

Final Project

Practical engineering, electronics & SW

Please note that this is an auto-translated version of the original file, and as a result, the data may not be presented as clearly as intended

Smart home - IOT Controller



Moderator: Shay Malul		Presenter: Idan Nave	
Date: 06/06/2019		Cycle: 'Shavit' 2017-2019	



Smart Home - IOT Controller



table of contents

The wording of the original project proposal	3	1.
6		2.
Introduction - "The Internet of (Internet Of Things) "The things		
Concepts from the world of networks	7	3.
The seven-layer model - OSI.....	7	4.
9		5.
TCP/IP - Transmission Control Program		
9		6.
DHCP - Dynamic Host Configuration Protocol		
Network component 8266ESP.....	10.....	7.
8266ESP WiFi Component Characteristics	11.....	8.
Logical and electrical scheme	12.....	9.
14.....	low power technology - expansion	10.
The switches and LEDs on the controller.....	15.....	11.
16.....	Burning AT internal firmware Commands	12.
17.....	Communication with the ESP controller - 13. 'AT' commands.....	13.
18.....	List of commands, parameters and the controller's response	14.
26.....	Demonstration of the actions to set up server My home (access point)	15.
27.....	Demonstration of the operations to connect Serve as an end user	16.
28.....	The 'smart home' interface The application on the cell phone	17.
29.....	The 'smart home' screen interface The touch on the microcontroller	18.



Hardware and sensors used in the design	19.
..... 31.....	The 'smart home'
Integrated climate control sensor -	20.
38.....	DHT-22
Proximity range monitoring sensor -HC	21.
40.....	SR04
other analog components	22.
..... 42.....	In the project
Real time clock component - RTC	23.
- 43.....	DS 1307
i ² C touch	24.
screen protocol "3.95 TFT LCD	54.....
The means of development- microcontroller	57.....
JTAG and real-time development.....	58.....
The µVisionIDE work environment	28.
58.....	Keil
59..... Configuration Wizard 2 Electrical	29.
properties of the controller	60.....
..... 61.....	clock/oscillator
..... 63.....	Port
memory for connecting devices	64.....
..... 68.....	Timers/counters
UART communication	74.....
String functions - deepening.....	78.....
Summary and personal tone	79.....
..... 80.....	Piahbiblioger
81 The sourcecode for the project.....	39.



1. The wording of the original project proposal

**Mahat - the government institute
for training**



The national school
For the major engineers
for electronics

Subject description:

: main layers The project implements a homelighting system with 3

1. **Interior lighting** - represents domestic lighting rooms, realized through switches the application and on the board.
2. **Yard lighting** - RTC-based and allows lighting to be turned on and off according to the time of day (automatically).
3. **Parking lighting** - based on a proximity sensor, allows the lighting to be turned on and off with identification a person/vehicle

in the parking lot

The system is used through a **SmartPhone application** that is accessible to all residents of the house. The project implements

the developing trend these days of devices connected to the Internet **IoT** (the Internet of Things).

We emphasize that several consumers can use the system at the same time as long as they are connected to the same home network (via local web server).

How to develop and implement aggravation:

The project will be developed in the language. The project will be burned on the internal flash memory of a 380F8051C microcontroller of the company SiliconLabs. During the development stages, the following actions were performed:

- Comprehensive research about the world of IoT and the layers of information included in it, in particular the layers of local communication (ie Client-Server within a home network).
 - Writing code in the Keil uVision development environment that allows for an extensive DEBUG and simulation process.
 - Activation and programming of the RTC component and ultrasonic sensor using UART and SPI communication protocols.
 - Extensive use of third-party applications (Commands AT communication, application activation and dedicated setting).
- To the system- **Wizard Configuration** for initializing the controller etc.
- Activating smart lighting and planning different configurations for this lighting.

Project goals:

How realized	purpose
The project is based on a microcontroller based on an 8051 processor - which was studied in detail within the study material, and was also tested in the final exams of the MHT.	Recognition and study of a SiliconLabs processor
Implemented in the project for defining the desired features from the microcontroller's UART protocol, timers, port initialization, clock frequency setting, etc.	Running the Configuration Wizard software to determine the initialization of the processor's chips and units.
I was studied extensively as part of a programming laboratory in the curriculum, protocol C and the products we reached were incorporated into the project.	Recognition and implementation of I2C Serial BUS protocol - synchronous
The home server is available for the connections of several end users through an application installed on their cell phones. The information traffic from the moment it was received by the server is carried out between it and the microcontroller via UART.	Recognition and implementation of the asynchronous serial BUS protocol and activation of the URAT serial communication unit.
A wide deepening of the layer model and planning in light of the model was carried out, focusing on and connecting the client and the server in the lower layers related to IP address identification	Recognition and implementation of the Internet's layered model with an emphasis on the TCP protocol and WiFi communication between client and server
Realized in the project - we will learn a wide set of commands beyond what is required. Communication is implemented both manually (for DEBUG purposes) and as an independent state machine that listens to user requests.	Using a set of commands WiFi module to define AT Commands
The RTC component can set the time date, and allows an autonomous function of the smart home to turn on the parking lights starting in the evening	Connecting an RTC component that uses the protocol I2C
A network controller that hosts a home server, available for the connections of several users. Can be downloaded using , through an application installed on their cell phones, the application store on Android-enabled devices.	ESP network controller connection using the TCP protocol
How realized	Achievements beyond what is required
Use of distance, temperature and humidity sensors that sample in fast+slow mode and provide information about the weather, inventions at the absence of the home vehicle and the home pet	Expanding the capabilities of the smart home
In order to operate the new sensors, it was necessary to deepen their features through the manufacturer's pages, independent learning of serial communication within a single double data line.	Recognition and implementation of arbitrary protocols from manufacturer pages
The entire hardware design was carried out in an Altium work environment and was actually built on a WireUp board that allows for an aesthetic presentation and quick replacement of faulty components.	Planning and implementation of hardware in a professional manner



- 1 humidity and temperature sensor 22-DHT.
 - 2 ultrasonic proximity sensors - HC-SR04
 - 1 V5 fan.
 - A dedicated application on a device Android.
 - 8 normal LEDs 3 RGB type and
 - USB to TTL converter for development needs
 - 380F8051C microcontroller with upgraded 8051 core
 - Network module .01Ver 8266ESP.
 - TFT-LCD touch screen. 3.95
 - Voltage stabilizer 1117AMS V3.0
 - WireUp panel - unique wiring for the system.
- RTC DS1307 Controller**

Smart home features:

• **Development**- the system was developed in the **C Embedded environment**, it is easy to upgrade and add features according to the customer's order.
Also, the interface is 'tailored' to those and the sensors he chooses to use, and the running of the system can be monitored in real time, thereby locating faults. •

Connectivity - the system connects independent sensors to the "Things of Internet" world using the popular 8266ESP network module- which is a controller that can connect to the network within the reception range and in this case hosts a network serve itself.

• **Security**- the smart home **hermetically protects the privacy of the residents of the home** even in the event of a cyber incident and hacking into it, through a software monitoring system that is completely separate from the basic security of a password-based WiFi network. **Each tenant has a personal password that can only be used to perform remote operations on the network.** • **Lighting control** - the system allows **lighting control** throughout the house, turning on and off specific bulbs and receive an indication of the total number of lights that are on.

The home- the system allows monitoring the presence of **Monitoring the home**, parking lot and pet living area of objects near a sensor, and in the current implementation allows the presence of the home pet in the parking lot as well as the presence of the pet in the kennel - in these places these sensors must be installed.

• **Climate control** - the system allows monitoring the **temperature and humidity** in the space where the sensor was installed.

• **An autonomous lighting and air conditioning system**- the system allows the activation of parking lighting and a central air conditioning system in the house based on a specified time (for each) for example in the evening.

• **Interface**- The user interface is divided into two parts. **Local interface** through an LCD screen which is a touchscreen (similar to a cell phone) and built close to the controller. And a personal interface **for all the residents of the house using their cell phones** through an application that can be downloaded from the 'app store' on Android.

• **Withdrawal of information by a remote client** - each tenant can independently withdraw information from the home server and receive a status in real time of reading the sensors in the house.

• **Controlling the status of the home server** - the system allows you to **remotely disable** the home server, and to receive information on the **display** about the connected tenants in real time.

• **Displaying a message sent from the cell phone through the display** - the system allows all tenants to send messages from the cell phone to the home controller, which will be displayed on the screen according to

minutes - a resident of a house that is disconnected. Power consumption and duration operation - if no request is made on the network within 5 automatically. The network controller is **voltage stabilized**, and should reset only once about half an hour. The system requires a permanent connection (standard V230) using a standard transformer. 9 to a household electrical outlet

Internet of Things"(Things Of Internet)roduction - "The



The world of IOT has developed a lot in the last 5 years and includes within the **combination of technology, society and the economy**. Almost every component that accompanies our lives is connected or can connect to the Internet and thus add information to assessment and analysis tools that are aimed at changing the way we work, live and play.

In this era, not only computers can communicate with each other and exchange information, now **also components such as refrigerators, microwaves, lights and audio sensors** are able to function in this way, and this with as little intervention as possible on the part of the user.

In addition to the many personal applications that have developed in the field - such as smart homes, sports bracelets, and medical components (monitors), there are also uses and contributions in the public sphere such as smart transportation systems that know how to regulate traffic according to the load on the roads, and ecological systems that utilize natural resources for the benefit of the residents. From air conditioning saving on municipal electricity consumption.

The connection between the various components, systems and sensors in their type and nature is carried out on the basis of one of the infrastructures Back-End Data-Sharing, Device-to-Gateway , Device-to-Cloud, Device-to-Device: the following

: for example Of course, next to the contribution of IoT applications there are also weaknesses and shortcomings

Security - components and servers with weak protection and encryption circuits endanger their environment and especially the information passing through them which could be **routed for malicious purposes**. For example, there are components that are able to automatically connect to components that are already connected to the network, thus opening a window of opportunity for hacking, surveillance or deliberate slowing down (such as DOS attacks) in the entire network.

Apart from the personal commitment of the developers and users of the IoT network, **there are enforcement organizations whose sole purpose is policy development and the eradication and prevention** of these security deficiencies - these are capabilities that are often reserved for countries and corporations with their own interests, which find it difficult to create broad regulation. And that sparked for thought: "**Where is the switch to disable the internet?**".

Privacy - as soon as a component connects to the network, information of one kind or another is collected regarding the frequency of use, the history of the operations performed to databases that include additional identical data from other users. As mentioned at the beginning, analysis and evaluation of this information will ultimately lead to the optimization of the entire system and the user experience in particular. And therefore - the fact that **there is a violation of the user's privacy for the development of the network** cannot be ignored.

3. Important concepts from the world of networks

, in advance We will focus our project on the **Gateway Device infrastructure**. The infrastructure allows components that have not been programmed but have features and capabilities that enable a direct connection to the Internet - to make their way with the help of applications on the phone. The mobile and use of existing protocols such as HTTP/TCP.

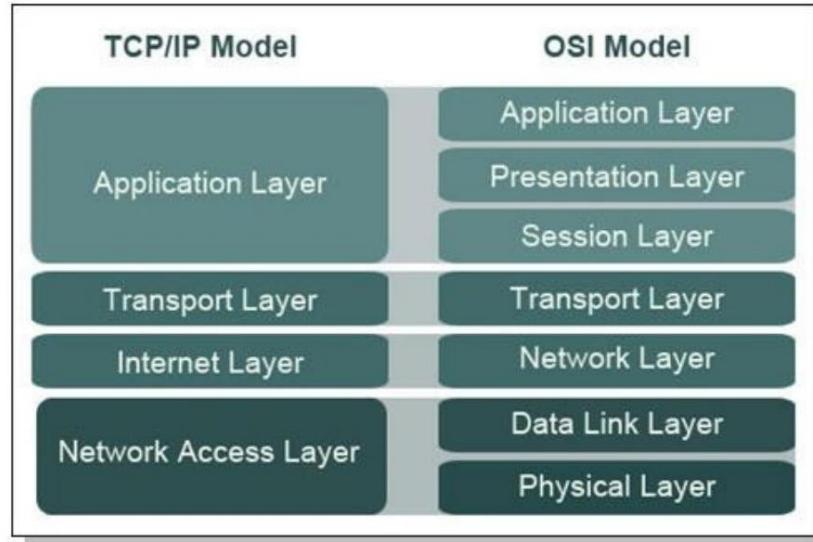
First we will explain that the concept of **protocol** is a collection of standards and rules that are important to follow in order to function efficiently, reliably and correctly transferring and exchanging information in the Internet space.

We will define the **Server Host** - as a server whose role is to handle the client's requests, from transferring and exchanging information with another client and a witness to transfer information from a webpage (which is on another server) to the client.

We will define the **Client** - as a client that requests to send information to a Host located in the environment far away from it for example from a page on the Internet.

In order to fully understand how information is exchanged between components, we will present the 'seven layers' model - Interconnect Systems Open OSI for communication on the Internet, and we will explain in detail about the IP/TCP model . From which the Transmission Control Protocol/Internet Protocol is derived

4. The seven-layer model - OSI



In the early 1980s, different networks consisted of hardware components from different manufacturers and different software familiar and adapted to their environment. This situation caused great difficulty in communication between computers from different networks.

As a result, the **OSI model** was developed by the International Organization for Standards (ISO). The model, like the protocol, defines a set of laws and rules for proper operation and reliable information transfer (without loss of information) between internet networks.



We will review the model hierarchically (from top to bottom):

1. **The application layer** - the layer closest to the client, where the client uses Gmail or posts a picture on Facebook.
2. **The presentation layer** - the layer responsible for how the information will be visually presented to the client. messaging (such as photo / video).
3. **The conversation layer** - the layer that manages the functions of starting or ending a call between two clients / servers. As soon as we stay connected to Gmail or Facebook even though we have moved to browse other sites we Basically using the conversation layer.
4. **The traffic layer** - in order to send information through the network, this layer divides the information into small packets (segments) and takes care of reassembling the information in an orderly manner on the other side (client / server). In addition, this layer where the exact identification is made between who the customer is and what is the application or service he is requesting, verification of the receipt of the information in the correct sequence and order and error checking for the various information packages is carried out.
5. **The network layer** - in order to identify the end users (clients / servers) this layer uses dedicated addresses on in order to route the information. A dedicated address, as it is more commonly known, an IP address with more than four octets. For example: server X requests to transfer information to computer at address 192.168.1.4. The network address is: 192.168.1 and in it i.e.,.. End users and their specific client address is: 192.168.1.4. For the purpose of the example, there are 255. The recipient is end user number 4.
6. **The link layer** - is responsible for linking the information between the end users and between the information packages themselves. Mean, all the information in the network passes, as mentioned, in small packets of information containing bits: 0 or 1. In this layer are the network cards and error control checks are performed at the bit level.
7. **Physical layer** - manages the movement of bits at the level of voltages and digital and analog signals that come from the end users and sets standards for frequencies (that is, how many bits will be transferred per second 1 Mbps or .) Mbps 20

In summary, the first four layers deal with the transfer of information between ends throughout the network. **three**
The last layers deal with providing service and routing the information between the applications. Below are the details of the protocols that operate within each of the layers - the protocols implemented in our structure will be emphasized:

HTTP, SMTP, FTP, RTP/IRC, SNMP, SIP, DNS, **DHCP** - the application layer.

