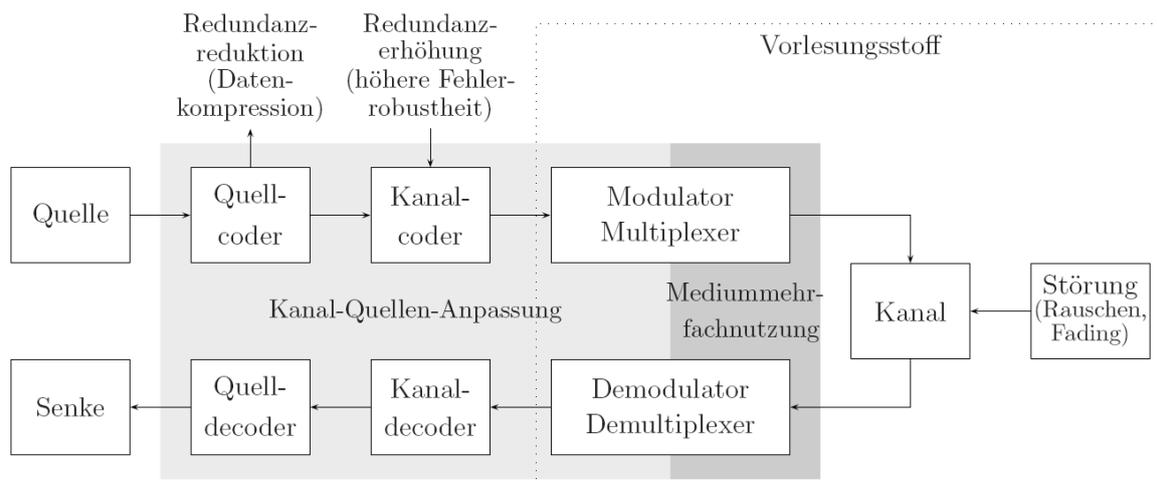
Abbildung 2.1: Allgemeines nachrichtentechnisches Übertragungssystem

Begriffe

Die Bausteine Quelle, Quellcoder, Kanalcoder, Modulator und Multiplexer werden unter dem Begriff Sender zusammengefaßt. Dementsprechend gehören zu dem Empfänger die Baugruppen Demultiplexer, Demodulator, Fehlerkorrekturelemente, Quelldecoder und eine Senke. Sender und Empfänger können sowohl stationär (z.B. Fernsehsender) als auch mobil (z.B. Handy) sein, sind aber immer leistungsbegrenzt. Der Kanal als Übertragungsmedium ist bandbreitenbegrenzt. Durch Störrauschen, Amplitudenschwankungen (fading, verursacht durch Bewegung und Abschattung), Interferenzerscheinungen, Zeit- (delay spread, verursacht durch Mehrwegeausbreitung) und Frequenzdispersion (Doppler spread, verursacht durch Bewegung von Sender, Empfänger und/oder Streuern/Reflektoren usw.) werden die gesendeten Informationen beeinflußt.
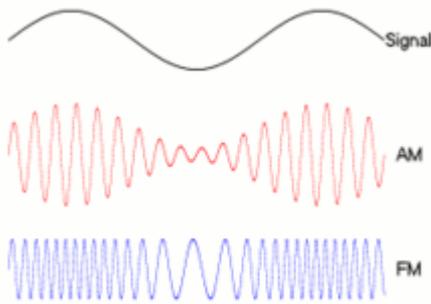
Modulator

analoge Amplitudenmodulation (AM) und Frequenzmodulation (FM) eines niederfrequenten Signals

Übertragungsmedien

Die Wahl des Übertragungsmediums hängt sehr stark von den Anforderungen an den Übertragungskanal ab (z.B. bezüglich Frequenzbereich oder Signalbandbreite, aber auch hinsichtlich der gleichzeitigen Anzahl der Nutzer). Mögliche Übertragungsmedien sind

- ''Twisted Pair`` (verdrillte Kupferkabel),

  - z.B. Telefonkabel (Endgeräteanschluß)

- Koaxkabel,

  - z.B. Antennenkabel, Kabelfernsehen

- Hohlleiter,

  - z.B. Antenneneinspeisung bei hohen Frequenzen (Giga-Hertz-Bereich)

- Lichtwellenleiter,

  - z.B. Übertragung mit sehr hohen Datenraten

- Funkkanal.

  - z.B. Mobilfunk, Hörrund- und Fernsehfunk

Im Funkbereich unterscheidet man auch zwischen Indoor- und Outdoor-Anwendungen. Ein typisches Beispiel für Indoor-Anwendungen könnte die Versorgung aller Räume eines Bürogebäudes mit einem WLAN (wireless local area network) sein. Outdoor-Anwendungen sind z.B. die bundesweit verbreiteten zellularen Mobilfunknetze, derzeit GSM-900, DCS-1800 (D1-, D2-, Eplus- und E2-Netz) und zukünftig auch UMTS.

Auch die Frequenz- bzw.Wellenlängenbereiche werden unterschieden, angefangen von den bekannten MW- und UKW-Bereichen bis hin zu den Millimeterwellen-Bereichen und weiter Infrarot-Bereichen der optischen Nachrichtentechnik.

Eigenschaften

Basics

Im folgenden werden einige Eigenschaften von Nachrichtenübertragungssystemen aufgezählt. Dabei wird keinerlei Anspruch auf Vollständigkeit erhoben.

Simplex/Duplex Ein Unterscheidungskriterium ist, ob Systeme im Simplex- oder Duplexmodus betrieben werden. Simplexbetrieb bedeutet, daß Nachrichten nur in eine Richtung übertragen werden (z.B. Rundfunk), während im Duplexbetrieb die Informationen in beide Richtungen übertragen werden (z.B. Telefonie).

Single-Cast/Multi-Cast Es gibt Single-Cast-Systeme (Telefon: 1 Quelle, 1 Empfänger) und Multi-Cast-Systeme (Rundfunk: 1 Quelle mit vielen Empfängern)

Paket/Leitungs-Vermittlung Ein weiteres Merkmal ist, ob Nachrichten leitungsvermittelt (z.B. "das gute alte" Telefon) oder paketvermittelt (z.B. Datenübertragung im Internet – IP-Protokoll) übertragen werden.

Signalpegel

Oftmals sind Signale mit großen Pegelunterschieden gegeben. Typische Werte für Signalleistungen P liegen zwischen $1\,\mu$W und 1kW. Das entspricht einem Unterschied von 10^9. Aus diesem Grund ist eine logarithmische Skala vorteilhaft. Eine solche Skala ist die dBm-Skala, bei der die Leistungspegel L_P auf P_{ref} = 1 mW normiert werden, also

$$P = \text{Leistung}(s(t)) \tag{2.1}$$

$$L_P = 10 \log_{10} \frac{P}{P_{\text{ref}}} \text{ dBm}$$

$$L_P = 10 \lg \frac{P}{P_{\text{ref}}} \text{ dBm} \quad \text{mit} \quad P_{\text{ref}} = 1 \text{ mW} \tag{2.2}$$

abs. Leistung in [mW]	0.1	0.5	1	2	4	8	10	100	1000
rel. Leistung in [dBm]	-10	-3	0	3	6	9	10	20	30

Tabelle 2.1: Absolutleistung vs. dBm-Pegel

• In der Tab. 2.1 sind einige absolute Leistungswerte und die dazugehörigen dBm-Werte angegeben.

• 2 W-Handy (D-Netz): P_{max} = 2 W, äquivalente Darstellung als Pegel L_{Pmax} = 10 lg $_{2W/1mW}$dBm = 10 lg(2 · 10^3)dBm = (10 lg(2) + 10 lg(10^3))dBm = 33dBm. Im GSM-Standard ist spezifiziert, daß der Pegel an der Basisstation mindestens -102 dBm betragen muß, d.h. es können sich Pegeldifferenzen von bis zu 135 dBm bzw. 10^{13} ergeben.

• 0.8 W-Handy (E-Netz): ≡ 29dBm. Da mit einer geringeren Leistung gesendet wird und bei 1.8 GHz wesentlich stärkere Dämpfungsverhältnisse vorliegen, ist im E-Netz eine größere Anzahl von Basisstationen gegenüber dem D-Netz erforderlich, was sich in den Infrastrukturkosten niederschlägt.

• Auch Signalspannungen können im logarithmischen Maßstab angegeben werden. Als Bezugsspannung wird meist 0.775 V verwendet und entspricht 0 dBu. (Wahl der Referenzzpannung: Welche Spannung ist notwendig, um an einem 600 Normwiderstand eine Leistung von 1 mW entstehen zu lassen? = $0.775^2 V^2/600$ = 1mW) Die Wahl eines anderen Normwiderstandes bzw. Referenzspannung verschiebt die dB-Skala entsprechend.

Ebenso lassen sich Verstärkungsfaktoren von Systemen äquivalent als Pegel angeben. Bezeichnen z.B. x und y den Ein- bzw. Ausgang eines Systems, so ergeben sich die Pegel zu

$$g_P = \frac{\text{Leistung}(y)}{\text{Leistung}(x)} \qquad \text{bzw.} \quad g_A = \frac{\text{Amplitude}(y)}{\text{Amplitude}(x)} \qquad (2.3)$$
$$- \text{Leistungsverstärkung } (gain) \qquad\qquad - \text{Amplitudenverstärkung}$$
$$L_{g_P} = 10 \lg(g_P) \text{ dB} \qquad\qquad L_{g_A} = 20 \lg(g_A) \text{ dB} \qquad (2.4)$$

Besonders bei passiven Systemen werden oft Dämpfungs- statt Verstärkungsfaktoren angegeben.

$$a_P = \frac{\text{Leistung}(x)}{\text{Leistung}(y)} \qquad \text{bzw.} \quad a_A = \frac{\text{Amplitude}(x)}{\text{Amplitude}(y)} \qquad (2.5)$$
$$a_P = \frac{1}{g_P} \qquad\qquad a_A = \frac{1}{g_A} \qquad (2.6)$$
$$- \text{Leistungsdämpfung } (attenuation) \qquad - \text{Amplitudendämpfung}$$
$$L_{a_P} = 10 \lg(a_P) \text{ dB} \qquad\qquad L_{a_A} = 20 \lg(a_A) \text{ dB} \qquad (2.7)$$
$$L_{a_P} = -L_{g_P} \qquad\qquad L_{a_A} = -L_{g_A} \qquad (2.8)$$

Oftmals sind die Pegelverhältnisse zwischen Nutz- und Störsignalen von Interesse. Das Verhältnis

$$SNR = 10 \lg \left(\frac{\text{Nutzsignalleistung}}{\text{Störsignalleistung}} \right) \text{dB} \qquad (2.9)$$

gibt das Nutzsignal-/Störsignalleistungsverhältnis (signal-to-noise-ratio) an. Beachten Sie bitte,

> • daß Pegelangaben in dB immer Verhältnisse zweier Leistungen oder Amplituden (z.B. Verstärkungsfaktor, Signal-Rausch-Abstand) bezeichnen

> • daß Pegelangaben in dBm (Referenz: Leistung P_{ref} = 1 mW), dbW (Referenz: Leistung P_{ref} = 1 W), dBu (Referenz: Spannung U_{ref} = 0.775 V) usw. immer absolute Leistungen oder Spannungen bezeichnen

> • daß sich beim Rechnen mit Pegeln folgende Einheiten ergeben:

Basics

$$dB \pm dB = dB \tag{2.10}$$
$$dBm \pm dB = dBm \tag{2.11}$$
$$dBm - dBm = dB \tag{2.12}$$
$$dBm + dBm = \text{nicht definiert} \tag{2.13}$$
$$\tag{2.14}$$

Kanal-coder/
Modulator/
Multiplexer

Quelle

Quell-coder

Kanal

Senke

Quell-decoder

Demodulator/
Demultiplexer/
Kanal-decoder

Communication Basics

From "Microwave and RF Design: A Systems Approach", Chapter 1 (Modulation, Transmitters and Receivers)(www.ece.ucsb.edu/yuegroup/Teaching/Lectures/steer_rf_chapter1.pdf):
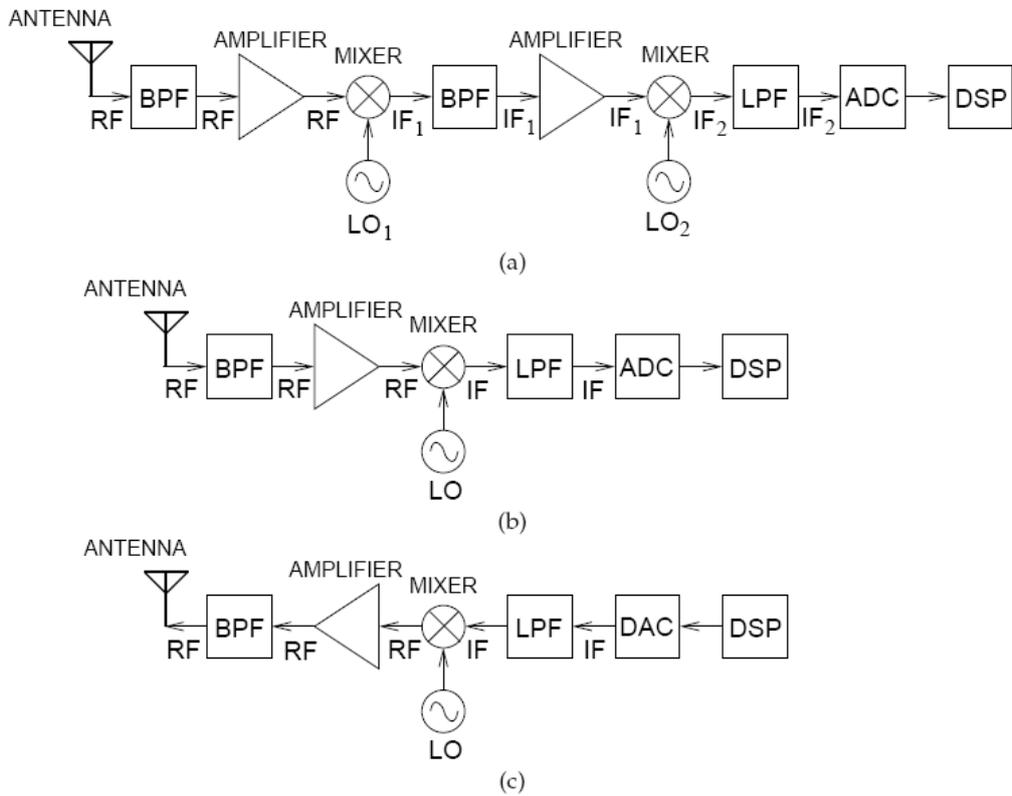
Figure 1-1 Unilateral RF frontend: (a) a receiver with two mixing stages; (b) a receiver with one heterodyne stage; and (c) a one-stage transmitter.
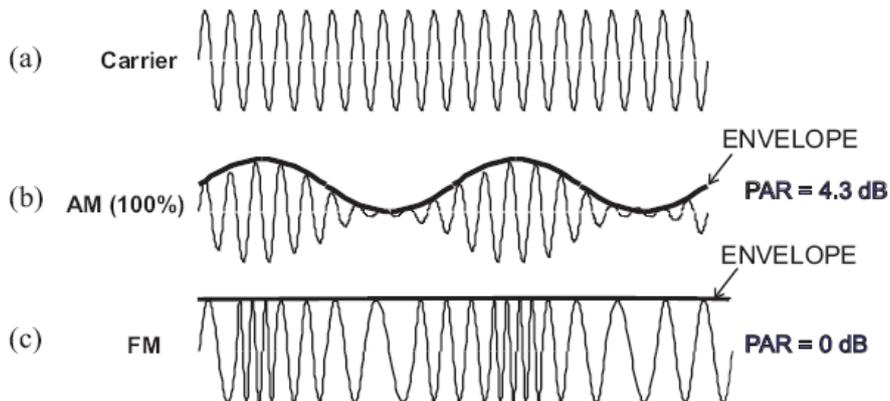
Figure 1-7 Comparison of 100% AM and FM highlighting the envelopes of both: (a) carrier; (b) AM signal with envelope; and (c) FM or PM signal with the envelope being a straight line or constant.
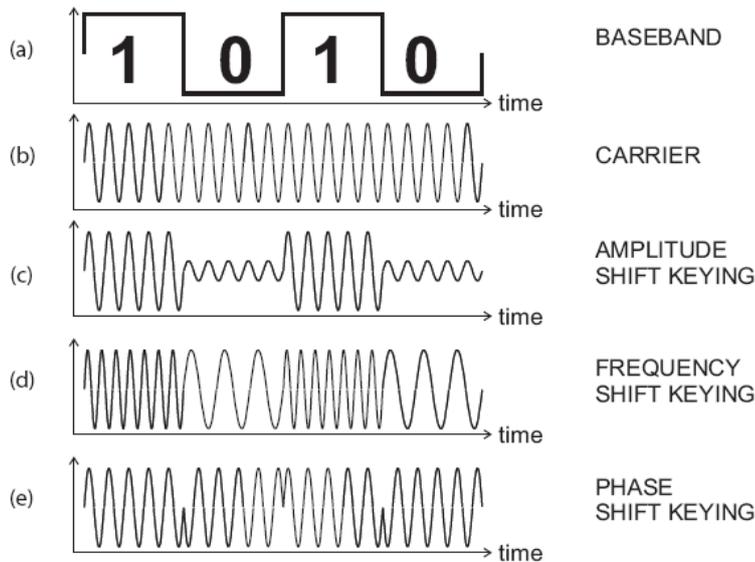
Figure 1-8 Modes of digital modulation: (a) modulating bit stream; (b) **carrier**; (c) carrier modulated using amplitude shift keying (ASK); (d) carrier modulated using frequency shift keying (FSK); and (e) carrier modulated using binary phase shift keying (BPSK).
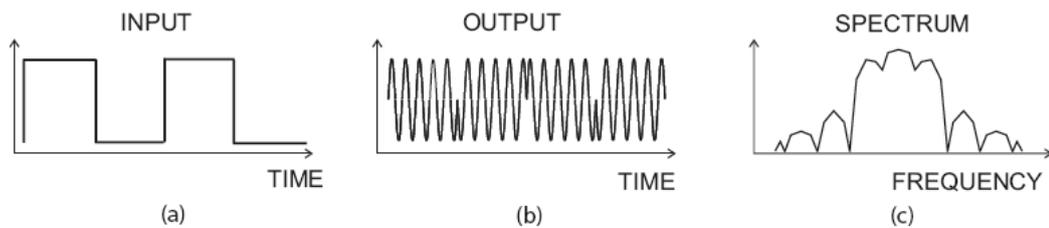
Figure 1-9 Characteristics of phase shift keying (PSK) modulation: (a) modulating bit stream; (b) the waveform of the carrier modulated using PSK with the phase determined by the 1s and 0s of the modulating bit stream; and (c) the spectrum of the modulated signal.
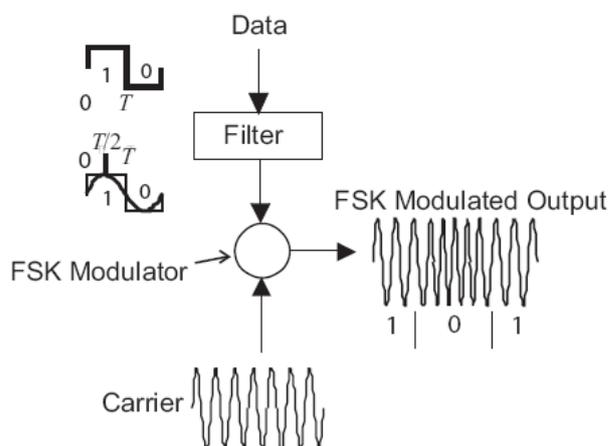
Figure 1-10 The frequency shift keying (FSK) modulation system.

Example 1.2 QPSK Modulation and Constellation

The bit sequence 110101001100 is to be transmitted using QPSK modulation. Show the transitions on a constellation diagram.

SOLUTION: The bit sequence 110101001100 must be converted to a two-bit wide parallel stream of symbols resulting in the sequence of symbols 11 01 01 00 11 00. The symbol 11 transitions to the symbol 01 and then the symbol 01 and so on. The states (or symbols) and the transitions from one symbol to the next required to send the bitstream 110101001100 are shown in Figure 1-13.
QPSK modulation results in the phasor of the carrier transitioning through the origin so that the average power is lower and the PAR is high. A more significant problem is that the phasor will fall below the noise floor, making carrier recovery almost impossible.
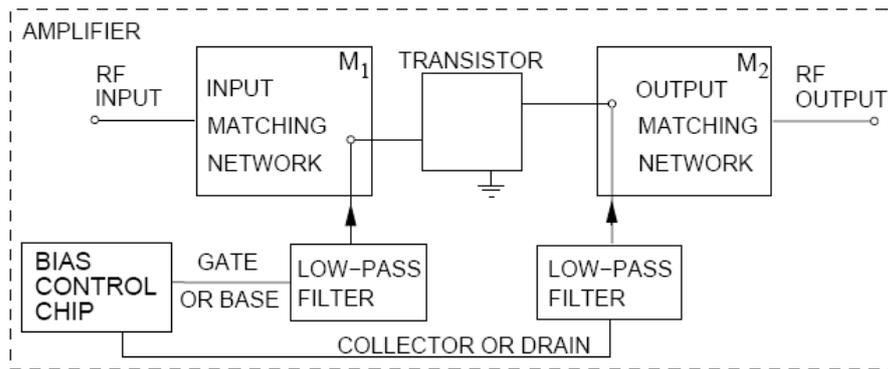
Figure 1-25 Block diagram of an RF amplifier including biasing networks.
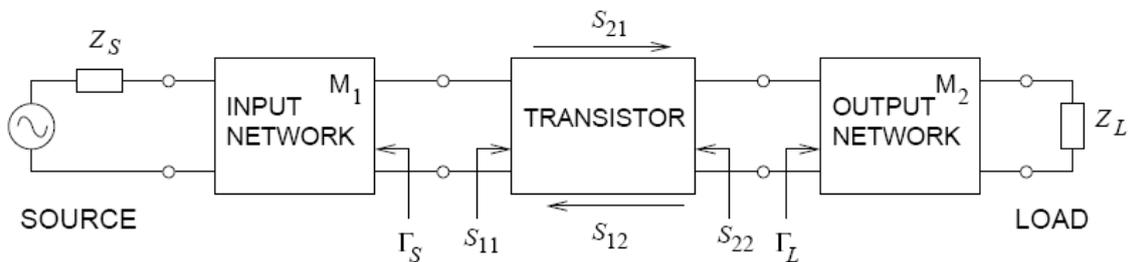
Figure 1-26 The scattering parameter (S_{nm}) and reflection coefficients (Γ) associated with a microwave transistor amplifier.
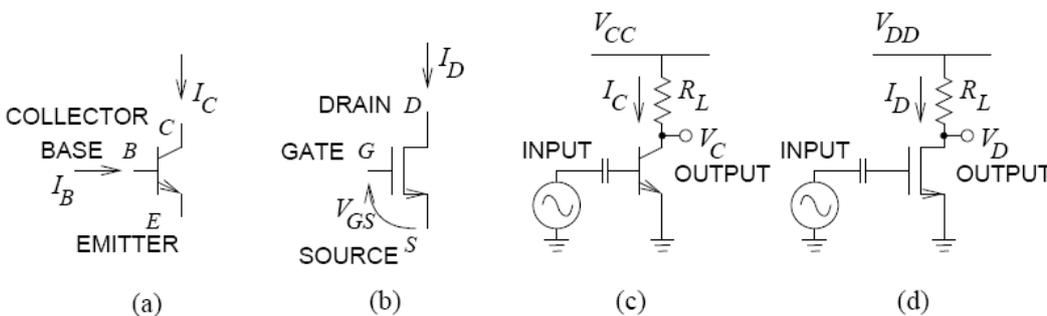
Figure 1-28 Class A single-ended amplifiers: (a) BJT transistor with B for base terminal, C for collector terminal, and E for emitter terminal; (b) MOSFET transistor with G for gate terminal, D for drain terminal, and S for source terminal; (c) single-ended BJT Class A amplifier with resistive bias; and (d) single-ended MOSFET Class A amplifier with resistive bias.
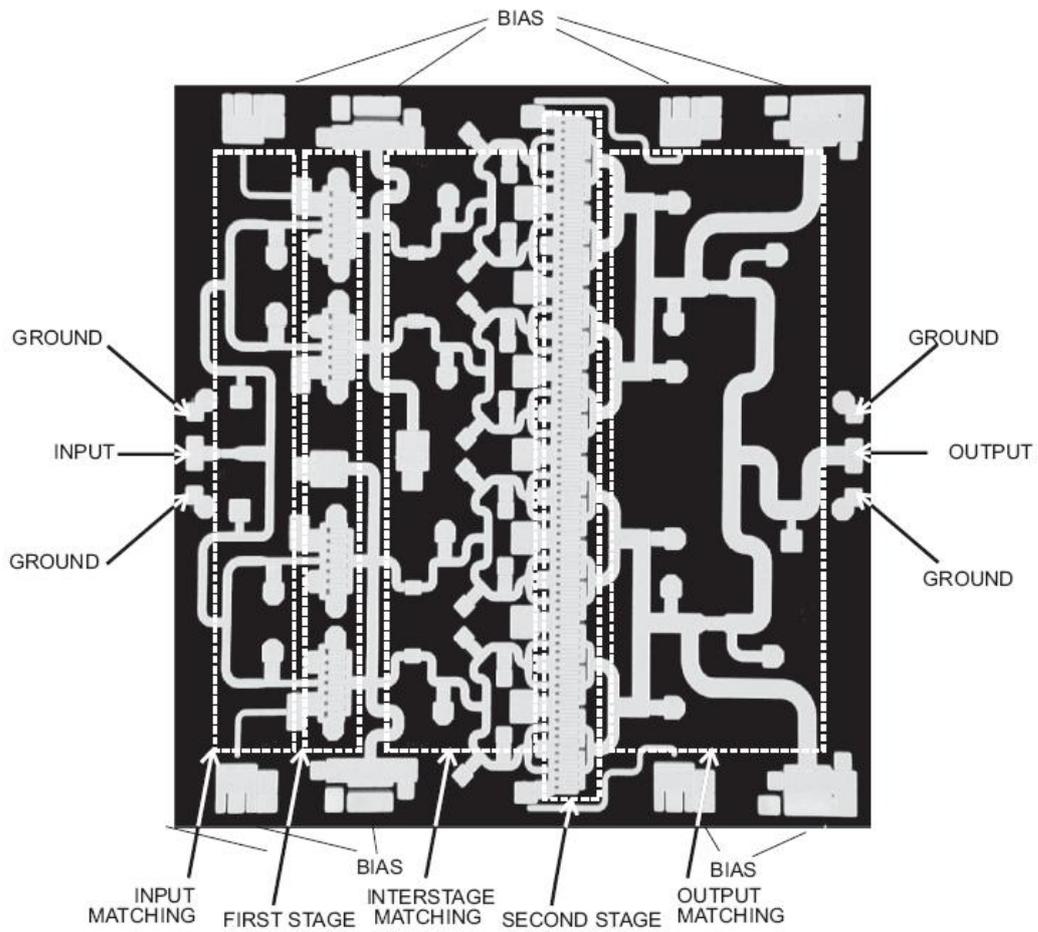
Figure 1-34 An 8–12 GHz MMIC amplifier producing approximately 1 W of output power with key networks identified. (Courtesy Filtronic, PLC, used with permission.)

Modern Transmitter Architectures

(a)

(b)
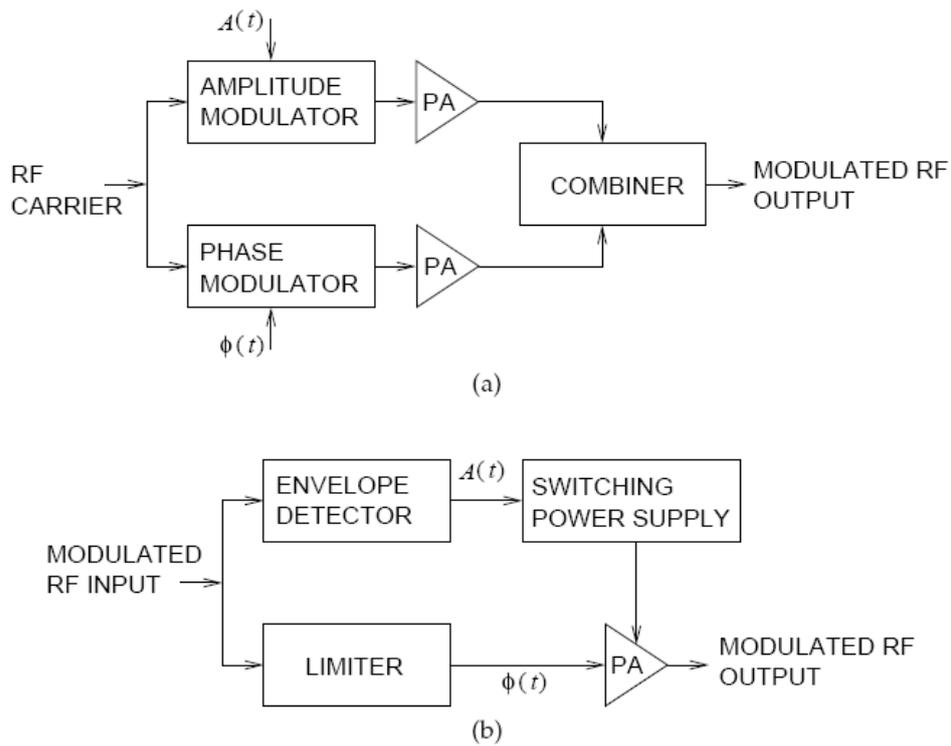
Figure 1-55 Polar modulator architectures: (a) amplitude and phase modulated components amplified separately and combined; and (b) the amplitude used to modulate a power supply driving a saturating amplifier with phase modulated input.
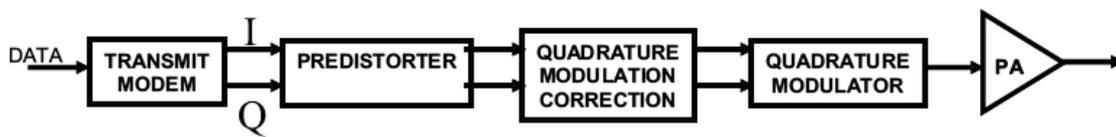
Figure 1-56 Architecture of a direct conversion transmitter.

Modern Receiver Architectures
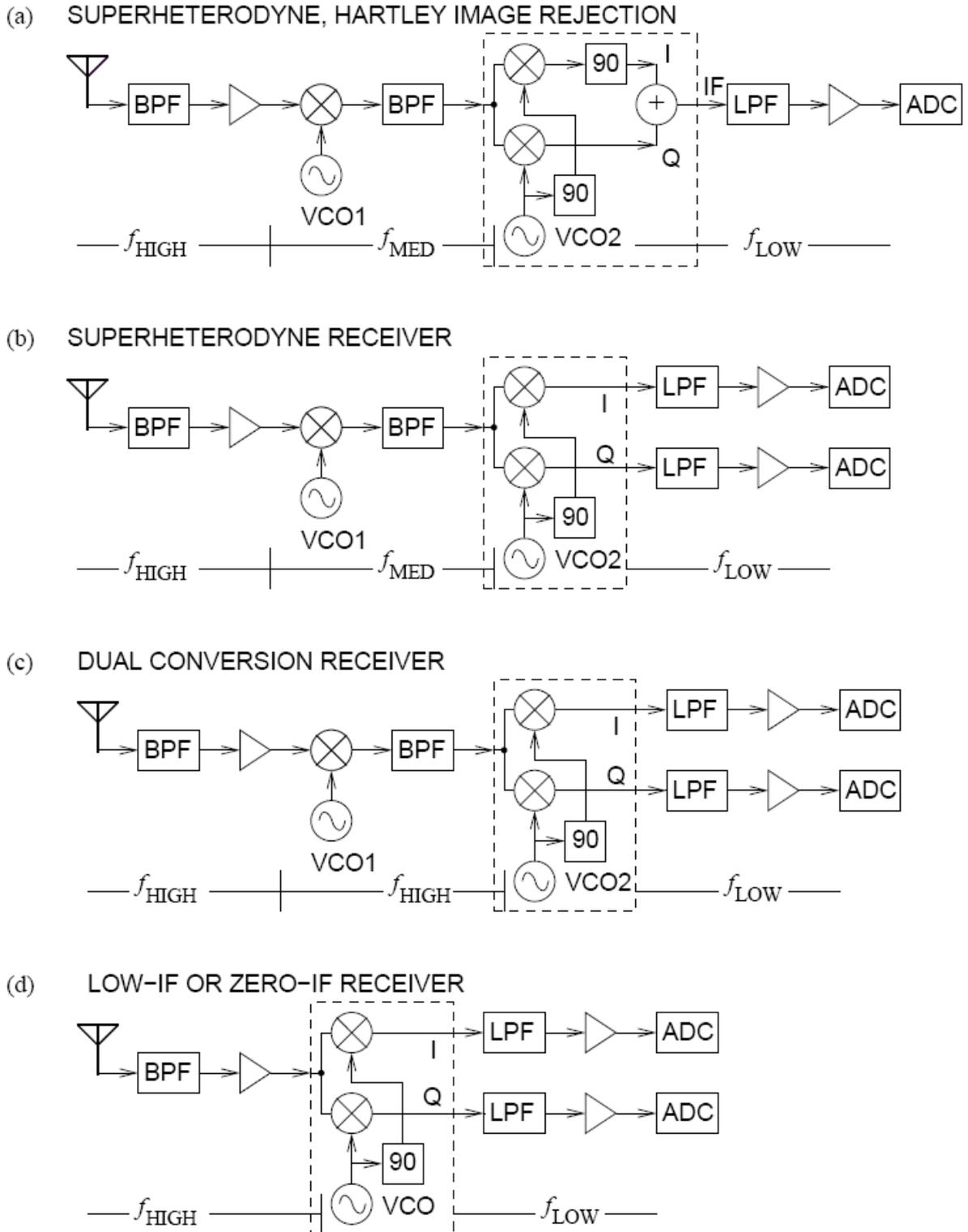
Figure 1-57 Architecture of modern receivers: (a) superheterodyne receiver using the **Hartley** architecture for image rejection; (b) superheterodyne receiver; (c) dual-conversion receiver; low-IF or zero-IF receiver. PBF, bandpass filter; LBF, LowPass Filter; ADC, analog to digital converter; VCO, voltage controlled oscillator; 90, 90° phase shifter; I, in-phase component, Q, Quadrature component; f_{HIGH}, f_{MED} and f_{LOW} indicate relatively high, medium and low frequencies in the corresponding section of the receiver.
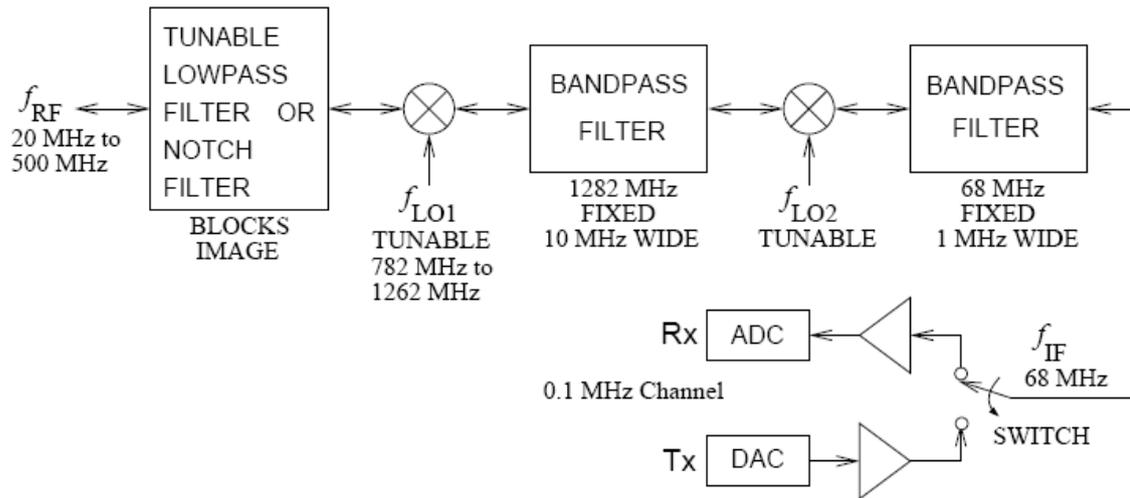
Figure 1-64 Bilateral double conversion transceiver for wideband operation.