MIME, **ASCII**, Unicode, SSL - the representation layer.

ASP, PPTP, SSH, NFS, RPC, SOCKS - the all layer

TCP, UDP, SCTP, DCCP - the traffic layer.

IP (IPv4, IPv6), ICMP, IPX, - the network routing layer.

Ethernet, Token ring, FDDI - the link layer

E1, 10Base-T, RS-232, DSL, SON - the physical layer

Now that we understand the seven layer model we can easily understand how two of the protocols work. The most significant in our design: IP/TCP and **DHCP**. One forms the **communication core** of the 01-ESP controller with all the customers connected to it, the second is important for the operation of the **application** that is installed on cell phones and is responsible for communication with the 'smart home' controller.



TCP/IP - Transmission Control Program 5

The data transfer control protocols built from only four layers meets the requirements and characteristics of the OSI model.

The link layer overlaps with the first and second layers in the OSI model and manages the data traffic through cards

the network

The Internetlayer - routes the traffic of packets throughout the network and uses additional protocols

Internet Control Message Protocol and Internet Protocol: such as the traffic layer - overlaps with the traffic and conversation layer in the OSI model and actually manages and routes the transfer of information to the application layer. Two important protocols operate in this layer: • **Protocol Control**

Transmission - TCP - This is a protocol for transferring information between two end users and it knows how to divide the information into the most optimal sizes, verify confirmation of each packet sent and report in case of failure.

- **User Datagram Protocol UDP** - this is a protocol for transferring information between two end users but it is not

As reliable as TCP, that is, there is no verification and confirmation of each packet transmitted between users.

- Synchronizes the use between the various applications, such as email or transferring a message. The app layer: for example, transferring photos, since each application has an adapted protocol for transferring information in the best way. There is a protocol called File Transfer Protocol (FTP), for sending mail there is a protocol called Simple Mail Transfer Protocol

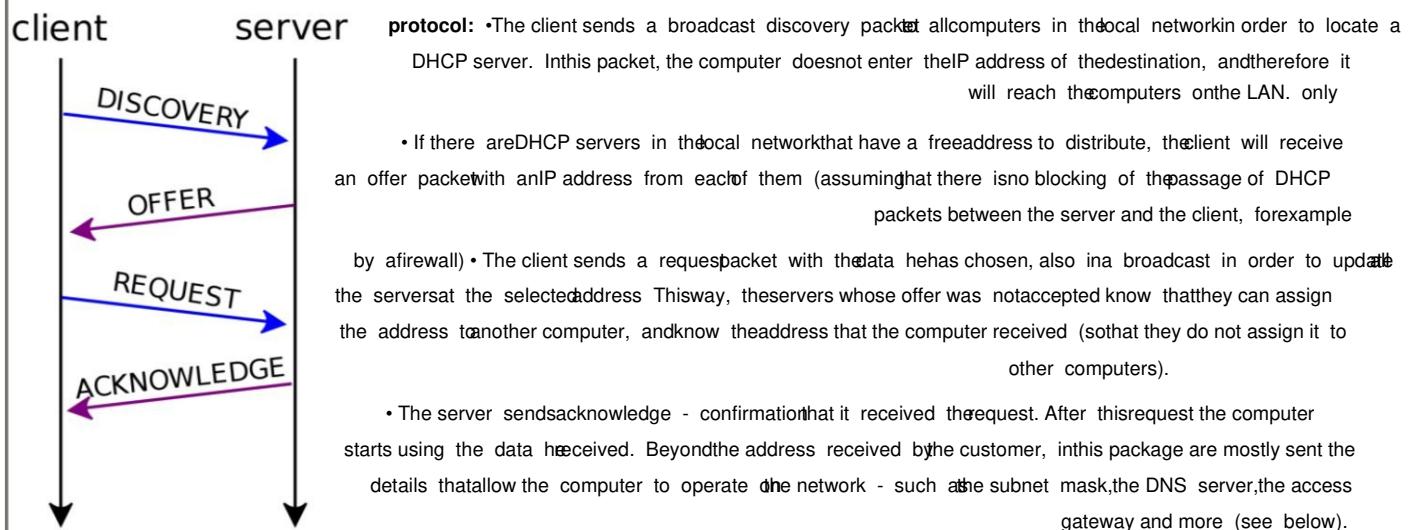
Simple Mail Transfer Protocol

DHCP - Dynamic Host Configuration Protocol 6

Dynamic Host Configuration Protocol is a communication protocol used to assign unique IP addresses to computers on a local area network (LAN) in addition to the IP address. A DHCP server will usually also provide the computer with data such as the subnet mask, DNS server address, and gateway address (so that the computer can start functioning in the network without the need for additional data).

DHCP functions in the application layer of the OSI model - and in the application layer of the TCP/IP model - it works above. As long as the client (client) and above the network protocol in IP (the UDP traffic protocol on ports 67-68 (server)) and has not received an IP address, he uses zeroes (0) to represent his address and is identified by the physical address.

The steps of the



7. Component 01-ESP from the 8266ESP Family

The communication component we used is a Wi-Fi module that allows microcontrollers to access the Wi-Fi network. This module is an independent controller (SOC), a system on a chip that communicates in the IP / TCP protocol and is able to host external network applications, and comes with a firmware that supports CommandAT - which are unique commands for communicating with it.

This means that the module can be used as an expansion card for an existing system but in addition it can be programmed to work as a microcontroller on its own. With the help of Wi-Fi communication, you can connect to the Internet as a client or alternatively host a web server with real web pages, thus connecting access points and connect to it with your smartphone.

There are many different modules available for this component, and many companies have even extended the existing configuration and different market it in the form of a complete development kit. Different controllers may have different pins, different Wi-Fi antennas or a different amount of flash memory on the board.

and development, The use of the 8266ESP component is very common among Arduino enthusiasts using the Arduino IDE environment. This environment is indeed very friendly for beginner developers, and users who are not looking professionally program microcontrollers.

However, we have chosen to 'sew' a unique system, without facilitating learning, while developing through language and environment - thus basically also summarizing the learned material. A job we got to know during our studies



It is important to note that 8266ESP is a general name for a group of chips that includes a series of components with additional characteristics and features suitable for development in different environments. Different manufacturers market development packages with varying features, but the hardware common denominators of the boards are: 3.3 V regulator for connecting the voltage, LEDs, antenna to extend the transmission range etc.

In light of the characteristics presented earlier, the 01-ESP model was chosen to be used by us in the project, mainly because we saw before our eyes the flexibility it offers in the structure that we intended to embed around it, taking into account that we will have to later suffer from possible instability and provide an answer to the current consumption of all the sensors that we intended to implement in the structure.

The low price compared to the functionality it offers was also taken into account, as well as the ability to use the component as a client connecting to a home server (in the project we use it as a server) something that will enable future expansion of the project.

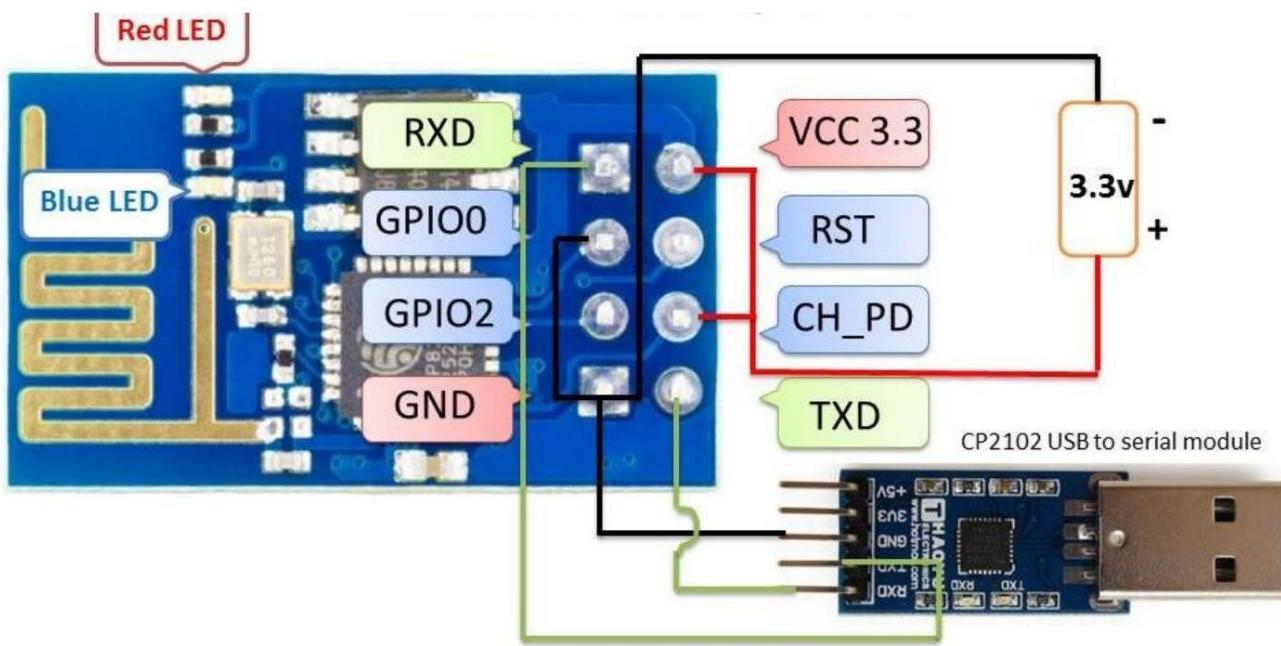
It is important to note that this is a final project with a limited time for its completion, and also the goals we sought to achieve were defined in advance so that they would be achievable, and the component allowed us to fulfill all of these - and to create a fairly independent home controller with a relatively low number of compromises.



8. 8266ESP WiFi component characteristics

meaning	explanation	bit32
advantage	In regressive operation MH80 which is not we workedwith Creates a bottleneckfor cattle 106L Tensilica RISC 32 8051 processor	processor characteristic
advantage	Power Supply:+3.3V only,Current Consumption: 100mA,I/O Voltage: 3.6V(max), I/Osource current: 12mA (max). These datamake the component suitablefor mobile devices, wearable electronics andIoT applications with extremely low power, whichtincludes Three operating modesactive mode,sleep modeand deep sleepmode. Thus,it is possible Adjust the home controllerto a'standby' working modewhen theusers The potentialschoose it.	architecture Energy saving
disadvantage	The controlleris able to operate consistently industrial environments,due to the range The wide range of temperaturesto which its adapted. It is alsocharacterized bytension Low activation, minimasize andlimited numberof minutes. However throughout The forumson the net doot lackdocumentation about instability in drawingcurrent and frustration A lot ofusers from his unexpectedactivity.	
allowing expansion future	kB 512that allow thestorage ofttemporary informationthat the component uses as well Storage of recentnetwork data to which thecomponent is connected. to assemblea system A small file storedn memory - System Flash SPIFFS (System. We do naise it, but itsexistence allows page storage Internet onthe component andthus run a fullweb server with HTMLpages written in advance.	Medium durability
advantage	The transmission ratfor uploading to ESP by default115200 baud but It can beprogrammed even 921,600. We preferretb usethe normal speedof light .Our 8051 Controller UARTcompatibility	 UART communication
Advantage + enables expansion future	Fusion in all the interfacesused todayby consumersand network products:three working modes.b / g / n / d / e / i / k / r support 802.11 different: Fi-Wstation, Fi-W access point - and a third mode - Both ifn Temporarily at theaccepted working frequency of 2.4 GHz. Hosting / connection capacity while maintaining commorencryptions suchas WPSand 2WPA / WPA . support	 WiFi workingmodes

9. Logicaland electricalschematic



The start of work with the component will be integrated into the learning and the protocols presented earlier, we did by direct connection to a laptop through a popular **TTL to USB type controller**, which appears in the lower right.

It can be seen that the component also provides external power to the component directly from the USB, and also the **2 data lines** of the USB are connected respectively to the 'send' and 'receive' lines which are the **Rx and Tx pins of the ESP controller**. In this way, you can send commands to it and define its work mode, which we will expand on later.

It should be noted that throughout the development we encountered **problems with the current consumption** of the ESP, which required us to supply it with external power through the TTL to USB component. This situation is **not ideal** because it requires the controller to be connected to a computer (or to a battery with a USB port), the final solution was to use a voltage stabilizer $V > 5 \text{ V } 3.3$ as will be shown later.

Each WiFi communication component contains a **transmitter + receiver** and the normal operation involves the transmission of high frequency radio waves, which is why we made sure to **calculate the total zero consumption** throughout the development and ensure compatibility with the design that may burden the features of the main controller.

The console, the core application we used, is common in maintenance ('open source') on a network called **Termite**. Throughout the book we will accompany the explanations with screenshots from it. An example of the AT commands (which will be explained later) that we used to send to the controller:

During the development we used the auxiliary connection with a laptop in order to watch in real time the traffic received both in the network controller and in the main controller:



Termite 3.3 (by CompuPhase)

COM4 115200 bps, 8N1, no handshake Settings Clear

```

AT
OK
AT+GMR
AT version:0.25.0.0(Jun 5 2015 16:27:16)
SDK version:1.1.1
Ai-Thinker Technology Co. Ltd.
Jun 23 2015 23:23:50
OK

```

The connection of the portable USB converter to the computer is wired to the controller on the development board. A module is connected to the controller through the network. This is how 'three-way' communication is carried out:

Sending orders , 1. Computer communication by network controller - pama surgical workserver in Karet against the network, specificity - are displayed on the console and learning the controller's response. An important process in the phase of learning the component. The responses

2. Network controller communication by Microcontroller 8051 - essentially enables the build of the smart home controller - independent work of the controller (using the software built into it) in front of the network module, without human intervention - - For oil it is intended except for the needs of requests

from the server3. Microcommunication - 8051 controller by computer was mainly used for DEBUG needs in the first stages of development - for listening to the strings produced by the microcontroller with the UART - and displaying them on the console

It goes without saying that the normal operation of this connectivity requires a common talk of all three at the rate of sending data identical - a figure that must be defined / verified in advance. For example, the rate was set to 115,200 BAUD on the console, the microcontroller, and for the network controller it is a default - but the command to change is as described:

Serial port settings

Port configuration		Transmitted text	
Port	COM4	<input type="radio"/> Append nothing	<input type="radio"/> Append CR
Baud rate	115200	<input type="radio"/> Append LF	<input checked="" type="radio"/> Append CR-LF
Data bits	8	<input type="checkbox"/> Local echo	
Stop bits	1	Received text	
Parity	none	Polling	100 ms
Flow control	none	Max. lines	
Forward	none	Font	default
		<input checked="" type="checkbox"/> Word wrap	
User interface language		English (en)	

Command AT+UART_CUR? will return the actual value of UART configuration parameters, which may have allowable errors compared with the set value because of the clock division.

For example, if the UART baud rate is set as 115200, the baud rate returned by using command AT+UART_CUR? could be 115273.

1. The configuration changes will NOT be saved in the flash.
 2. The use of flow control requires the support of hardware:
 - MTCK is UART0 CTS
 - MTDO is UART0 RTS
 3. The range of baud rates supported: 110~115200*40.
- AT+UART_CUR=115200,8,1,0,3



10. As the power supply technology- expansion

The 8266ESP enables relatively low current consumption for its purpose - especially for portable devices such as electronics. This option is achieved by switching it between three modes: **'active mode'**, **'sleep mode'** and **'deep sleep mode'**.

The controller can be programmed to wake up within a certain period of time by using RTC. The component will wake up in a manner automatic according to the set time. This feature can be applied to SOC for mobile devices, and to optimize more the average current consumption of the system.

If the programmers refer to the **power of the network detected** by the component (existing option), it is also possible to determine whether addition, **turn off/on the component** in the applied software, thus maintaining a continuous connection only with networks with reception favor.

, 3.3 The following data is based on manufacturer data, and was tested (according to this commitment) by connecting an external V power supply to WiFi and 90% cycle time (CD). Continuous transmission mode °C antenna with connection at an ambient temperature of 25 can be observed different current consumption depending on the strength of the communication and the transmission protocol used, but the value . with us. The typical one is about that which is also measured

Mode	Typical	Units
802.11b, CCK 1Mbps, POUT = +19.5 dBm 802.11b,	215	mA
CCK 11Mbps, POUT = +18.5 dBm 802.11g, OFDM	197	mA
54Mbps, POUT = +16 dBm 802.11n, MCS7, POUT = +14 dBm 802.11b, packet size of 1024 bytes, -80 dBm 802.11b, packet size of 1024 bytes, -70 dBm 802.11b, packet size of 1024 bytes, -65 dBm	145	mA
Standby Deep sleep Saving mode DTIM 1 Saving mode DTIM 3	135	mA
Shutdown	60	mA
	62	mA
	0.9	uA
	10	mA
	1.2	mA
	0.86	mA
	0.5	uA



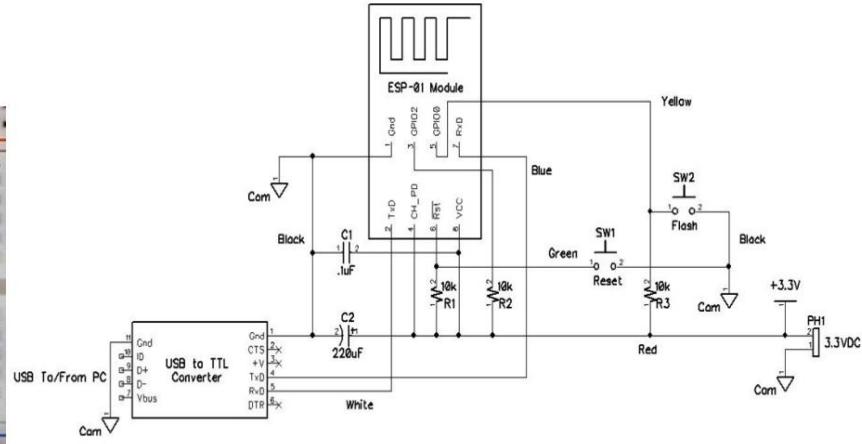
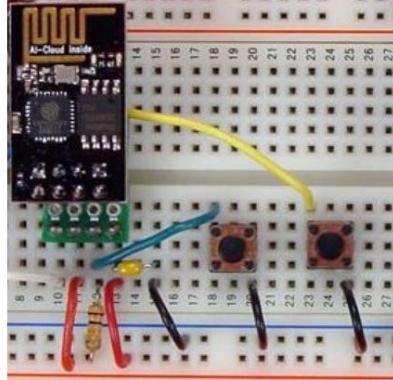
11. Andfasten LEDs on the controller

trigger	name	alias	Description ofuse	Additional uses
1	Ground	-	Permanently connected to theground ina way	-
2	TX	GPIO – 1	Used as a TX data trigger	Can be usedas a trigger for use General, ifnot used as a trigger TX information
3	-2GPIO	-	The trigger for general usefor connecting O/I	-
EN_CH 4		-	devices is permanently connected toDV voltage 3.3 V - forthe purpose ofenabling the component	-
5	GPIO -0	Flash	General purpose triggerfor connecting devices O/I	'holding' the triggein a logic state low for a fewseconds Puts the controller into position Firmware burning, as will be demonstrated in thenext episode
6	Reset	-	Connecting it toground resets thecomponent and deletes alprevious settings carried out in it. The component willimmediately load defaultsettings defaults tokeep the memoryof the FLASH, or settings configured as such Let them be loaded from memory thenext session.	-
7	3 - GPIO RX		Used as anRX datatrigger	Can be usedas a trigger for use General, if not used as a trigger RX information
8	Vcc	-	Connected toDV voltage 3.3 V permanently	It is importantto emphasize thatthe board receives A voltage of V 3.3 and if youmistaken and connect it toV 5 actually burn it.
LEDs	PWR Red		Emitters a constanted light as longs thecontroller receives power	-
LEDs	TXD Blue		Distributes a single pulse during Reset	-

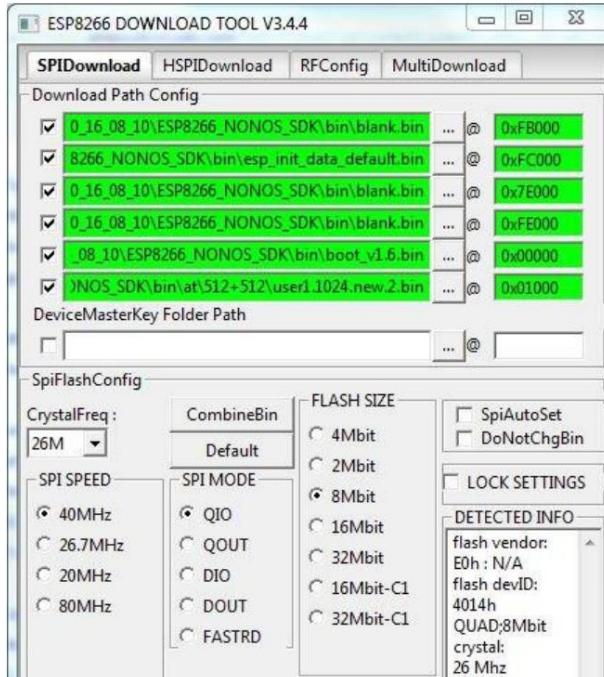


12. Burning internal firmware Commands AT

As mentioned, connecting the component and starting work is a simple matter, although it should be noted that the firmware installed on the component when we received **was not up-to-date**, so the connection process was preceded by the above for **burning the most recent version available on the network**, which is from **August 2018**. For the purpose of burning, we were required to connect the burning connection component as described above, which included **adding the RST and PROGRAM buttons below:**



Finally we flashed the latest firmware using a guide we found online:



[How to Flash ESP-01 Firmware](https://www.allaboutcircuits.com/projects/update-the-firmware-in-your-esp8266-wi-fi-module/)

Update the Firmware in Your ESP8266 Wi-Fi Module

October 28, 2015 by Charles R. Hampton

Objective and Rationale

This is the author's second article about the ESP8266 integrated circuit, a relatively new chip comprising a full-featured 32-bit RISC µC and a built-in 802.11 b/g/n Wi-Fi circuit. The [first article](#) described using the Arduino IDE to program the ESP8266, and contains some important background information that will not be repeated here. If you haven't read it, please do.

There is no shortage of information on the Web about the 8266; in fact, there may be too much...of the wrong type. The developers of the IC, [Espressif](#), have apparently chosen to not only outsource manufacturing, but



13. Communication with the ESP controller 'AT' commands

The 'Commands AT' command system is **command** language -unlike a conventional **programming** language . The commands are based They are based on a communication method **previously called 'set command Hayes'** which was originally developed by Dennis Hayes, who founded and ran a company to create modems in the 1970s. -80 The **wA&T** is an abbreviation of **Attention** - from the component That is, **receiving attention**

The command line consists **of series of short text strings** than can be combined to form executable commands and a definition of modems of the type used in the 80s and 90s (which were connected to the telephone line **using dial-up method**). This system expanded as more manufacturers included it, but kept **the basic syntax**.

Modems are typically run on direct telephone lines that always begin and end with a known modem at each end. , but the microcomputer evolution The modems operated in "source" or "answer" mode for small network of computer customers. The 1970s led the introduction of low-cost modems and the idea of a **point-to-point link was no longer adequate**, since **hundreds of thousands of users** might want to talk to each other.

However, this method of communication remains popular, and is implemented in the component we have chosen since it is very easy to manipulate on strings that are received and sent to the component, and also the expected communication volume for IoT applications of the same type as the target audience of 8266ESP is limited anyway. (developers, students)

Below is the collection of commands relevant to the smart homecontroller project that we implemented, this collection is an extract from a database of commands we used wider, most of which was irrelevant for us. Below is a general breakdown of the types of

commands	type
AT+RST, AT, AT+GMR, ATE, AT+GSPL	Basic
AT+CWMODE, AT+CWJAP, AT+CWLAP, AT+CWQAP, AT+CWSAP, AT+CWLIF, AT+CWDHCP, AT+CIPSTAMAC, AT+CIPAPMAC, AT+CIPSTA, AT+CIPAP	WiFi layer
AT+CIPSTATUS, AT+CIPSTART, AT+CIPSEND, AT+CIPCLOSE, AT+CIFSR, AT+CIPMUX, AT+CIPSERVER, AT+CIPMODE, AT+CIPSTO, AT+CIUPDATE, +IPD	TCPIP Layer

Syntax and use of commands:

Similar to strings, the commands contain repeating parameters and rules, for example at the end of each command there is to send a termination line termination line that the 8266ESP component expects. The syntax for this ending is: . LineFeed and CarriageReturn meaning >CR><LF>

while a command that requires ,A command that stands on its own (without parameters) will be sent as it is with the ending string.The transfer in advance of information in the form of parameters will be sent when the sign '=' is attached to, followed by the information in a defined format. For each command, we have attached the answer received from the component, which usually will be shown below. Also, success/failure of its execution.



14. Details of commands, parameters and controller response

- Initial boot

AT -Test AT startup

Variant Command Response Function		
Execute AT	OK	Test if AT system works correctly

- Secondary boot

AT+RST -Restart module

Variant Command Response Function		
Execute AT+RST	OK	Reset the module

ESP-01 Output after reset:

Jan 8 2013,rst cause:4, boot mode:(3,7) wdt reset
load

0x40100000, len 24444, room 16 tail 12
checksum

0xe0 ho 0 tail
12 room 4 load

0x3ffe8000, len 3168 , room 12 tail 4

checksum 0x93

load 0x3ffe8c60, len 4956, room tail 8

checksum 0xbd
csum 0xbd

ready

- Print firmwareversion installed on the component

AT+GMR -View version info

Variant Command Response Function		
Execute AT+GMR	version, OK	Printfirmware version

Parameters:

- version: firmwareversion number

ESP-01 output:

00160901

- Putting the component into'sleep' mode

AT+GSLP -Enter deep-sleep mode

Variant Command	Response Function
set	AT+GSLP=time timeOK Enter deepsleep mode for time milliseconds

parameters:

- time: Time to sleep inmilliseconds

**Example:**

- AT+GSLP=1500

Note:

- Hardware has to support deep-sleep wake-up (Reset pin has to be High).

- Enable or cancel receiving an ECHO of the command we sent before outputting the answer**

ATE - Enable / Disable echo

Variant	Command	Response	Function
Execute	ATE0	OK	Disable echo (Doesn't send back received command)
Execute	ATE1	OK	Enable echo (Sends back received command before response)

- Setting work mode server, client or both at the same time**

AT+CWMODE - WIFI mode (station, AP, stationAP)

Variant	Command	Response	Function
Test	AT+CWMODE=?	+CWMODE:(1-3) OK	List valid modes
Query	AT+CWMODE?	+CWMODE: mode OK	Query AP's info which is connected by ESP8266.
Execute	AT+CWMODE=mode OK		Set AP's info which will be connected by ESP8266.

Parameters:

- mode: An integer designating the mode of operation either 1, 2, or 3.
 - 1 = Station mode (client)
 - 2 = AP mode (host)
 - 3 = AP + Station mode (Yes, ESP8266 has a dual mode!)

- Connecting to the access point using a name and password known to us**

AT+CWJAP - Connect to AP

Variant	Command	Response	Function
Query	AT+CWJAP?	+ CWJAP:ssid OK	Prints the SSID of Access Point ESP8266 is connected to.
Execute	AT+CWJAP=ssid, pwdOK		Commands ESP8266 to connect an SSID with supplied password.

Parameters:

- ssid: String, AP's SSID
- pwd: String, not longer than 64 characters

Example:

AT+CWJAP="my-test-wifi","1234test"

**Example AT+CWJAP?ÿ**

+CWJAP:"my-test-wifi"

- Print all the communication networks that are in the reception range

AT+CWLAP - Lists available APs

Variant	Command	Response	Function
Set	AT+CWLAP=ssid,mac,ch +CWLAP:ecn,ssid,rssi,mac OK	Search available APs with specific conditions.	
Execute AT+CWLAP	AT+CWLAP:ecn,ssid,rssi,mac OK		Lists available Access Points.