Summary

This chapter presented the RF frontend architectures used from the beginnings of wireless communications up to those used in modern systems. Similar architectures are used in the frontends of radar and sensor systems. Wireless systems proliferate, and even in established domains such as cellphones, architectures are evolving to achieve greater efficiency, greater multifunctionality, and lower cost primarily by monolithically integrating and digitizing as much as possible of the RF frontend. Size drives the replacement of superheterodyne architecture by eliminating large intermediate filters.

3.1.1 Transmitter design from http://en.wikibooks.org/wiki/Electronics/Transmitter_design

Radio transmitter design is a complex topic which can be broken down into a series of smaller topics.

Contents

1 Frequency synthesis and frequency multiplication

1.1 Synthesis

1.1.1 Fixed frequency systems

1.1.2 Variable frequency systems

1.2 Multiplication

2 Frequency mixing and Modulation

2.1 AM modes

2.1.1 Low level and High level

2.1.1.1 Low level

Basics

2.1.1.2 High level

2.1.2 Types of AM modulators

2.1.2.1 Plate AM modulators

2.1.2.2 Screen AM modulators

2.2 Other modes which are related to AM

2.2.1 Single-sideband modulation

2.2.1.1 Filter method

2.2.1.2 Phasing method

2.2.2 Vestigial-sideband modulation

2.2.3 Morse

2.3 FM modes

2.3.1 Direct FM

2.3.2 Indirect FM

3 RF power amplifiers

3.1 Valves

3.1.1 Advantages of valves

3.1.2 Disadvantages of valves

3.2 Solid state

4 Linking the transmitter to the aerial

5 EMC matters

5.1 RF leakage (defective RF shielding)

5.2 Spurious emissions

5.2.1 Harmonics

5.2.2 Local oscillators and unwanted mixing products

5.2.3 Instability and parasitic oscillations

6 Reference

7 Further reading

3.1.1.1 Frequency synthesis and frequency multiplication

Synthesis

Fixed frequency systems

For a fixed frequency transmitter one commonly used method is to use a resonant quartz crystal in a Crystal oscillator to fix the frequency. For transmitter where the frequency has to be able to be varied then several options can be used.

Variable frequency systems

An array of crystals—This approach uses several oscillators, each tuned to a different fixed frequency.

Variable frequency oscillator (VFO)

Phase locked loop (PLL) frequency synthesizer

Multiplication

It is often the case for VHF transmitters that it is not possible to operate the crystal controlled or variable frequency oscillator at the frequency of the final output. Also, for reasons including frequency stability, it is better to multiply the frequency of the free running oscillator up to the final frequency which is required.

If the output of a amplifier stage is tuned to a multiple of the frequency which the stage is driven with, the stage is optimised to give a larger harmonic output than that found in a linear amplifier. In a push-push stage, the output will only contain the even harmonics. This is because the currents which would generate the fundamental and the odd harmonics in this circuit (if one valve was removed) are canceled out by the second valve. Note that in these diagrams that the bias supplies and the neutralization have been omitted for clarity. In a real system it is likely that tetrodes would be used as plate to grid capacitance in a tetrode is lower so making the stage less likely to be unstable.
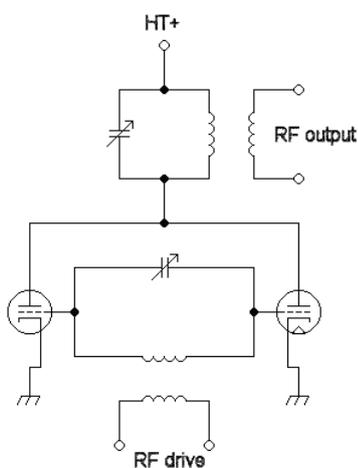
Here in the push-pull stage the output will only contain the odd harmonics because of the canceling effect.
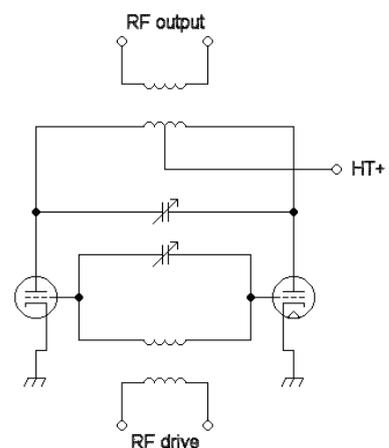
3.1.1.2   Frequency mixing and Modulation

The task of many transmitters is to transmit some form of information using a carrier wave. This process is called modulation. There are many types of RF modulation, and the choice of modulation often depends on the type of information being transmitted.

For instance, audio information is continuous in time and value, and scaling by a constant (i.e. signal inversion, volume control) is acceptable, so AM and FM transmission work. But for digital communications, the signal is discrete in time and discrete in value, and inversion of the signal is unacceptable, so AM and FM are not (on their own) satisfactory. For digital communications, a modulation such as frequency shift keying (FSK) or on-off keying (OOK) over FM would be better.

AM modes

In many cases the carrier wave is mixed with another electrical signal to impose upon it the information. This occurs in Amplitude modulation (AM).

Low level and High level

Low level

Here a small audio stage is used to modulate a low power stage, the output of this stage is then amplified using a linear RF amplifier.

Advantages

The advantage of using a linear RF amplifier is that the smaller early stages can be modulated, which only requires a small audio amplifier to drive the modulator.

Disadvantages

The great disadvantage of this system is that the amplifier chain is less efficient, because it has to be linear to preserve the modulation. Hence class C amplifiers cannot be employed.

An approach which marries the advantages of low-level modulation with the efficiency of a Class C power amplifier chain is to arrange a feedback system to compensate for the substantial distortion of the AM envelope. A simple detector at the transmitter output (which can be little more than a loosely coupled diode) recovers the audio signal, and this is used as negative feedback to the audio modulator stage. The overall chain then acts as a linear amplifier as far as the actual modulation is concerned, though the RF amplifier itself still retains the Class C efficiency. This approach is widely used in practical medium power transmitters, such as AM radiotelephones.

High level

Advantages

One advantage of using class C amplifiers in a broadcast AM transmitter is that only the final stage needs to be modulated, and that all the earlier stages can be driven at a constant level. These class C stages will be able to generate the drive for the final stage for a smaller DC power input. However in many designs in order to obtain better quality AM the penultimate RF stages will need to be subject to modulation as well as the final stage.

<u>Disadvantages</u>

A large audio amplifier will be needed for the modulation stage, at least equal to the power of the transmitter output itself. Traditionally the modulation is applied using an audio transformer, and this can be bulky. Direct coupling from the audio amplifier is also possible (known as a cascode arrangement), though this usually requires quite a high DC supply voltage (say 30V or more), which is not suitable for mobile units.

Types of AM modulators

A wide range of different circuits have been used for AM. While it is perfectly possible to create good designs using solid-state electronics, valved (tube) circuits are shown here. In general, valves are able to easily yield RF powers far in excess of what can be achieved using solid state. Most high-power broadcast stations still use valves.

Plate AM modulators

In plate modulation systems the voltage delivered to the stage is changed. As the power output available is a function of the supply voltage, the output power is modulated. This can be done using a transformer to alter the anode (plate) voltage. The advantage of the transformer method is that the audio power can be supplied to the RF stage and converted into RF power.
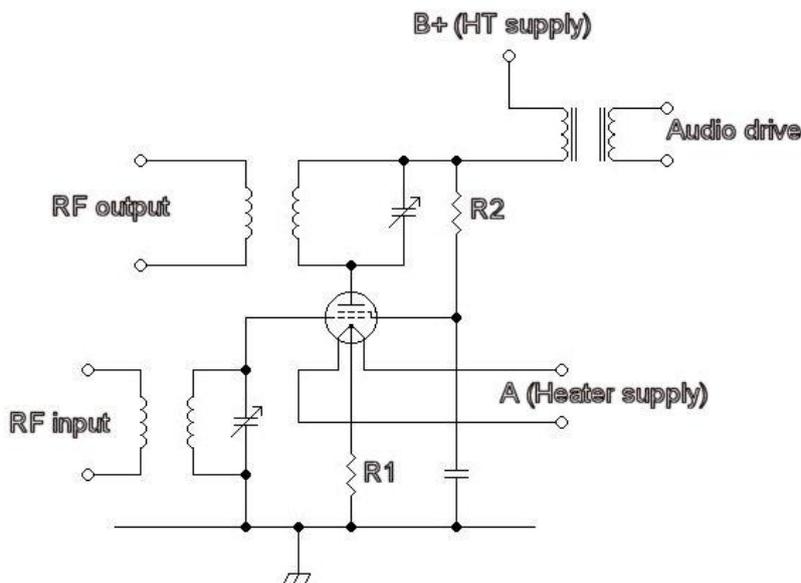
Anode modulation using a transformer. The tetrode is supplied with an anode supply (and screen grid supply) which is modulated via the transformer. The resistor R1 sets the grid bias, both the input and outputs are tuned LC circuits which are tapped into by inductive coupling.

Basics
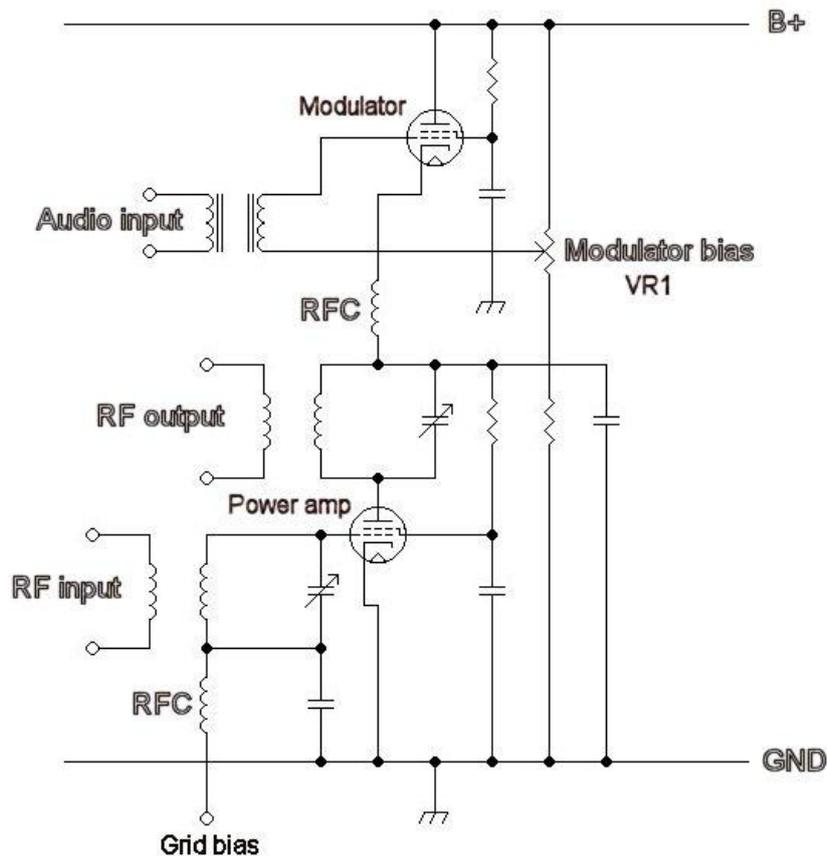
An example of a series modulated amplitude modulation stage. The tetrode is supplied with an anode supply (and screen grid supply) which is modulated by the modulator valve. The resistor VR1 sets the grid bias for the modulator valve, both the RF input (tuned grid) and outputs are tuned LC circuits which are tapped into by inductive coupling. When the valve at the top conducts more than the potential difference between the anode and cathode of the lower valve (RF valve) will increase. The two valves can be thought of as two resistors in a potentiometer.

Screen AM modulators
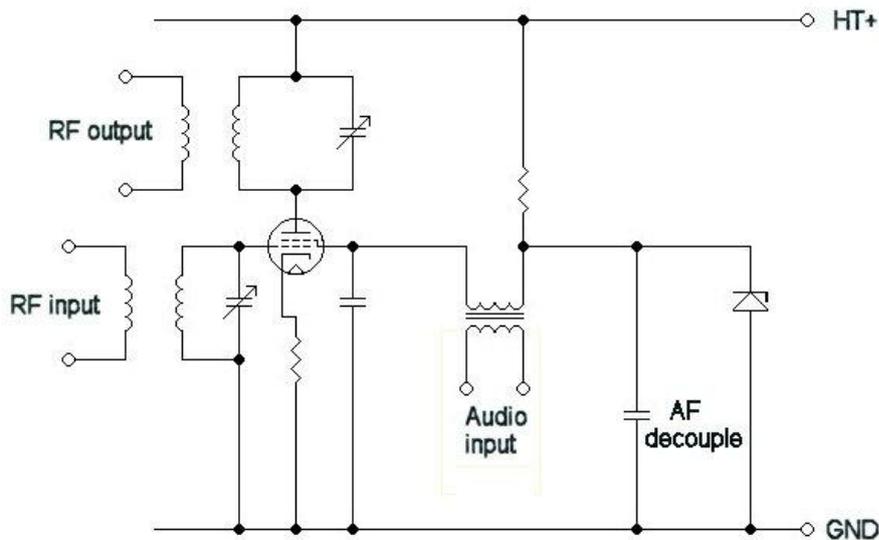
Under steady state conditions (no audio driven) the stage will be a simple RF amplifier where the grid bias is set by the cathode current. When the stage is modulated the screen potential changes and so alters the gain of the stage.

Other modes which are related to AM

Several derivatives of AM are in common use. These are

Single-sideband modulation

(SSB, or SSB-AM single-sideband full carrier modulation), very similar to single-sideband suppressed carrier modulation (SSB-SC)

Filter method

Using a balanced mixer a double side band signal is generated, this is then passed through a very narrow bandpass filter to leave only one side-band. By convention it is normal to use the upper sideband (USB) in communication systems, except for HAM radio when the carrier frequency is below 10 MHz here the lower side band (LSB) is normally used.

Phasing method

The phasing method is another way to generate of single sideband signals. One of the weaknesses

of this method is the need for a network which imposes a constant 90° phase shift on audio signals

throughout the entire audio spectrum. By reducing the audio bandwidth the task of designing the phaseshift network can be made more easy.

Imagine that the audio is a single sine wave $E = E^o$ sine (ωt)

The audio signal is passed through the phase shift network to give two identical signals which differ by 90°.

Basics

So as the audio input is a single sine wave the outputs will be

$E = E^o$ sine (ωt)

and

$E = E^o$ cosine (ωt)

These audio outputs are mixed in non linear mixers with a carrier, the carrier drive for one of these mixers is shifted by 90°. The output of these mixers is combined in a linear circuit to give the SSB signal.

Vestigial-sideband modulation[]

Vestigial-sideband modulation (VSB, or VSB-AM) is a type of modulation system commonly used in TV systems, it is normal AM which has been passed through a filter which removes one of the sidebands.

Morse

Strictly speaking the commonly used 'AM' is double-sideband full carrier. Morse is often sent using on-off keying of an unmodulated carrier(Continuous wave), this can be thought of as an AM mode.

FM modes

Direct FM

Direct FM (true Frequency modulation) is where the frequency of an oscillator is altered to impose the modulation upon the carrier wave. This can be done by using a voltage controlled capacitor (Varicap diode) in a crystal controlled oscillator. The frequency of the oscillator is then multiplied up using a frequency multiplier stage, or is translated upwards using a mixing stage to the output frequency of the transmitter.

Indirect FM

Indirect FM employs varicap diode to impose a phase shift (which is voltage controlled) in a tuned circuit which is fed with a plain carrier. This is termed Phase modulation, the modulated signal from a phase modulated stage can be understood with a FM receiver but for good audio quality the audio applied to the phase modulation stage.
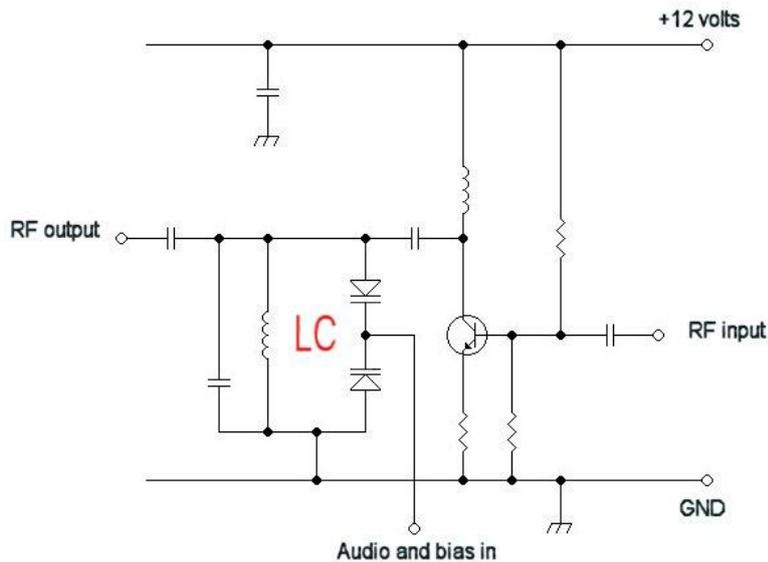
This is a solid state circuit, on the right a RF drive is applied to the base of the transistor, the tank circuit (LC) connected to the collector via a capacitor contains a pair of varicap diodes. As the voltage applied to the varicaps is changed the phase shift of the output will change.

Sigma-delta modulation ($\sum\Delta$)

3.1.1.3   RF power amplifiers

Valves

For high power systems it is normal to use valves, please see Valved RF amplifiers for details of how valved RF power stages work.

Advantages of valves

Good for high power systems

Electrically very robust, they can tolerate overloads for minutes which would destroy bipolar transistor systems in milliseconds

Disadvantages of valves

Heater supplies are required for the cathodes

High voltages (*Threat of death*) are required for the anodes

Valves have a shorter working life than solid state parts because the heaters tend to fail

Solid state

For low and medium power it is often the case that solid state power stages are used. Sadly for high power systems these cost more per Watt of output power than a valved system.

3.1.1.4   Linking the transmitter to the aerial

The vast majority of modern equipment is designed to operate with a resistive load driven via coaxial cable of one particular impedance, often 50 ohms. To connect the aerial to this coaxial cable transmission line a matching network and/or a balun may be required. Commonly a SWR meter and/or a Antenna analyzer are used to check the goodness of the match between the aerial system and the transmission line (feeder).

See Antenna tuner and balun for details of matching networks and baluns respectively.

3.1.1.5   EMC matters

While this section was written from the point of view of a radio ham with relation to Television interference (radio transmitter interference) it applies to the construction and use of all radio transmitters, and other electronic devices which generate high RF powers with no intention of radiating these. For instance a dielectric heater might contain a 2000 Watt 27 MHz source within it, if the machine operates as intended then none of this RF power will leak out. However if the device is subject to a fault then when it operates RF will leak out and it will be now a transmitter. Also computers are RF devices, if the cases is poorly made then the computer will radiate at VHF.

For example if you attempt to tune into a weak *FM* radio station (88 to 108 MHz, band II) at your desk you may lose reception when you switch on your PC. Equipment which is not intended to generate RF, but does so through for example sparking at switch contacts is not considered here, for a consideration of such matters please see Television interference (electrical interference) for further details.

RF leakage (defective RF shielding)

All equipment using RF electronics should be inside a screened metal box, all connections in or out of the metal box should be filtered to avoid the ingress or egress of radio signals. A common and effective method of doing so for wires carrying DC supplies, 50 Hz AC connections, audio and control signals is to use a feedthrough capacitor. This is a capacitor which is mounted in a hole in the shield, one terminal of the capacitor is its metal body which touches the shielding of the box while the other two terminal of the capacitor are the on either side of the shield. The feed through capacitor can be thought of as a metal rod which has a dielectric sheath which in turn has a metal coating.

In addition to the feed through capacitor, either a resistor or RF choke can be used to increase the filtering on the lead. In transmitters it is vital to prevent RF from entering the transmitter through any lead such as a power, microphone or control connection. If RF does enter a transmitter in this way then an instability known as motorboating can occur. Motorboating is an example of a self inflicted EMC problem.

If a transmitter is suspected of being responsible for a television interference problem then it should be run into a dummy load, this is a resistor in a screened box or can which will allow the transmitter to generate radio signals without sending them to the antenna. If the transmitter does

not cause interference during this test then it is safe to assume that a signal has to be radiated from the antenna antenna to cause a problem. If the transmitter does cause interference during this test then a path exists by which RF power is leaking out of the equipment, this can be due to bad shielding. This is a rare but insidious problem and it is vital that it is tested for.

You are most likely to see this leakage on homemade equipment or equipment which has been modified. It is also possible to observe RF leaking out of microwave cookers.

Spurious emissions

Early in the development of radio technology it was recognized that the signals emitted by transmitters had to be 'pure'. For instance Spark-gap transmitters were quickly outlawed as they give an output which is so wide in terms of frequency. In modern equipment there are three main types of spurious emissions.

The term Spurious emissions refers to any signal which comes out of a transmitter other than the wanted signal. The spurious emissions include harmonics, out of band mixer products which are not fully suppressed and leakage from the local oscillator and other systems within the transmitter.

Harmonics

These are multiples of the operation frequency of the transmitter, they can be generated in a stage of the transmitter even if it is driven with a perfect sine wave because no real life amplifier is perfectly linear. It is best if these harmonics are designed out at an early stage. For instance a push-pull amplifier consisting of two tetrode valves attached to an anode tank resonant LC circuit which has a coil which is connected to the high voltage DC supply at the centre (Which is also RF ground) will only give a signal for the fundamental and the odd harmonics.
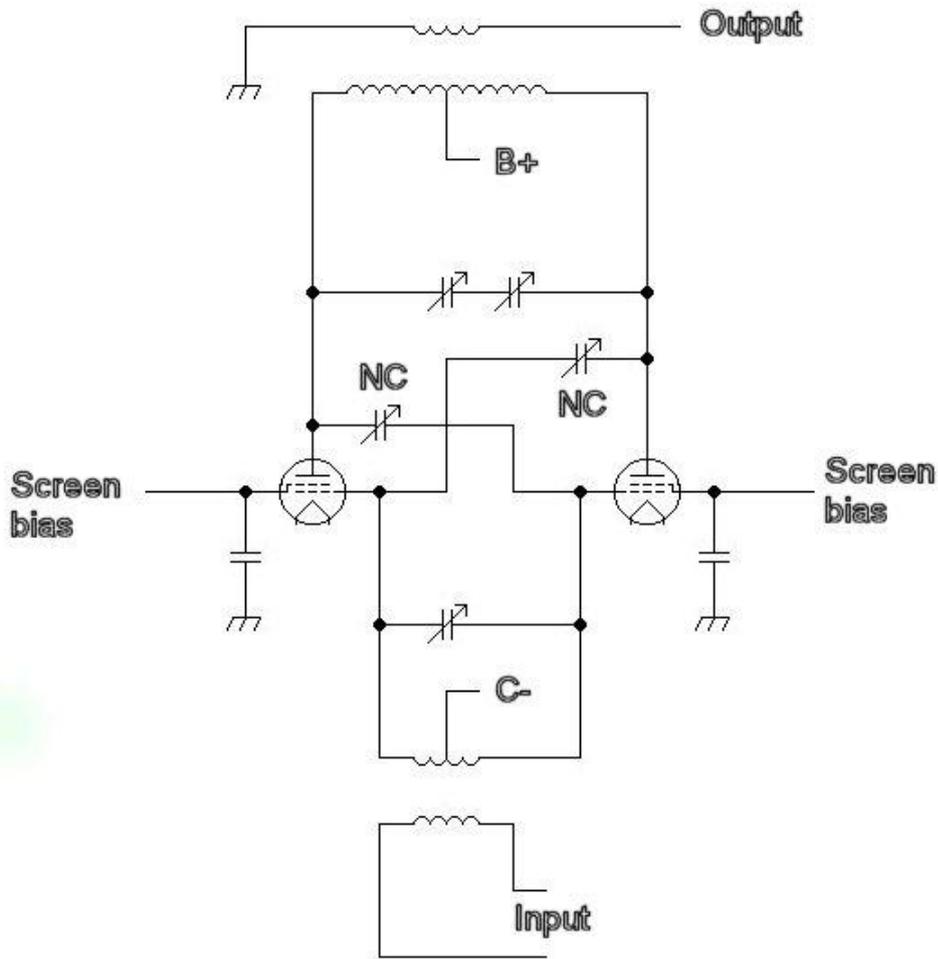
Here is a slightly worse design which only has one tetrode, while perfectly good designs have been made using this circuit it does have more potential shortcomings than the above circuit.
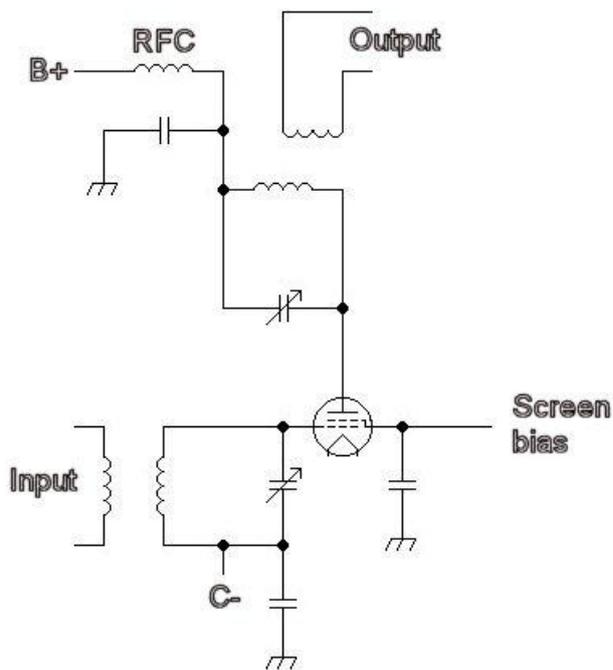
In addition to the good design of the amplifier stages, the transmitter's output should be filtered with a low pass filter to reduce the level of the harmonics.

The harmonics can be tested for using a RF spectrum analyser (expensive) or with an absorption wavemeter (cheap). If a harmonic is found which is at the same frequency as the frequency of the signal wanted at the receiver then this spurious emission can prevent the wanted signal from be received.

Local oscillators and unwanted mixing products

Imagine a transmitter, which has an intermediate frequency (IF) of 144 MHz, which is mixed with 94 MHz to create a signal at 50 MHz, which is then amplified and transmitted. If the local oscillator signal was to enter the power amplifier and not be adequately suppressed then it could be radiated. It would then have the potential to interfere with radio signals at 94 MHz in the FM audio (band II) broadcast band. Also the unwanted mixing product at 238 MHz could in a poorly designed system be radiated. Normally with good choice of the intermediate and local oscillator frequencies this type of trouble can be avoided, but one potentially bad situation is in the construction of a 144 to 70 MHz converted, here the local oscillator is at 74 MHz which is very close to the wanted output. Good well made units have been made which use this conversion but their design and construction has been challenging. This problem can be thought of as being related to the Image response problem which exists in receivers.

One method of reducing the potential for this transmitter defect is the use of balance and double balanced mixers. If the equation is assumed to be

$E = E_1 \cdot E_2$

and is driven by two simple sine waves, f_1 and f_2 then the output will be a mixture of four frequencies

f_1

$f_1 + f_2$

$f_1 - f_2$

f_2

If the simple mixer is replaced with a balanced mixer then the number of possible products is reduced. Imagine that two mixers which have the equation $\{I = E_1 \cdot E_2\}$ are wired up so that the current outputs are wired to the two ends of a coil (the centre of this coil is wired to ground) then the total current flowing through the coil is the difference between the output of the two mixer stages. If the f_1 drive for one of the mixers is phase shifted by 180° then the overall system will be a balanced mixer.

Basics
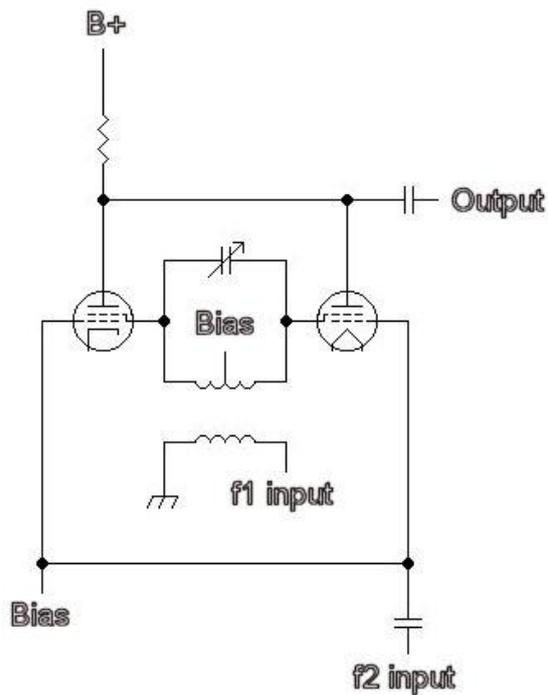
$$E = K \cdot E_{f2} \cdot \Delta E_{f1}$$

So the output will now have only three frequencies

$f_1 + f_2$

$f_1 - f_2$

f_2

Now as the frequency mixer has fewer outputs the task of making sure that the final output

is *clean* will be simpler.

Instability and parasitic oscillations

If a stage in a transmitter is unstable and is able to oscillate then it can start to generate RF at either a frequency close to the operating frequency or at a very different frequency. One good sign that it is occurring is if a RF stage has a power output even without being driven by an exciting stage. Another sign is if the output power suddenly increases wildly when the input power is increased slightly, it is noteworthy that in a class C stage that this behaviour can be seen under normal conditions. The best defence against this transmitter defect is a good design, also it is important to pay good attention to the neutralization of the valves or transistors.

Reference

Radiocommunication handbook (RSGB), ISBN 0900612584[2]

3.2 Receiver Design from http://en.wikipedia.org/wiki/Tuner_(electronics)
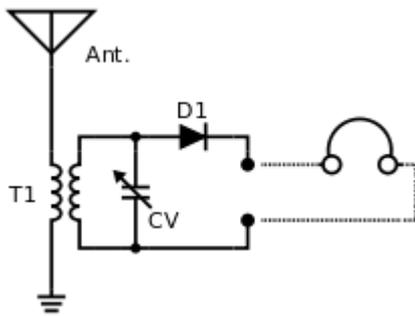
Inductively coupled crystal radio receiver

The simplest tuner consists of an inductor and capacitor connected in parallel, where the capacitor or inductor is made to be variable. This creates a resonant circuit which responds to an alternating current at one frequency. Combined with a detector, also known as a demodulator, (diode D-1, in the circuit), it becomes the simplest radio receiver, often called a crystal set.

Practical radio tuners use a superheterodyne receiver. Older models would realize manual tuning by means of mechanically operated ganged variable capacitors. Often several sections would be provided on a tuning capacitor, to tune several stages of the receiver in tandem, or to allow switching between different frequency bands. A later method used a potentiometer supplying a variable voltage to varactor diodes in the local oscillator and tank circuits of front end tuner, for electronic tuning. Still later, phase locked loop methods were used, with microprocessor control.

In a self-contained radio receiver for audio, the signal from the detector after the tuner is run through a volume control and to an amplifier stage. The amplifier feeds either an internal speaker or headphones. In a tuner component of an audio system (for example, a home high-fidelity system or a public address system in a building), the output of the detector is connected to a separate external system of amplifiers and speakers.

The broadcast audio FM band (88 - 108 MHz in most countries) is around 100 times higher in frequency than the AM band and provides enough space for a bandwidth of 50 kHz. This bandwidth is sufficient to transmit both stereo channels with almost the full bandwidth of the human ear. Sometimes, additional subcarriers are used for unrelated audio or data transmissions. The left and right audio signals must be combined into a single signal which is applied to the modulation input of the transmitter; this is done by the addition of an inaudible subcarrier signal to the FM broadcast signal. FM stereo allows left and right channels to be transmitted. The availability of FM stereo, a quieter VHF broadcast band, and better fidelity lead to the specialization of FM broadcasting in music, tending to leave AM broadcasting with spoken-word material.