Parameters:

- ECN:
 - o 0= OPENo 1 = WEP o2 = WPA_PSK o3 = WPA2_PSK o4 = WPA_WPA2_PSK
- SSID of AP rssi: signal strength •
- mac: String, MAC address

Example AT+CWLAP:

```
+CWLAP:(3,"CVBJB",-71,"f8:e4:fb:5b:a9:5a",1)
+CWLAP:(3,"HT_00d02d638ac3",-90,"04:f0:21:0f:1f:61",1)
+CWLAP:(3,"CLDRM",-69,"22:c9:d0:1a:f6:54",1)
+CWLAP:(2,"AllSaints",-88,"c4:01:7c:3b:08:48",1)
```

- Disconnecting from the access number were currently connected to

AT+CWQAP - Disconnect from AP

Variant	Command	Response	Function
Execute AT+CWQAP	OK		Disconnect ESP8266 from the AP it is currently connected to.

Note:

After running this command, if you run AT+CWJAP? it still shows the AP you were connected to before.

- Setting server status access point hosted by the rival and setting name and password

AT+CWSAP - Configuration of softAP mode

Variant	Command	Response	Function
Query	AT+CWSAP?	+CWSAP: ssid,pwd,ch,ecn OK	Query configuration of ESP8266 softAP mode.
Set	AT+CWSAP=ssid,pwd,ch,ecn OK		Set configuration of softAP mode.

**Parameters:**

- ssid: String, ESP8266's softAPSSID
- pwd: String, Password longerthan 64 characters
- id
- ECN:
 - 0= OPEN**
 - = WPA_PSK**
 - 3 = WPA2_PSK**
 - 4 = WPA_WPA2_PSK**

Example

AT+CWSAP="esp_123","1234test",5,3
 AT+CWSAP? => +CWSAP:"esp_123","1234test",5,3

- Printing all the clients(Clients) who are connected to the accesspoint wehost**

AT+CWLIF -List clientsconnected to ESP8266 softAP

Variant	Command	Response	Function
	Execute AT+CWLIF [ip,other]	OKList information	on of connected clients.

Parameters:

ip: IPaddress of a client connected to theESP8266 softAP other: Otherinfo, look at example. don't know what it means yet.

Example:

AT+CWLIF

192.168.4.100,3fff50b4:3fff50ba:3fff50c0:3fff50c6:3fff50cc:3fff50d2

OK

- EnableDHCP as explained in thechapter 'The Internet Layer Model'**

AT+CWDHCP - Enable/Disable DHCP

Variant	Command	Response	Function
Set	AT+CWDHCP=mode,en	OK	Enable or disableDHCP for selected mode

Parameters:

- mode:
 - 0 : set ESP8266 as a softAP**
 - 1 : set ESP8266 as a station**
 - 2 : set both softAP and a station**
- en :
 - 0 : Enable DHCP**
 - 1 : Disable DHCP**

Note:

This command doesn't seemto work on firmware 00160901 (ESP-01) nor 0018000902-AI03 (ESP-12).



- Receiving general information about the current connection status

AT+CIPSTATUS - Information about connection

Variant	Command	Response	Function
Test	AT+CIPSTATUS=? OK		
Execute	AT+CIPSTATUS STATUS: status + CIPSTATUS: id, type,addr, port, type OK		Get information about connection.

Parameters:

- status:
 - o 2: Got IP
 - 3: Connected
 - 4: Disconnected
- id: id of the connection (0~4), for multi-connect type

String, "TCP" or "UDP"

- addr: String, IP address.
- port: port number
- type: 0 =

ESP8266

runs as client 0 = ESP8266 runs as a server

- Establishing TCP communication in the case of a server or UDP in the case of a client

AT+CIPSTART - Establish TCP connection or register UDP port and start connection

Variant	Command	Response	Function
Set	AT+CIPSTART=type addr, port	OK	Start a connection as client. (Single connection mode)
Set	AT+CIPSTART=id,t ype, addr, port	OK	Start a connection as client. (Multiple connection mode)
Test	AT+CIPSTART=?	[+CIPSTART:]id()“type “(,)”ip address“(,)port() OK	(List possible command variations)

Parameters:

- id: 0-4, id of connection
- type: String, "TCP" or "UDP"
- addr: String, remote IP
- port: String, remote port



• Preparation for sending free strings (like SMS 2)- possible modes

AT+CIPSEND - Send data

Variant	Command	Response Function	
Test	AT+CIPSEND=?	OK	
Set	AT+CIPSEND=length SEND	OK	Set the length of the data that will be sent. For normal sending (single connection).
Set	AT+CIPSEND=id, length SEND	OK	Set the length of the data that will be sent. For normal sending (multiple connection).
Execute	AT+CIPSEND		Send data. For unvarnished transmission mode.

Parameters:

- id: ID no. of transmit connection
- length:
data length, MAX 2048 bytes

• Disconnecting TCP communication in the case of a server or UDP in the case of a client

AT+CIPCLOSE - Close TCP or UDP connection

Variant	Command	Response Function	
Test	AT+CIPCLOSE=? OK		
Set	AT+CIPCLOSE=id OK		Close TCP or UDP connection. For multiple connection mode
Execute	AT+CIPCLOSE OK		Close TCP or UDP connection. For single connection mode

Parameters:

- id: ID no. of connection to close, when id=5, all connections will be closed.

Note:

- In server mode, id =5 has no effect!

• Obtaining a local IP address of the controller

AT+CIFSR - Get local IP address

Variant	Command Response	Function
Test	AT+CIFSR=? OK	
Execute	AT+CIFSR +CIFSR:ip OK	Get local IP address.

Parameters:

- ip: IP address of the ESP8266 as a client.

Example AT+CIFSR:

10.101.10.134



- Enable the number of connections allowed to connect to the access point or a single connection in case of client mode

AT+CIPMUX - Enable multiple connections or not

Variant	Command	Response	Function
Set	AT+CIPMUX=mode OK		Enable / disable multiplex mode (up to 4 connections)
Query	AT+CIPMUX?	+CIPMUX:mode OK	Print current multiplex mode.

Parameters:

- mode: 0: Single connection
- 1: Multiple connections (MAX 4)

Note:

This mode can only be changed after all connections are disconnected. If server is started, reboot is required.

- Server initialization is required after setting mode

AT+CIPSERVER - Configures server

Variant	Command	Response	Function
Set	AT+CIPSERVER=mode[,port] OK		Configure ESP8266 as server

Parameters:

- mode: 0: Delete server (need to follow by restart)
- 1: Create server
- port: port number, default is 333

NOTE:

1. Server can only be created when AT+CIPMUX=12. Server monitor will automatically be created when Server is created.
3. When a client is connected to the server, it will take up one connection, beginning an id.

- Setting parameters for string sending mode

AT+CIPMODE - Set transfer mode

Variant	Command	Response	Function
Query	AT+CIPMODE?	+CIPMODE:mode OK	Set transfer mode, normal or transparent transmission.
Set	AT+CIPMODE=mode OK		Set transfer mode, normal or transparent transmission.

Parameters:

- mode:
- 0: normal mode
- 1: unvarnished transmission mode



- Setting a period of time to disable the server

AT+CIPSTO - Setserver timeout

Variant	Command	Response	Function
Query	AT+CIPSTO?	+CIPSTO:time Query	Server timeout.
Set	AT+CIPSTO=time OK		Set server timeout.

Parameters:

- time: server timeout, range 0~7200 seconds

- Server output along with information received from it

+IPD - Receive network data**Variant Command Response Function**

Execute +IPD,len:data Receive network data from a single connection.

Execute +IPD,id,len:data Receive network data from multiple connections.

Parameters:

- id: id no. of connection
- len: data length
- data: data received



15. Demonstration of the operations for setting up a home server (access point)

In order to set up a home server to which all end users (residents of the house) can connect, the controller must be configured using a sequence of commands as described above, but while maintaining a certain order and inserting correct parameters.

Of course, we do not intend to bother the residents of the house to enter the AT commands to start the server, **even if**

And therefore the collection of commands that will be presented below has been embedded in the source code, **The smart home** that they want to use the controller, as part of functions that run independently when activating the 8051 controller through which the design is implemented. **in the software**

The commands were embedded in a state machine that the controller initializes and activates in an infinite loop and so throughout the duration of its operation the smartphone, and is ready to present them. **The controller listens to requests from the app users** making an interface on **the local monitor** - with which all operations can be performed even without the cellphones.

: Multiple Connections as TCP Server on a multi-use hosting server

To allow several residents of a house to connect to the server at the same time, we will set the network controller to the working mode of an access point. There is Note that there is a **limit of up to 4 clients** that are connected at the same time - a limit that comes from the component's hardware (limited memory). Details about each of the commands can be found in detail in the AT commands chapter:

2. Setting the station mode: 3 = CWMODE + AT (softAP), 1. **Initializing the component** - operating commands

The controller's response: confirmation basicity.

: Controller response , 3. **Allow multiple connections**: 1 = CIPMUX+ AT

confirmation We will sometimes receive ownership strings

Controller response: , 4. **Creating a TCPserver**: 1= CIPSERVER + AT

'Garbage' values - normal condition that instructs on

Shred the beef.

confirmation

The image shows two side-by-side windows of the Termite 3.4 serial terminal. Both windows have the title 'Termite 3.4 (by CompuPhase)' and show a connection to 'COM7 115200 bps, 8N1, no handshake'.

Left Window (Configuration):

- Shows the 'Settings' tab selected.
- Displays the following configuration parameters:
 - 2nd boot version : 1.5
 - SPI Speed : 40MHz
 - SPI Mode : DIO
 - SPI Flash Size & Map: 8Mbit(512KB+512KB)
 - jump to run user1 @ 1000
- Shows a large amount of raw hex data starting with '\n[03]x[1B],\r\nrf\n[0C][0C][0C][0C][04][0C][04][03],x([1]>C]1]\r\nB,[0C][04][0C]\$`[02]`Tr^p,[0C][02]\r\n1F][00][0C]\$`[02]`([12)o[02]`xc[04],`b([0C]cp`>)\r\n`c[12]`\$cvn\n1][02]\r\nB]on03][00][03][10]\r\n,`p[03][04][0C],[0C]\$[04][04][0C][1F]\$(`[03)o`[00],`xc[04]\r\n[04]b[04][03][02]`[0C]\$[0C],`bpr\$\r\nAi-Thinker Technology Co. Ltd.
- Shows the command 'AT+CWMODE=3' followed by 'OK'.
- Shows the command 'AT+CIPMUX=1' followed by 'OK'.
- Shows the command 'AT+CIPSERVER=1' followed by 'OK'.

Right Window (Boot Log):

- Shows the 'About' tab selected.
- Logs the following boot process:
 - ready
 - AT
 - OK
 - AT+RST
 - OK
 - ets Jan 8 2013,rst cause:4, boot mode:(3,6)
 - wdt reset
 - load 0x40100000, len 1856, room 16
 - tail 0
 - checksum 0x63
 - load 0x3ffe8000, len 776, room 8
 - tail 0
 - checksum 0x02
 - load 0x3ffe8310, len 552, room 8
 - tail 0
 - checksum 0x79
 - csum 0x79
- Shows the '2nd boot version : 1.5' and 'SPI Speed : 40MHz' at the bottom.



16. Demonstration of the operationshang companies on the server as an eror

3. It is important to note that the IP address You can see the web page using the cell phone. Another option is to send the AT+CFSR+AT command via the computer, and receive the information via the console screen.

```
Termite 3.4 (by CompuPhase)
COM7 115200 bps, 8N1, no handshake

OK
AT+CIPSERVER=1

AT+CFSR

busy p...

OK
AT+CWLIF

OK
AT+CFSR

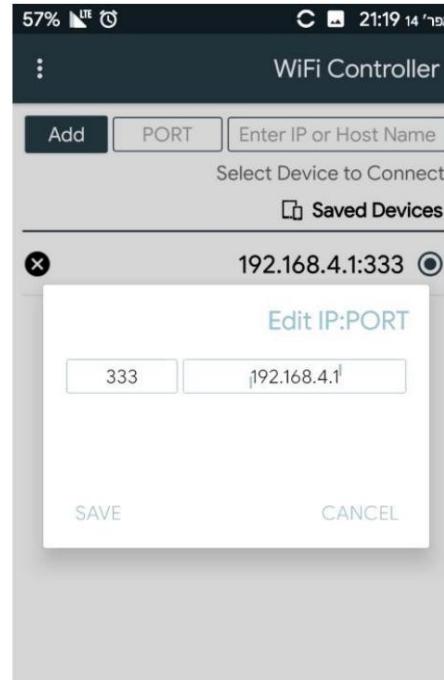
+CFSR:APIP,"192.168.4.1"
+CFSR:APMAC,"5e:c7:ff:fd:19:17"
+CFSR:STAIP "0.0.0.0"
+CFSR:STAMAC,"5c:c7:ff:fd:19:17"

OK
0.CONNECT

+IPD,0,18:Send-This-String
AT+CIPSEND=0,50

Send-Another-String
```

2. Using a WiFi application
Controller - connect to the TCP server that the server is currently hosting (IP must be entered once and the settings are saved for the next time).



1. Activating WiFi transmission and connecting to the network on the access cell phone that the component distributes. ESP8266 softAP



In fact, the actions presented here are one-time, since the WiFi network we connected to is saved in the cellphone for future uses, and the IP address is also saved in the application to connect the next time.

It is important to note that the Timeout mechanism controls when the 8266ESP works as a TCP server. If a TCP client connects to the TCP 8266ECP server, but no request was received within 5 minutes - the server will terminate the connection with the timeout. To avoid such problems, a data transfer cycle should be set every 5 seconds. We did not make this setting in order to preserve the battery life time in cell phones and prevent data traffic that burdens the home network.

Of course, everything mentioned here is intended to present the communication protocol between the server, the end users, and the controller. 8051 The intervention in the transfer of the information in the console - intended to illustrate the processing of the commands presented in the previous chapter.

The network, as will be shown in the next chapter, is a direct result of an intermediate state in which a machine is located to the controller, sending the commands to the states of the microcontroller, that is, it is automatic - and the result of all end-user requests that constitute a trigger for changing the states of the machine.

Disabling the server, and terminating the connection proactively, is also a possible remote operation using a button in the application that holds the Kill command. After it has been sent and decoded in the software, the string AT 0= CIPCLOSE+ " is sent to the network controller. This option is also provided through the local system (the touch screen) attached to the microcontroller.