---

[2] http://en.wikibooks.org/wiki/Special:BookSources/0900612584
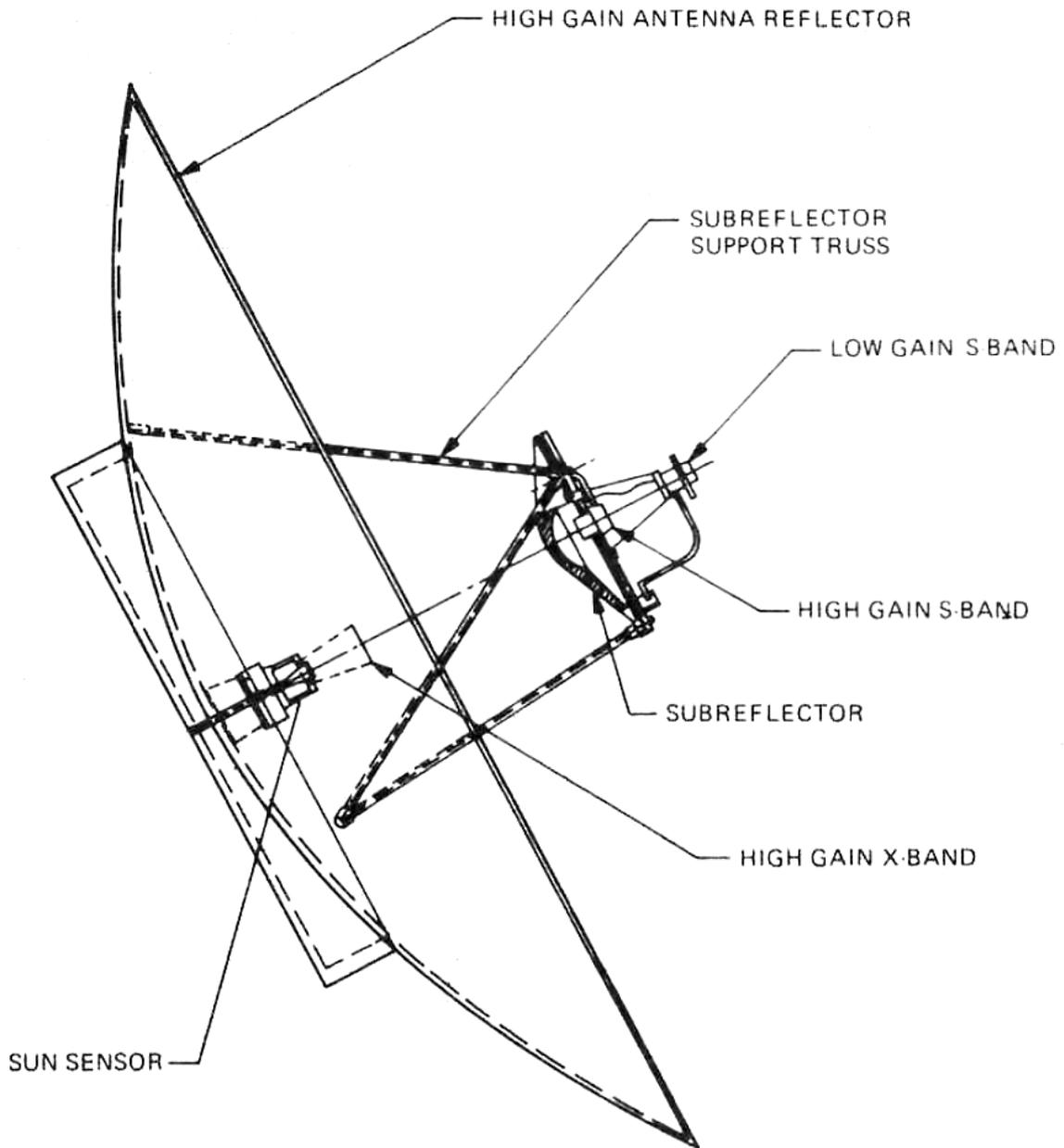
3.3 Antenna

An **antenna** (or **aerial**) is an electrical device which converts electric power into radio waves, and vice versa.[1] It is usually used with a radio transmitter or radio receiver. In transmission, a radio transmitter supplies an oscillating radio frequency electric current to the antenna's terminals, and the antenna radiates the energy from the current as electromagnetic waves (radio waves). In reception, an antenna intercepts some of the power of an electromagnetic wave in order to produce a tiny voltage at its terminals, that is applied to a receiver to be amplified.

Antennas are essential components of all equipment that uses radio. They are used in systems such as radio broadcasting, broadcast television, two-way radio, communications receivers, radar, cell phones, and satellite communications, as well as other devices such as garage door openers, wireless

microphones, bluetooth enabled devices, wireless computer networks, baby monitors, and RFID tagson merchandise.

Typically an antenna consists of an arrangement of metallic conductors (elements), electrically connected (often through atransmission line) to the receiver or transmitter. An oscillating current of electrons forced through the antenna by a transmitter will create an oscillating magnetic field around the antenna elements, while the charge of the electrons also creates an oscillating electric field along the elements. These time-varying fields radiate away from the antenna into space as a moving transverse electromagnetic field wave. Conversely, during reception, the oscillating electric and magnetic fields of an incoming radio wave exert force on the electrons in the antenna elements, causing them to move back and forth, creating oscillating currents in the antenna.

Antennas may also include reflective or directive elements or surfaces not connected to the transmitter or receiver, such as parasitic elements, parabolic reflectors or horns, which serve to direct the radio waves into a beam or other desired radiation pattern. Antennas can be designed to transmit or receive radio waves in all directions equally (omnidirectional antennas), or transmit them in a beam in a particular direction, and receive from that one direction only (directional or high gain antennas).
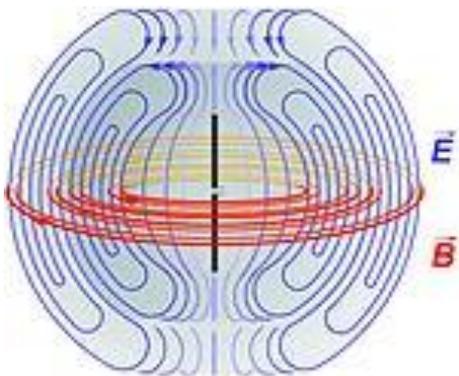
Diagram of the electric fields (blue) and magnetic fields (red) radiated by adipole antenna (black rods)during transmission.

Large parabolic antenna for communicating with spacecraft

Antenna tuner

An **antenna tuner**, **transmatch** or **antenna tuning unit** (**ATU**) is a device connected between a radio transmitter or receiver and its antenna to improve power transfer between them by matching the impedance of the radio to the antenna. An antenna tuner matches a transceiver with a fixed impedance (typically 50 ohms for modern transceivers) to a load (feed line and antenna) impedance which is unknown, complex or otherwise does not match. An ATU allows the use of one antenna on a broad range of frequencies. An antenna and transmatch are not as efficient as a resonant antenna due to feedline losses due to the SWR (multiple reflections) and losses in the ATU itself. An ATU is an antenna matching unit, and cannot change the resonant frequency of the

aerial. Similar matching networks are used in other equipment (such as linear amplifiers) to transform impedance.
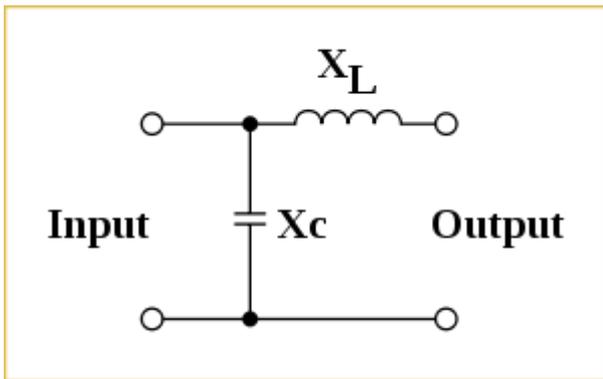
Basic network for a antenna tuner

3.4 Software Defined Radio (SDR)

Software Defined Radio is a Wireless communication in which the transmitter modulation is generated or defined by a computer, and the receiver uses a computer to recover the signal intelligence? To select the desired modulation type, the proper programs must be run by microcomputers that control the transmitter and receiver
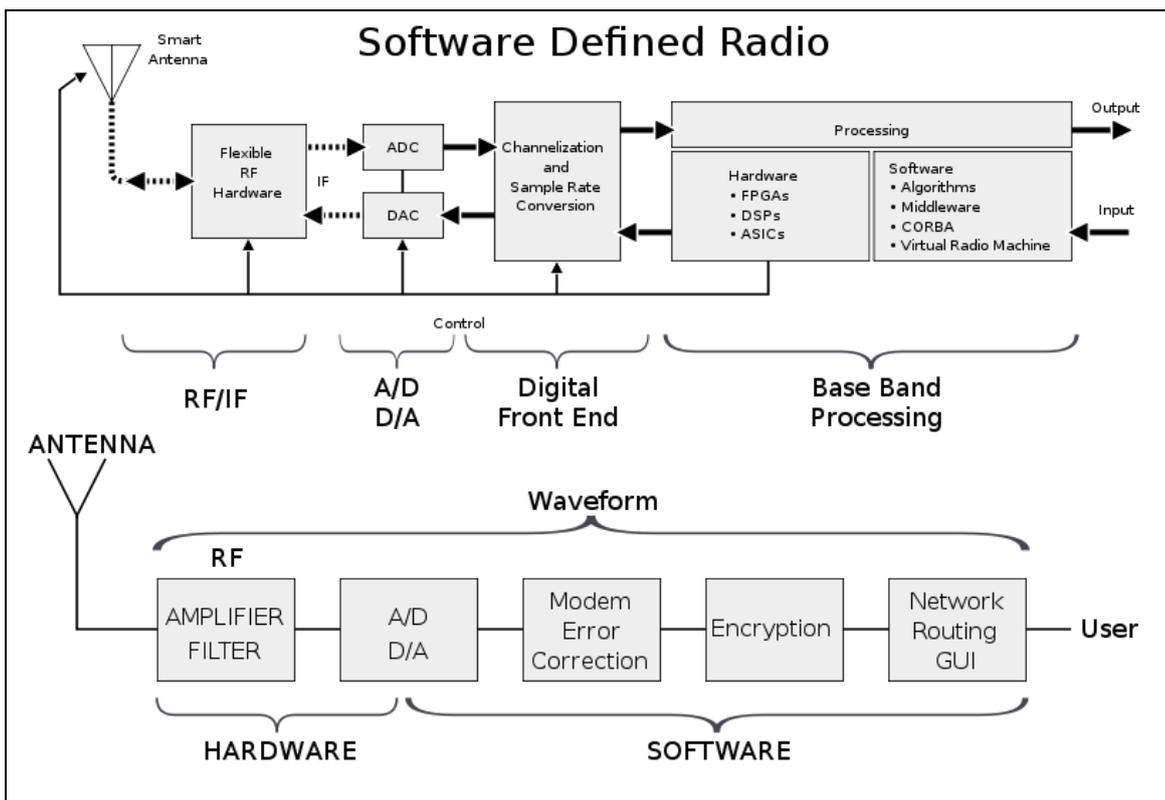
SDR block diagram

3.5 HDSDR (High Definition Software Defined Radio)

HDSDR (**High Definition Software Defined Radio**) is a freeware SDR program for Microsoft Windows 2000/XP/Vista/7/8. Typical applications are Radio listening, Ham Radio, SWL, Radio

ExtIO.dll

Astronomy, NDB-hunting and Spectrum analysis. HDSDR (former WinradHD) is an advanced version of Winrad, written by Alberto di Bene

3.6 ExtIO.dll

The HDSDR software doesn't communicate with the SDR hardware directly. It communicates with the SDR radio hardware through an **External Input Output Dynamic Link Library** (ExtIO-DLL) file, which is a type of plug-in. Alberto di Bene designed the DLL interface so that Winrad can operate with a wide range of SDR hardware. We extended the DLL-interface to support TX switching. Winrad and HDSDR can support new hardware radios using an ExtIO-DLL file without the need to change the HDSDR software. ExtIO DLL files are written by anyone who wishes to provide support for any particular SDR hardware. In this manner, several radios can be used with a single piece of software. The software in this case is HDSDR.

3.7 How do I develop an ExtIO.dll ?

We assume that you are a software developer familiar with C/C++ programming. Here is a "hopefully" well documented header file [3], which specifies the interface between HDSDR and an ExtIO-DLL. Here is also an example ExtIO DLL [4] with sources as public-domain, developed with Microsoft Visual C++ 2008 Express ion.

3.8 Visual C++ 2008 Express

Visual C++ is part of the *Visual Studio Programming Suite*. A light *express* version is freely available. Visual Studio is an **Integrated Development Environment** (IDE) for developing web applications, client applications, and Windows Phone mobile applications. It supports C, C++, C#, Visual Basic

3.9 Qt

Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems. The easiest way to start application development with Qt is to download and install Qt 5. It contains Qt libraries, examples, documentation, and the necessary development tools, such as the Qt Creator integrated development environment (IDE)

3.10 RF hardware (USB Stick)

For the RF hardware there are some existing SDR USB sticks we can use it:

3.10.1   TERRATEC ran T stick DVB-T/DAB/DAB + Stick USB 2.0 [5]

---

[3] Guide\LC_ExtIO_Types.h - reference: http://www.hdsdr.de/download/LC_ExtIO_Types.h

[4] Guide\ExtIO_Demo_101\.. - reference: http://hdsdr.de/download/ExtIO/ExtIO_Demo_101.zip

[5] Reference: http://www.amazon.de/Terratec-ran-T-Stick-DVB-T-schwarz/dp/B007EB995U/

Price: EUR 29.98

The ultimate all-rounder for the digital TV and radio reception on your PC

USB stick for DVB-T (TV) and DAB / DAB + (radio). Direct recording and programming via EPG recording Software for both television and radio reception. Support for all major DVB-T Features

3.10.2   Hackrf (an-open-source-SDR-platform) [6]

Price: about 300$

Transmit or receive any radio signal from 30 MHz to 6000 MHz on USB power with HackRF.

HackRF is an open source hardware project to build a Software Defined Radio (SDR) peripheral.

3.11 RF Overview

Radio Frequency (RF) is a rate of oscillation in the range of about 9 kHz to 300 GHz, which corresponds to the frequency of radio waves, and the alternating currents which carry radio signals. It is the use of radio signals to communicate real-time data from the warehouse floor to the WMS database and back to the floor.

This expes processing in the warehouse. Scanners collect the data and transmit it via radio frequency to antennas located throughout the warehouse. From the antennas, the signal proceeds to an access point that communicates with the warehouse management system. This process reduces paper, data entry time delays, cycle count processing, out of stock quantities, typing errors...

---

[6] The primary web page for HackRF is: http://greatscottgadgets.com/hackrf/

3.12 RF Frequencies policies

The LNFT allocates Lebanon's radiofrequency spectrum into a number of frequency bands relevant to ITU regulations and specifies the general purposes for which the bands may be used. This process is referred to as the allocation of frequency bands to radio communication services. The primary objectives to be achieved with the radio spectrum are:

-To harmonize spectrum use with international developments. In this regard, Lebanon follows closely the work of the ITU, the CEPT, the league of Arab States and the local regional organization

-To manage the radio spectrum within Lebanon taking into account the governmental requirements and the needs of the various commercial sectors

-To stimulate technological innovation and competitiveness

The LNFT will be updated from time to time dependant on international initiatives and national decisions. The main source documentation used in the development of this version of LNFT was the ITU Radio Regulations and the Provisional Final Acts of the ITU WRC07.

Each band may be allocated to one or more services. The services printed in capitals are called "primary" services; the names which printed in small characters are called "secondary" services. Stations of a secondary service shall not cause harmful interference to stations of primary service and cannot claim protection from harmful interference from stations of a primary service.

For our purpose we should use the Amateurs service bands which are specified in the following table[7]:

Frequency Band (kHz MHz or GHz)	International Region 1 Allocation	National Allocation	Main application	Notes
135.7 – 137.8 kHz	FIXED MARITIME MOBILE Amateur 5.4C03 5.64 5.67 5.4C04	FIXED MARITIME MOBILE 5.64 5.67 5.4C04	SRD Maritime applications Ultra Low Power Active Medical Implants	ERC REC 62-01
1810-1830 kHz	AMATEUR 5.98 5.99 5.100 5.101	AMATEUR FIXED 5.98 MOBILE except aeronautical mobile 5.100	Amateur applications	
1830-1850 kHz		AMATEUR	Amateur applications	
3500-3800 kHz	AMATEUR FIXED MOBILE aeronautical 5.92	AMATEUR FIXED MOBILE aeronautical 5.92	Amateur applications	

---

[7] This table is a part from the LNFT document publish on 28-06-2008

7000-7100 kHz	AMATEUR AMATEURSATELLITE 5.140 5.141 5.141A	AMATEUR AMATEURSATELLITE	Amateur applications	
7100-7200 kHz	AMATEUR 5.141A 5.141B 5.141C 5.142	Amateur LBN 3 BROADCASTING 5.141C		
10100-10150 kHz	FIXED Amateur	FIXED Amateur		
14000-14250 kHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE		
14250-14350 kHz	AMATEUR 5.152	AMATEUR		
18068-18168 kHz	AMATEUR AMATEURSATELLITE 5.154	AMATEUR AMATEURSATELLITE		
21000-21450 kHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE		
24890-24990 kHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE		
28000-29700 kHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE		
50.0000 52.0000 MHz	BROADCASTING 5.164 5.162A	BROADCASTING LAND MOBILE 5.164 Amateur LBN 4, LBN 6		Geographical sharing with wind profiler radars in the range 46-68 MHz
144-146 MHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE	Amateur	
430-432 MHz	AMATEUR RADIOLOCATION 5.271 5.272 5.273 5.274 5.275 5.276 5.277	FIXED 5.276 MOBILE except aeronautical mobile AMATEUR RADIOLOCATION		
432-433.05 MHz	AMATEUR RADIOLOCATION Earth Exploration Satellite (active) 5.279A 5.138 5.271 5.272 5.273 5.274 5.275 5.276 5.277 5.280 5.281 5.282	FIXED MOBILE except aeronautical mobile AMATEUR RADIOLOCATION Earth Exploration Satellite (active) 5.279A 5.276 5.277		
433.05-434.79 MHz	AMATEUR RADIOLOCATION Earth Exploration-Satellite (active) 5.279A 5.138 5.271 5.272 5.276 5.277 5.280 5.281	FIXED MOBILE except aeronautical mobile AMATEUR RADIOLOCATION Land Mobile Earth Exploration-Satellite (active) 5.279A 5.138 5.276	ISM SRD	
434.79-435 MHz	AMATEUR RADIOLOCATION Earth Exploration-Satellite (active) 5.279A 5.138 5.271 5.272 5.276 5.277 5.280 5.281 5.282	FIXED MOBILE except Aeronautical Mobile AMATEUR AMATEURSATELLITE RADIOLOCATION Earth Exploration-Satellite (active) 5.279A 5.276		Amateur Satellite Service restricted to 435-438 MHz.
435-438 MHz		FIXED AMATEUR AMATEURSATELLITE RADIOLOCATION		

		5.276		
438-440 MHz	AMATEUR RADIOLOCATION 5.271 5.273 5.274 5.275 5.276 5.277 5.283	FIXED MOBILE except aeronautical mobile AMATEUR RADIOLOCATION 5.276		
1240-1300 MHz	EARTH EXPLORATION SATELLITE (active) RADIOLOCATION SPACE RESEARCH (active) RADIONAVIGATIONSATELLITE (S/E)(S/S) 5.329 5.329A 5.328B Amateur 5.282 5.330 5.331 5.335A	RADIOLOCATION EARTH EXPLORATION SATELLITE (active) SPACE RESEARCH (active) RADIONAVIGATION RADIONAVIGATIONSATELLITE 5.329 5.329A 5.328B Amateur Amateur-Satellite 5.282 5.330 5.331 5.335A	DME Radio navigation Amateur	This band 1260-1300 MHz is proposed to be protected to distance measurement equipment (DME) Wind profiler radars between 1270 MHz and 1295 MHz
2300-2450 MHz	FIXED MOBILE 5.384A Amateur Radiolocation 5.150 5.282 5.395	FIXED MOBILE Amateur Radiolocation	IMT (2300-2400 MHz) Fixed links	The band 2300-2400 MHz identified for IMT (WRC07)
		FIXED MOBILE Amateur Amateur Satellite 5.150 5.282	FIXED Links Amateur SRDs RLAN AVI RFID WLAN ISM	The band 2400-2483.5 MHz is designated for ISM applications. Radio communications must accept any interference caused by ISM apparatus in this band.
5650-5725 MHz	RADIOLOCATION MOBILE except aeronautical mobile 5.450A 5.446A Amateur Space Research (deep space) 5.282 5.451 5.453 5.454 5.455	FIXED 5.453 MOBILE RADIOLOCATION Amateur 5.282 LBN1	Defense systems Wireless Access RLANs Shipborne and VTS Radar Amateur applications	ERC REC 70-03 Amateur Satellite Service (Earth to space), 5650-5670 MHz from RR 5.282.
5725-5830 MHz	FIXED-SATELLITE (E/S) RADIOLOCATION Amateur 5.150 5.451 5.455 5.456 5.453	FIXED 5.453 MOBILE FIXED-SATELLITE (E/S) RADIOLOCATION Amateur 5.150 LBN1	Amateur applications SRD ISM Radars BFWA	ERC REC 70-03 ISM 5725-5875 MHz RTTT 5805-5815 MHz SRDs 5725-5875 MHz
5830-5850 MHz	FIXED-SATELLITE (E/S) RADIOLOCATION Amateur Amateur-Satellite (S/E) 5.150 5.451 5.455 5.456 5.453	FIXED MOBILE FIXED-SATELLITE (E/S) RADIOLOCATION Amateur Amateur-Satellite (S/E) 5.150 5.453	Fixed links Amateur applications SRD ISM Radars	Amateur Satellite 5830-5850 MHz (S/E)
10.00-10.15 GHz	FIXED MOBILE RADIOLOCATION Amateur 5.479	FIXED MOBILE RADIOLOCATION Amateur 5.479	Fixed links SAB	
10.15-10.30 GHz			Fixed links FWA	ERC REC 12-05 for fixed service ERC REC 13-04 for FWA 10.15-10.30/10.5-10.65 GHz
10.30-10.45 GHz			Fixed links SAB	
10.45-10.50 GHz	RADIOLOCATION Amateur Amateur Satellite 5.481	FIXED RADIOLOCATION MOBILE Amateur Amateur Satellite	Fixed links SAB	ERC REC 12-05 for fixed service

Basics

24.00-24.05 GHz	AMATEUR AMATEURSATELLITE 5.150	AMATEUR AMATEURSATELLITE 5.150	Amateur	ISM 24-24.5 GHz
24.05-24.25 GHz	RADIOLOCATION Amateur Earth exploration Satellite (active) 5.150	RADIOLOCATION Amateur Earth exploration Satellite (active) Fixed Mobile 5.150	Amateur ISM SAB SRD Motion sensors	ERC REC 70-03 ISM 24-24.5 GHz
47.00-47.20 GHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE	Amateur applications Amateur satellite applications	
48.20-48.54 GHz	FIXED FIXED-SATELLITE (E/S) 5.552 (S/E) 5.516B 5.554A 5.555B MOBILE	FIXED FIXED-SATELLITE (E/S) 5.552 (S/E) 5.516B 5.554A 5.555B MOBILE Amateur	Fixed satellite applications SAB	ERC REC 25-10 Feeder link band for 40GHz broadcasting satellites
76.00-77.50 GHz	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur-Satellite Space Research (S/E) 5.149	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur-Satellite Space Research (S/E) 5.149	Radio astronomy applications RTTT Amateur applications Amateur satellite applications Civil radiolocation	Spectral line and wide band continuum observations Road Transport and Traffic Telematics 76-77 GHz Radar
77.50-78.00 GHz	AMATEUR AMATEUR SATELLITE Radio Astronomy Space Research (S/E) 5.149	AMATEUR AMATEUR SATELLITE Radio Astronomy Space Research (S/E) 5.149	Radio astronomy applications	Spectral line and wide band continuum observations
78.00-79.00 GHz	RADIOLOCATION Amateur Amateur Satellite Radio astronomy Space Research (S/E) 5.149 5.560	RADIOLOCATION Amateur Amateur-Satellite Radio astronomy Space Research (S/E) 5.149 5.560	Radio astronomy applications	Spectral line and wide band continuum observations
79.00-81.00 GHz	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur Satellite Space Research (S/E) 5.149	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur Satellite Space Research (S/E) 5.149	Radio astronomy applications	Spectral line and wide band continuum observations
122.25-123 GHz	FIXED INTER-SATELLITE MOBILE 5.558 Amateur 5.138	FIXED INTER-SATELLITE MOBILE 5.558 Amateur 5.138	Amateur applications Amateur satellite applications SRD	ERC REC 70-03
134-136 GHz	AMATEUR AMATEURSATELLITE Radio Astronomy	AMATEUR AMATEURSATELLITE Radio Astronomy	Amateur applications Amateur satellite applications	
136-141 GHz	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur satellite 5.149	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur satellite 5.149	Radio astronomy applications Amateur applications Amateur satellite applications	Spectral line and wide band continuum observations
241-248 GHz	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur-Satellite 5.138 5.149	RADIO ASTRONOMY RADIOLOCATION Amateur Amateur-Satellite 5.138 5.149	Amateur applications Amateur satellite applications	Spectral line and wide band continuum observations ERC REC 70-03
248-250 GHz	AMATEUR AMATEURSATELLITE	AMATEUR AMATEURSATELLITE		

	Radio Astronomy 5.149	Radio Astronomy 5.149		

The yellow rows indicate the frequency band which we are going to use it.

From 433.05 to 434.79 MHz	AMATEUR RADIOLOCATION Earth Exploration- Satellite (active) 5.279A 5.138 5.271 5.272 5.276 5.277 5.280 5.281	FIXED MOBILE except aeronautical mobile AMATEUR RADIOLOCATION Land Mobile Earth Exploration- Satellite (active) 5.279A 5.138 5.276	ISM SRD

3.13 RF modules

An RF Module is a (usually) small electronic circuit used to transmit, receive, or transceive radio waves on one of a number of carrier frequencies. RF Modules are widely used in consumer application such as garage door openers, wireless alarm systems, industrial remote controls, smart sensor applications, and wireless home automation systems. They are often used instead of infrared remote controls as they have the advantage of not requiring line-of-sight operation.

In this project we will use an RF module to send and receive message between two programmable microcontrollers with some condition related to the frequency, range and module speed. For this purpose there are several RF modules which may do this, bellows we will take a look on some of it:

3.13.1 STD-402

The transceiver to be used is **MB-STD-RS232.** It is a bi-directional semi-duplex radio modem having RS232 serial interface. It uses CIRCUIT DESIGN's standard 434 MHz FM Narrow Band transceiver module **STD-402** transceiver for RF part. This transceiver was selected because of its frequency of 434 MHz For this frequency in Germany there is no extra permission necessary. Another reason is that this transceiver is a cheap one.

The **STD-402** transceiver is an UHF Narrow Band Multi channel Transceiver. The UHF FM-Narrow Band semi-duplex radio data module **STD-402** equipped PLL controller in its robust metal housing. Unlike other transceivers, the **STD-402** is ready to transmit RF data without complicated controller board. The compact size and low power consumption of the **STD-402** make it ideal for battery operated applications where its interference rejection and practical distance range are much better than similar RF modules based on Wide Band SAW – resonator frequency devices.

Most of RF settings are done by internal microcomputer, which allows the user to manipulate the module without professional knowledge of RF circuit.

3.13.1.1 Special for *MB-STD-RS232*
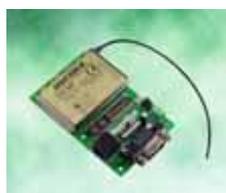
Basics

<p align="center">**Figure 5.3.1: MB-STD-RS232 – CIRCUIT DESIGN**</p>

The RF part complies with the European radio, EMC and safety requirements and has been notified in major European countries under the R & TTE directive. The **MB-STD-RS232** provides long range data link at low/medium data rate for various industrial telemetry and data transfer applications. Also this board can be used as a test board of the **STD-402 TR**.

Features

- CE compliance **STD-402** 434 MHz RF module on the board.
- RS232 interface with D-sub 9pins connector or Modular 6pin jack.
- Fixed frequency / Auto frequency setting selectable.
- Cross / Straight cable selection SW.

Applications

- Serial data transmission (RS232C communication)
- Telemeter (FA line, Sensor information)
- Wireless connection between PC and peripheral RS232 equipment

General Description

MB-STD-RS232 is designed to make it possible for the user to connect between RS232 equipments with the radio. **STD-402** 434 MHz narrow band radio module that complies with EN300220 is equipped on the board. 64 channels are pre-programmed in the module.

There are two frequency setting are available. **In fixed (manual) setting**, RF channel can be set on board switches. **In auto setting**, RF channel is set to vacant channel automatically.

Operation mode and communication set up (Ack, parity, data rate) can be selected by on board dip-switch. The **operation mode 1** is designed for two-way communication and the **operation mode 2** is designed for one-way communication (TX -> RX).

1:N communication is possible by using unique module ID number that designated to each RF module.

Specification

RF parameter

Communication mode	Half-Duplex
Frequency range	433.200 to 434.775 MHz
CH step	25 kHz
Number of CH	64 CH
CH setting	Fix / Auto (8Gr*8ch)
Modulation data speed	9600bps
Modulation	2FSK
Emission class	F1D
Transmission power	10 mW

Serial Interface

Interface	RS-232C

RF modules

Data format	Asynchronous communication (UART)
Data speed of RS	1200/2400/4800/9600 bps
Flow control	RS / CS hardware control
Buffer	Transmission 2kB, Reception 2KB
Interface connector	D-Sub 9P / Modular 6P

Other

Switches	Power, Frequency, Operation Mode, Cable (Cross/Straight)
LED indication	TX, RX, RSSI, LD, LE
Dimension	85*53*15mm
Supply Voltage	4.0 to 9V DC.
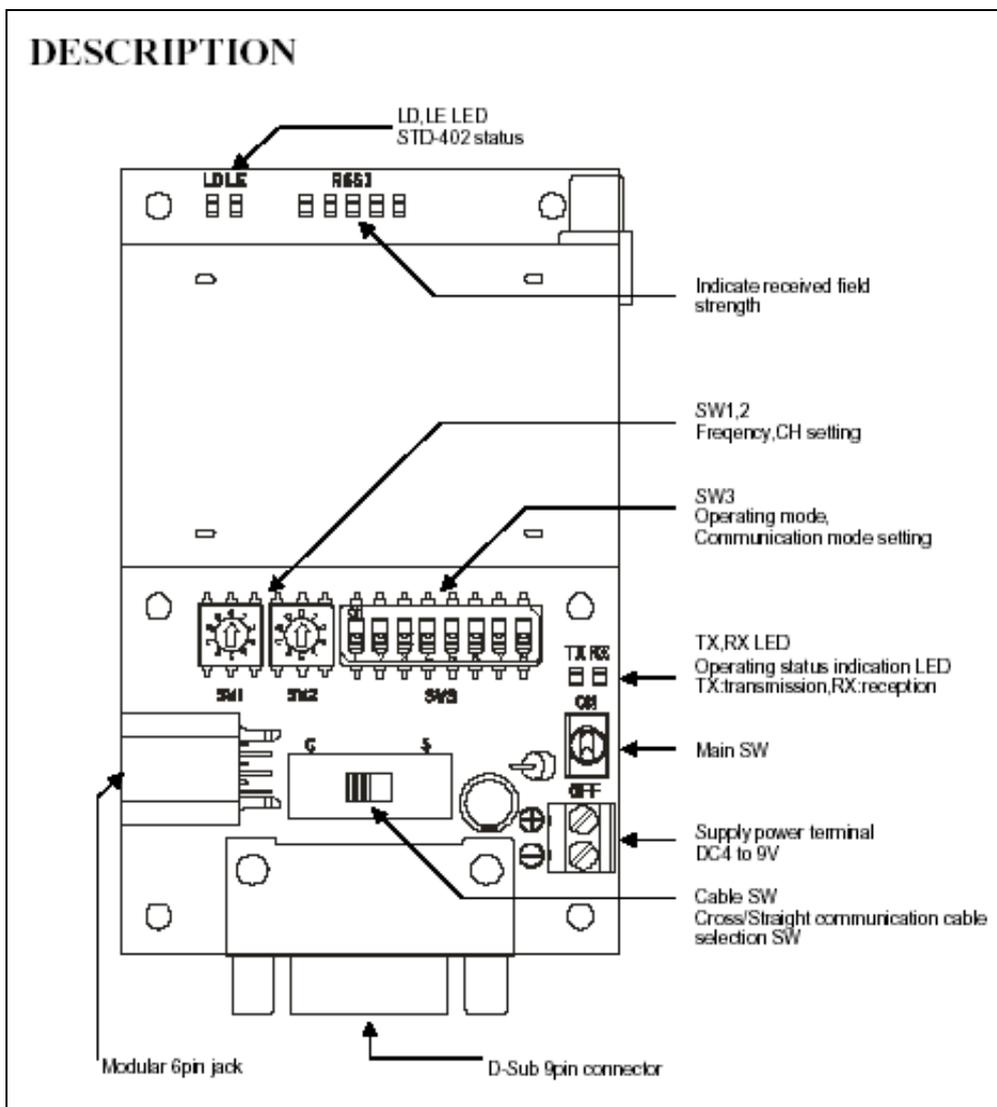
Figure 5.3.2: MB-STD-RS232 – CIRCUIT DESIGN

For more details about this board, refer to Annex A.

3.13.1.2 Special for STD-402 (Transceiver)
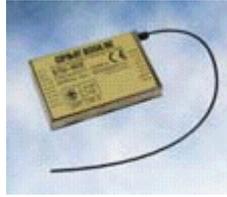
Figure 6.6: STD-402 Transceiver – *CIRCUIT DESIGN*

The **STD-402** transceiver is an UHF Narrow Band Multi channel Transceiver.

The UHF FM-Narrow Band semi-duplex radio data module **STD-402** equipped PLL controller in its robust metal housing. Unlike other transceiver, the **STD-402** is ready to transmit RF data without complicated controller board. The compact size and low power consumption of the **STD-402** make it ideal for battery operated applications where its interference rejection and practical distance range are much better than similar RF modules based on Wide Band SAW – resonator frequency devices.

Most of RF settings are done by internal microcomputer, which allows the user to manipulate the module without professional knowledge of RF circuit.

Features

➢ European EN300 200 standard compliance.
➢ High technology into compact module for easy operation.
➢ Low voltage operation from 3.6 V DC.
➢ Low current consumption, ideal for battery operated applications.
➢ 9600bps data rate.
➢ Carrier sense output for Multi-Channel access operation.

Application

➢ Remote control system.
➢ Security systems.
➢ Bi-directional communication systems.
➢ Telemetry systems
➢ Handy terminal.

STD-402 characteristics

❖ **Common**

Communication form	Semi-duplex
Frequency range	433.200 MHz to 433.775 MHz
Channel step	25 KHz.
Baud rate	9600bps max.

Supply voltage	3.6 – 12 V DC (Direct Mode).
Dimensions	53 × 35 × 12 mm.

❖ **Transmitter**

RF output power	9 mW ± 1mW.
Data input level	3.6 – 12V (Direct Mode).
Input signal	Digital
Spurious emission	< -60 dBm (< 1 GHz).
Supply current	36 mA.

❖ **Receiver**

Receiver type	Double super heterodyne PLL synthesizer.
Selectivity	± 4 kHz at –6dB point.
Data output	Digital.

- **The** STD-402 **transceiver has 3 mode operation guides:**

1. **Direct Mode Operation Guide (For more details about this mode, refer to Annex B)**
2. **Auto Mode Operation Guide. (For more details about this mode, refer to   Annex C)**
3. **Auto Mode Operation Guide for CPU interface. (For more details about this mode, refer to Annex D)**

Or the MB-STD-RS232 **equips** STD-402 **transceiver module and performs packet communication using CPU interface mode of the transceiver.**

3.13.2   RFM42B-RFM31B 433MHz
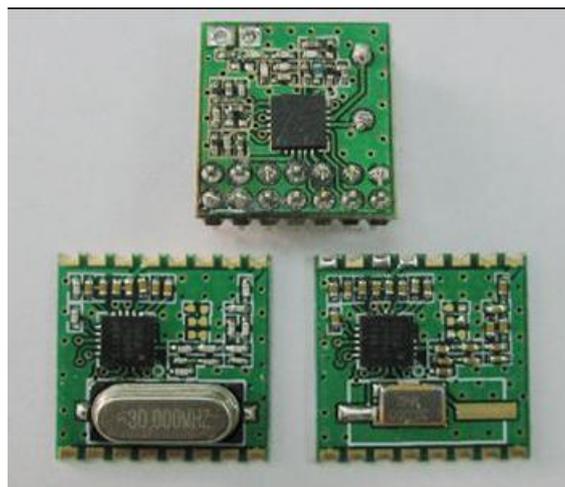
Features:
- 433/868/915MHz ISM bands Frequency range:
- Low Power Consumption
- Data Rate = 0.123 to 256 kbps
- FSK, GFSK, and OOK modulation

Basics

- Power Supply = 1.8 to 3.6 V
- Ultra low power shutdown mode
- Wake-up timer
- TX 64 byte FIFO
- Low battery detector
- Temperature sensor and 8-bit ADC
- –40 to +85 °C temperature range
- Integrated voltage regulators
- Frequency hopping capability
- On-chip crystal tuning
- 14-PIN DIP & 16-PIN SMD package
- Low cost
- Power-on-reset (POR)

Special for RFM42B (Transmiter):

- Output Power Range
  - +1 to +20dBm (RFM42B)
  - –8 to +13dBm (RFM43B)
- Integrated 32 kHz RC or 32 kHz XTAL
- Configurable packet handler

Special for RFM31B (Receiver):

- Sensitivity = –121 dBm
- Digital RSSI
- Auto-frequency calibration (AFC)
- Clear channel RX BW 2.6–620 kHz
- Programmable assessment
- Programmable packet handler
- Programmable GPIOs
- Embedded antenna diversity algorithm
- Configurable packet handler
- Preamble detector
- RX 64 byte FIFO

Application example:

RFM31B module
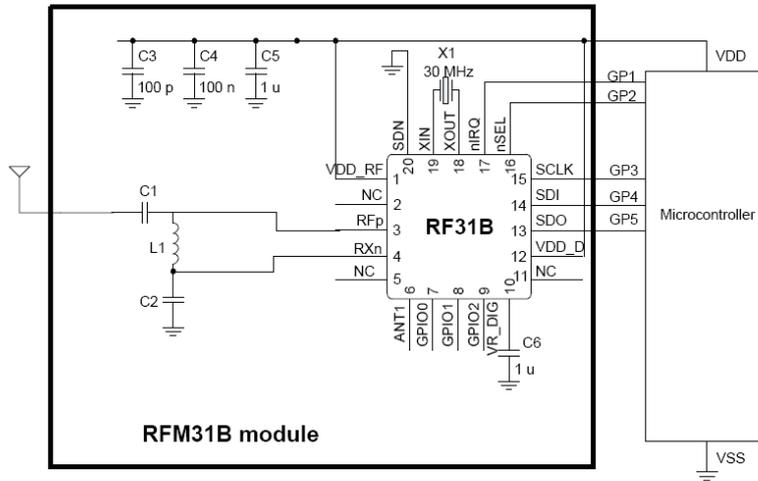
3.13.3 BOWITZ W.T.

من كتاب الإرسال اللاسلكي و البث

الكتاب الثامن من موسوعة عالم الالكترونيات للمهندس أمين فهمي
دار الراتب الجامعية

الباب الثامن:
أجهزة الارسال و الاستقبال التجارية
إبتداء من صفحة: 165

3.13.4 Comparison between modules

module	Frequency	Range (m)	Speed (bps)	other
SHY-J6122TR	300 – 450 MHz			Made In chine Available in Lebanon
Rx Tx 315Mhy	315 MHz 433.92 MHz	> 500 m	< 10Kbps	Made In chine Available in Lebanon
RFM12 433MHz	433 MHz		> 115.2 Kbps	Programmable TxRx bandwidth
RFM12 915 MHz	915 MHz			SPI interface Made In chine Available in Lebanon
STD-402	434 MHz	500 m	9600 bps	Serial com. Need extra circuit (max232) Not available anymore
BOWITZ W.T.				Full W.T. project, We should build it by ourselves using the BOWITZ open source information

4   Specification

4.1 System Requirements

[SysReq 1] The system shall be a demonstration plattform for customers. The customers can then specify their individual requirements. Afterwards the IAP ECS paltform shall be migrated in every customer project to the specific needs

[SysReq 2] Our goal is to design an Emergency communication system (voice and information) for the red halfmoon, Red Cross or police to stay on touch in emergency situations.

4.2 Hardware Requirements

[HWReq1] The system shall use a SDR (software defined radio) with RF transmission technology.

[HWReq 2] The sending and receiving HW shall be a cheap off-the-shell system so that the project can be finished in December 2013.

4.3 Software Requirements

tbd

5   System Design

5.1 System Overview
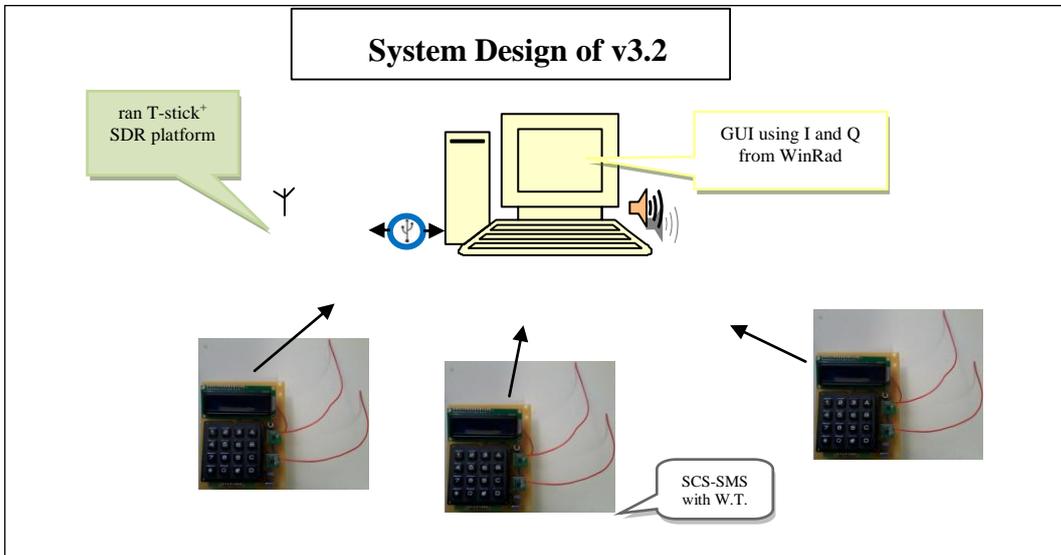
5.2 Central Station

5.2.1 Architecture
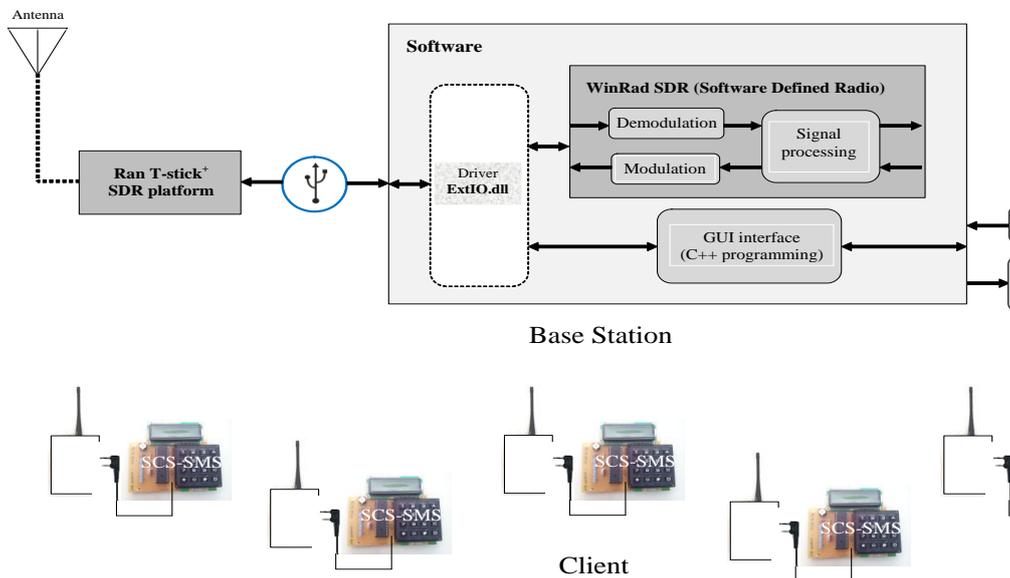
5.2.2 SDR development side

In our project, we need the Software Defined Radio code which is included in **HDSDR** software. But as we know the **HDSDR** software is not open-source software while **WinRad** is. Then we have two potential choices to do this step:

For the HDSDR: we can change and develop the ExtIO.dll file of this program to input from my STD hardware source and to output on my GUI. In this case the HDSDR software will work o the background of our GUI software.

This choice means that we should use the HDSDR software in the two communication sides. This means also that we should use PC on the two sides again.

For the WinRad: as we say before WinRad is Open source then we can use its code. We should read its code to know where is the SDR code to copy it to our GUI.

This choice has the following problem: with WinRad we can only receive while we need to send and receive. This means that we should develop the code such that also sending is possible.

5.2.3 Graphical User Interface

The Graphical User Interface is the interaction interface between the user and the system. The goal of this interface is to monitor the location status: road status, problems on road, hospital status, and also it will have a messaging box to write notes for each other. Also the user interface should have the HDSDR option (change frequencies, send/receive, volume up/down) with an extra button to open the HDSDR software when user want.
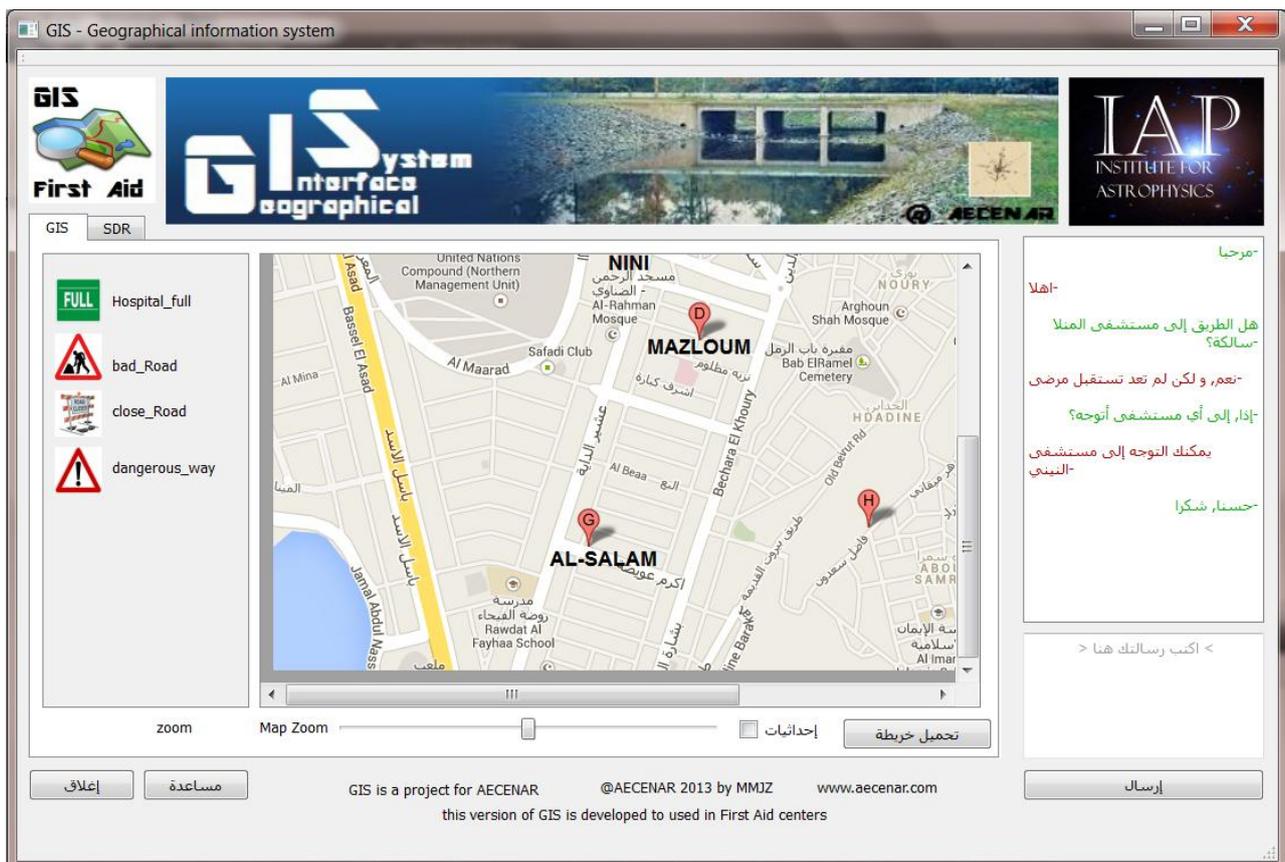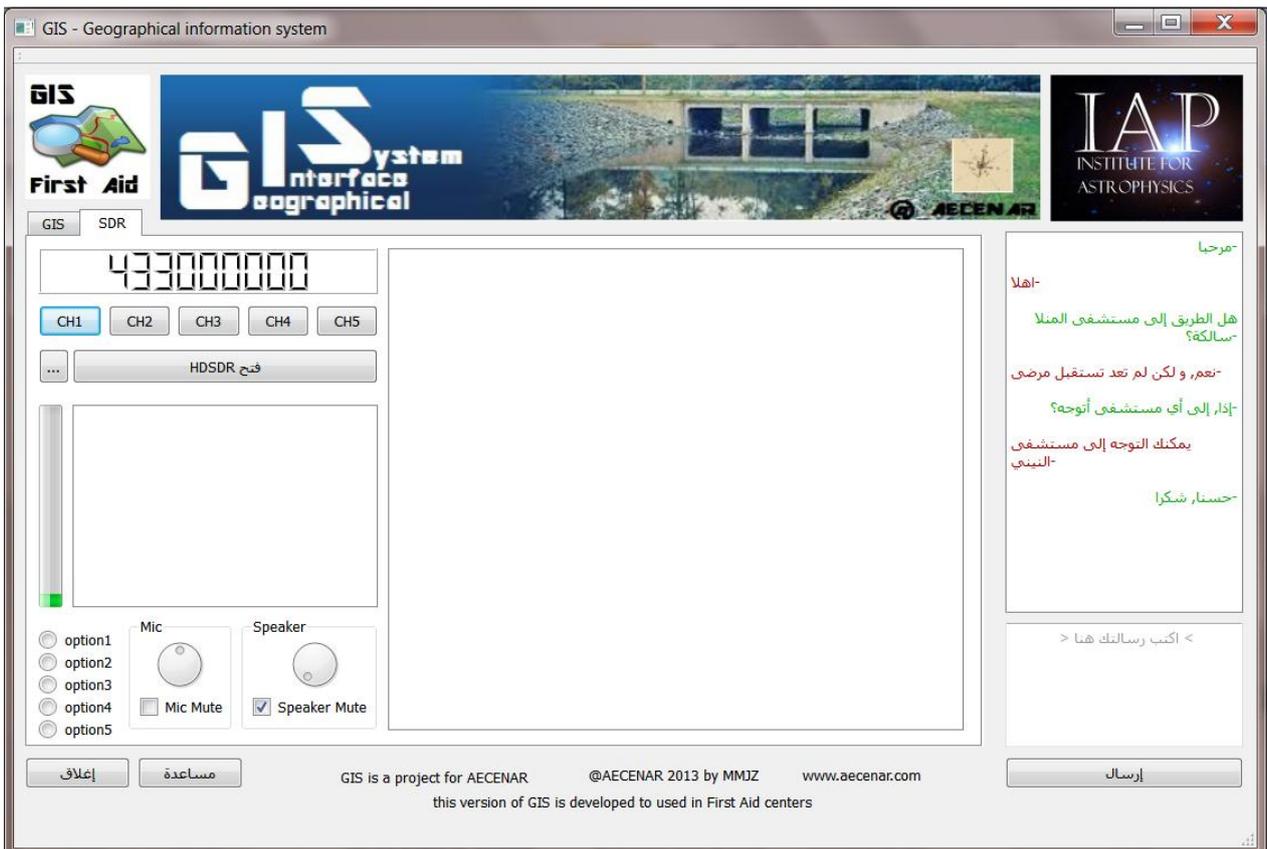
5.3 Mobile Stations

In the client side we can use the hardware of **SCS-SMS** project instead of PC but we should first add an RF transceiver to it with doing some modification on it

The main modifications are:

- Adding the RF hardware (amplifier, filter, antenna)
- Adding the A/D and D/A converter
- Put the SDR code on its processor

System Design

Base Station

Client
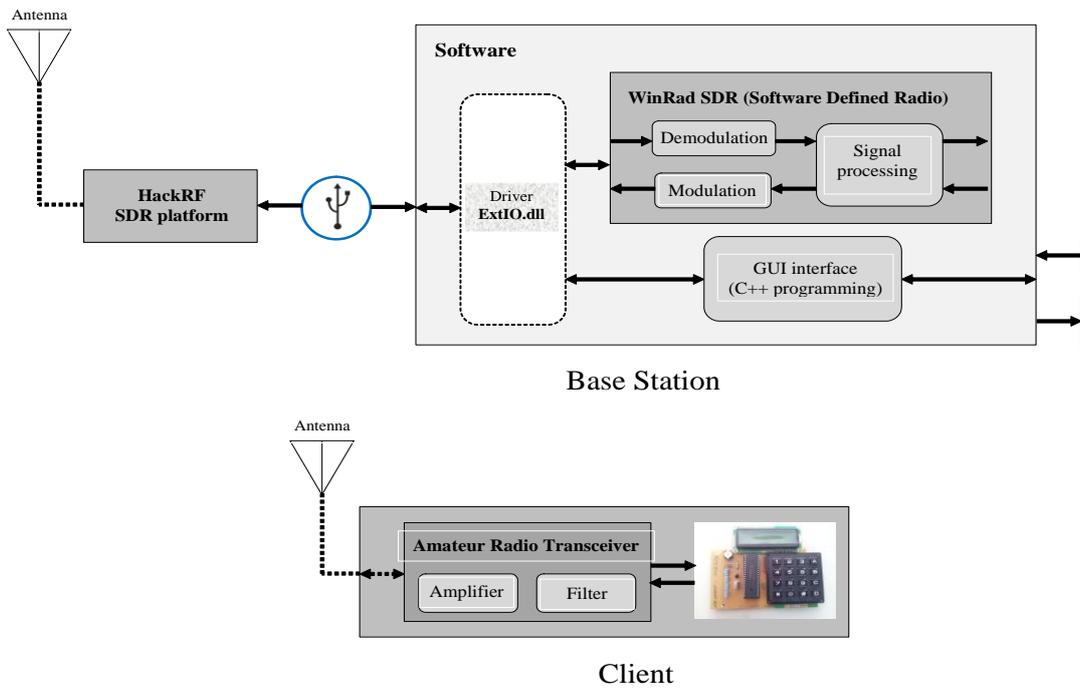
In this version we need to change on the hardware with the software of the SCS-SMS project to be able to use it in our project. The basic changes are:

- Adding the RF hardware (amplifier, filter, antenna)
- Adding the A/D and D/A converter
- Put the SDR (HDSDR) on a second processor

The new block diagram will be:

SCS-SMS after changes to GIS-STD
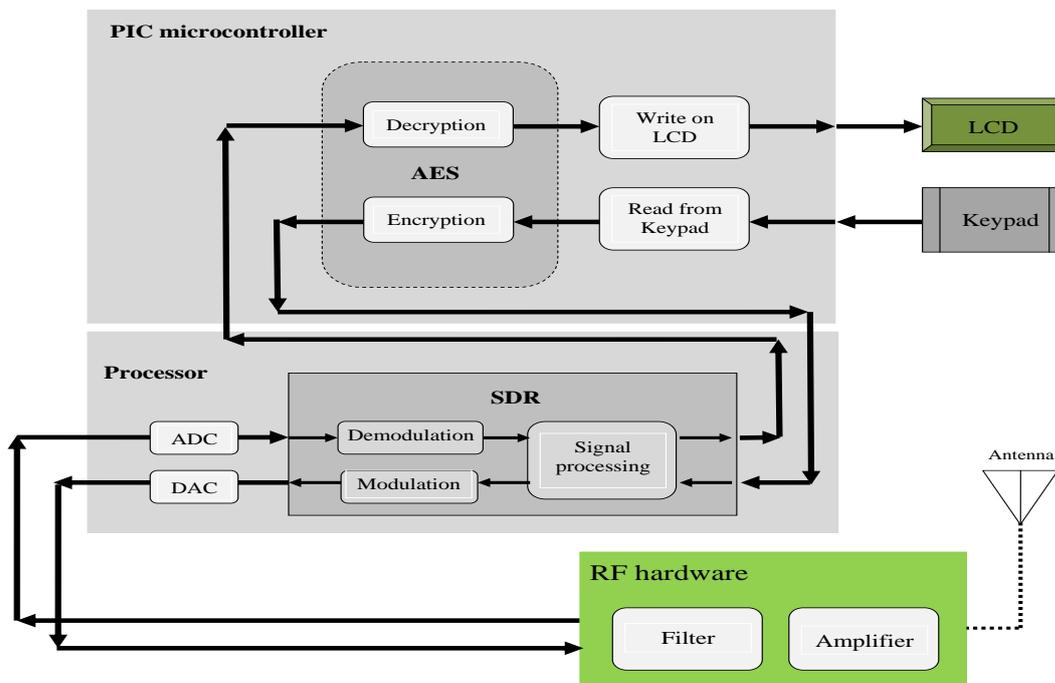
6   Mechanics

6.1 Mechanical Design
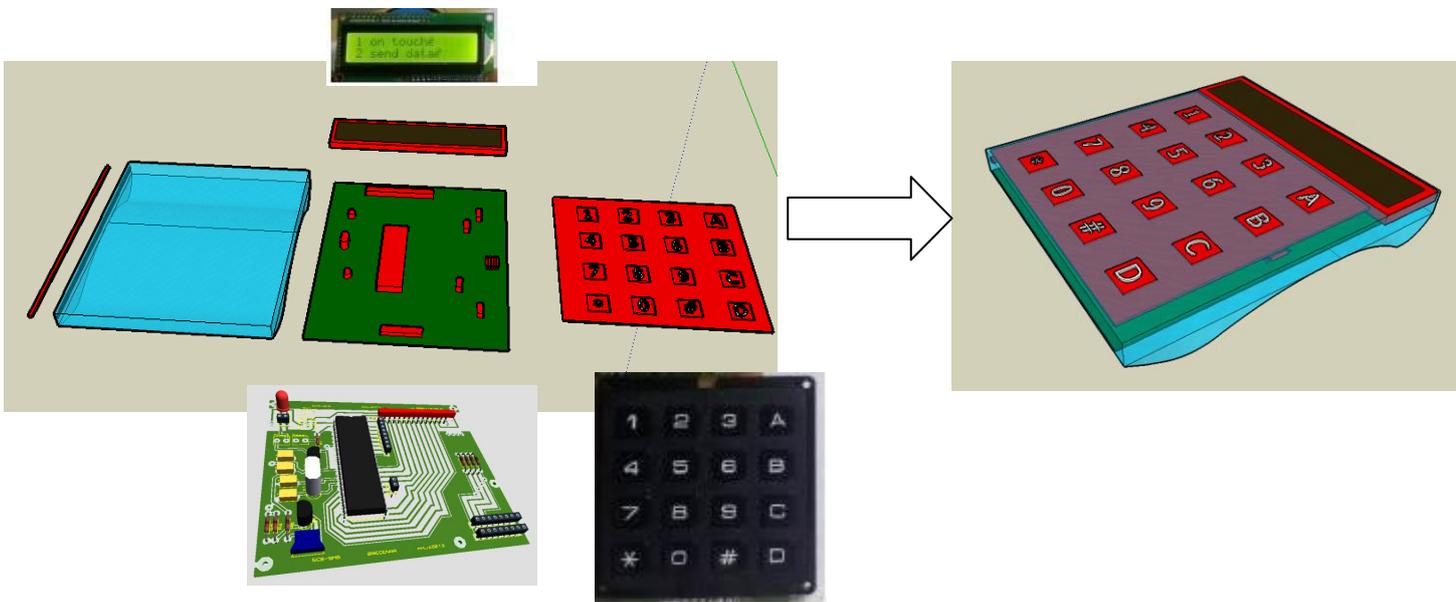
6.2 Prototype without cover
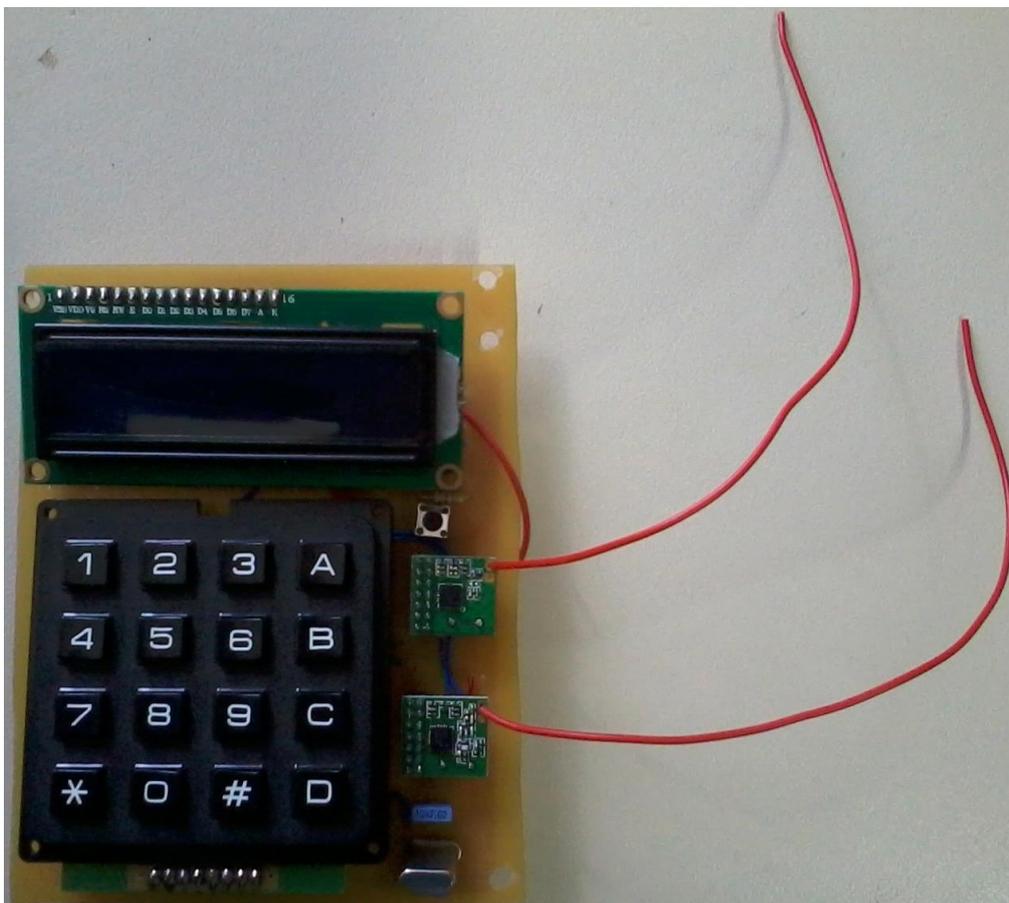
7   SCS-SMS

Secured communication System

7.1 Abstract of SCS-SMS

The goal of this project is to sending SMS securely on several way of communication (i. e.: telephone, mobile, RF transceiver…)

7.2 System design

System plan:
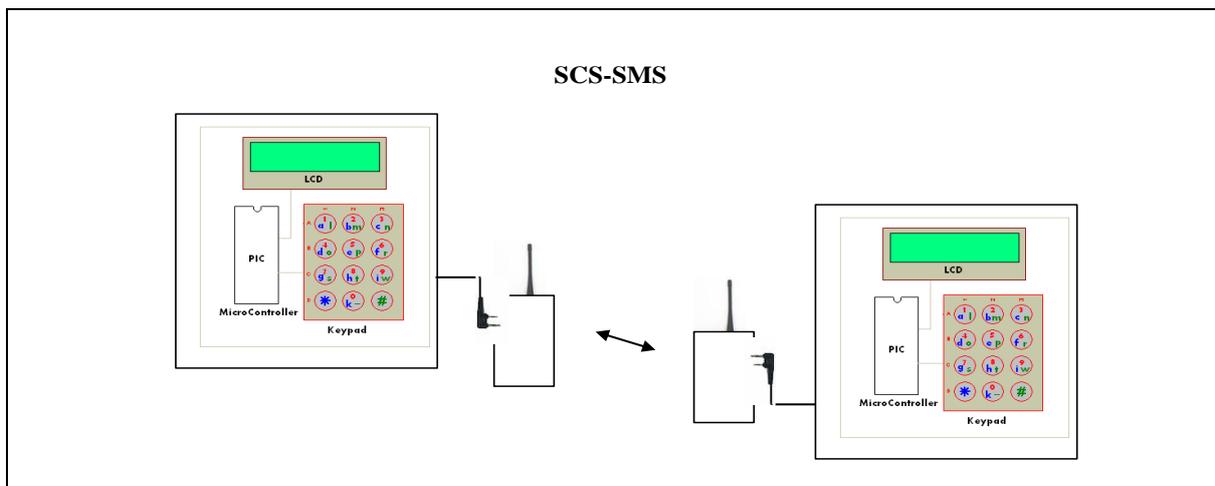
Block diagram:
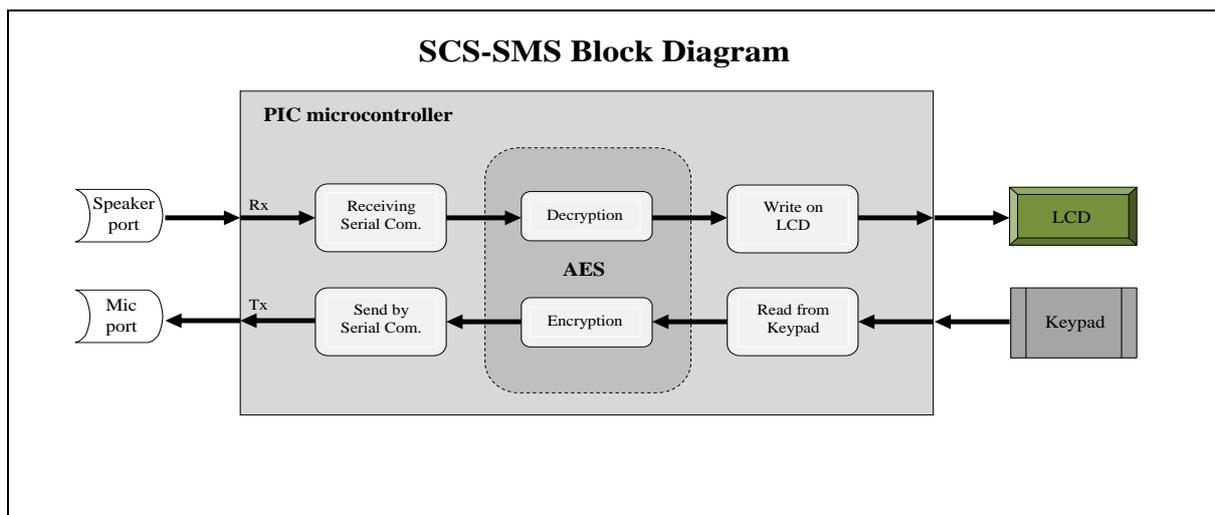
New:
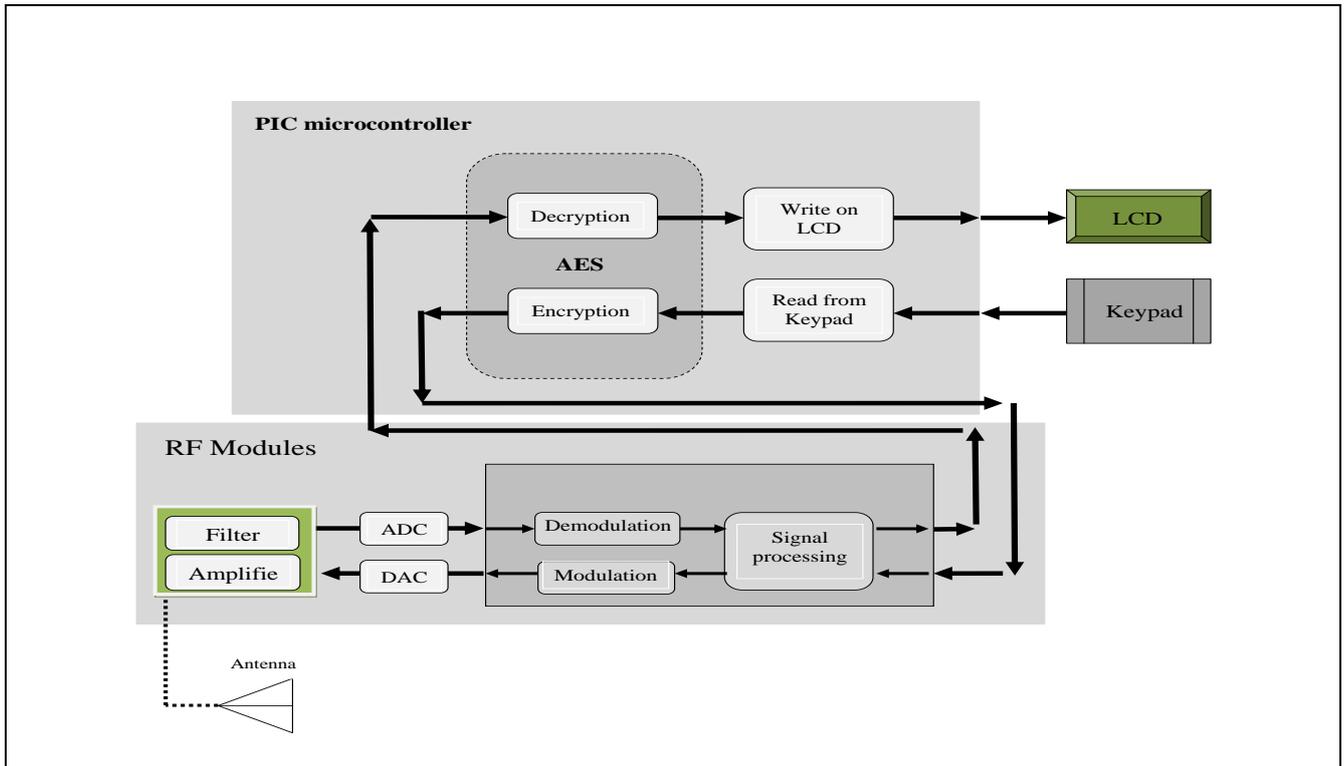
7.3 Architectures

The architectures of this project is contain as a basic component a PIC microcontroller with a 4x4 keypad with a LCD and with some other electronic components

Equipment used:

PIC microcontroller	18f4550	1
LCD	2x16 (line X character)	1
Keypad	4x4	1
Resistor	4.7 kΩ	1
Pot resistor	10 kΩ max	1
Capacitor	1µF	4
Crystal	48000 MHz	1
Voltage regulator	78L05	1
Push button		1

PIC microcontroller:

The microcontroller needs a programmer with a compiler to write your program in it. The most widely compiler is the MPLAB software which we will use it in this project, the MPLAB

already have an Assembly compiler for the PIC but if you want to write your program in C language then you need a C compiler for your PIC. On our project the compiler we use it is MCC18 which is for the 18 PIC series. We will discuss the c program in the next chapter

LCD:

2x16 LCD is used in this project to display character on 2 lines 16 characters Liquid Crystal Display. This LCD has 14 input pins to write on and to control LCD

LCD PINs description:

Pin	Symbol	Description
1	Vss	Ground
2	Vcc	+5 V power supply
3	Vee	Power supply to control contrast
4	RS	RS=0 to select command register RS=1 to select data register
5	RW	RW=0 for write RW=1 for read
6	E	Enable
7	DB0	The 8-bit data bus
8	Db1	The 8-bit data bus
9	DB2	The 8-bit data bus
10	DB3	The 8-bit data bus
11	DB4	The 8-bit data bus
12	DB5	The 8-bit data bus
13	DB6	The 8-bit data bus
14	DB7	The 8-bit data bus

On each time you need to write on LCD or to send a command to LCD you should set the Enable pin **E** before then disable it after sending.