On the cell phone 17the 'smart Home' - arinterface application

3. The information saved on the server is transmitted

Also automatically via the UART protocol to the 8051 controller (because the ESP is wired to the Rx and Tx protocol lines). This can be

proven by listening to the traffic on the server using the console. You can see

an initial response to connecting to the server and immediately after receiving the string that was sent.

```
+CIFSR:APIP,"192.168.4.1"
+CIFSR:APMAC,"5e:cf:7f:fd:19:17"
+CIFSR:STAIP,"0.0.0.0"
+CIFSR:STAMAC,"5c:cf:7f:fd:19:17"

OK
0.CONNECT

+IPD,0,18:Send-This-String
AT+CIPSEND=0,50
```

AT + CIPSEND = 0, n. At the end of the command, the controller expects to receive strings that are transmitted at the rate at which it is set (BAUD selection). Indeed, a default of 115,200 will be received - marking of input ">". At the end of the sending, the controller's response: n RECV Bytes confirmation of bytes that we were notified in advance about sending. If the number of bits received is different from the defined size (n): the controller will reply "Busy" to send only the n bytes.

After them he will inform that was actually successful 'OK.'

```
> +CIPSEND=0,20
```

```
busy s...
```

```
Recv 50 bytes
```

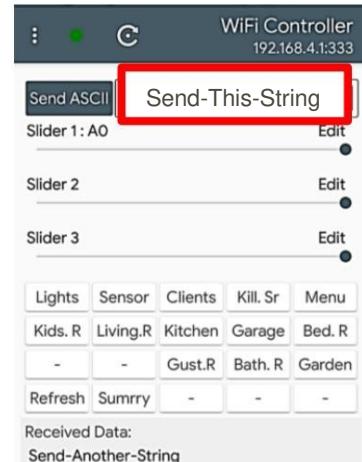
```
SEND OK
```

```
Send-Too-Long-String
```

2. A button interface now appears

and sending text to the server (also the buttons are actually trigger for sending a string and are changeable).

Choosing one of the sending options (text/button) will publish the string on the server.



1. Upon connecting to the server

The indicator in the application will change from 'red' to 'green'.



5. Information sent from the server to the user

will be printed at the bottom of the application. The source of this information is in

the 8051 microcontroller which sends users feedback about the state of the sensors in the house. This closes a circuit that began with the user's request for information from the network, and the return of the information to him from the network as well. Data reception: when the ESP received data from a client, it generates

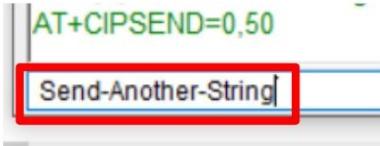
4. Conversely, the server also publishes information to clients connected to it.

In order to trace the information, we will start a manual attempt to send a string using the console as if the origin of the string is in the software from the 8051 microcontroller.)

message as follows: that is, IPD,0, n:

xxxxxxxxxx + the controller received n bytes the information itself =

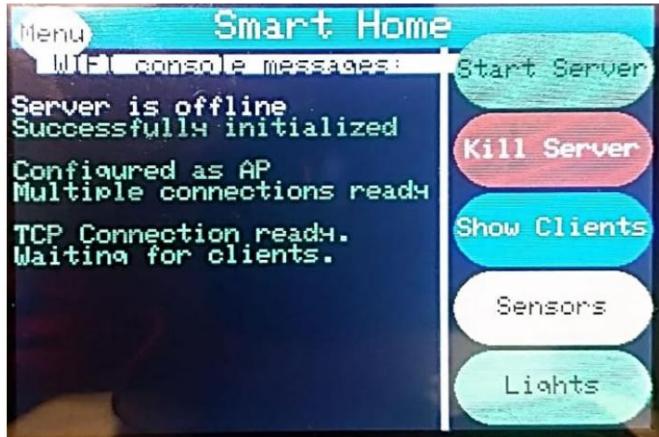
xxxxxxxxxxxx





18. The 'smart home' - touchscreen interface onthe microcontroller

A major part of the project is also realization of local interface that is accessible to the residents of the house and makes itpossible to receive all the information comingfrom the sensors in onplace, withoutthe need foran Internet connection . Theimplementation is on touch screen - in the form of astatus menu,with a fixed structurefor each page, andthe use of buttons for operation.



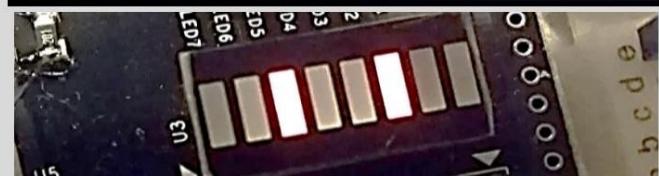
Main screen, appears when the microcontroller boots. In the menu on the right are buttonsthat enable basic operations suchas activating and deactivating the home server.

You canalso watchall theclients that are connected to the server in reale. Buttons for 'Lighting' and 'Sensors' allow a transition to theub-menu related to them. On theleft side there is amulti-console Let's use thatshows the informationtraffic and the commands sent the server.



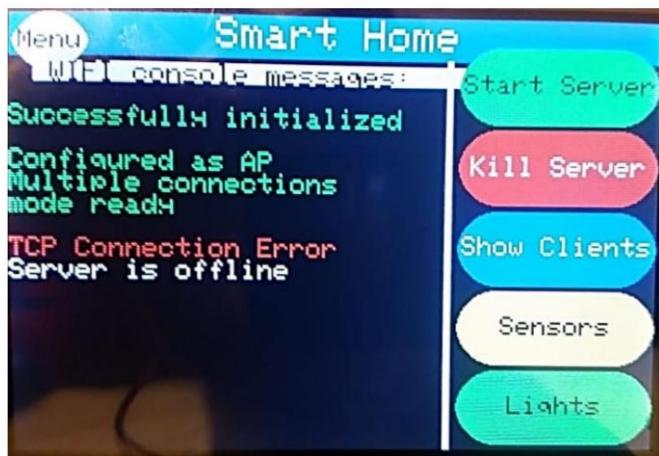
Home lighting controlscreen - enables turningon and off upto 8light bulbs in different rooms in the house. Turning on lighting turns the button green, and turning it off turns it back tored. A counter at the bottom of the screengives an indication of the total number oflights on - andthe mainuse of the messageat the bottomof the screenis to holda string that wilbe sent to the end users' celphones .

LEDs onthe development board represent the lights of the house.Activating lightsin the lighting menu causes themto be turned onand alternately to be turned off. Itshould be noted that moving between menus does not disable the selectionsmade, as would normallyhappen.



Control screenfor sensors aroundthe house - allowsreal-time reading of the state of thetemperature and humidityin the house (bank where the sensor is located)Two other sensors indicate the presenceor absenceof the objects they aronitoring. Wechoose to monitor the car in theparking lot and thedomestic pet .

The screen automaticallyrefreshes the reading of thesensors, although an additional refresh is notaumatic but requires pressing the refresh button the sensors intoone string at atime



The string is sent to all users ,The click
to the end server connected

An example of an indication of a fault detected when connecting to the server. The indicator is one example out of many that have been embedded in the code and are ready to be displayed as soon as such is reported by the network controller.



From the example of console prompts when requesting to display all end users (4) up to those connected to the server.



An example of a string containing all the data gathered from the sensors, after clicking the 'Summary' button in the Sensors menu. It should be noted that the click also redirects back to the main screen so that you can view this communication with the server through the console.

Received Data:
SmartHome Report:
Humidity=.
Temperature=.
Car is In Use.
Pet is Missing.
0 Lights are ON at Home

The string is of course sent to the end users and appears in the application .



An example of an SMS message received from one of the residents of the house, which was sent as part of the application on the cell phone. The message is sent to the console and also to all other keyboard users, and this is how home chat group actually takes place.



19. Hardware and sensors used in the 'smart home' design

As mentioned, the control of the system is carried out in two ways - **locally** (touch screen) and **wirelessly** (an application that communicates with a **network controller**). The following is a breakdown of the uses of the hardware in the system:

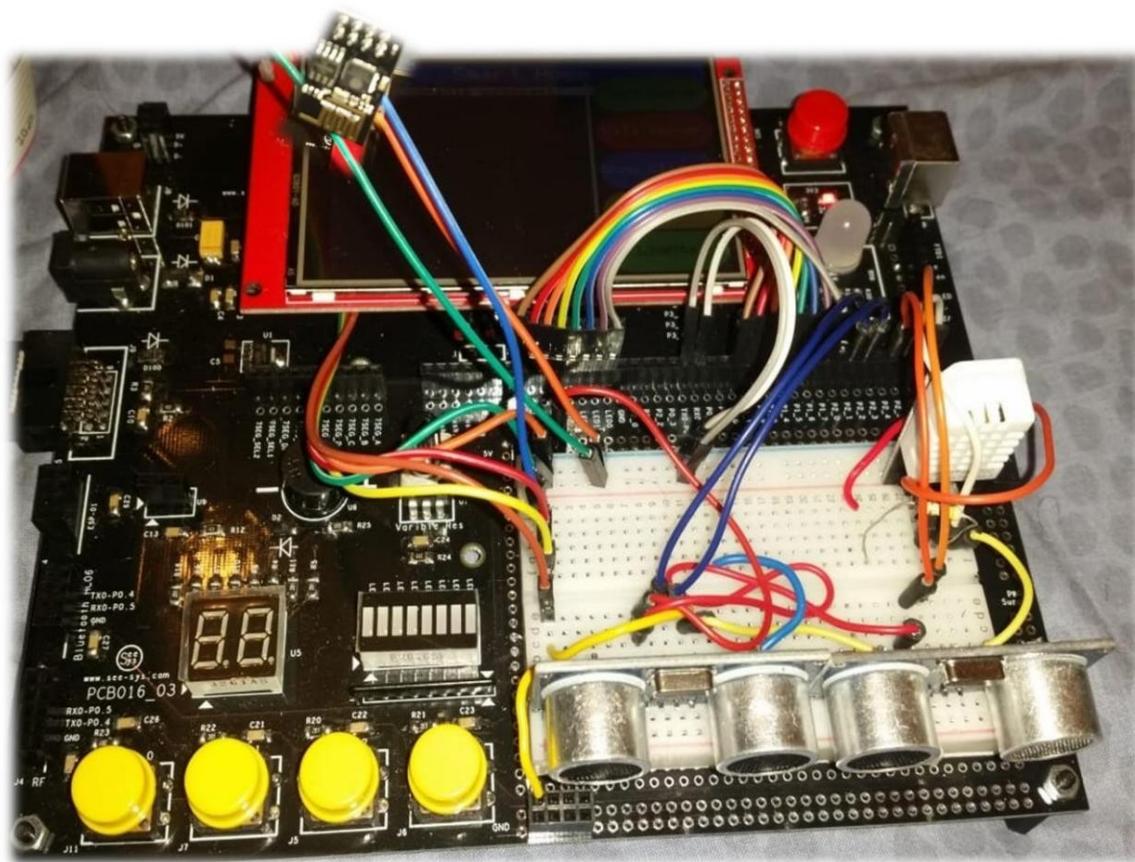
- **Microcontroller 380F8051** with an **upgraded 8051 core** - the heart of the system on which the software is written.
- **Network module 01 Ver 8266ESP** - communication component for hosting house residents (users) and control in the remote controller.
- **1117AMS voltage stabilizer** for the system.
- **1307DS RTC component** time controller for autonomous lighting and home air conditioning.
- **Humidity and temperature sensor DHT** - temperature monitor and humidity in the air in which it was assembled at home.

Installed under the fan to show actual measurement capabilities.

- **2 ultrasonic proximity sensors 04SR-HC** simulate sensors for monitoring the inventions of a pet in the area assigned to her and monitoring the vehicle finds of the family members in the parking lot assigned to her.
- **V5 fan** - simulates a home air conditioner.
- **3 RGB type LEDs** colored lights that indicate the status of the system or automatic activation setting
- **8 normal LEDs** - simulating 8 household consumers/bulbs around the house.
- **Cellphone with a dedicated application (Android)** - for remote control via Cellphone.

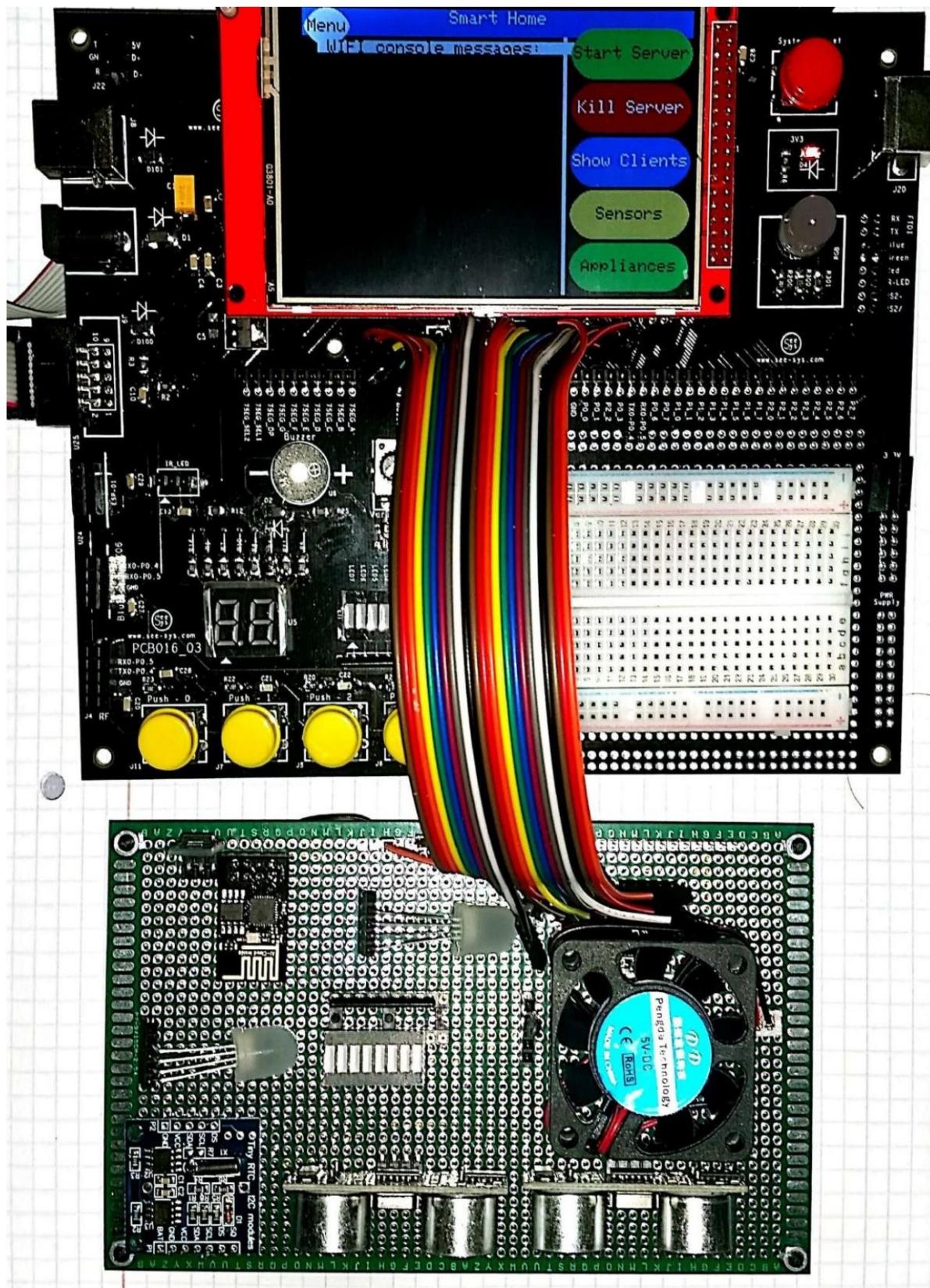
The air conditioning and the park light.