Also you should do a harmony between the speed of the LCD receiving with the speed of PIC data sending, and this will be do it by the PIC software using a **delay**s function.

Finally, a Pot resistor should be connecting to the **R** pin of the LCD to control the LCD light brightness.

Keypad:

The keypad we use is 4x4 keypad with 8 I/O pins; the 4x4 keypad design show in the picture bellow:
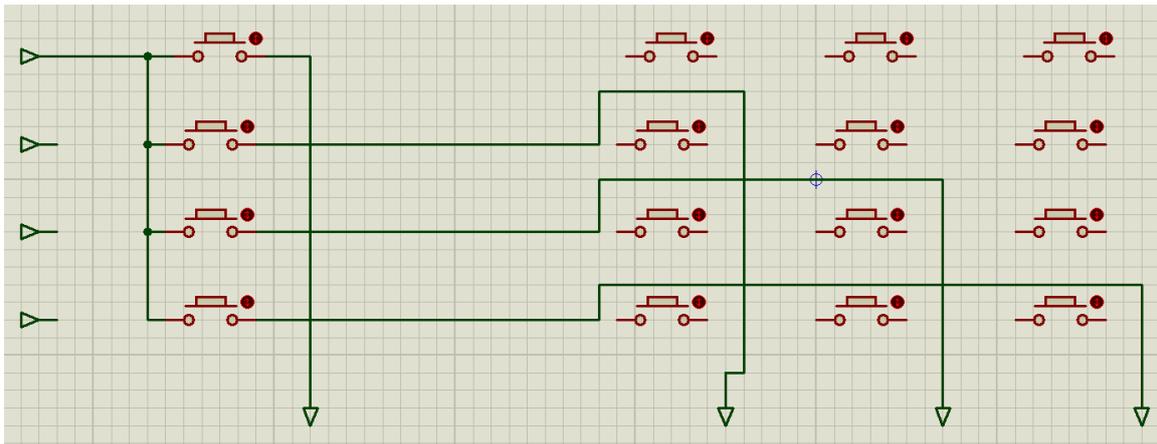
As we see in the picture above, we nave 4 input pins and 4 output pins. We should set each input one by one and in each time we should read the output on the 4 output pins to know which key was pressed.

In the software program you should specify the meaning of each key. In our project, we specify our key as follow:
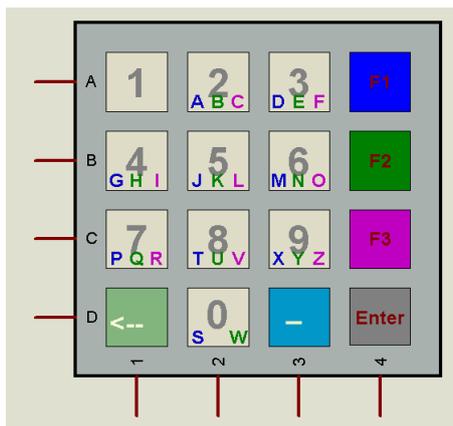
In the normal case the keypad will write numbers if **F1** (Function1 key) pressed then the key will write the Blue character (A, D, G, J, M, P, T, X, S). IF **F2** (Function2 key) pressed then the key will write the Green character (B, E, H, K, N, Q, U, Y, W). If the **F3** (Function3 key) pressed then the key will write the Red character (C, F, I, L, O, R, V, Z). There are also three other key which are: (**Enter**) to send data, (_) to write a space, (←) to delete the last character.

System Design:

The design of the system was developed on Proteus to simulate and test before the built of the real system.

The figure bellow shows us the system design:

Architectures
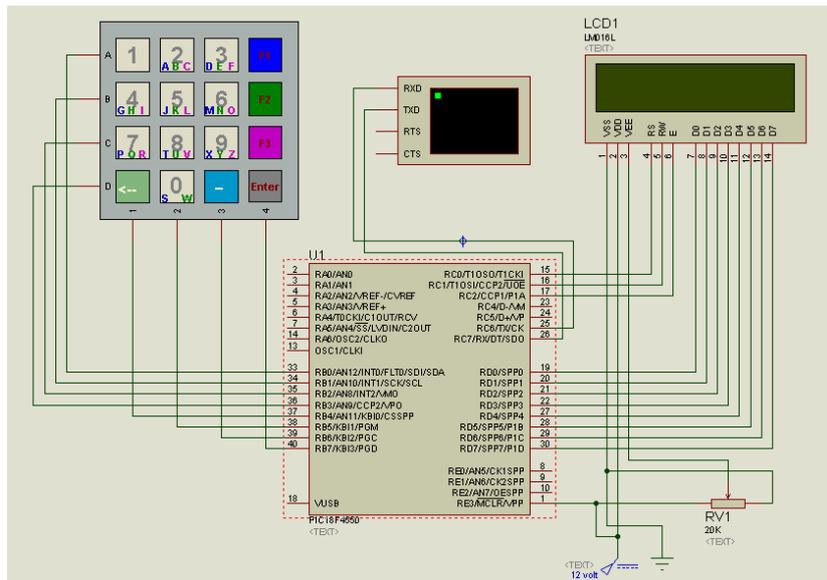
As we see, our design is too simple. It has three base components: PIC, keypad, and a LCD. The Keypad is connected to port B of the PIC, and LCD connected to the port C and D. port D connected to LCD Data register and port C (first three pins only) connected to LCD Controller. The virtual machine is added for testing purpose.

The design is enough in Proteus simulation but it is not in the real hardware while we should add some components for: timing resonator, voltage regulator, and reset button.

Timing components resonator:

A PIC microcontroller requires an external clock circuit (some PIC microcontrollers have built-in clock circuits) to function accurately. For accurate timing applications, the clock circuitry consists of a crystal, and two small capacitors. Figure bellow shows the circuit diagram of a PIC microcontroller with a 4-MHz crystal clock circuit. The crystal and the capacitors are connected to the OSC1 and OSC2 inputs of the microcontroller.
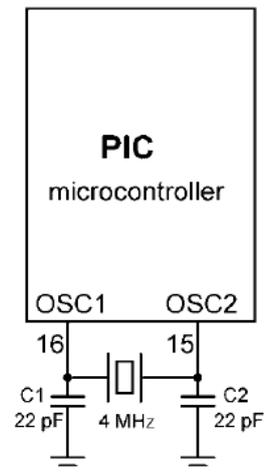
Power source and Reset circuit

A PIC microcontroller starts executing the user program from address 0 of the program memory when power is applied to the chip. As shown in Figure bellow, the reset input (MCLR) of the microcontroller is usually connected to the 5V supply voltage through a 4.7K resistor.
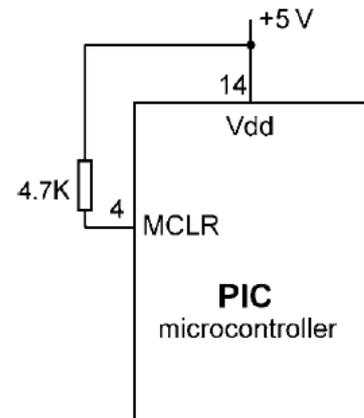
There are many applications where the user may want to force reset action e.g. by pressing an external button so that the program re-starts to execute from the beginning. External reset is very useful during microcontroller-based system development and testing. Figure bellow shows how an external reset button can be connected to a PIC microcontroller. Normally the MCLR input is at logic 1, and goes to logic 0 which resets the microcontroller when the reset button is pressed. The microcontroller goes back to the normal operating mode when the button is released.
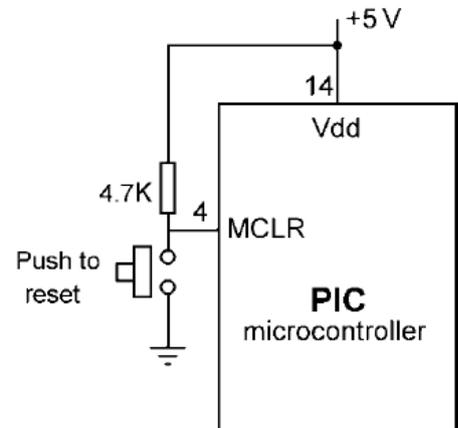
Power supply

Every electronic circuit requires a power supply to operate. The required power can either be provided from a battery, or the mains voltage can be used and then reduced to the required level before it is used in the circuit (e.g. a mains adaptor). In this section, we shall look at the design of a power supply circuit to power our PIC microcontroller circuits.

PIC microcontrollers can operate from a power supply voltage in the range 2 to 6V. The standard power supply voltage in digital electronic circuits is 5 V and this is the voltage with which the PIC microcontrollers are mostly operated. Unfortunately, it is not possible to obtain 5 V using standard alkaline batteries only. The nearest we can get is by using three batteries, which gives 4.5 V and this is not enough to power standard logic circuits. The simplest solution to drop the voltage from 9 to 5 V is by using a potential divider circuit using two resistors. Although a potential divider circuit is simple, it has the major disadvantage that the voltage at the output depends on the current drawn from the circuit. As a result of this, the output voltage will change as we add or remove components from our circuit. Also, the output voltage falls as the battery is used. A voltage regulator circuit is needed to convert the 9 V battery voltage into 5V, independent of the current drawn from the supply. A basic voltage regulator circuit consists of a regulator integrated circuit and filter capacitors. Figure bellow shows a low-cost voltage regulator circuit using the **78L05**-type voltage regulator IC, and two filter capacitors. **78L05** is a 3-pin IC with a maximum current capacity of 100 mA.
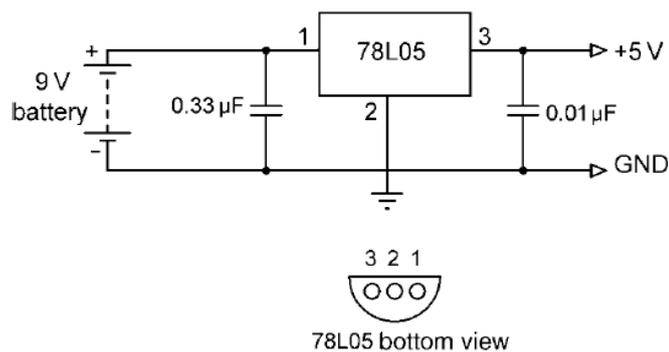
Architectures

One of the pins of **78L05** is connected to the +V terminal of the battery in parallel with a 0.33-uF capacitor. One of the pins is connected to the -V terminal of the battery. The third pin provides the +5 V output and a 0.01-uF capacitor should be used in parallel with this pin. In applications where a larger current is required, the **7805** regulator IC can be used. This is pin compatible with the low-power **78L05** and it has a maximum current capacity of 1 A. **78L05** should be used with a suitable heat-sink in applications drawing more than a few hundreds of mill-amperes.

The complete circuit diagram of our PIC based basic system, together with the power supply, is shown in Figure bellow. The circuit is now fully functional, what is required now is to write our program and load it into the program memory of the microcontroller.
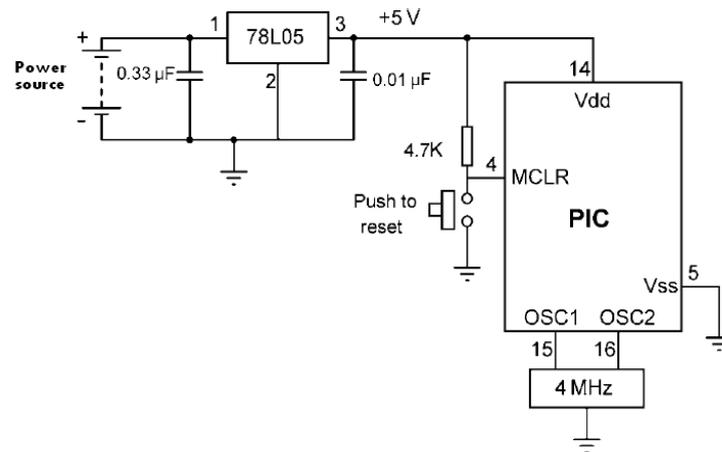
The final circuit is show in the figure bellow:
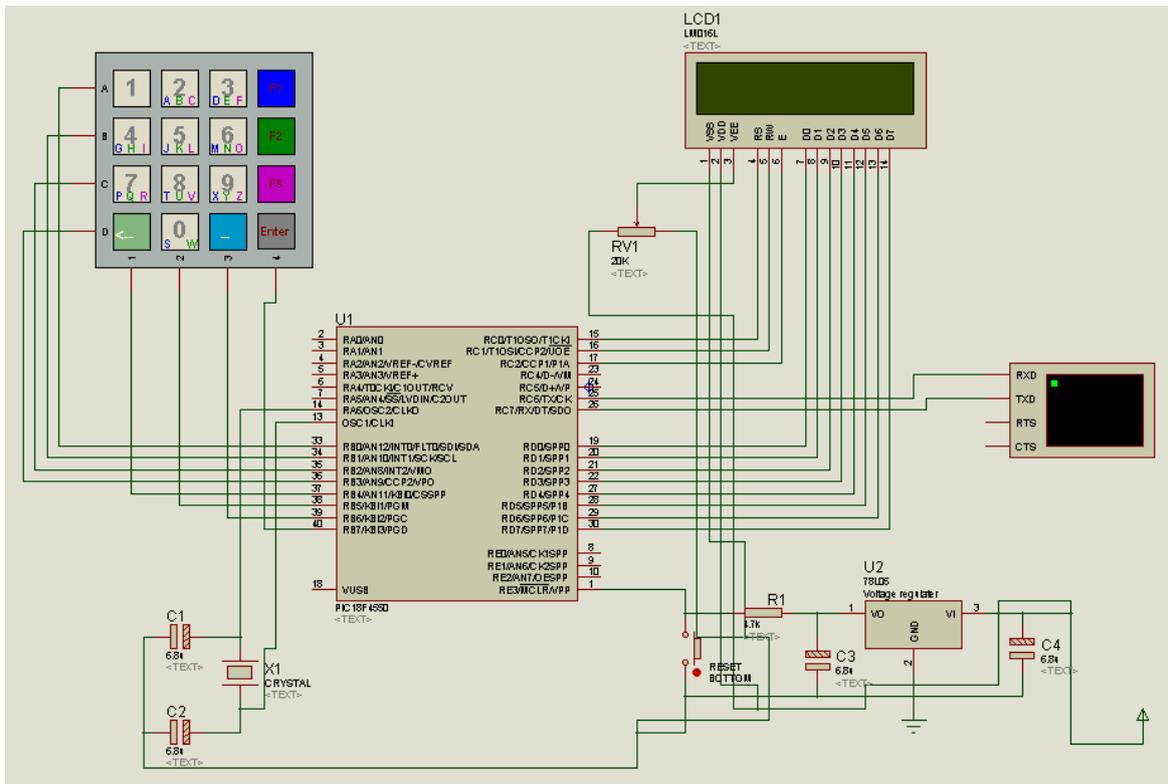
The final circuit on PCB board is show in the figure bellow (see layout on Appendix B):
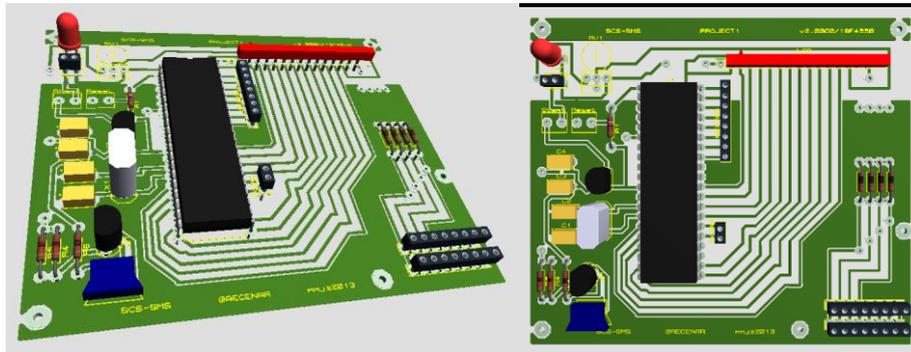
7.4 PIC software

As we see in the section before, we will write our program using C language. So, we should install the MCC18 C compiler for our MPLAB software.

In our program, some (.h) libraries which will be use in the program should be include on the head of our program

The (.h) libraries we include it is:

P18f4550.h,   stdio.h,   dalays.h,   string.h

And this is doing by this C code:

```
#include<p18f4550.h>
#include<stdio.h>
```

PIC software

```
#include<delays.h>
#include<string.h>
```

Now, let's start with our program. The program is containing some special functions a side to the main function and the initialization functions.

The special functions are:

```
void INIT_PORT(void);
void InitSerial(void);
char Encryption(char PC);
char Decryption(char CC);
void SendToSerial(char m);
void SendStringToSerial(char msg[]);
char ReceiveFromSerial(void);
void LCD_CMD(unsigned int value);
void WRITE_CHAR_LCD(unsigned char value);
void WRITE_STRING_LCD(char value[]);
char KeyPad(void);
void on_touch(void);
void send_SMS(void);
void send_data(void);
void Menu(void);
```

INIT_PORT function

The goal of this function is to initialize the input output port/pin for the microcontroller by setting the **TRIS** for each port or pin used. This function also clears all port from any past setting. This function has no return no calling input.

C code:

```
void INIT_PORT(void)
{
    ADCON1=0x0E; //for using analog port RA0
    TRISAbits.TRISA0 = 1; //input pin
    TRISAbits.TRISA1 = 1; //input pin
    PORTA =0x00;
    TRISB =0b11110000;
    PORTB =0x00;
    TRISCbits.TRISC0 = 0; //output pin for LCD RS
    TRISCbits.TRISC1 = 0; //output pin for LCD RW
    TRISCbits.TRISC2 = 0; //output pin for LCD E
    TRISCbits.TRISC6 = 0; //output pin
    TRISCbits.TRISC7 = 1; //input pin
```

```
    PORTC =0x00;
    TRISD =0x00; //output port 00000000
    PORTD =0x00;
}
```

InitSerial function

To communicate with external components such as computers or microcontrollers, the PIC microcontroller uses a component called **USART - Universal Synchronous Asynchronous Receiver Transmitter.** This component can be configured as:

- A Full-Duplex asynchronous system that can communicate with peripheral devices, such as CRT terminals and personal computers

- A Half-Duplex synchronous system that can communicate with peripheral devices, such as A/D or D/A integrated circuits, serial EEPROMs, etc.

To enable the serial communication with PIC microcontroller we must set different parameters within two registers:

1. TXSTA - Transmit Status and Control Register

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0
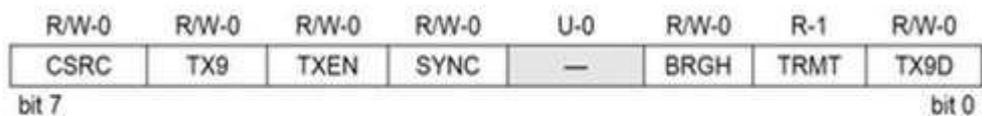
MicrocontrollerBoard.com

2. RCSTA - Receive Status and Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0
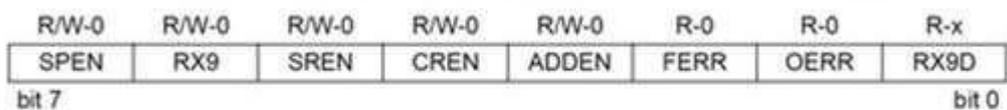
MicrocontrollerBoard.com

The send information will be stored inside **TXREG** register, which acts as a temporary buffer storage of information prior to transmission. While the receive information will be store in the **RCSTA** register, which acts as a temporary buffer storage.

Each transmission is transmitted in the particular rate (BAUD). The baud rate is measured in units of **bps** (bit per second). This is done by setting the system clock to the value needed. To do so, we need to "write" a hexadecimal number to the **SPBRG** register. The value written to the **SPBRG** register set the clock cycle to the value we want for the BAUD rate.

The size of **SPBRG** register is 8-bit. In asynchronous mode, the baud rate of transmission of the information can be set to high speed or to low speed. The rate selection, as already seen, is made by the BRGH bit in TXSTA register:

1 = High speed   0 = Low speed

For each baud rate we need to calculate the value being placed in the SPBRG differently:

SPBRG = (Fosc / (16 x Baud rate)) - 1,   BRGH = 1 High Speed

SPBRG = (Fosc / (64 x Baud rate)) - 1,   BRGH = 0 Low Speed

In our case, we have: Fosc=48 Mhz,   Baud rate=9600

For High Speed =>   SPBRG = 0x137   hex

For Low Speed =>   SPBRG = 0x4D   hex

After the calculation of this tree register we can set it by this function as follow, as we see this function also have no return no calling input

C code:

```c
void InitSerial(void)
{
    SPBRG = 0x4D;       // 4D hex or 77 decimal (baud rate=9600), Low
speed: SPBRG = (Fosc / (64 x Baud rate)) - 1
    TXSTA = 0x22;       // determinng the setting for the transmitter
    RCSTA= 0x90; //determining the setting for the receiver
}
```

SendToSerial function

this function receive a character as calling input and put it into the buffer of the transmission register to be send and a while loop used with the **TRMT** register to wait until message was sent. No return for this function, it has only char as calling input.

C code:

```c
void SendToSerial(char m)
{
    TXREG = m;
    while(TXSTAbits.TRMT==0){}
}
```

SendStringToSerial function

SCS-SMS

This function is to send a more than one character to serial by only one command. This function has no return but it need a string array from the caller. (@) is specified for **ENTER** meaning.

C code:

```c
void SendStringToSerial(char msg[])
{
    int i, lenght;
    lenght = strlen(msg);
    for(i=0; i <= lenght; i++){
        if(msg[i]=='@') break;
        SendToSerial(msg[i]);
    }
}
```

ReceiveFromSerial function

This function is to receive the data send from other and a while loop used with the **RCIF** register to wait until message was received. No calling input for this function but it return the receiving data for the caller.

C code:

```c
char ReceiveFromSerial(void)
{
    PORTAbits.RA4 = 0;  // reset the alert when data received
    while(PIR1bits.RCIF==0); // Wait until RCIF gets low
    return RCREG; // Retrieve data from reception register
}
```

LCD_CMD function

The goal of this function is to send a command for the LCD. To send a command for the LCD, after the enabling of **E** register you should clear the **RS** and **RW** register. When you clear this two register the LCD know that the data will be receive it is a command. After sending the hexadecimal number of the command the enable register **E** should be disabling again.

Commands Hexadecimal code:

0x01	CLEAR DISPLAY SCREEN
0x02	RETURN HOME
0x04	SHIFT CURSER TO LEFT
0x06	SHIFT CURSER TO RIGHT
0x05	SHIFT DISPLAY TO RIGHT
0x07	SHIFT DISPLAY TO LEFT
0x0C	CURSER OFF

0x0E	CURSER BLINKING
0x10	SHIFT CURSOR POSITION TO LEFT
0x14	SHIFT CURSOR POSITION TO RIGHT
0x18	SHIFT THE ENTIRE DISPLAY TO LEF
0x1C	SHIFT THE ENTIRE DISPLAY TO RIGHT
0x80	FORCE CURSOR TO BEGINNING OF 1ST LINE Or you can start from where you want 0x8X ex: 0x83
0xC0	FORCE CURSOR TO BIGINNING OF 2ND LINE Or you can start from where you want 0xCX ex: 0xC7
0x38	TO WRITE ON THE TWO LINES

A delay should be executing to harmony the speed of the LCD and PIC; PIC will wait the LCD to be ready to receive a new order. No return for this function but it receives an integer of the command value from the caller.

C code:

```c
void LCD_CMD(unsigned int value)
{
    PORTCbits.RC2=1; //E
    PORTCbits.RC0=0; //RS
    PORTCbits.RC1=0; //RW
    Delay10KTCYx(15);
    PORTD=value;
    Delay10KTCYx(15);
    PORTCbits.RC2=0; //E
    Delay10KTCYx(30);
}
```

WRITE_CHAR_LCD function

As the LCD_CMD function, the goal of this function is to send a data to the LCD to write it. To write on LCD, after the enabling of **E** register you should set the **RS** register and clear the **RW** register. When you do that the LCD know that the data will be receive it should be write on it. Now we can send the data to be written on LCD then the enable register **E** should be disabling again.

Also, a delay should be executing to harmony the speed of the LCD and PIC; PIC will wait the LCD to be ready to receive a new data. No return for this function but it receives the character from the caller.

C code:

```c
void WRITE_CHAR_LCD(unsigned char value)
{
            PORTCbits.RC2=1;   //E
            PORTCbits.RC0=1;   //RS
```

```
                PORTCbits.RC1=0;    //RW
                Delay1KTCYx(5);
                PORTD=value;
                Delay1KTCYx(5);
                PORTCbits.RC2=0;    //E
}
```

WRITE_STRING_LCD function

This function is to send more than one character (a string) to LCD by only one command. This function has no return but it need a string array from the caller.

C code:

```
void WRITE_STRING_LCD(char value[])
{
                int i, lenght;
                lenght = strlen(value) - 1;
                for(i=0; i <= lenght; i++){
                 WRITE_CHAR_LCD(value[i]);
                }
                //Delay10KTCYx(10);
}
```

KeyPad function

The goal of this function is to get a key pressed by the user. We discussed in the previous chapter how the keypad it works. In this function we write the IF condition which will specific the meaning of each pressed key.

In out project we specify that the normal way of the keypad is to write number with enter and space and we specify a three functional key (f1, f2, f3) to change from the normal way to one of the three functional way which will write the English alphabetic instead of numbers. Figure bellow shows the keypad key specification.
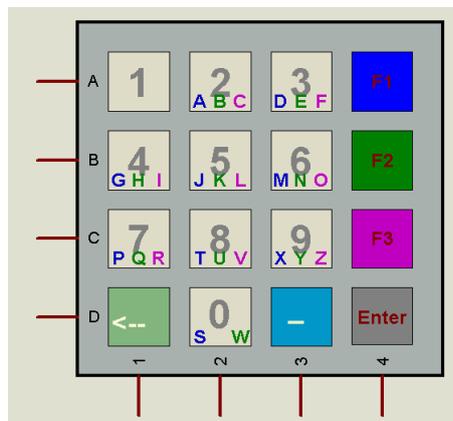
This function has no receiving input from the caller but it returns the pressed character for the caller

C code:

PIC software

```c
char KeyPad(void)
{
                char msg='*';    // to be sure that a key was pressed
                PORTB =0x00;

                msg = KeyPressed;
                if(msg != '*')
                {
                 KeyPressed = '*';
                 return msg;
                }

waiting:

                PORTBbits.RB0=1;
                if(PORTBbits.RB4==1)return('1');
                else if(PORTBbits.RB5==1)return '4';
                else if(PORTBbits.RB6==1)return '7';
                else if(PORTBbits.RB7==1)return '<';

                PORTB =0x00;
                PORTBbits.RB1=1;
                if(PORTBbits.RB4==1)return '2';
                else if(PORTBbits.RB5==1)return '5';
                else if(PORTBbits.RB6==1)return '8';
                else if(PORTBbits.RB7==1)return '0';

                PORTB =0x00;
                PORTBbits.RB2=1;
                if(PORTBbits.RB4==1)return '3';
                else if(PORTBbits.RB5==1)return '6';
                else if(PORTBbits.RB6==1)return '9';
                else if(PORTBbits.RB7==1)return ' ';

                PORTB =0x00;
                PORTBbits.RB3=1;
                if(PORTBbits.RB4==1)       // F1
                {
                 PORTB =0x00;
                 msg = '*';    // to be sure that a key was pressed
f1:
                 PORTBbits.RB0=1;
                 if(PORTBbits.RB5==1)return 'G';
                 else if(PORTBbits.RB6==1)return 'P';

                 PORTB =0x00;
                 PORTBbits.RB1=1;
                 if(PORTBbits.RB4==1)return 'A';
                 else if(PORTBbits.RB5==1)return 'J';
                 else if(PORTBbits.RB6==1)return'T';
                 else if(PORTBbits.RB7==1)return'S';

                 PORTB =0x00;
                 PORTBbits.RB2=1;
                 if(PORTBbits.RB4==1)return 'D';
                 else if(PORTBbits.RB5==1)return 'M';
                 else if(PORTBbits.RB6==1)return 'X';

                 PORTB =0x00;
                 if(msg=='*') goto f1;
                }
```

```
                  else if(PORTBbits.RB5==1)        // F2
                  {
                   PORTB =0x00;
                   msg = '*';   // to be sure that a key was pressed
f2:
                   PORTBbits.RB0=1;
                   if(PORTBbits.RB5==1)return 'H';
                   else if(PORTBbits.RB6==1)return 'Q';

                   PORTB =0x00;
                   PORTBbits.RB1=1;
                   if(PORTBbits.RB4==1)return 'B';
                   else if(PORTBbits.RB5==1)return 'K';
                   else if(PORTBbits.RB6==1)return'U';
                   else if(PORTBbits.RB7==1)return'W';

                   PORTB =0x00;
                   PORTBbits.RB2=1;
                   if(PORTBbits.RB4==1)return 'E';
                   else if(PORTBbits.RB5==1)return 'N';
                   else if(PORTBbits.RB6==1)return 'Y';

                   PORTB =0x00;
                   if(msg=='*') goto f2;
                  }
                  else if(PORTBbits.RB6==1)      // F3
                  {
                   PORTB =0x00;
                   msg = '*';   // to be sure that a key was pressed
f3:
                   PORTBbits.RB0=1;
                   if(PORTBbits.RB5==1)return 'I';
                   else if(PORTBbits.RB6==1)return 'R';

                   PORTB =0x00;
                   PORTBbits.RB1=1;
                   if(PORTBbits.RB4==1)return 'C';
                   else if(PORTBbits.RB5==1)return 'L';
                   else if(PORTBbits.RB6==1)return'V';

                   PORTB =0x00;
                   PORTBbits.RB2=1;
                   if(PORTBbits.RB4==1)return 'F';
                   else if(PORTBbits.RB5==1)return 'O';
                   else if(PORTBbits.RB6==1)return 'Z';
                   else if(PORTBbits.RB7==1)return '^';

                   PORTB =0x00;
                   if(msg=='*') goto f3;
                  }

                  else if(PORTBbits.RB7==1)return '@';

                  PORTB =0x00;

                  if(msg=='*') goto waiting;

                  return msg;

}
```

send_SMS function

The goal of this function is to wait the user to write a SMS to be sending via serial. (@) is specified for **ENTER** meaning.

C code:

```c
void send_SMS(void)
{
                int i=0;
                unsigned char SMS[16];
                SMS[0]='*';

Loop:
                SMS[i]=KeyPad();

                if(SMS[i]=='<')
                {
                 if(i!=0)
                  {
                        i--;
                        LCD_CMD(0x10);  //shift  cursor  position  to
left
                  }
                 goto Loop;
                }
                else if(SMS[i]=='@' && i!=0)
                {
                 SendCyphierToSerial(SMS);
                 SMS[i]='*';
                 LCD_CMD(0xC0); //Second Line
                 goto Loop;
                }
                else if(SMS[i]!='^')
                {
                 WRITE_CHAR_LCD(SMS[i]);

                 Delay10KTCYx(20);

                 if(i==15)
                  {
                        SendCyphierToSerial(SMS);
                        LCD_CMD(0xC0); //Second Line
                        i=0;
                  }
                 else i++;
                 goto Loop;
                }
                else SMS[i]='*';
}
```

on_touch function

This function is staying on receiving mode till the user press MENU key.

C code:

```c
void on_touch(void)
{
                int i=0;
```

```
                     unsigned char waiting[15]="WAITING SMS...";
                     char rc;
                     LCD_CMD(0x01); //CLEAR SCREEN
                     LCD_CMD(0x80); //FIRST LINE
                     WRITE_STRING_LCD(waiting);

                     LCD_CMD(0xC0); //second LINE

                     while(1)
                     {
/*
                      LCD_CMD(0xC0); //second LINE

                      ReceiveCypherText();
                      Delay10KTCYx(50);

                      LCD_CMD(0x01); //Clear screen
*/
                      rc = ReceiveFromSerial();
                      WRITE_CHAR_LCD(rc);
                      i++;
                      if(i==16)
                      {
                           LCD_CMD(0x01); //Clear screen
                           LCD_CMD(0xC0); //second LINE
                      }
                     }
}
```

send_data function

This function ask the user to choice how he want to send his SMS, to specific one of for all by BROADCAST.

C code:

```
void send_data(void)
{
                     unsigned char broadcast[12]="4 broadcast";
                     unsigned char to_one[9]="5 to one";
                     unsigned char writeM[11]="write SMS:";
                     unsigned char EntID[10]="Enter ID:";
                     unsigned char GoOut[7]="Go Out";
                     char Key_Pressed;
                     int i=0;

screen2:
                     LCD_CMD(0x01); //CLEAR SCREEN
                     LCD_CMD(0x80); //FIRST LINE
                     WRITE_STRING_LCD(broadcast);
                     LCD_CMD(0xC0); //second LINE
                     WRITE_STRING_LCD(to_one);

                     Key_Pressed = KeyPad();
                     if(Key_Pressed=='4')
                     {
                      // broadcast choice
                      LCD_CMD(0x01); //CLEAR SCREEN
                      LCD_CMD(0x80); //FIRST LINE
                      WRITE_STRING_LCD(writeM);
                      LCD_CMD(0xC0); //second LINE
```

PIC software

```
                while(1)
                {
                        send_SMS();
                        Key_Pressed = KeyPad();
                        if(Key_Pressed == '@') break;
                        else
                        {
                                LCD_CMD(0xC0); //Second Line
                                for(i=16;  i>0;  i--)  WRITE_CHAR_LCD('
');
                                LCD_CMD(0xC0); //Second Line
                                send_SMS();
                        }
                }
        }
        else if(Key_Pressed=='5')
        {
         // to one choice
         LCD_CMD(0x01); //CLEAR SCREEN
         LCD_CMD(0x80); //FIRST LINE
         WRITE_STRING_LCD(EntID);
         LCD_CMD(0xC0); //second LINE
         Key_Pressed = KeyPad();
         if(Key_Pressed=='M')
         {
                LCD_CMD(0x01); //CLEAR SCREEN
                LCD_CMD(0x80); //FIRST LINE
                WRITE_STRING_LCD(writeM);
                LCD_CMD(0xC0); //second LINE
                while(KeyPad()!='@')
                {
                        send_SMS();
                }
         }
         else
         {
                WRITE_STRING_LCD(GoOut);
                Delay10KTCYx(50);
         }
        }
    }
}
```

Menu function

This function is the MENU which the user choice in it if he want to stay on touch for any coming SMS or if he want to send SMS. This function have no input (caller input) no output (return).

C code:

```
void Menu(void)
{
                unsigned char OnTouch[11]="1 on touch";
                unsigned char SendData[12]="2 send data";
                unsigned char wrong[8]="UnValid";
                unsigned char choice[7]="choice";
                char Key_Pressed;

screen1:
                LCD_CMD(0x01); //CLEAR SCREEN
```

```
                LCD_CMD(0x80); //FIRST LINE
                WRITE_STRING_LCD(OnTouch);
                LCD_CMD(0xC0); //second LINE
                WRITE_STRING_LCD(SendData);

                Key_Pressed = KeyPad();
                if(Key_Pressed=='1')
                {
                 // on touch choice
                 on_touch();
                }
                else if(Key_Pressed=='2')
                {
                 // send data choice
                 send_data();
                }
                else
                {
                 LCD_CMD(0x01); //CLEAR SCREEN
                 LCD_CMD(0x84); //FIRST LINE specific place
                 WRITE_STRING_LCD(wrong);
                 LCD_CMD(0xC4); //second LINE specific place
                 WRITE_STRING_LCD(choice);
                }
}
```

Encryption function

This function is to encrypt message before sending it. This will be encrypting via AES encrypt theorem in the next part.

C code:

```
char Encryption(char PC)
{
    int n=0;
    n = ((int)PC) + 1;
    return ((char)n);
}
```

Decryption function

This function is to decrypt received message before display on LCD. This will be decrypting via AES encrypt theorem in the next part.

C code:

```
char Decryption(char CC)
{
    int n=0;
    n = ((int)CC) - 1;
    return ((char)n);
```

Test

```
}
```

Main function

Main function of the program

C code:

```
void main()
{
                unsigned char welcometo[11]="welcome to";
                unsigned char SCS[8]="GIS-ECS";

                PORTAbits.RA4 = 1;

                INIT_Interrupt();
                INIT_PORT();
                InitSerial();

                LCD_CMD(0x01);
                LCD_CMD(0x0E);
                LCD_CMD(0x38);

                Delay10KTCYx(100); // wait for 500ms

                PORTAbits.RA4 = 0;

                LCD_CMD(0x83); //FIRST LINE
                WRITE_STRING_LCD(welcometo);
                LCD_CMD(0xC4); //second LINE
                WRITE_STRING_LCD(SCS);
                Delay10KTCYx(10);

begin:

                Menu();

                goto begin;
}
```

7.5 Test

Before build the hardware system we should test the system on any simulation software. We choice the Proteus ISIS simulation to do the system testing

On Proteus:

In the simulation software Proteus ISIS professional the system work as follows:

- After connecting components together a virtual terminal should be add in place of the second side to imagine the interaction between two systems.

- When you press the start button, the LCD display the welcome screen (see the figure bellow)
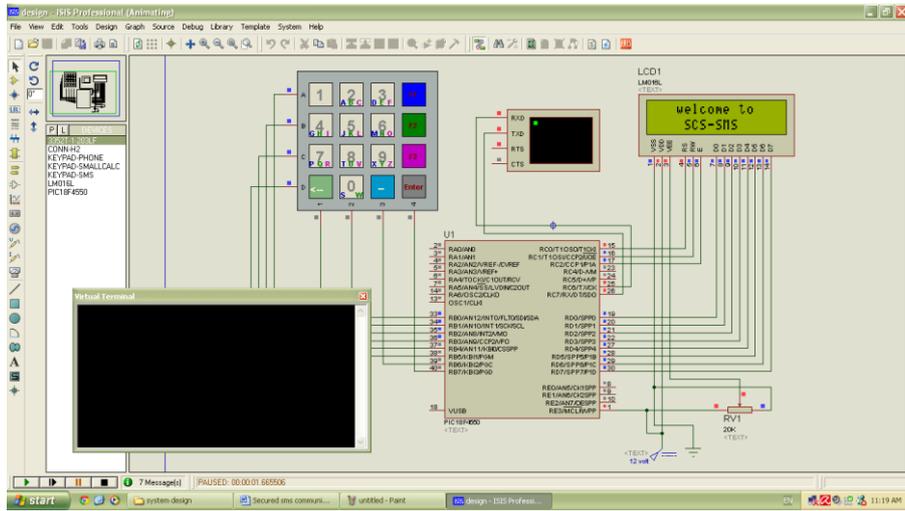
- Then the system work as follows:
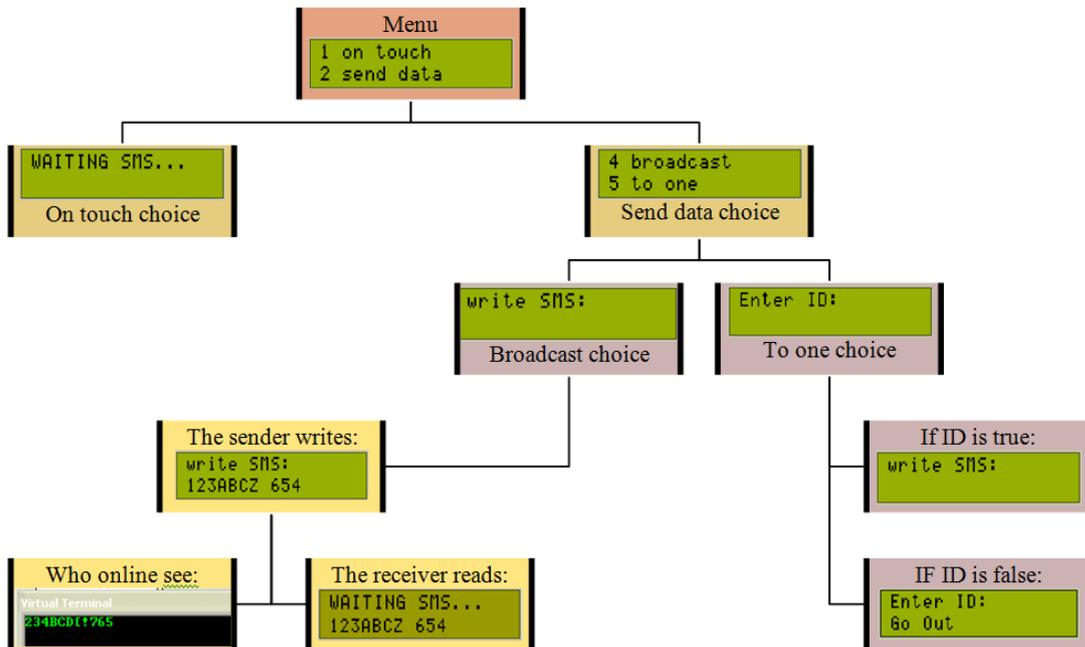
- Also, we test the sender\receiver side together with the following design:
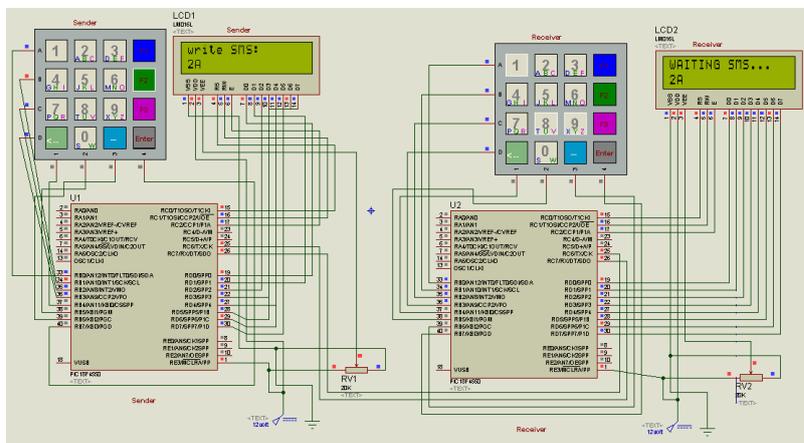
Test

On Real Hardware:
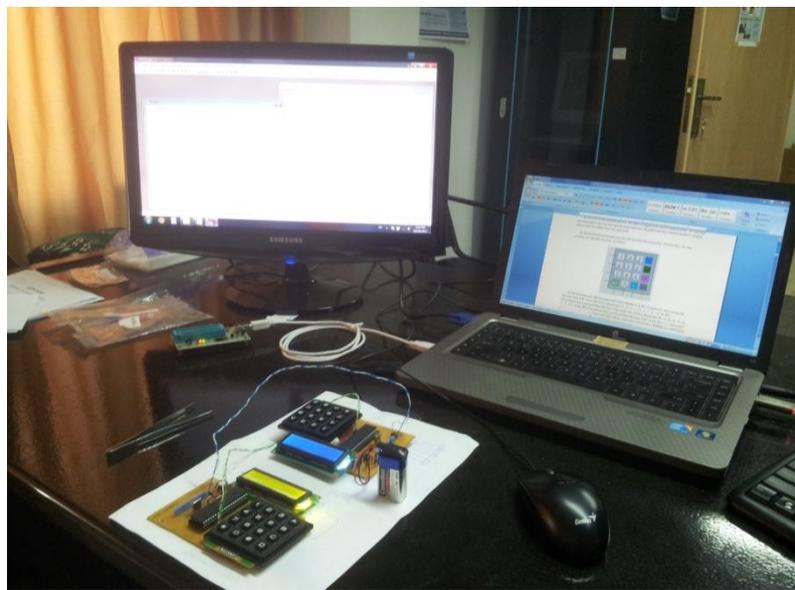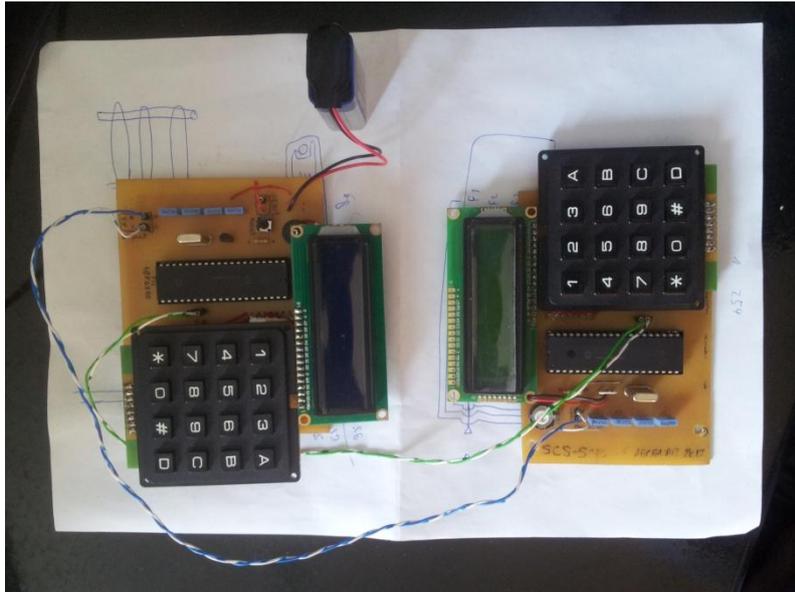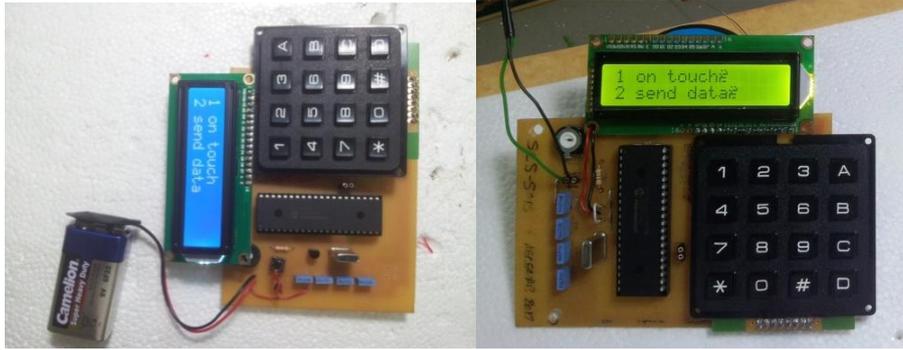
8   AES encryption

Advanced Encryption Standard

One of the most widely used block cipher algorithms is the **Data Encryption Standard** (**DES**), adopted in 1977 by the American National Standards Institute (ANSI).

After more than twenty years of use with continuous aging due to advances in cryptography, the National Institute of Standards and Technology (NIST). On 2 October 2000 the NIST announced that the new encryption technique, named **Advanced Encryption Standard** (**AES**), would use the <u>Rijndael</u> algorithm, designed by two well-known specialists, Joan Daemen and Vincent Rijmen from Belgium.

8.1 Introduction

AES is based on a design principle known as a substitution-permutation network, and is fast in both software and hardware. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification *per se* is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4×4 column-major order matrix of bytes, termed the *state*, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher-text. The number of cycles of repetition is as follows:

* 10 cycles of repetition for 128-bit keys.
* 12 cycles of repetition for 192-bit keys.
* 14 cycles of repetition for 256-bit keys.

Each round consists of several processing steps, each containing five similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform cipher-text back into the original plaintext using the same encryption key.

8.2 AES Algorithm[8]

i.   KeyExpansion: round keys are derived from the cipher key using Rijndael's key schedule.
ii. InitialRound
    a. `AddRoundKey`—each byte of the state is combined with the round key using bitwise **xor**.
iii. Rounds
   a. `SubBytes`—a non-linear substitution step where each byte is replaced with another according to a lookup table.

---

[8] <u>Reference</u>: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

      b. `ShiftRows`—a transposition step where each row of the state is shifted cyclically a certain number of steps.

      c. `MixColumns`—a mixing operation which operates on the columns of the state, combining the four bytes in each column.

      d. `AddRoundKey`

  iv.   Final Round (no `MixColumns`)

      a. `SubBytes`

      b. `ShiftRows`

      c. `AddRoundKey`

A. The SubBytes step

In the SubBytes step, each byte in the state matrix is replaced with a SubByte using an 8-bit substitution box, the Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.
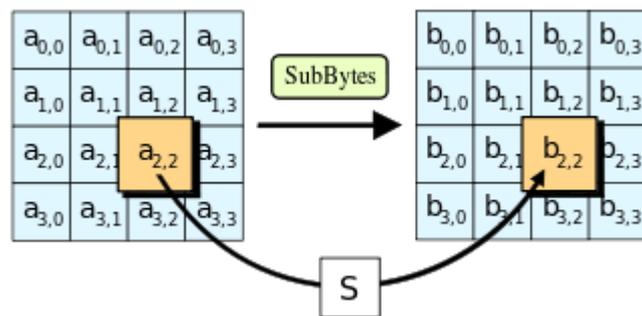
B. The ShiftRows step

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row n is shifted left circular by n-1 bytes. In this way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 2 bytes

and 3 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to make columns not linear independent if so, AES becomes four independent block ciphers.
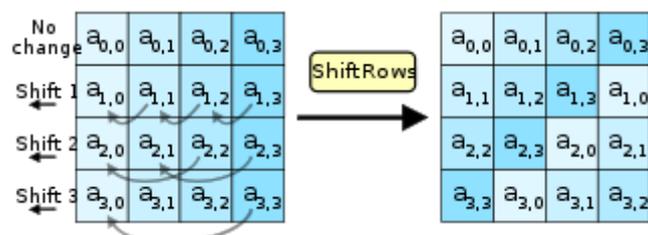
C. The MixColumns step
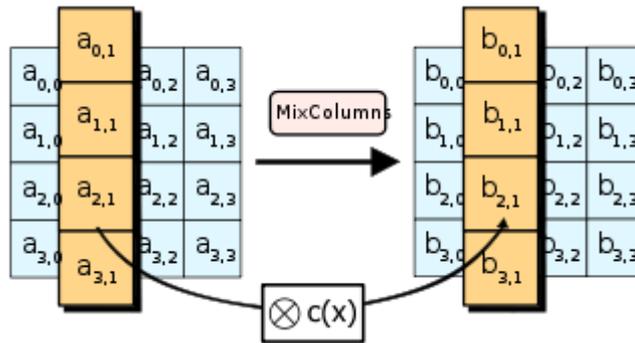
In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. TheMixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher.

During this operation, each column is multiplied by the known matrix that for the 128-bit key is:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

The multiplication operation is defined as: multiplication by 1 means no change, multiplication by 2 means shifting to the left, and multiplication by 3 means shifting to the left and then performing XOR with the initial unshifted value. After shifting, a conditional xor with 0x1B should be performed if the shifted value is larger than 0xFF.

In more general sense, each column is treated as a polynomial over GF(28) and is then multiplied modulo x4+1 with a fixed polynomial c(x) = 0x03 · x3 + x2 + x + 0x02. The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from GF(2)[x]. The MixColumns step can also be viewed as a multiplication by a particular MDS matrix in a finite field. This process is described further in the article Rijndael mix columns.

D. The AddRoundKey step

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main key usingRijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.
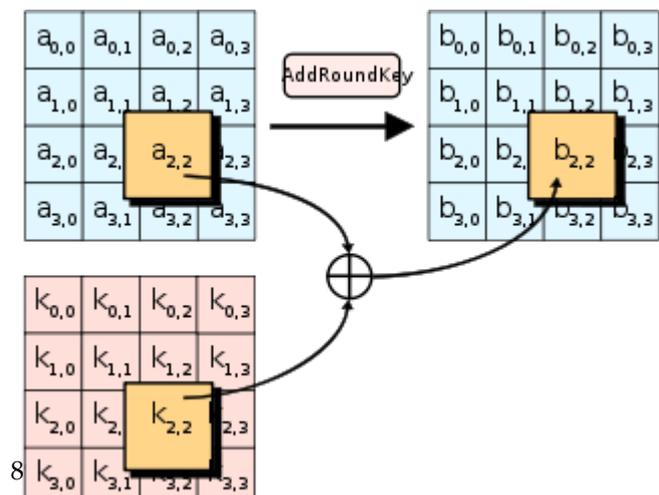
8.3 Coding [9]

AES is a symmetric key block cipher algorithm. The algorithm executes a series of rounds. The intermediate results of the rounds over the block are called states. In order to prepare for the round transformations, a "`KeyExpansion`" operation must be executed. This operation uses the original key to create several round keys. Each round key, including the original one, will be used in one of the rounds.

This operation is performed by this C code:

```c
void KeyExpansion()
{
    unsigned char i,j;
    unsigned char temp[4],k;

    for(i=0; i<4; i++)
    {
        RoundKey[i*4]=Key[i*4];
        RoundKey[i*4+1]=Key[i*4+1];
        RoundKey[i*4+2]=Key[i*4+2];
        RoundKey[i*4+3]=Key[i*4+3];
    }

    while (i < (4 * (11)))
    {
        for(j=0;j<4;j++)
        {
            temp[j]=RoundKey[(i-1) * 4 + j];
        }
        if (i % 4 == 0)
        {
                    k = temp[0];
                    temp[0] = temp[1];
                    temp[1] = temp[2];
                    temp[2] = temp[3];
                    temp[3] = k;

                    temp[0]=sbox[temp[0]];
                    temp[1]=sbox[temp[1]];
                    temp[2]=sbox[temp[2]];
                    temp[3]=sbox[temp[3]];
```

---

[9] <u>Reference</u>: Microchip AN821 Advanced Encryption Standard Using the PIC16XXX

Coding

```
                    temp[0] =  temp[0] ^ Rcon[i/4];
        }
        else if (4 > 6 && i % 4 == 4)
        {
                    temp[0]=sbox[temp[0]];
                    temp[1]=sbox[temp[1]];
                    temp[2]=sbox[temp[2]];
                    temp[3]=sbox[temp[3]];
        }
        RoundKey[i*4+0] = RoundKey[(i-4)*4+0] ^ temp[0];
        RoundKey[i*4+1] = RoundKey[(i-4)*4+1] ^ temp[1];
        RoundKey[i*4+2] = RoundKey[(i-4)*4+2] ^ temp[2];
        RoundKey[i*4+3] = RoundKey[(i-4)*4+3] ^ temp[3];
        i++;
    }
}
```

Where `Rcon` represent a vector of round constant, `RoundKey` represent the array that stores the round keys, `Key` is the encryption key, and **sbox** is the encryption substitution table and later we will see **rsbox** which is the decryption substation table.

These variables are saved in the ROM of the PIC under this code:

```
const rom unsigned char sbox[256] =    {
   //0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
   0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0
   0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1
   0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2
   0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3
   0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4
   0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5
   0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6
   0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7
   0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8
   0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9
   0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A
   0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B
   0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C
   0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D
   0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E
   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F


const rom unsigned char Rcon[10] = {
   0x36, 0x1B, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};


const rom unsigned char rsbox[256] = {
   0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
   0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
   0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
   0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
   0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
   0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
   0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
   0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
   0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
   0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
   0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
   0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
   0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
   0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
```

AES encryption

```
      0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
      0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };

unsigned char out[16], state[4][4];

unsigned char RoundKey[160];

unsigned char Key[16]="YOUR_SECURE_KEY";
```

In the encryption process, each of the ten rounds (with the exception of the last one) is composed of four stages:

• byte_sub

```c
void substitution_s()
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[i][j] = sbox[state[i][j]];

        }
    }
}
```

• shift_row

```c
void enc_shift_row()
{
    unsigned char temp;

    // Rotate first row 1 columns to left
    temp=state[1][0];
    state[1][0]=state[1][1];
    state[1][1]=state[1][2];
    state[1][2]=state[1][3];
    state[1][3]=temp;

    // Rotate second row 2 columns to left
    temp=state[2][0];
    state[2][0]=state[2][2];
```

Coding

```
    state[2][2]=temp;

    temp=state[2][1];
    state[2][1]=state[2][3];
    state[2][3]=temp;

    // Rotate third row 3 columns to left
    temp=state[3][0];
    state[3][0]=state[3][3];
    state[3][3]=state[3][2];
    state[3][2]=state[3][1];
    state[3][1]=temp;
}
```

- mix_column

```
void mix_column()
{
              int i;
              unsigned char Tmp,Mem;
              for(i=0;i<4;i++)
              {
               Mem = state[0][i];
               Tmp = state[0][i] ^ state[1][i] ^ state[2][i]
^ state[3][i];
               state[0][i]  ^=  Tmp  ^  xtime(state[0][i]  ^
state[1][i]);
               state[1][i]  ^=  Tmp  ^  xtime(state[1][i]  ^
state[2][i]);
               state[2][i]  ^=  Tmp  ^  xtime(state[2][i]  ^
state[3][i]);
               state[3][i] ^= Tmp ^ xtime(state[3][i] ^ Mem);
              }
}
```

While the xtime function is done under this process:

```
char xtime(char x)
{
              if(x < 0x80)
               return (x <<= 1);
              else
               return (x = (x << 1) ^ 0x1b);
}
```

93

AES encryption

- `key_addition`

```
void key_addition(unsigned char round)
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[j][i] ^= RoundKey[round * 16 + i * 4 + j];
        }
    }
}
```

In the decryption process, each of the ten rounds (with the exception of the first one) is composed of four stages:

- `inv_byte_sub`

```
void substitution_si()
{
    unsigned char i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[i][j] = rsbox[state[i][j]];
        }
    }
}
```

- `inv_mix_column`

```
void inv_mix_column()
{
    unsigned char i;
            unsigned char Tmp0, Tmp1, Tmp2, Tmp3;

            for(i=0;i<4;i++)
            {
             Tmp0 = state[0][i] ^ state[1][i] ^ state[2][i]
^ state[3][i];
             Tmp1 = xtime(state[0][i] ^ state[2][i]);
             Tmp2 = xtime(state[1][i] ^ state[3][i]);
             Tmp3 = xtime( xtime( Tmp1 ^ Tmp2 )) ^ Tmp0;
```

Coding

```
        state[0][i]  ^= xtime(state[0][i]^ state[1][i]  ^  Tmp1)
^  Tmp3;
        state[1][i]  ^= xtime(state[1][i]^ state[2][i]  ^  Tmp2)
^  Tmp3;
        state[2][i]  ^= xtime(state[2][i]^ state[3][i]  ^  Tmp1)
^  Tmp3;
        state[3][i]  = state[0][i]  ^  state[1][i]  ^  state[2][i]
^  Tmp0;

            }
}
```

• dec_shift_row

```
void dec_shift_row()
{
    unsigned char temp;

    // Rotate first row 1 columns to right
    temp=state[1][3];
    state[1][3]=state[1][2];
    state[1][2]=state[1][1];
    state[1][1]=state[1][0];
    state[1][0]=temp;

    // Rotate second row 2 columns to right
    temp=state[2][0];
    state[2][0]=state[2][2];
    state[2][2]=temp;
    temp=state[2][1];
    state[2][1]=state[2][3];
    state[2][3]=temp;

    // Rotate third row 3 columns to right
    temp=state[3][0];
    state[3][0]=state[3][1];
    state[3][1]=state[3][2];
    state[3][2]=state[3][3];
    state[3][3]=temp;
}
```

The original key schedule functions use several RAM positions, in order to save all round keys used in the encryption/decryption process.

To reduce the RAM consumption, the implementation of the round keys was done on-the-fly. To do this, three different functions were added:

AES encryption

1. `enc_key_schedule (key)`: This function takes the actual key and generates the next round key that is placed in the same RAM positions.

C code

```c
void enc_key_schedule()
{
            char Rcon = 0x01;
            Key[0] ^= sbox[13];
            Key[1] ^= sbox[14];
            Key[2] ^= sbox[15];
            Key[3] ^= sbox[12];
            Key[0] = Key[0] ^ Rcon;
            Rcon = xtime(Rcon);
            Key[4] ^= Key[0];
            Key[5] ^= Key[1];
            Key[6] ^= Key[2];
            Key[7] ^= Key[3];
            Key[8] ^= Key[4];
            Key[9] ^= Key[5];
            Key[10] ^= Key[6];
            Key[11] ^= Key[7];
            Key[12] ^= Key[8];
            Key[13] ^= Key[9];
            Key[14] ^= Key[10];
            Key[15] ^= Key[11];
}
```

2. `dec_key_schedule(key)`: This function takes the actual key and generates the previous round key that is placed in the same RAM positions.