The micro controller in development configuration, before mounting the hardware on a dedicated board



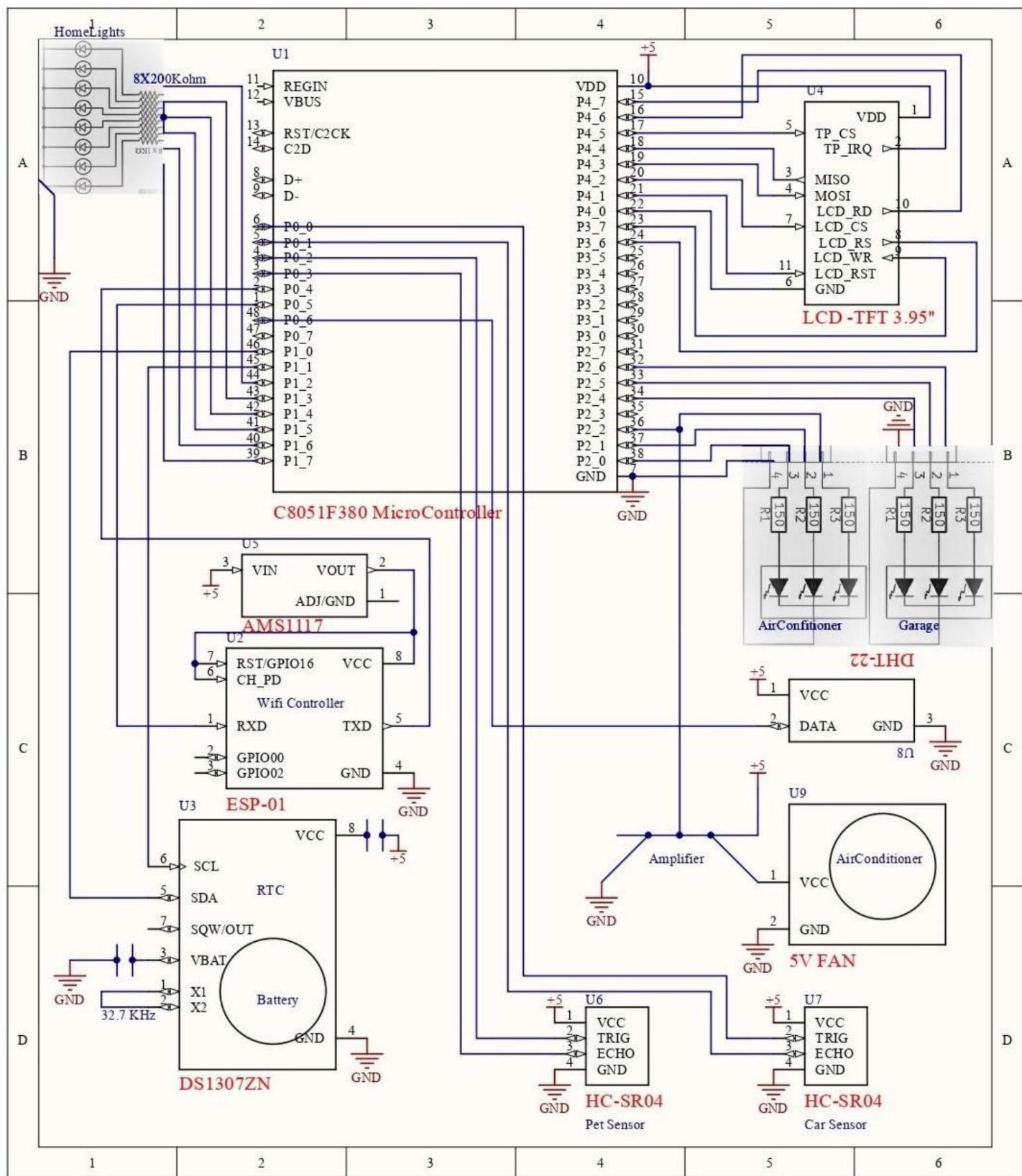


The connection ports of a microcontroller to the wiring we created on the WireUp board

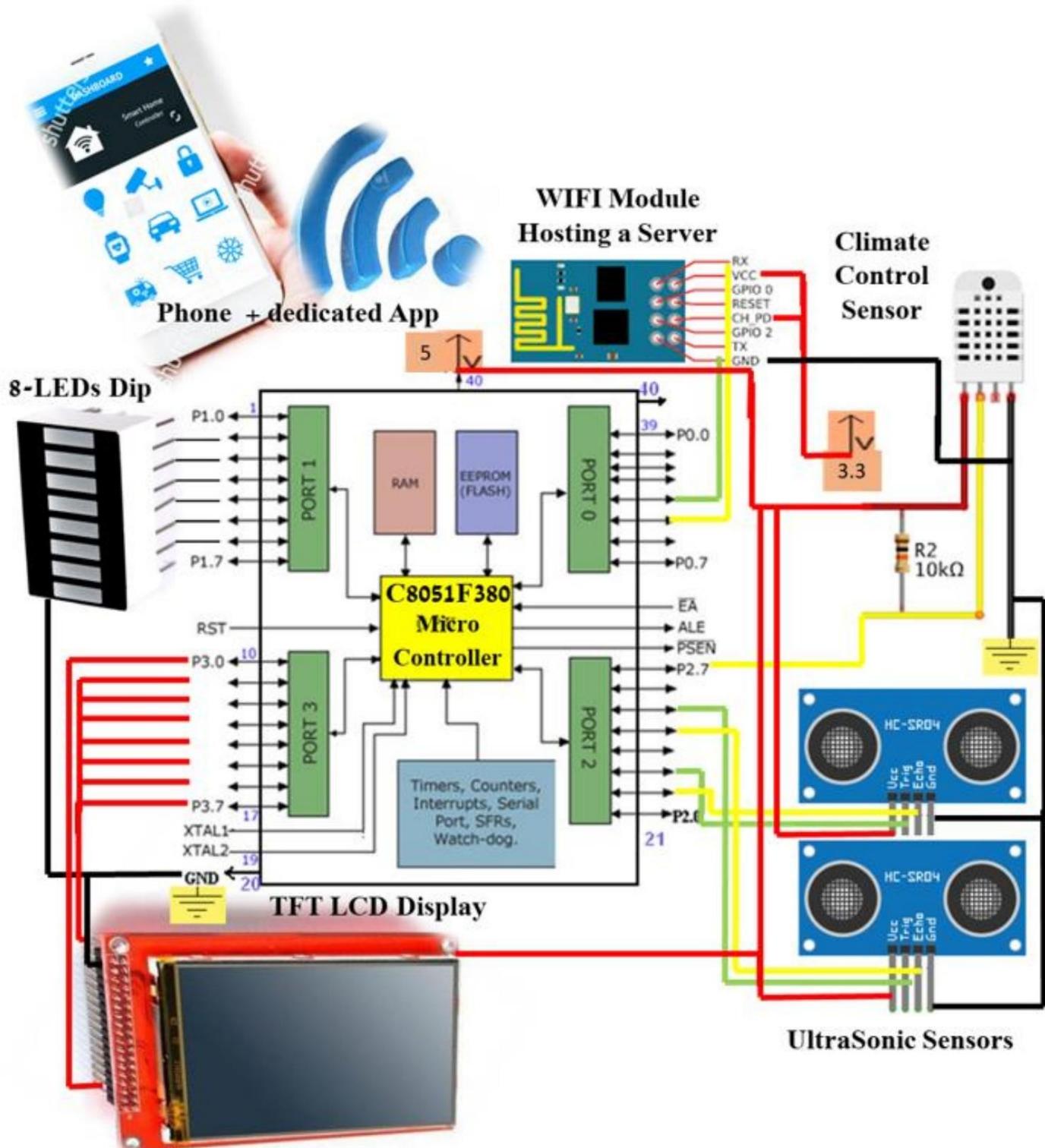


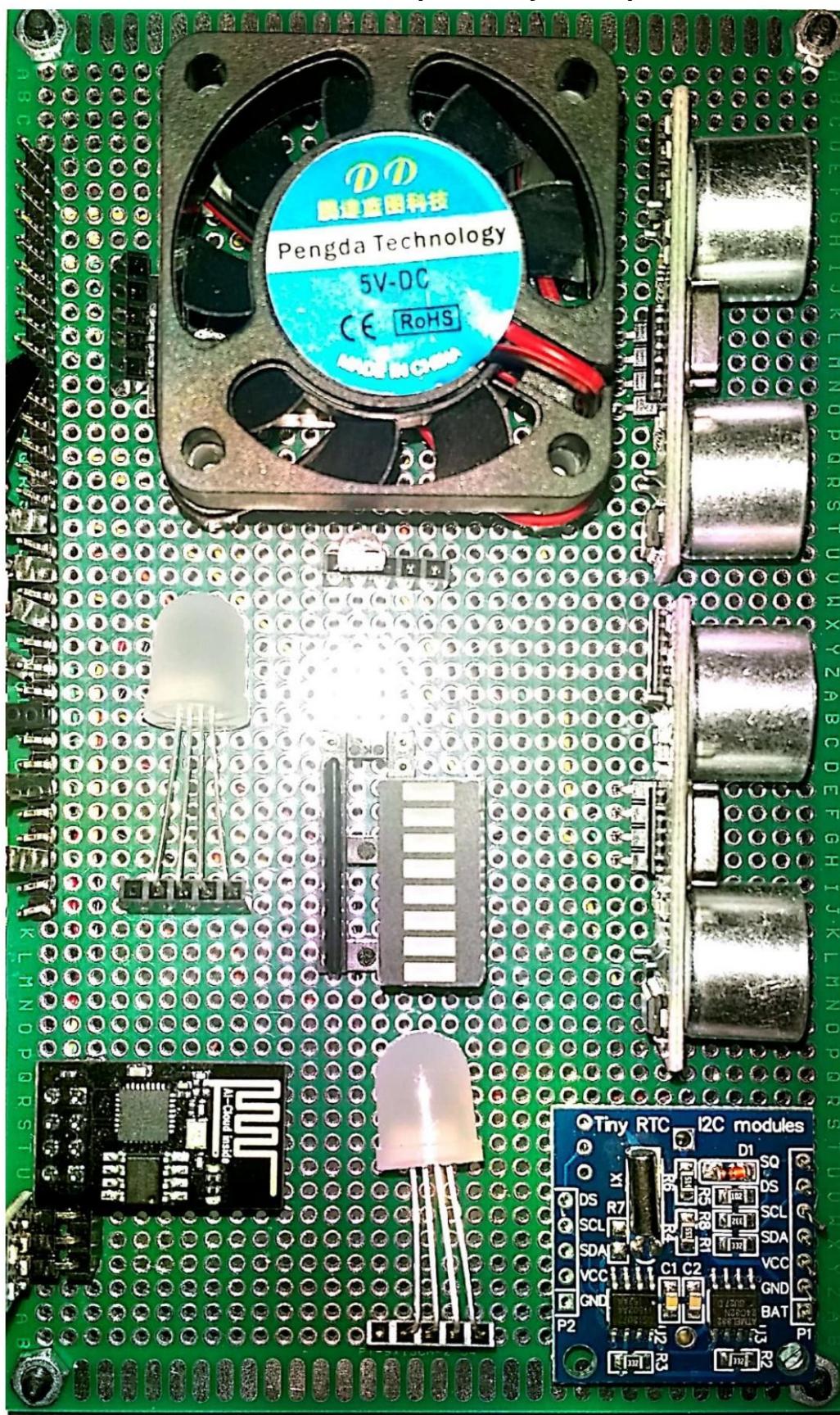


Electrical scheme (built using Altium 19 ver. software)

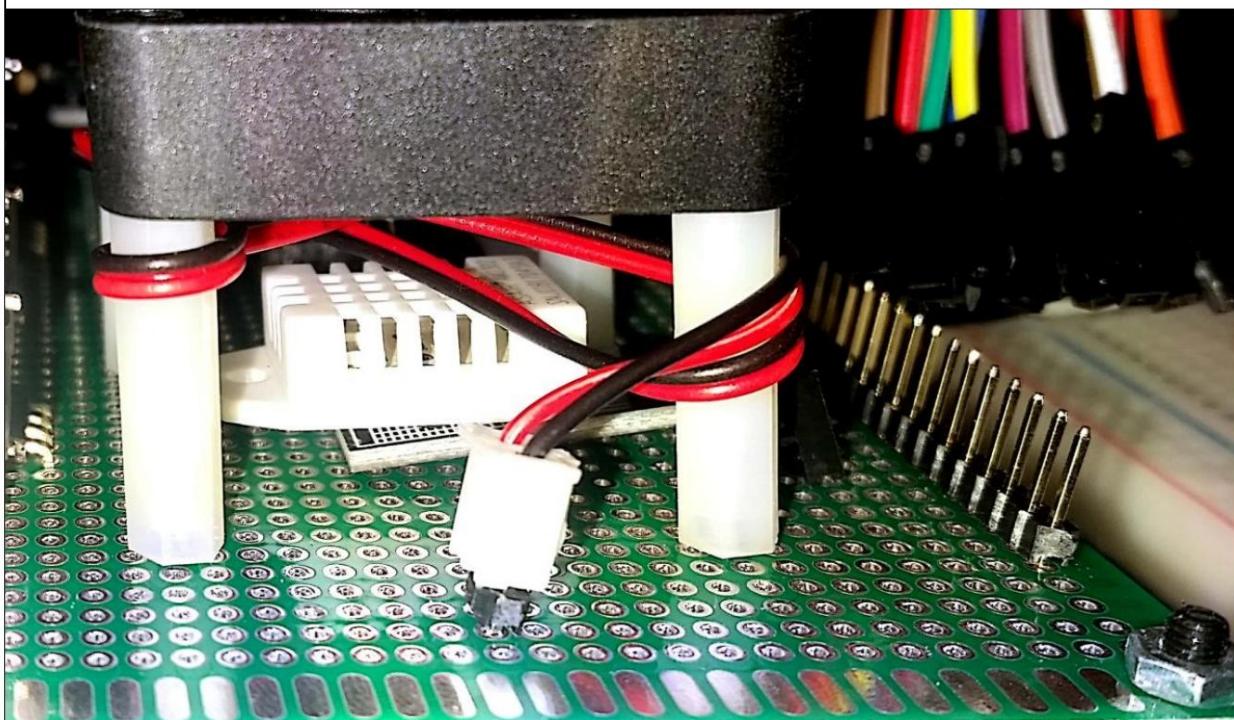


main components during development: Rectangles, the visual chemistry of the material's

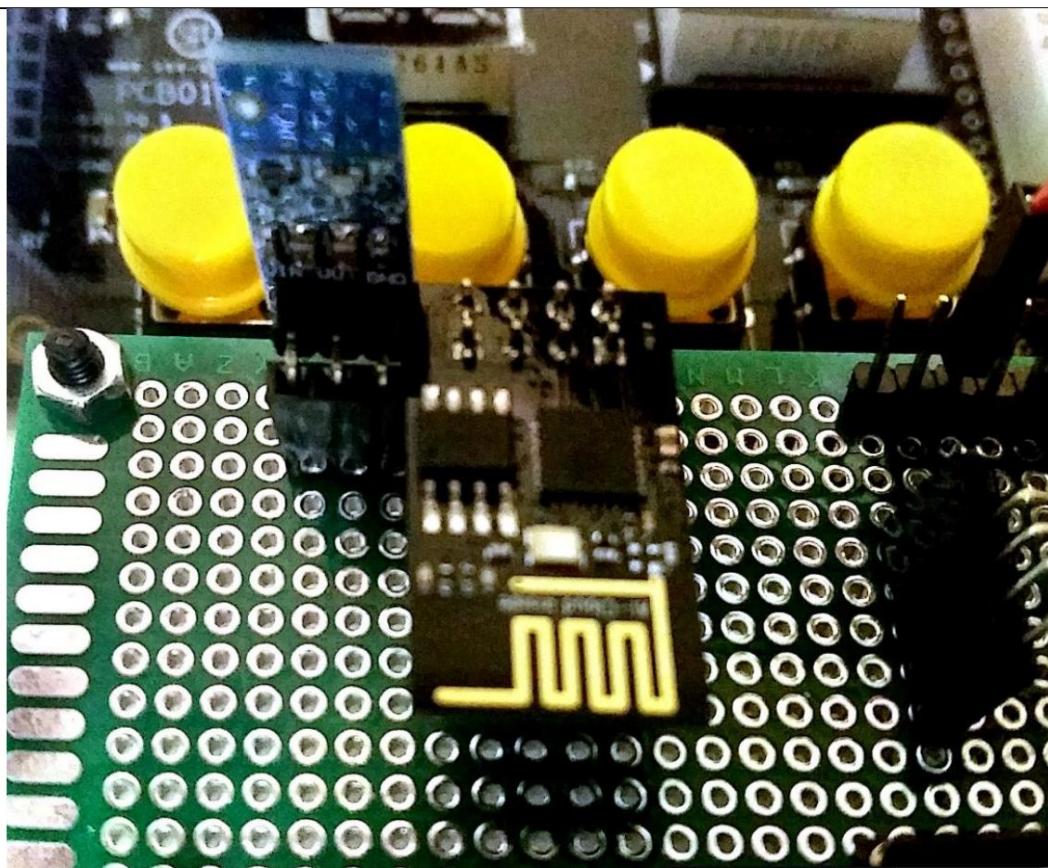


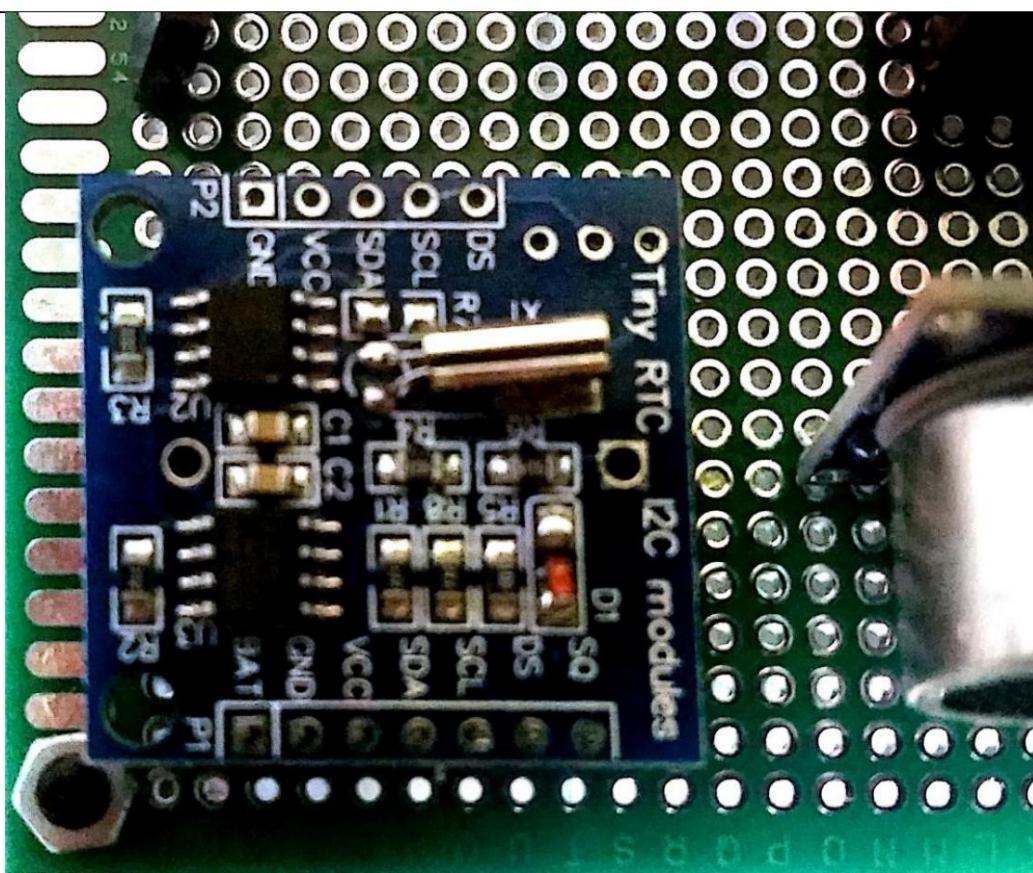
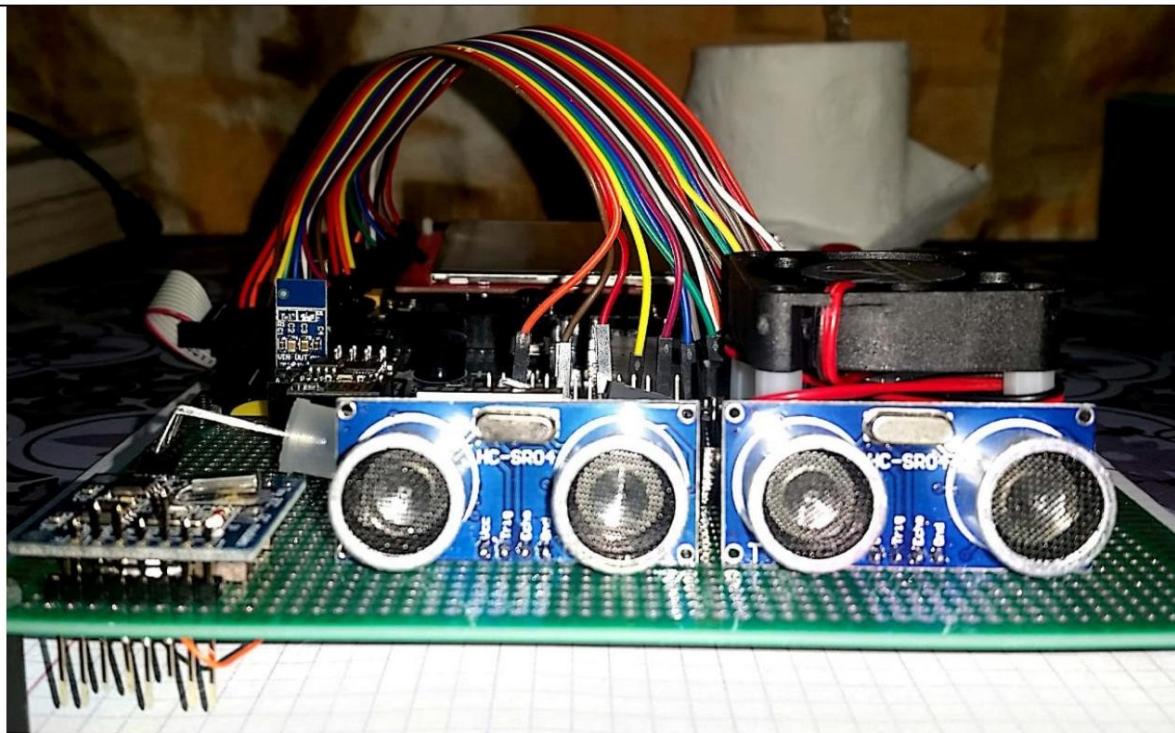
List of all components by WireUp

A 22-DHT humidity and temperature sensor that is fixed under fan and monitors the change in data accordingly

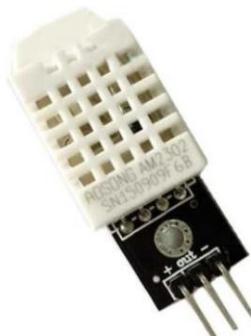


Network module .01Ver 8266ESP voltage stabilizer



RTC DS1307 component**2 ultrasonic proximity sensors 04SR-HC**

20 Integrated climate control sensor - -22DHT



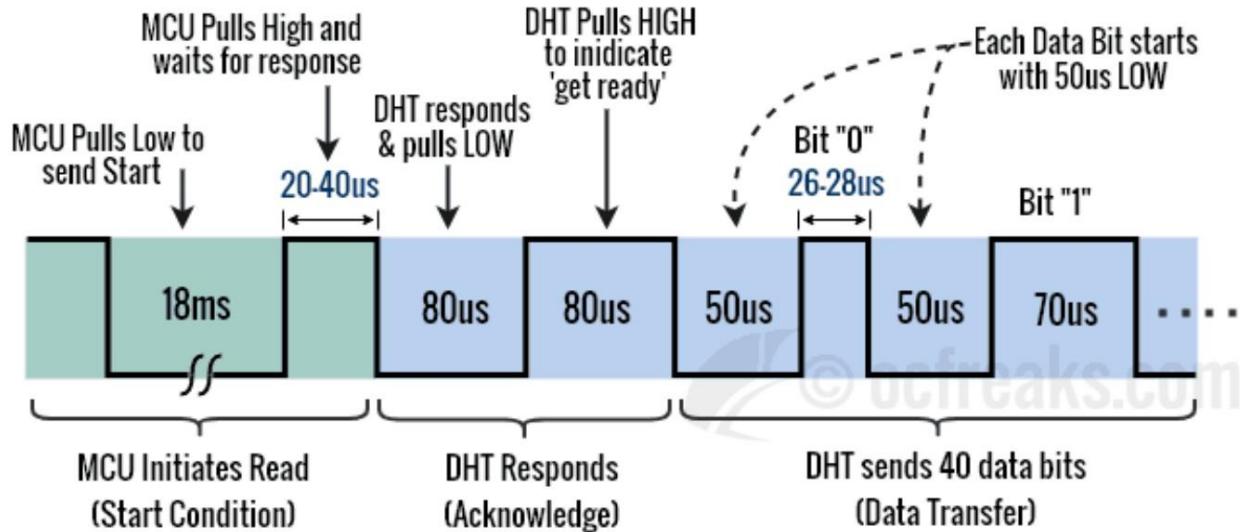
Pin Name	Pin Description
VCC	3.3 to 5.5 Volt DC, also 4.7K - 10K PullUp resistor is needed between this and DATA pin.
DATA	Digital output pin
GND	Ground

-22DHT is a sensor for monitoring temperature and humidity. The humidity value is measured in percentages (with an error range of 2-5%) and values in degrees Celsius in a working range of C80° C-40° (with an error range of ±0.5°C). The metric temperature is measured by the method.)

The measurement method is electrical, that is, through the conductivity of a material that is affected by the above data, and these are converted digital. The allowed reference rate for the component is 0.5 Hz, which means that 2 data can only be 'pulled' from it every second.

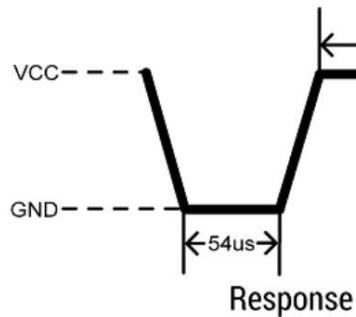
One is used for communication. The communication process is divided into three stages, the first sending a request to the sensor, the second responding to the request, and the third sending data. The first stage means only stages, the first sending a request to the sensor, in both the sensor will send a pulse (pulse) a response and then in the third stage 40 bits for the microcontroller that runs it. Total starts sending data

DHT11 / DHT22 Protocol

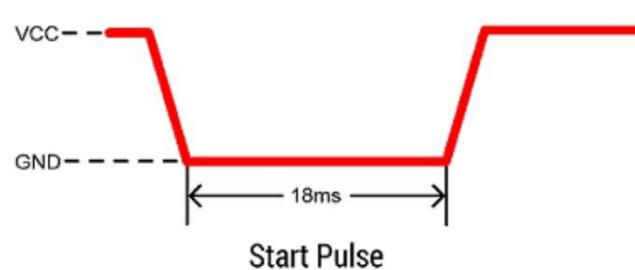




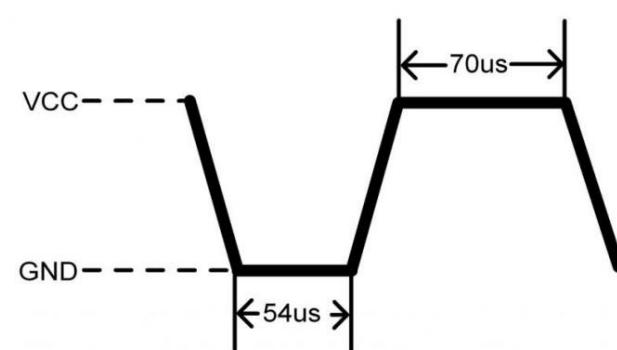
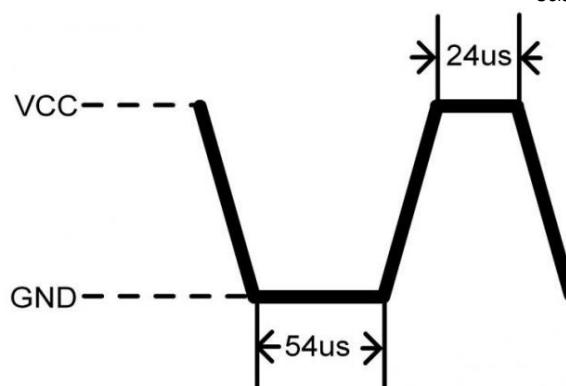
As a response, a pulse is received below with a length of 54 us. Immediately after it is a high pulse, after which the sensor pushes the data in the information line.