C code

```c
void dec_key_schedule()
{
            char Rcon = 0x01;
            Key[12] ^= Key[8];
            Key[13] ^= Key[9];
            Key[14] ^= Key[10];
            Key[15] ^= Key[11];
            Key[8] ^= Key[4];
            Key[9] ^= Key[5];
            Key[10] ^= Key[6];
            Key[11] ^= Key[7];
            Key[4] ^= Key[0];
            Key[5] ^= Key[1];
```

Coding

```
        Key[6] ^= Key[2];
        Key[7] ^= Key[3];
        Key[0] ^= sbox[Key[13]];
        Key[5] ^= sbox[Key[14]];
        Key[6] ^= sbox[Key[15]];
        Key[7] ^= sbox[Key[12]];
        Key[0] = Key[0] ^ Rcon;

        if(Rcon & 0x01)
         Rcon = 0x80;
        else
         Rcon >>1;
}
```

AES encryption

Code flow chart

A. Encryption flow chart
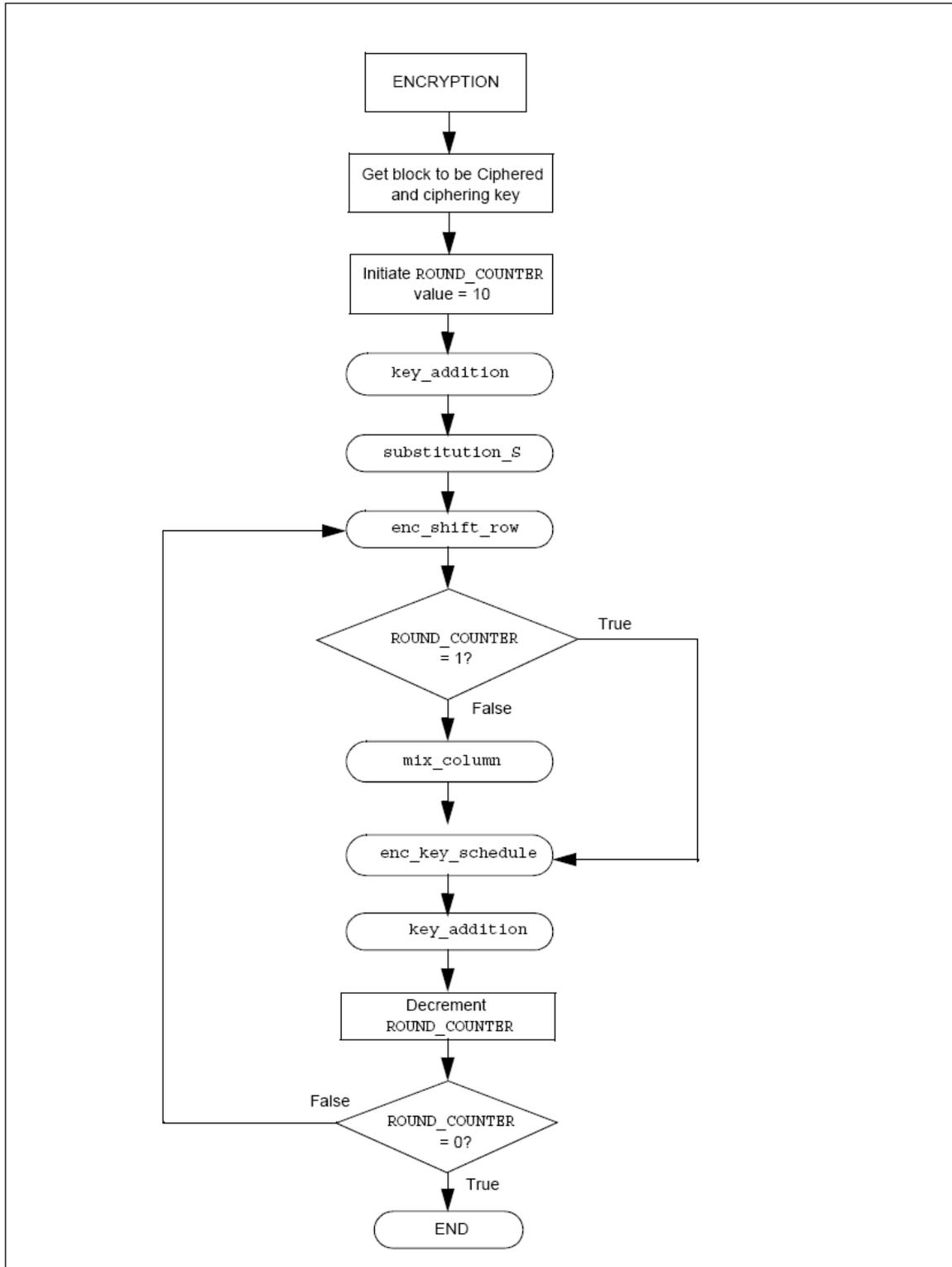
Coding

The structure of the encryption program is:

```
void encrypts(char in[])
{
    unsigned char i,j,round;

            KeyExpansion();

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[j][i] = in[i*4 + j];
        }
    }

    key_addition(0);

    for(round=1;round<=10;round++)    // 10 rounds
            {
             substitution_s();
             enc_shift_row();
             if( round != 10 ) // last round is done
without mix_column
                   mix_column();
             enc_key_schedule();  //  direct  key_schedule
executed on-the-fly
             key_addition(round);
            }

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            out[i*4+j]=state[j][i];
        }
    }
}
```

AES encryption

B. Decryption flow chart
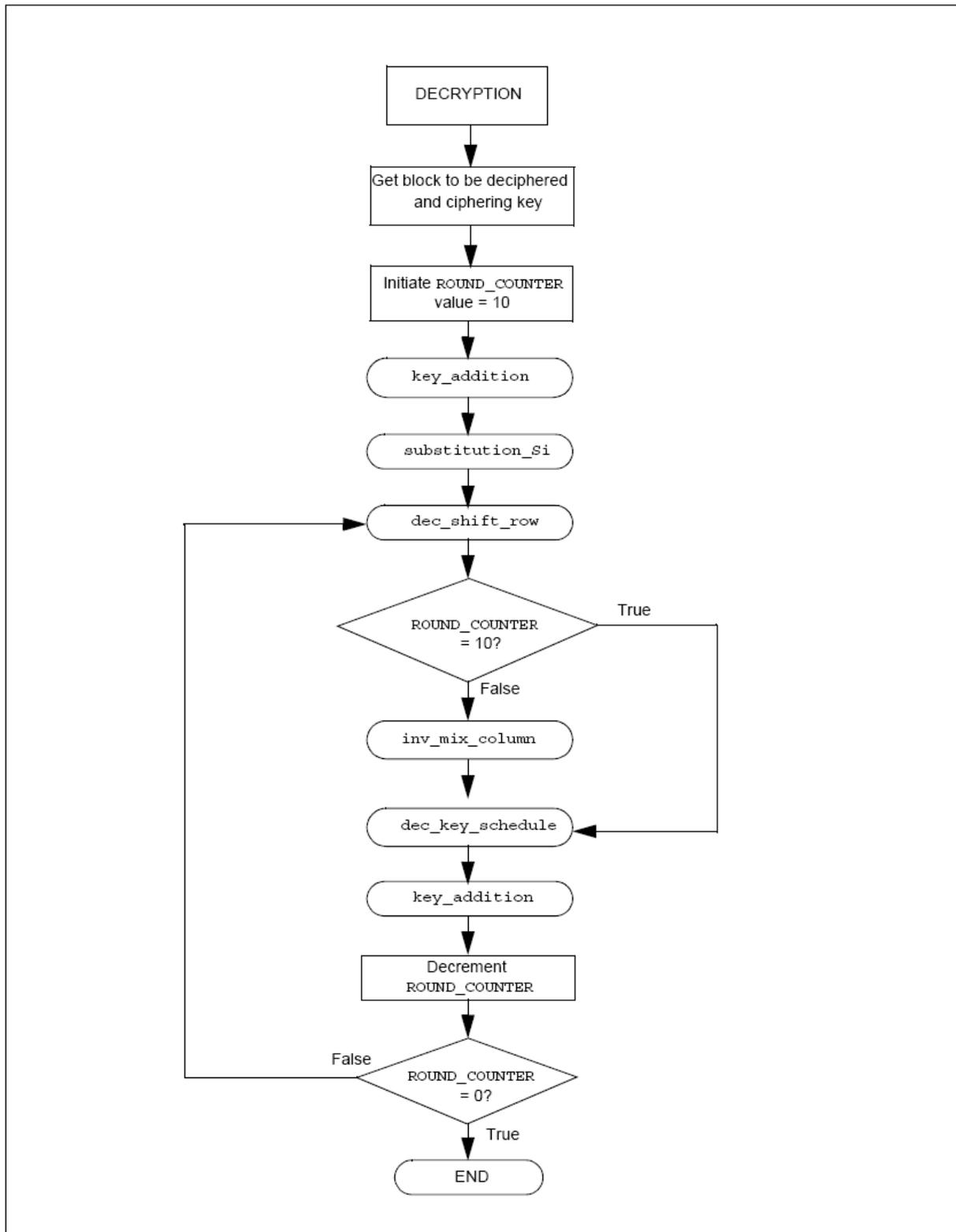
Coding

Then structure of the decryption program is:

```
void decrypts(char in[])
{
    unsigned char i,j,round;

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[j][i] = in[i*4 + j];
        }
    }

                key_addition(10);

                round = 9;
    for(round;round>0;round--)
    {
                substitution_si();
                dec_shift_row();
        key_addition(round);
        inv_mix_column();
                dec_key_schedule();  //  inverse  key_schedule
executed on-the-fly
    }
    substitution_si();
    dec_shift_row();
                dec_key_schedule();  //  inverse  key_schedule
executed on-the-fly
    key_addition(0);

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            out[i*4 +j]=state[j][i];
        }
    }
}
```

9 Hardware of ECS Demo System

9.1 Realization of RF Module

9.1.1 Using STD-402

In SCS-SMS project we use the USART theory which is included in the PCI microcontroller to send and receive data via serial. So, we can use the STD-402 to send and receive message by adding the max232 circuit. We use the max232 IC to translate the Tx/Rx of microcontroller (0V – 5V) to the RS232 Tx/Rx (-10V, +10V).

- **The test of the transceiver is shown in following lines**:

The MB-STD-RS232 board which equips the STD-402 transceiver module has stored unique module identification number in the radio module. When one unit is set up for master and other unit is for slave, slave modem unit operate with same ID as master unit.

1. Connect the serial cable to the D-SUB 9pin connector.

2. Set "Cable SW" (Cross / straight) according to the cable.

RSSI
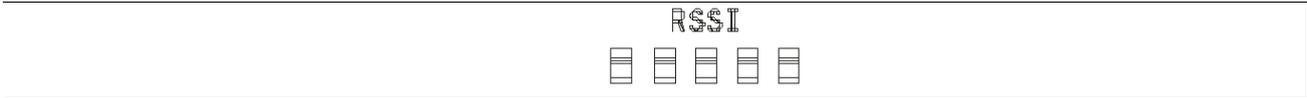
TX RX
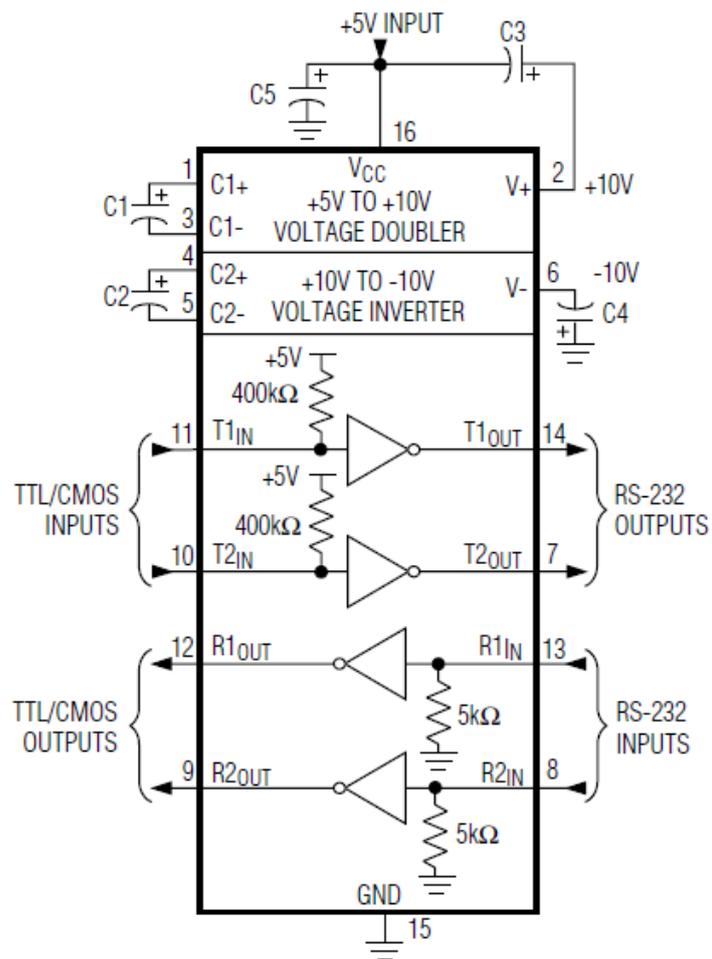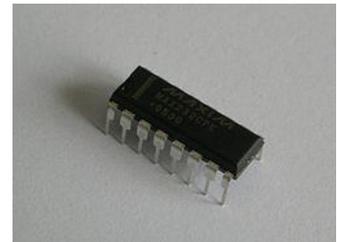
Realization of RF Module
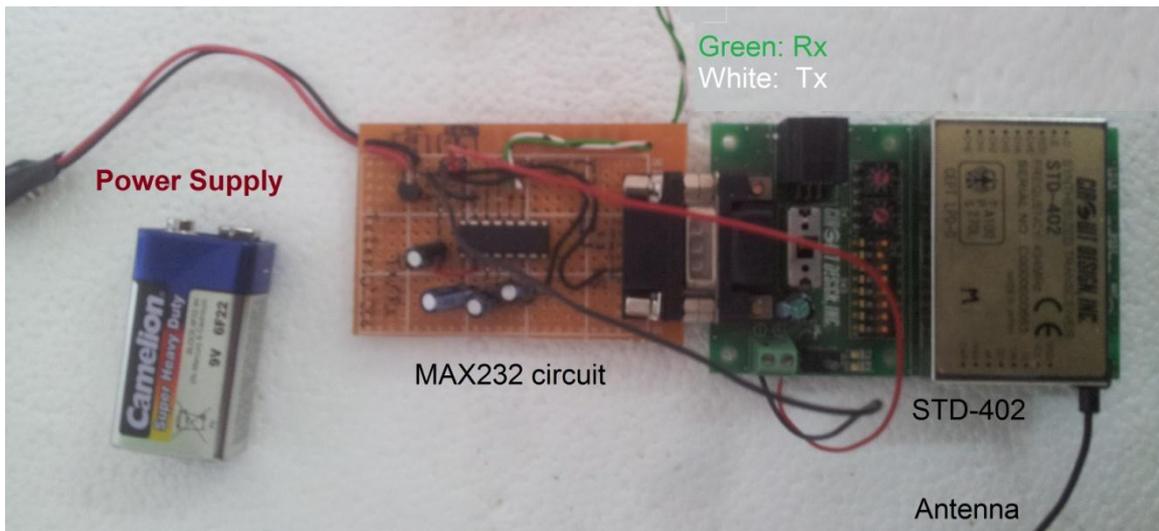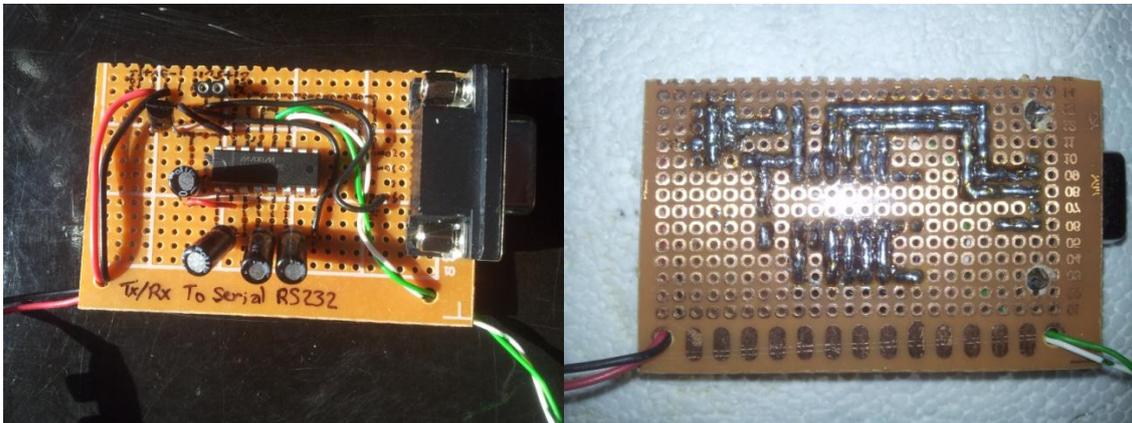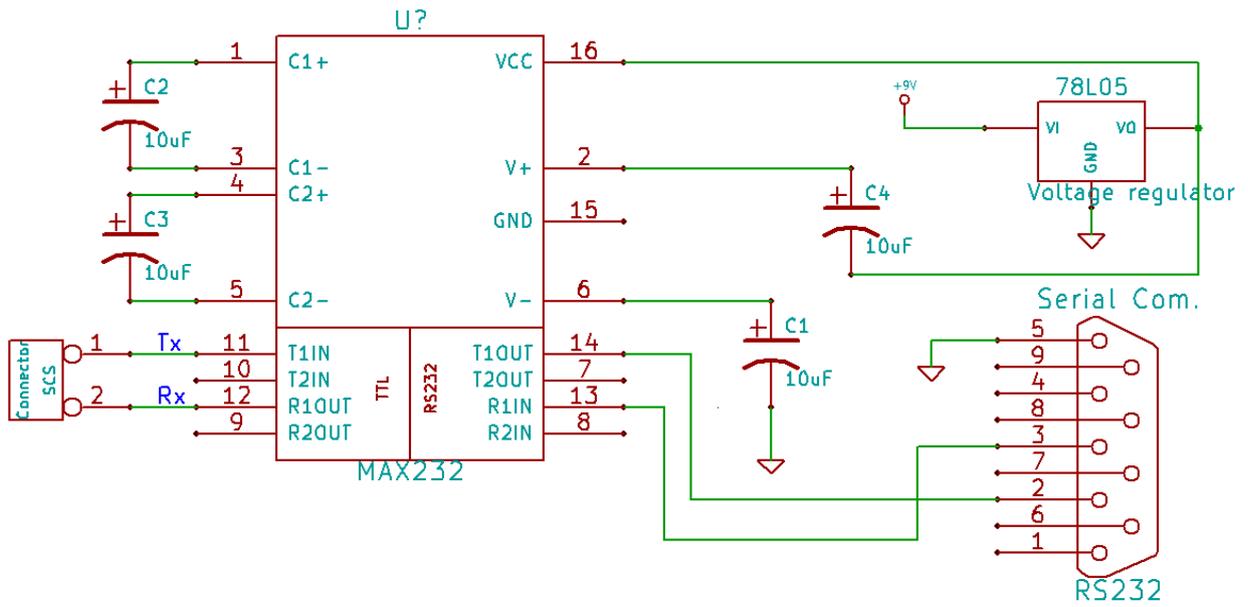
9.1.2 Realization of RF Module Using RFM42B-RFM31B - 433MHz

The RFM42B and the RFM31B use the SPI communication theory to send (RFM42B) or to receive (RFM31B) data. So, to use this two RF module some software and hardware change should be implement to the SCS-SMS project.

9.1.2.1 Serial Periferal interface (SPI)

The RFM31B/42B communicates with the host MCU over a standard 3-wire SPI interface: SCLK, SDI, and nSEL. The host MCU can read data from the device on the SDO output pin. A SPI transaction is a 16-bit sequence which consists of a Read-Write (R/W) select bit, followed by a 7-bit address field (ADDR), and an 8-bit data field (DATA) as demonstrated in Figure 2. The 7-bit address field is used to select one of the 128, 8-bit control registers. The R/W select bit determines whether the SPI transaction is a read or writes transaction. If R/W = 1 it signifies a WRITE transaction, while R/W = 0 signifies a READ transaction. The contents (ADDR or DATA) are latched into the RFM31B/42B every eight clock cycles. The SCLK rate is flexible with a maximum rate of 10 MHz.

To read back data from the RFM31B/42B, the R/W bit must be set to 0 followed by the 7-bit address of the register from which to read. The 8 bit DATA field following the 7-bit ADDR field is ignored n the SDI pin when R/W = 0. The next eight negative edge transitions of the SCLK signal will clock out the contents of the selected register. The data read from the selected register will be available on the SDO output pin. The READ function is shown in Figure 3. After the READ function is completed the SDO pin will remain at either logic 1 or logic 0 state depending on the last data bit clocked out (D0). When nSEL goes high the SDO output pin will be pulled high by internal pullup.

The figure bellow shows us the PIN description:
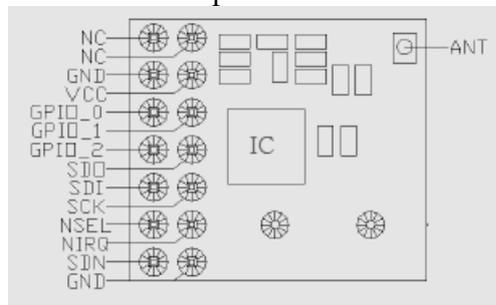
9.1.2.2 The new hardware design

The change which will be happen is caused by adding the two RFM module, the two RFM module connect to the same pins (5 pins connect to the PIC and three connect to the power supply). The RFM module is low power consumption, it's need about 85mA with supply voltage range between 1.8V and 3.6V, and you can see in its datasheet that the best is 3.0V. For this reason and because of our system supply voltage is 5V, so we need to regulate the supply voltage of the RFM to 3V. We do that by voltage divider theory by adding two high impedence (MΩ) resistors.

We know that our supply voltage is 5V which is regulate by 78L05 and the goal is to get 3V, so using voltage divider theory as you can see in the next figure we can get it. The cause of using two high impedence resistors is to eliminate the effect of the interior impedence in the system and the impedence in the RFM module.

Realization of RF Module
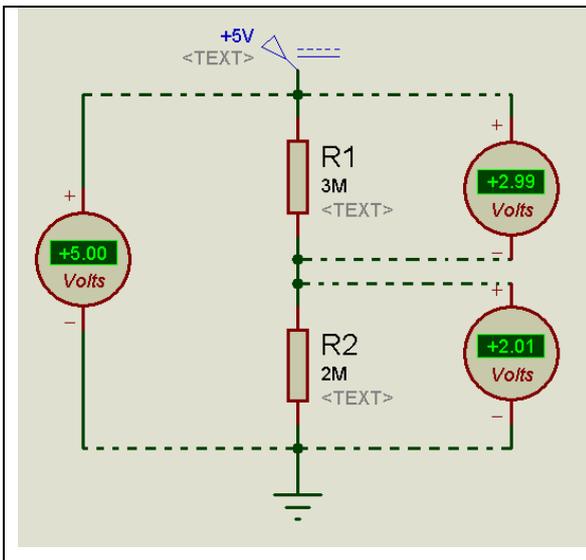
	Input Voltage = 5V R1 = 3 MΩ V out = (3 / 5) x 5 = 3V R2 = 2 MΩ Vout = (2 / 5) x 5 = 2V

Now, let connect the RFM modules to the PIC. The figure bellow shows us how the module connects to PIC microcontroller:
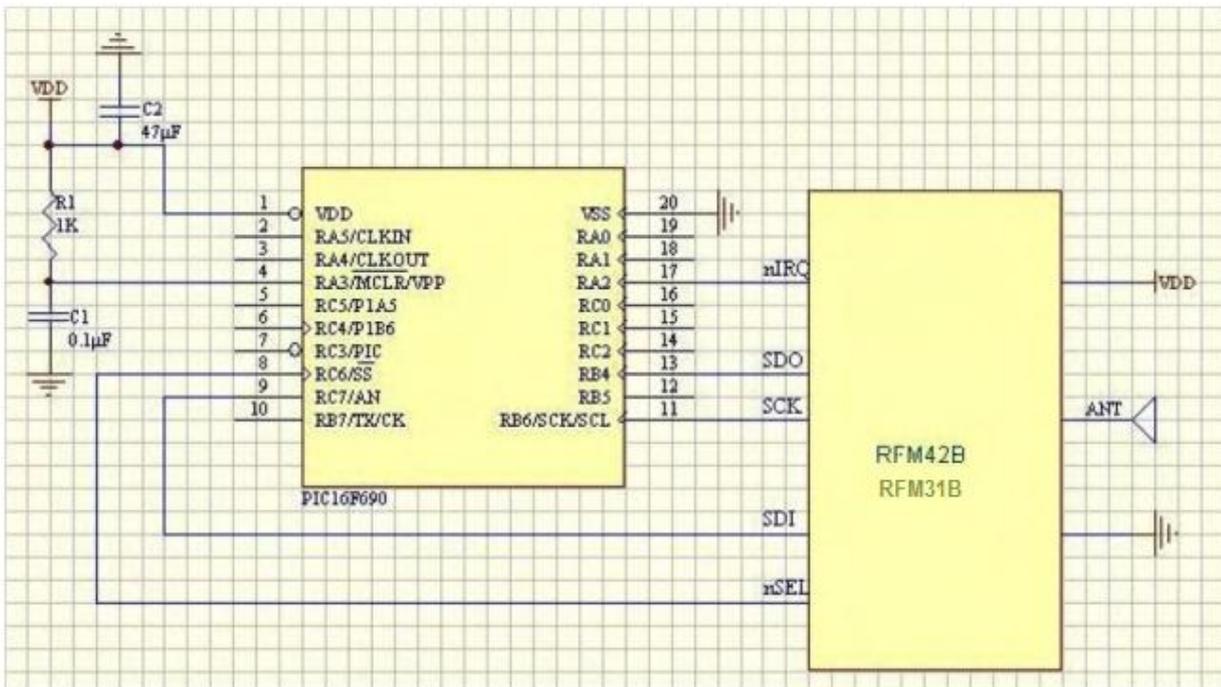
In SCS-SMS we use the PIC 18f4550, so the connection should be doing as follows:

RFM module pins	PIC pins
SDO	SDI (RB0 PIN)
SCK	SCK (RB1 PIN)
NIQR	INT2 (RB2 PIN)
SDI	SDO (RC7)
NSEL	SS (RA5) for RFM31B & (RB3) for RFM42B

While two else pins should connected to the Ground and one else should connect to +3V, and all the remaining pins doesn't connected anywhere.

As we see in the table above, we have 3 connections to the port B which already used in the SCS-SMS project for the KEYPAD. So, we need to change the connection of the KEYPAD to another pins (we take pins RA0 to RA3 for input keypad pins and RB4 to RB7 for the output keypad pins). Finally the new circuit design became as followes:
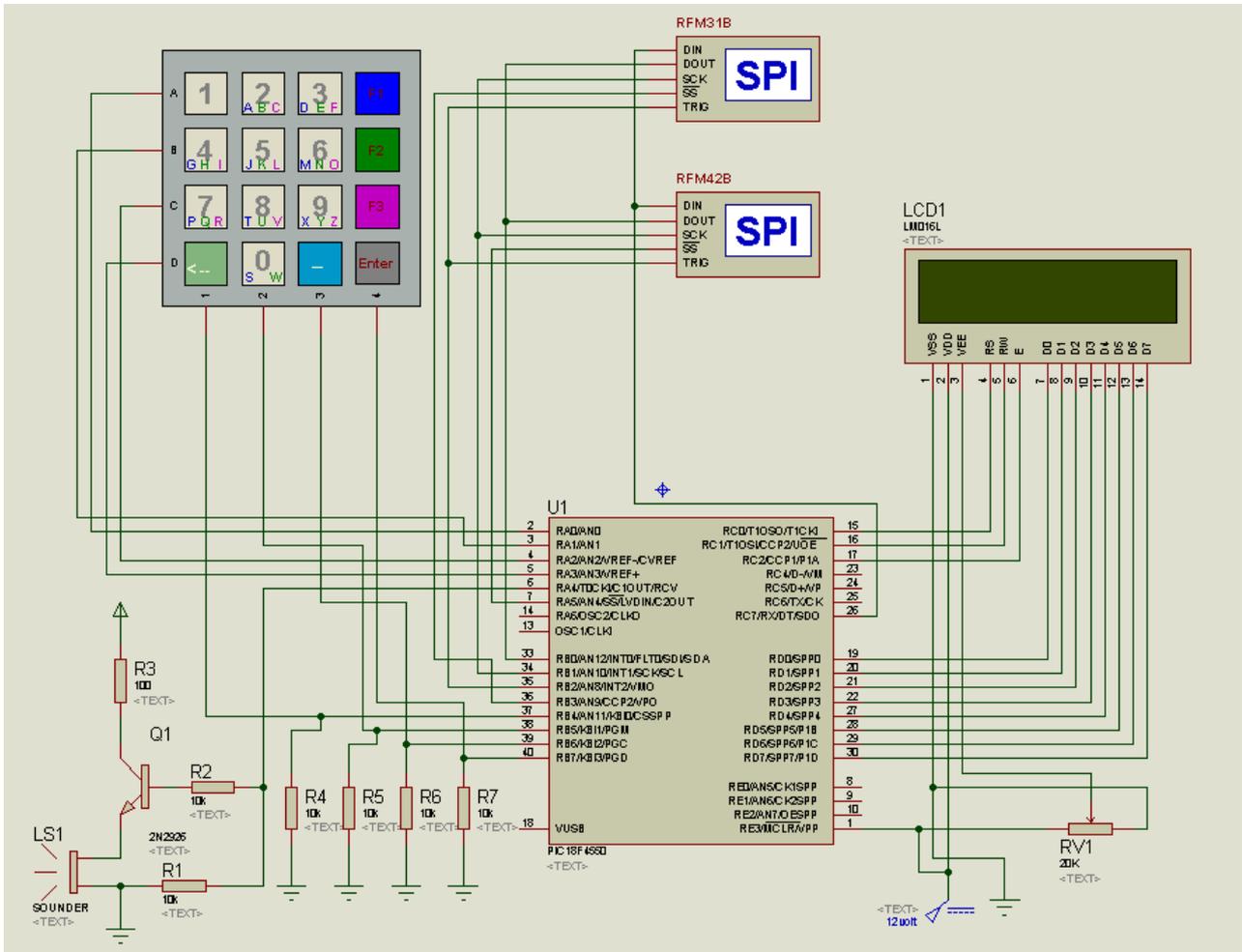
9.1.2.3   MSSP module to establishing (SPI)[10]

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communication with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. the MSSP module can operate in one of two methods:

- Serial Peripheral interface (SPI)
- Inter-integrated circuit (I2C)

▪ **Control Registers:**

The MSSP module has three associated control registers. These include a status register (SSPSTAT) and two control registers (SSPCON1 and SSPCON2). The use of these registers and

---

[10] PIC18F4550 datasheet, chapter 19, page 197

their individual Configuration bits differ significantly depending on whether the MSSP module is operated in SPI or I2C mode.

- **SPI mode:**

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four modes of the SPI are supported. To accomplish communication, typically three pins are used:

- Serial Data Out (**SDO**) – RC7/RX/DT/SDO

- Serial Data In (**SDI**) – RB0/AN12/INT0/FLT0/SDI/SDA

- Serial Clock (**SCK**) – RB1/AN10/INT1/SCK/SCL

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select (**SS**) – RA5/AN4/SS/HLVDIN/C2OUT

- **Registers:**

The MSSP module has four registers for SPI mode operation. These are:

- MSSP Control Register 1 (**SSPCON1**)
- MSSP Status Register (**SSPSTAT)**
- Serial Receive/Transmit Buffer Register (**SSPBUF**)

SSPCON1 and SSPSTAT are the control and status registers in SPI mode operation. The SSPCON1 register is readable and writable. The lower six bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write. SSPBUF is the buffer register to which data bytes are written to or read from.

In receive operations; SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not doublebuffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

SSPSTAT register:

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE[1]	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF

bit 7   bit 0

SMP: sample bit, in master mode (1: data sampled at end, 0: at middle), in slave mode (SMP=0)
CKE: SPI clock select bit (1: transmit on transition from active to Idle, 0: from Idle to active)
D/A, **P**, **S**, **R/W**, **UA**: used in I2C only
BF: Buffer full status bit when receive (1: SSPBUF full, 0: receive not complete)

SSPCON1 register:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV[(1)]	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

WCOL: write collision Detect bit (on transmitting) (1: collision, 0: no collision)
SSPOV: receive overflow indicator bit (slave mode) (1: overflow, 0: no overflow)
SSPEN: master Synchronous Serial Port Enable bit (1: enable serial pot, 0: serial port be I/O)
CKP: clock polarity select bit (1: idle state for clock is High, 0: is low)
SSPM3:**SSPM0**: Master Synchronous serial Port Mode Select bits
> 0101 = SPI Slave mode, clock=SCKpin, Sspin control disabled, SS can be used as I/Opin
> 0100 = SPI Slave mode, clock=SCKpin, Sspin control enabled
> 0011 = SPI Master mode, clock=TMR2 output/2
> 0010 = SPI Master mode,, clock=Fosc/64
> 0001 = SPI Master mode,, clock=Fosc/16
> 0000 = SPI Master mode,, clock=Fosc/4

In our case, we need to use the micro controller in Master mode to connect it the the 2 slave modules (RFM31B and RFM42B) we have.

- ▪ **Master mode**:

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave is to broadcast data by the software protocol.

In Master mode, the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a "Line Activity Monitor" mode.

The clock polarity is selected by appropriately programming the CKP bit (SSPCON1<4>). This, then, would give waveforms for SPI communication. In Master mode, the SPI clock rate (bit rate) is user-programmable to be one of the following:
- • FOSC/4 (or TCY)
- • FOSC/16 (or 4 • TCY)
- • FOSC/64 (or 16 • TCY)
- • Timer2 output/2

This allows a maximum data rate (at 48 MHz) of 12.00 Mbps.

When used in Timer2 Output/2 mode, the bit rate can be configured using the PR2 Period register and the Timer2 prescaler. However, writing to SSPBUF does not clear the current TMR2 value in hardware. Depending upon the current value of TMR2 when the user firmware writes to SSPBUF, this can result in an unpredictable MSb bit width.

When the CKE bit is set, the SDO data is valid before there is a clock edge on SCK. The change of the input sample is shown based on the state of the SMP bit. The time when the SSPBUF is loaded with the received data is shown.

Realization of RF Module

 ▪ **Implementation**:

In our case, the PIC is used as SPI in master mode and inially set the two regidter as followes;

<u>SSPSTAT</u>: (0xC0) SMT=1, CKE=1, all the remaing bit are readeble not writable as we see before.

<u>SSPCON1</u>: (0x30) WCOL=0, SSPOV=0, SSPEN=1, CKP=1, SSPM=0000.

Bellows is the SPI code in C programing with comment when necessary: (SPI.c)

```c
#include<p18f4550.h>
#include<string.h>
#include<delays.h>
#include"SPI.h"

void InitDAC(void)
{
       NSEL1pin = 0;
       NSEL2pin = 0;
       SDIpin = 1;
       SCKpin = 0;
       NIQRpin = 1;
       SDOpin = 0;
       ADCON1=0x0f;  //Turn off A/D

       SSPSTAT=0xC0; //SMP: SPI master mode, CKE: active to idle clock state, 0, 0, 0,
0, 0, 0
       SSPCON1=0x30;// Mode 1,1 SPI Master Mode, 1/4 Tosc bit
}

void Send_char_SPI(char data)
{
       SS1 = 0;      // Enable SS1 Output (low)
       SS2 = 1;      // Disable SS2 Output (high)
       SSPBUF=data; // sending the upper 8 bits serially
       while(!SSPSTATbits.BF);   // wait until the upper 8 bits are sent
       SS1 = 1;      // Disable SS1 Output (high)
       SS2 = 0;      // Enable SS2 Output (low)
}

void Send_int_SPI(int data)
{
       unsigned int c;
       unsigned int lower_bits, upper_bits;
       c = ((data+1)*16) -1;      // here we obtain 12 bit data

       //first obtain the upper 8 bits
       upper_bits = c/256; // obtain the upper 4 bits
       upper_bits = (48) | upper_bits;  // append 0011 to the above 4 bits


       //now obtain the lower 8 bits
       lower_bits = 255 & c;      // ANDing separates the lower 8 bits

       SS1 = 1;      // Disable SS1 Output (high)
       SS2 = 0;      // Enable SS2 Output (low)

       PORTBbits.RB0=0;
       SSPBUF=upper_bits;  // sending the upper 8 bits serially
       while(!SSPSTATbits.BF);    // wait until the upper 8 bits are sent
       SSPBUF=lower_bits;  // sending the lower 8 bits serially
       while(!SSPSTATbits.BF);    // wait until the lower 8 bits are sent
       PORTBbits.RB0=1;

       SS1 = 1;      // Disable SS1 Output (high)
       SS2 = 0;      // Enable SS2 Output (low)
}

void Send_string_SPI(char s[])
```

Hardware of ECS Demo System

```
{
      int i, lenght;
      lenght = strlen(s);

      for(i=0; i <= lenght; i++)
      {
            Send_char_SPI(s[i]);
            Delay10KTCYx(17);   // wait until the 8 bits char are sent
                                // Delay Subroutine: 169129 Clock cycles ~= 17(10KTCY)
~= 0.014 seconds
      }
}

char Receive_data_SPI(void)
{
      while(!SSPIFbits.PIR1);   // Interrupt flag set when transmission/reception is
complete
      return SSPBUF;

}
```

Bellowe is the Header file for this c librery: (SPI.h)

```
#ifndef SPI_H
#define SPI_H

#define Clock_Khz 48000       //Fosc = 48Mhz

#define NSEL1pin TRISAbits.TRISA5   // connect to the NSEL pin of the Tx
#define NSEL2pin TRISBbits.TRISB3   // connect to the NSEL pin of the Rx
#define SDIpin TRISBbits.TRISB0     // connect to the SDO pin of SPI module
#define SCKpin TRISBbits.TRISB1     // connect to the SCK pin of SPI module
#define NIQRpin TRISBbits.TRISB2    // connect to the NIQR pin of SPI module
#define SDOpin TRISCbits.TRISC7     // connect to the SDI pin of SPI module

#define SS1 PORTAbits.RA5    // selection slave 1 (transmiter)
#define SS2 PORTBbits.RB3    // selection slave 2 (receiver)

void InitSPI(void);
void Send_char_SPI(char n);
void Send_int_SPI(int n);
void Send_string_SPI(char s[]);
char Receive_data_SPI(void);

#endif
```

10 Further Work: System Integration and Integration Test of ECS Demo System

tbd

Appendix A: Alternative Project Plans

A.1: Three Alternatives for Central Station / Mobile Stations

As we see before, each side of this project has multiple potential choices. And each choice has its extra tasks. In this part we will specific the tasks with the period of each choice of each side.

- **For the STD hardware.**

    a. Using existing STD hardware (ran) in the two sides (connect to computer).
       No extra potential tasks for this choice **(only 1 week for testing)**

    b. Using existing STD hardware (ran) in the base side and using the new SCS-SMS hardware on the client side

       Tasks for this choice:

       - Change the SCS-SMS hardware to be able to use **(need among 3 weeks)**
       - Change and develop the input part of the ExtIO file to be compatible with the client side hardware **(2 weeks)**

       Note that, if this choice is taken you can't take the HDSDR as a choice for the SDR.

    c. Develop a hardware to be able to connect to 4 antenna on the base side and to one antenna on the client side (base on HackRF project)

       Tasks for this choice:

       - Develop the hardware for one antenna for the client side **(3 weeks)**

       - Develop the hardware for 4 antennas for the base side **(2 extra weeks)**

       - Change and develop the input part of the ExtIO file to be compatible with the client side hardware (one antenna) **(2 weeks)**
       - Change and develop the input part of the ExtIO file to be compatible with the base side hardware (4 antenna) **(1 extra week)**

- **For the SDR code**

    a. Using HDSDR by develop its ExtIO.
       Tasks for this choice:

       - Change and develop the output part of the ExtIO file to be compatible with the GUI interface software **(4 weeks)**
       - Develop our GUI interface **(4 weeks)**

    b. Using the source code of WinRad

       Tasks for this choice:

       - Take the SDR code from the WinRad software **(3 weeks)**
       - Develop the SDR code to send and receive **(2 weeks)**

- Change and develop the output part of the ExtIO file to be compatible with the GUI interface software **(4 weeks)**
- Develop our GUI interface **(4 weeks)**

A side to this task and duration there are the testing and documentation tasks and period. Bellow is two project plans:

Project 1: choice (a) for STD hardware choice (a) for the SDR code

Using existing STD hardware (ran) in the two sides (connect to computer). With using HDSDR by develop its ExtIO for the SDR code

Event	Time
Getting start with software (Qt, VC++, HDSDR)	2 weeks
Using HDSDR to receive and transmit Radio wave	1 week
Getting Start with DLL and see demo ExtIO	1 week
Change and develop the output part of the ExtIO file to be compatible with the GUI interface software	4 weeks
Develop our GUI interface	4 weeks
System testing (task **9**)	2 weeks
Documentation and Final report (task **10**)	3 weeks

Approximately **4 months and 1 week** with a possibility of delay

Project 2: choice (b) for STD hardware choice (b) for the SDR code

Using existing STD hardware (ran) in the base side and using the new SCS-SMS hardware on the client side. With using of the source code of WinRad for the SDR code

Event	Time
Getting start with software (Qt, VC++, WinRad)	2 weeks
Using WinRad to receive Radio wave	1 week
Read with understanding the WinRad code	1 week
Take the SDR code from the WinRad software	2 weeks
Develop the SDR code to send and receive	2 weeks
Getting Start with DLL and see demo ExtIO	1 week
Change and develop the output part of the ExtIO file to be compatible with the GUI interface software	4 weeks
Develop our GUI interface	4 weeks
Change the SCS-SMS hardware to be able to use	3 weeks
Change and develop the input part of the ExtIO file to be compatible with the client side hardware	2 weeks
System testing (task **9**)	2 weeks
Documentation and Final report (task **10**)	3 weeks

Approximately **6 months and 2 weeks** with a possibility of delay

Project 3: choice (c) for STD hardware choice (b) for the SDR code

Develop hardware to be able to connect to 4 antennas on the base side and to one antenna on the client side with using of source code of WinRad

Event	Time
Getting start with software (Qt, VC++, WinRad)	2 weeks

Using WinRad to receive Radio wave	1 week
Read with understanding the WinRad code	1 week
Take the SDR code from the WinRad software	2 weeks
Develop the SDR code to send and receive	2 weeks
Getting Start with DLL and see demo ExtIO	1 week
Change and develop the output part of the ExtIO file to be compatible with the GUI interface software	4 weeks
Develop our GUI interface	4 weeks
Develop the hardware for one antenna for the client side	3 weeks
Develop the hardware for 4 antennas for the base side	2 weeks
Change and develop the input part of the ExtIO file to be compatible with the client side hardware (one antenna)	2 weeks
Change and develop the input part of the ExtIO file to be compatible with the base side hardware (4 antenna)	1 week
System testing (task **9**)	2 weeks
Documentation and Final report (task **10**)	3 weeks

Approximately **7 months and 2 weeks** with a possibility of delay

A.2: Demo System Integration with different developers

Event	Time
Using WinRad to receive Radio wave using exist SDR platform	1 week
Introduction to HackRF SDR platform	2 weeks
Build our SDR platform	2 weeks
Using WinRad to receive Radio wave via new SDR platform	1 week
Build our Amateur Radio Transceiver (ART)	3 weeks
Connect the SCS-SMS hardware to the ART with testing	2 weeks
Take I and Q from WinRad to a file	1 week
Develop GUI interface to read SMS from file	2 weeks
System testing	2 weeks
Documentation and Final report	2 weeks

Approximately **18 weeks** with a possibility of delay

These tasks are dividing to a three work packages, which are:

1st package: building of the SDR platform

2nd package: building of the Amateur Radio Transceiver

3rd package: WinRad and interface software

ID	Name	Start	Finish	October 2013				November 2013		
				07	14	21	28	04	11	18
1	introducation to WinRad	10/7/2013	10/13/2013							
2	take I & Q from WinRad to a file	10/14/2013	10/20/2013							
3	develop user interface to read SMS	10/21/2013	11/3/2013							
4	system testing	11/4/2013	11/10/2013							

Appendix B: All about HackRF
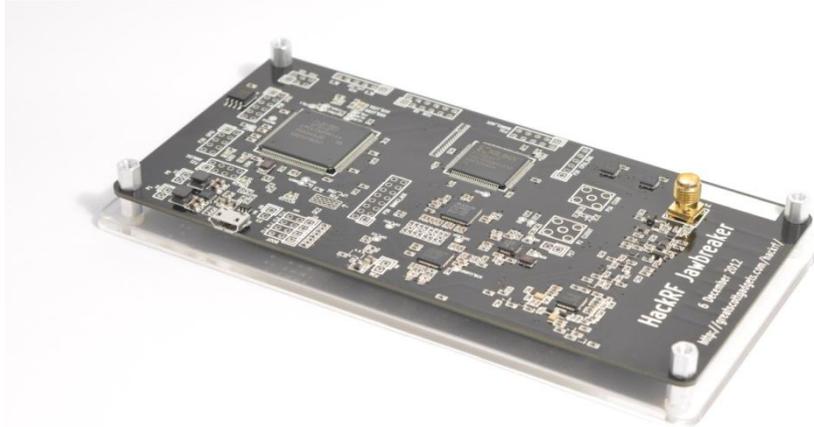
HackRF is a project to produce a low cost, open source software radio platform.

Principal author: Michael Ossmann: mike@ossmann.com
Home hackRF website: https://github.com/mossmann/hackrf

HackRF is an open source hardware project to build a Software Defined Radio (SDR) peripheral.

B.1 HackRF overview

SDR is the application of Digital Signal Processing to radio waveforms. It is similar to the software-based digital audio techniques that became popular a couple of decades ago. Just as a sound card in a computer digitizes audio waveforms, a software radio peripheral digitizes radio waveforms. It's like a very fast sound card with the speaker and microphone replaced by an antenna. A single software radio platform can be used to implement virtually any wireless technology (Bluetooth, ZigBee, cellular technologies, FM radio, etc.).

Digital audio capabilities in general purpose computers enabled a revolution in the sound and music industries with advances such as hard disk recording and MP3 file sharing. Today's computers are fast enough to process radio waveforms in similar ways, and the radio communications industry is going through the same sorts of changes. One critical advance is finally taking place now, and that is the availability of low cost tools enabling anyone to take part in the revolution.

✓ **Wide Operating Frequency Range:**

HackRF operates from 30 MHz to 6 GHz, a wider range than any SDR peripheral available today. This range includes the frequencies used by most of the digital radio systems on Earth. It can operate at even lower frequencies in the MF and HF bands when paired with the Ham It up RF up converter.

✓ **Transceiver:**

HackRF can be used to transmit or receive radio signals. It operates in half-duplex mode: it can transmit or receive but can't do both at the same time. However, full-duplex operation is possible if you use two HackRF devices.

✓ **Low Cost:**

HackRF was designed to be the most widely useful SDR peripheral that can be manufactured at a low cost. The estimated future retail price of HackRF is $300, but you can get one for even less by backing the Kickstarter project today.

✓ **Wideband:**

The maximum bandwidth of HackRF is 20 MHz, about 10 times the bandwidth of TV tuner dongles popular for SDR. That means that HackRF could be used for high speed digital radio applications such as LTE or 802.11g.

✓ **Open Source:**

The most important goal of the HackRF project is to produce an open source design for a widely useful SDR peripheral. All hardware designs and software source code are available under an open source license. The hardware designs are produced in KiCad, an open source electronic design automation tool. You can download the Jawbreaker (HackRF beta) design and build your own HackRF today!

✓ **Compatible:**

HackRF beta units are already being used on Linux, OS X, and Windows platforms. The device takes full advantage of USB 2.0, an interface found on almost every general purpose computer. HackRF already works with the popular GNU Radio software framework, and HackRF support can be added to other SDR software.

✓ **Tested:**

The Jawbreaker design depicted above is the fully functional HackRF beta design. Hundreds of Jawbreakers have been distributed to developers and beta testers. HackRF has already been used for Digital Audio Broadcasting (DAB), Bluetooth monitoring, spectrum sensing, wireless microphones, AIS, FM radio, and more. I plan to use feedback from beta testers to make your HackRF even better than Jawbreaker.

B.2 Jawbreaker[11]

Jawbreaker is the first complete HackRF platform, a wideband software radio transceiver with a USB interface.

Hardware notes:
Schematic and layout files were designed in **KiCad**, an open source electronic design automation package.

```
order of copper layers:
    Copper 1: Front
    Copper 2: Inner3
    Copper 3: Inner2
    Copper 4: Back

PCB description: 4 layer PCB 0.062 in
    Copper        1   0.5 oz foil plated to approximately 0.0017 in
    Dielectric  1-2   0.0119 in
    Copper        2   1 oz foil (0.0014 in)
    Dielectric  2-3   0.0280 in
    Copper        3   1 oz foil (0.0014 in)
    Dielectric  3-4   0.0119 in
    Copper        4   0.5 oz foil plated to approximately 0.0017 in
FR4 or similar substrate with Er=4.5 (+/- 0.1)
double side solder mask black
double side silkscreen white
6 mil min trace width and
6 mil min isolation
```

B.3 Jellybean[12]

---
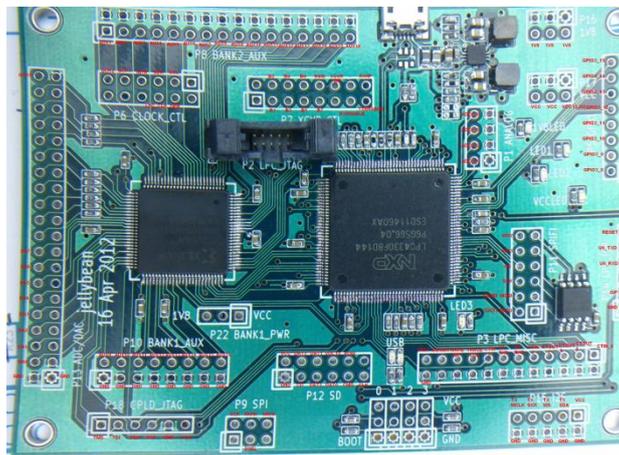
[11] ..\\6-ECS-SDR\5-Guide\HackRF\hardware\jawbreaker
This file contain also the Hardware design file, but it should open by **KiCad** software

[12] ..\\6-ECS-SDR\5-Guide\HackRF\hardware\jellybean
This file contain also the PCB design as pdf, with the Hardware design file, but it should open by KiCad software

Jellybean is a microcontroller platform based on the LPC43xx. It is designed to control Lemondrop.

Hardware notes:
Schematic and layout files were designed in **KiCad**, an open source electronic design automation package.

```
order of copper layers:
    Front
    Inner3
    Inner2
    Back

PCB description: 4 layer PCB 1.6 mm
    Copper        1    35 um
    Dielectric  1-2    0.35 mm
    Copper        2    18 um
    Dielectric  2-3    0.76 mm
    Copper        3    18 um
    Dielectric  3-4    0.35 mm
    Copper        4    35 um
DE104iML or equivalent substrate (Er=4.42@2.4GHz TanD=0.016)
double side solder mask black
double side silkscreen white
6 mil min trace width and
6 mil min isolation
```

This file contain also the PCB design as pdf, with the Hardware design file, but it should open by KiCad software

B.4 Lemondrop[13]

Lemondrop is a 2.3 to 2.7 GHz wireless transceiver with a 22 Msps ADC/DAC and flexible clocking for software radio applications.

Hardware notes:
Schematic and layout files were designed in **KiCad**, an open source electronic Design automation package.

```
order of copper layers:
    Front
    Inner3
    Inner2
    Back

PCB description: 4 layer PCB 1.6 mm
    Copper        1    35 um
    Dielectric  1-2    0.35 mm
```

---

[13] ..\\6-ECS-SDR\5-Guide\HackRF\hardware\lemondrop
This file contain also the Hardware design file, but it should open by **KiCad** software

```
       Copper          2    18 um
       Dielectric   2-3    0.76 mm
       Copper          3    18 um
       Dielectric   3-4    0.35 mm
       Copper          4    35 um
DE104iML or equivalent substrate (Er=4.42@2.4GHz TanD=0.016)
double side solder mask black
double side silkscreen white
6 mil min trace width and
6 mil min isolation
```

B.5 HackRF Hardware
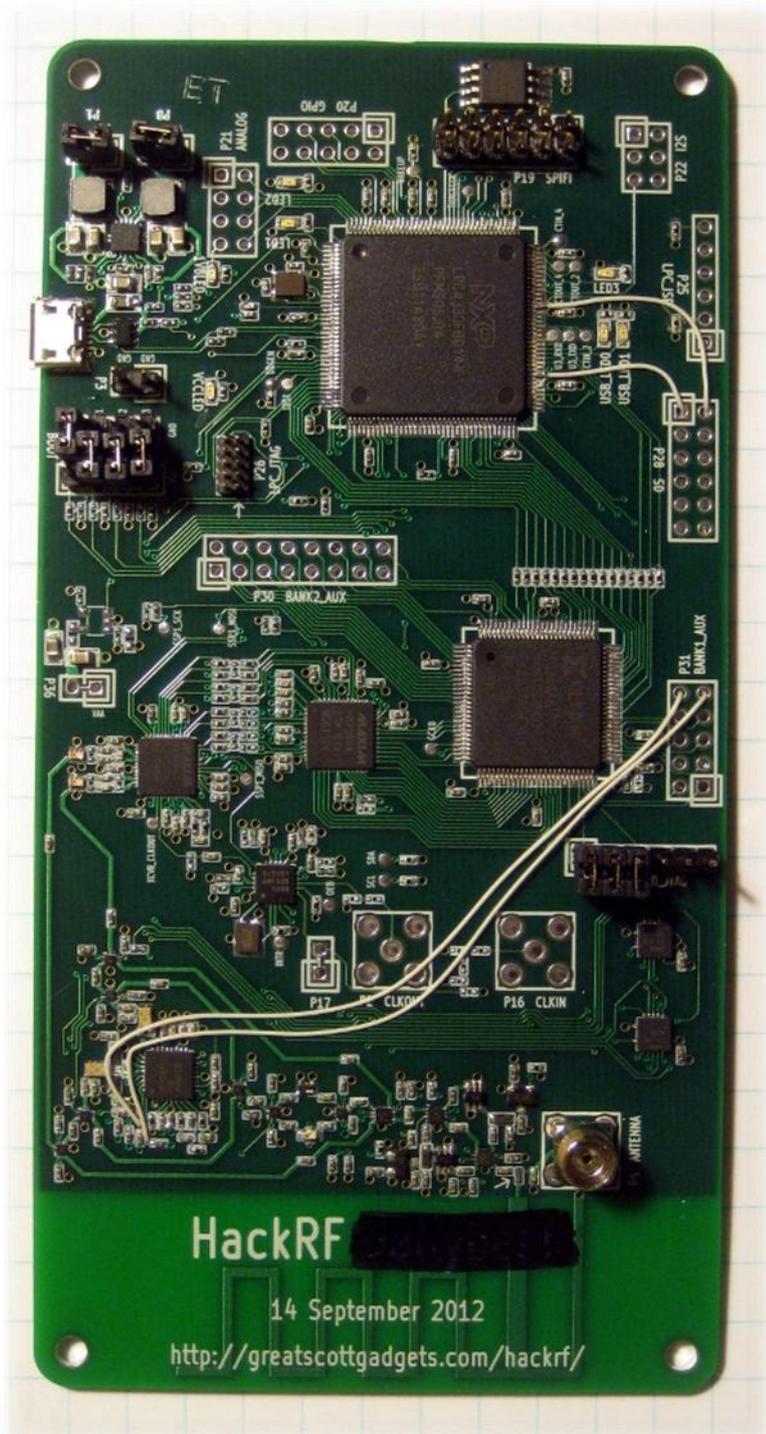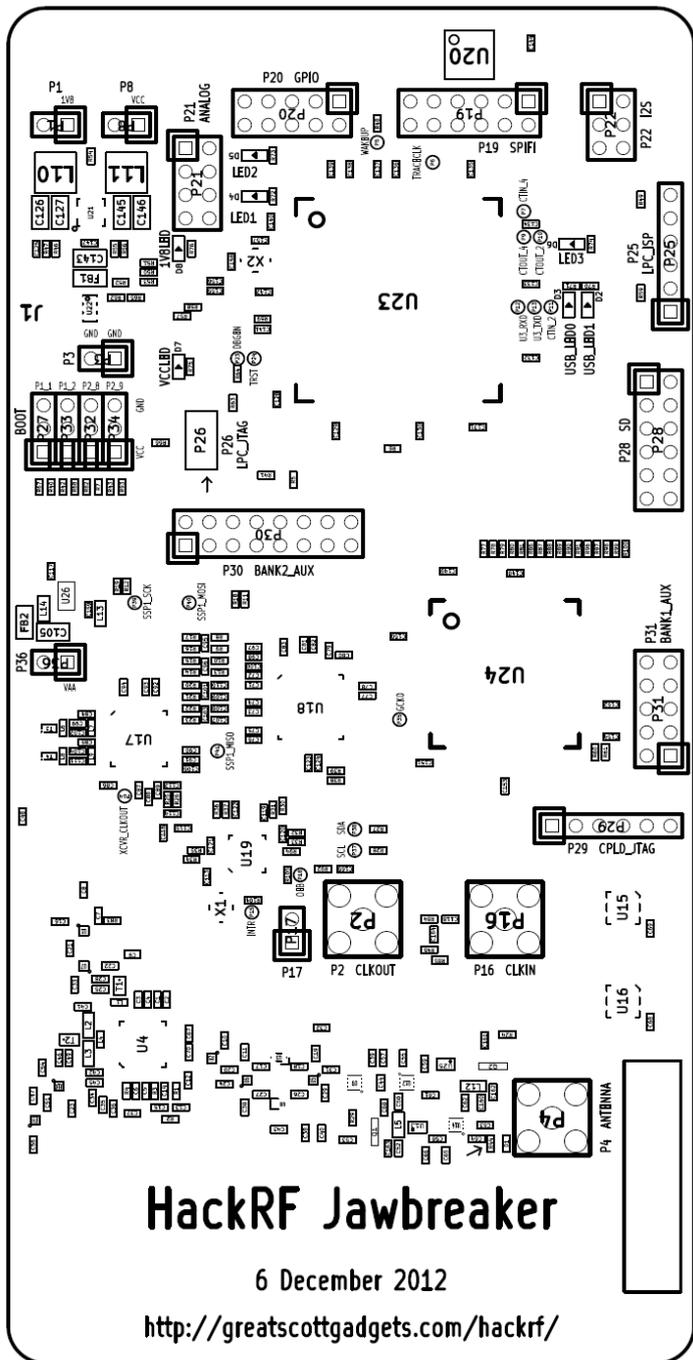
- **Board IC:**

U1-U2-U5-U6-U7-U10-U11:SKY13350;Skyworks;SKY13350-385LF;0.01-6.0 GHz GaAs SPDT Switch

U3:RX_LOWPASS_FILTER;AVX;LP0603A1880ANTR;FILTER LOW PASS 1880MHZ 0603 SMD

U4:RFFC5072;RFMD;RFFC5072TR7;WIDEBAND SYNTHESIZER/VCO WITH INTEGRATED 6GHz MIXER

U8:RX_HIGHPASS_FILTER;TDK;DEA162400HT-8004B1;FILTER HIGHPASS WLAN&BLUETOOTH

U9-U12-U14:SKY13317;Skyworks;SKY13317-373LF;20 MHz-6.0 GHz pHEMT GaAs SP3T Switch

U13–U25:MGA-81563;Avago;MGA-81563-TR1G;0.1-6 GHz 3 V

U15:GSG-74HC04;Texas Instruments;SN74AHC04RGYR;IC HEX INVERTERS 14-QFN

U16:GSG-74HC08;Texas Instruments;SN74AHC08RGYR;IC QUAD 2IN POS-AND GATE 14-QFN

U17:MAX2837;Maxim;MAX2837ETM+;IC TXRX 2.3GHZ-2.7GHZ 48TQFN

U18:MAX5864;Maxim;MAX5864ETM+;IC ANLG FRONT END 22MSPS 48-TQFN

U19:SI5351C;Silicon Laboratories Inc;SI5351C-B-GM;IC CLK GENERATOR 160MHZ 20QFN

U20:W25Q80BV;Winbond;W25Q80BVSSIG;IC FLASH 8MBIT 8SOIC

U21:TPS62410;Texas Instruments;TPS62410DRCR;IC BUCK SYNC DUAL ADJ 0.8A 10SON

U22:GSG-IP4220CZ6;NXP;IP4220CZ6

U23:LPC43XXFBD144;NXP;LPC4330FBD144

U24:GSG-XC2C64A-7VQG100C;Xilinx;XC2C64A-7VQG100C;IC CR-II CPLD 64MCELL 100-VQFP

U26:RF LDO;DNP

- **Other component:**

FB1:FILTER;Murata;BLM21PG221SN1D;FERRITE CHIP 220 OHM 2000MA 0805
FB2:FILTER;Murata;BLM21PG221SN1D;FERRITE CHIP 220 OHM 2000MA 0805

Q1:MOSFET_P;Fairchild;BSS84;MOSFET P-CH 50V 130MA SOT-23
Q2:MOSFET_P;Fairchild;BSS84;MOSFET P-CH 50V 130MA SOT-23

T1:MIX_IN_BALUN;Anaren;B0310J50100AHF;Ultra Low Profile 0805 Balun 50 to 100 ohm Balanced
T2:MIX_OUT_BALUN;Anaren;B0310J50100AHF;Ultra Low Profile 0805 Balun 50 to 100 ohm Balanced
T3:RX_BALUN;Johanson Technology;2500BL14M100T;BALUN CERAMIC CHIP WIMAX 2.5GHZ
T4:TX_BALUN;Johanson Technology;2500BL14M100T;BALUN CERAMIC CHIP WIMAX 2.5GHZ

X1:GSG-XTAL4PIN;AVX;CX3225GB25000D0HEQZ1;CRYSTAL 25.000MHZ 8PF SMD
X2:MCU_XTAL;TXC;7V-12.000MAAE-T;CRYSTAL 12.000 MHZ 12PF SMD

With a lot of: capacitors, resistors, inductors, ports with jumpers

B.6 Extra file

➢ `LPCXpresso Flash Debug Tutorial(pdf file)`[14]

This pdf file contains the following points:
- Hardware required:
  - NXP LPC-Link board included with any LPCXPRESSO Board
  - LPC43xx board

- Software required:
  - LPCXpresso v4.2.3 build 292

- Starting LPCXpresso IDE

- *Create a project*

- *Flashing ".bin" or ".elf" in SPIFI flash memory*

- *Debugger configuration*

➢ `LPCXpresso Flash Debug Tutorial(pdf file)`[15]

This file contains the following folders:

```
Blinky
Blinky_rom_to_ram
Common
Cpld
Cpldjtagprog
Cpldjtagprog_rom_to_ram
Hackrf_usb
Hackrf_usb_rom_to_ram
Mixertx
sgpio
sgpio_passthrough_rom_to_ram
sgpio_rx
simpletx
spiflash
startup
startup_systick
startup_systick_perfo
startup_systick_perfo_rom_to_ram
```

---

[14] `..\\6-ECS-SDR\5-Guide\HackRF\doc\LPCXpresso_Flash_Debug_Tutorial.pdf`

[15] `..\\6-ECS-SDR\5-Guide\HackRF\firmware`

With makefile and this readme note:

The primary firmware source code for USB HackRF devices is hackrf_usb. Most of the other directories contain firmware source code for test and development. The common directory contains source code shared by multiple HackRF firmware projects. The cpld directory contains HDL source for the CPLD present on the Jawbreaker and Jellybean designs.

The firmware is set up for compilation with the GCC toolchain available `here`:

`https://code.launchpad.net/gcc-arm-embedded`

`Required dependency:`

`https://github.com/mossmann/libopencm3`

Another file named **`firmware-bin`** contain **`hackrf_usb_rom_to_ram`**`.bin`

B.7 Host build[16]

How to build host software on Windows:

Prerequisite for `cygwin` or `mingw`:

```
* cmake-2.8.10.2 or more see http://www.cmake.org/cmake/resources/software.html
```

```
* libusbx-1.0.14 or more see
http://sourceforge.net/projects/libusbx/files/latest/download?source=files
```

```
* Install Windows driver for HackRF hardware or use Zadig see
http://sourceforge.net/projects/libwdi/files/zadig
  - If you want to use Zadig select HackRF USB device and just install/replace
it with WinUSB driver.
```

```
* Build libhackrf before to build this library, see host/libhackrf/Readme.md.
```

For Cygwin:

```
cmake -G "Unix Makefiles" -DCMAKE_LEGACY_CYGWIN_WIN32=1 -
DLIBUSB_INCLUDE_DIR=/usr/local/include/libusb-1.0/
make
make install
```

For Mingw:

```
#normal version
cmake -G "MSYS Makefiles" -DLIBUSB_INCLUDE_DIR=/usr/local/include/libusb-1.0/
#debug version
cmake -G "MSYS Makefiles" -DCMAKE_BUILD_TYPE=Debug -
DLIBUSB_INCLUDE_DIR=/usr/local/include/libusb-1.0/
make
make install
```

---

[16] `..\\6-ECS-SDR\5-Guide\HackRF\host\hackrf-tools`

Appendix C: Alternative System Designs

System Design of v1.0

HackRF
SDR platform

GUI using I and Q
from Winrad

Amateur Radio
Transceiver

SCS-SMS

Amateur Radio
Transceiver

SCS-SMS
with ART

Amateur Radio
Transceiver

SCS-SMS

System Design of v2.0

ran T-stick[+]
SDR platform

GUI using I and Q
from WinRad

SCS-SMS

SCS-SMS

SCS-SMS
with W.T.
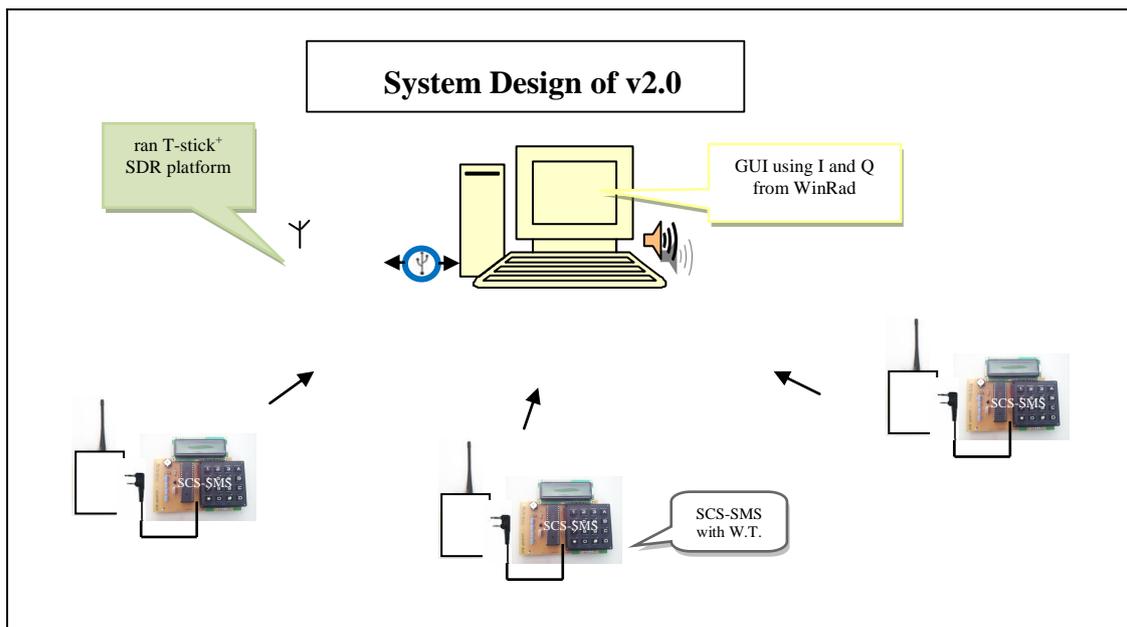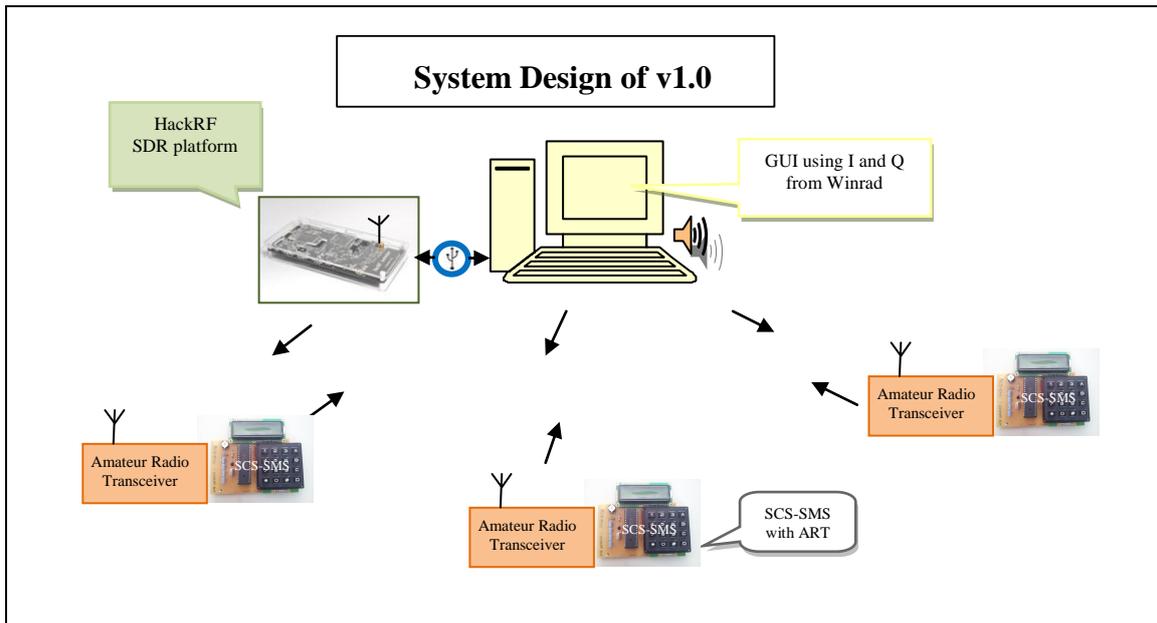
Fig. 5.1 A and B: System Overview

Literature

[HarckRF] …

http://en.wikibooks.org/wiki/Special:BookSources/0900612584

…