To provide the initial pulse, a low pulse is provided 18 ms long.



The data frame has a total length of 40 bits and it contains segments (bytes) and each segment is 8 bits. In these 5 segments, the first two segments contain a moisture value in the form of decimal number. This value gives us relative humidity percentages. The first 8 bits are an integer part and the next 8 bits are a decimal part. The next two bits contain a temperature value in the form of decimal integer. This value gives us a temperature in Celsius. The figure is sent in a similar way to the form in which the previous figure was sent.



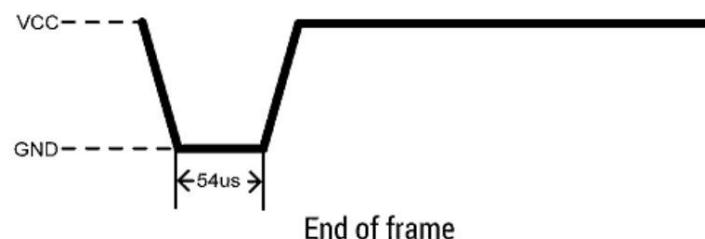
Example 40 bits read from the sensor and their decoding: 0000 1110 1110 1111 0101 0001 0000 1100 1000 0010

- 16 bits RH data: 0000 0010 1000 1100 → 652 → RH=652/10=65.2%RH
- 16 bits T data: 0000 0001 0101 1111 → 351 → T=351/10=35.1

What is called 'is 1' means that the temperature is negative (below 0): (e.g. 1000 0000 0110 0101, T= minus 10.1)

- 8) A bit a CheckSum which is summation of the first four sectors, in this form provides an option to verify the correctness of the received data calculation according to the examples given: 1110 1110=1111 0001+0101 1100+0000 0010+1000 0000= sum of bytes

At the end of sending, the sensor goes into low power consumption mode until receives a start pulse, not before sending Last pulse indicating the end of data transfer:



- 04SR-HC 21. Proximity sensor rangemonitoring



Pin Name	Pin Description
VCC	5 Volt DC
TRIG	Trigger input of sensor. Microcontroller applies a 10us trigger pulse to the HC-SR04 ultrasonic module.
ECHO	Echo output of sensor. Microcontroller reads/monitors this pin to detect the obstacle to find the distance.
GND	Ground

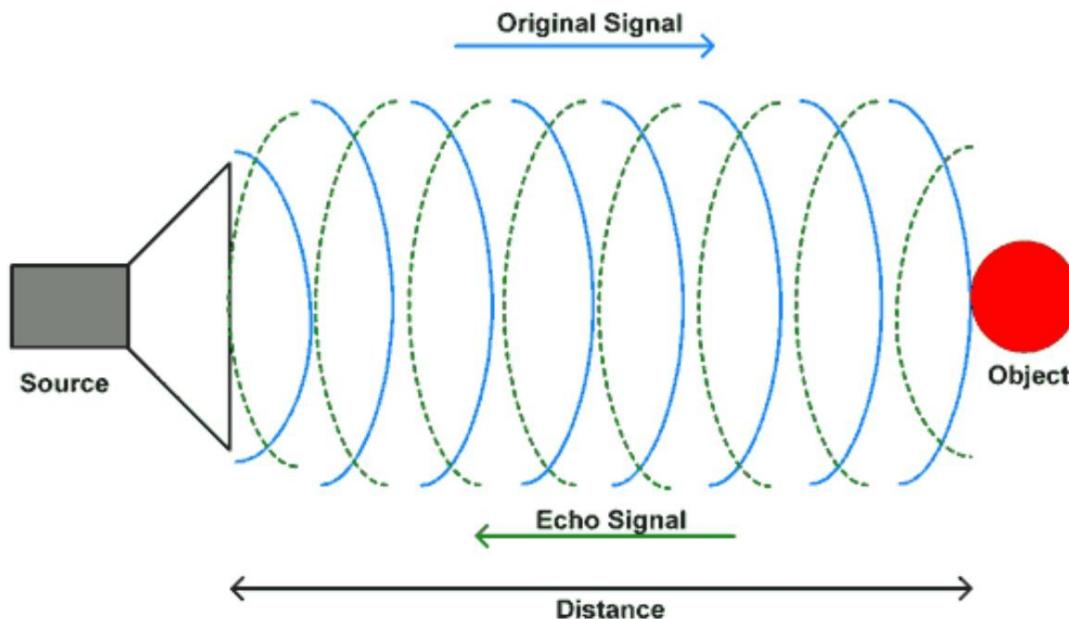
04SR-HC is a sensor for measuring distances between 2cm and 3meters. So that it is possible to measure large distances. The sensor should be placed higher than the floor, otherwise the signal reflected from the floor could disrupt the measurements. location

The sensor 8-10 cm above the floor will be enough to measure ranges of about a meter from the sensor.

The sensor works on the principle on which SONAR or RADAR are based, which are used to determine the distance to an object reflected and direct to the sensor and therefore essentially contains a transmitter and a receiver (ultrasound), but not perceptible to the human ear.

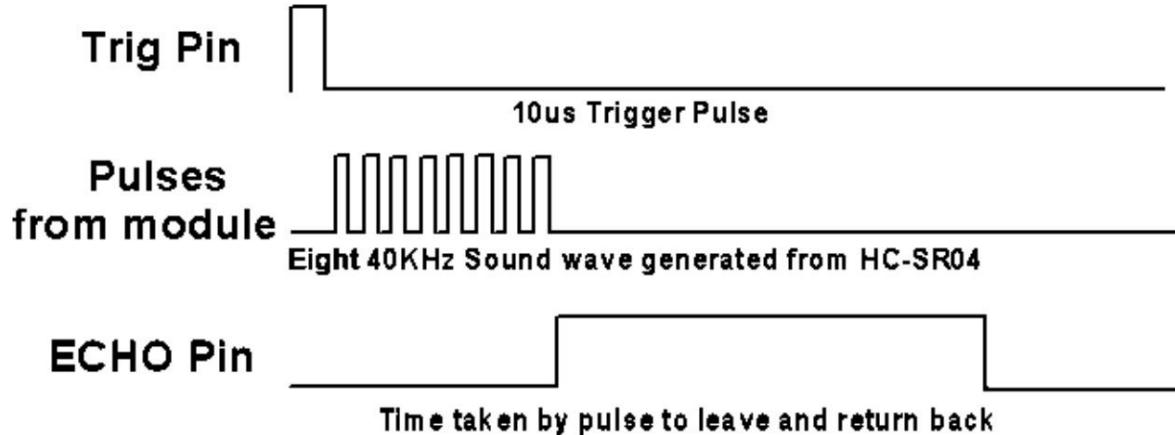
An ultrasonic sensor creates high-pitched sound waves

(40KHz frequency) when they hit an object, they return as an echo that is felt by the receiver as shown in the diagram below.



Calculating the range from the object:

By measuring the time it takes for an echo to reach the receiver we can calculate the distance. There is the measurement of time to start immediately with the sending of the sound waves by the transmitter, and therefore we use a timer (in the project - 0TIMER), for 5 microseconds. immediately upon giving the high component Trigger pulse for activation



Ultrasound 8 - pulses of 40kHz by the sensor. As soon as they are completed, a line rises. The axis of the pulse causes me, and waits in this state until the voice is not received back. **to logical level the Echo**

And we have to respond in the software to stop the counter. **To logical level '0'** with the return of the soundwaves, **the echo line objectively disappears**. The resulting pulse width represents the time it took for the soundwaves to complete the journey from the transmitter

to the shelter

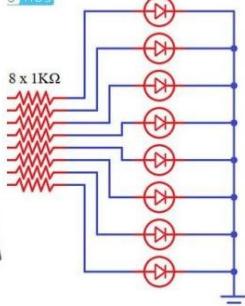
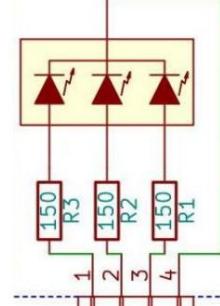
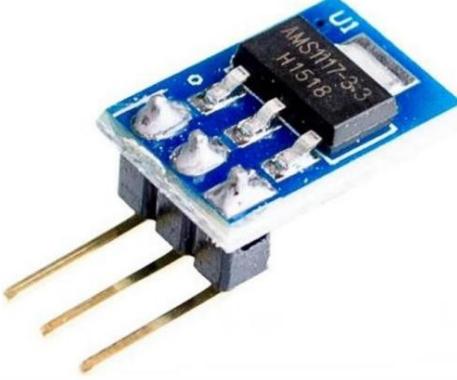
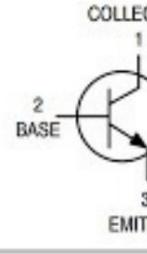
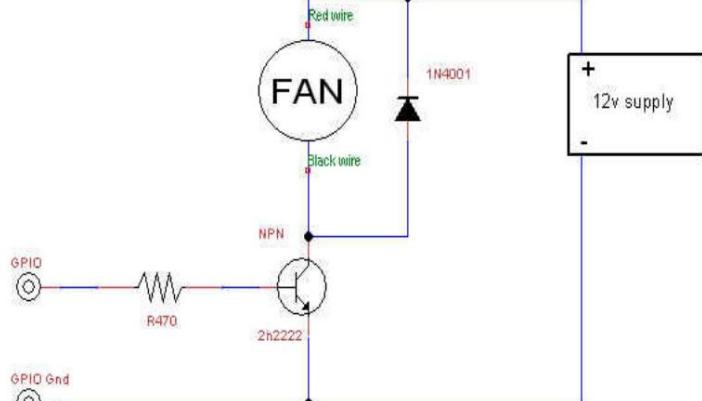
Another figure that's important to complete the equation is **the speed of sound** - which is a constant figure (in a good approximation) and has a value of 343 m/s. It can be calculated. Since speed multiplied by time is equal to range

$$\text{Total Distance} = \frac{343 \times \text{Time of High(Echo) Pulse}}{2}$$

As a general approximation, it can be said that the distance in cm to the measured object will be the length of the returned pulse in microseconds divided by 58

It is important to see that **the total distance divided by 2 because it is important for us to consider only the distance traveled by the sound waves from the transmitter to the object without including their way back to the receiver.**

22. Other analog components in the project

Pack of 8 bulbs - for simulating home lighting	RGB LED - for automatic clock indication for parking lighting
 	 
1117AMS voltage stabilizer for constant voltage supply to the home server, and stabilization of the ESP component	Ventilator V- 5 simulates an air conditioning system at home
	
Bipolar transistor used to increase current in order to turn on the fan	Electrical diagram for connecting the fan in the project
  TO-92 CASE 29 STYLE 17 1 2 3	



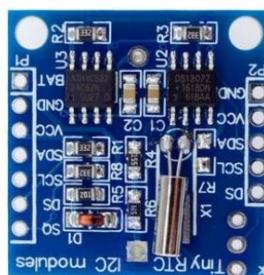
23 Real time clock component -RTC 1307 -DS

Pin Name	Pin Description
<u>The minutes</u> <u>2.1</u> <u>X1 X2</u>	Between the two triggers is connected a quartz crystal with a frequency of HZ 32.768 which provides a square wave to activate the circuit inside the watch. 1X is the input of the internal oscillator and instead of a crystal, an external oscillator with a frequency of KHZ 32.768 can be connected to this input or another squarewave source at this frequency. The accuracy of the crystal will determine the accuracy of the watch.
<u>Trigger</u> <u>-3</u> <u>VBAT</u>	The positive voltage input of the backup battery. The backup battery is a 3 volt battery (you can connect a battery between 2 and 3.5) volts. The second trigger of the backup battery will be connected to ground. If the external backup battery is not used, connect the .10 years Trigger 3 to the ground. mAh 48 battery can hold
<u>Trigger - 4</u> <u>GND</u>	. The soil of the thin component
<u>Trigger - 5</u> <u>SDA</u> <u>DATA -</u> <u>SERIAL</u>	The two-way data trigger of the C2I serial communication interface.
<u>Trigger - 6</u> <u>SCL</u> <u>SERIAL -</u> <u>CLOCK</u>	This trigger is the clock input in the C2I interface; it is used to synchronize the data movement in the serial interface. Connect to a resistance clamp External PULLUP because it is of the 'open funnel' type (drain open).
<u>Trigger - 7</u> <u>SQW/OUT</u>	When the software enables the output of a square wave at a desired frequency (KHZ 1, KHZ 4, KHZ 8, KHZ 32), the wave is output through this trigger up square. Also to this port, a pull-up resistor must be connected to
<u>Trigger - 8</u> <u>VCC</u>	the positive voltage trigger of the power supply.

General description:

- A real-time clock component that connects via C2I serial communication with a microcontroller. The component has low power consumption with time and date and another 56 bytes of static RAM for general use according to the user's wishes.
- The clock provides us with time that includes seconds, minutes and hours and the date that includes the day of the week, day of the month and the year. The month is automatically updated in months that have less than 31 days including leap year corrections.
- The clock can work in a 24-hour format or in a 12-hour format with an indication of AM and PM.
- Inside the component there is a circuit that senses a voltage drop and automatically switches the circuits to battery voltage Backup. Creating the time continues even if the component is powered from the external backup battery.

- The component is activated in the C2I serial communication protocol and is used as a SLAVE component in this communication.

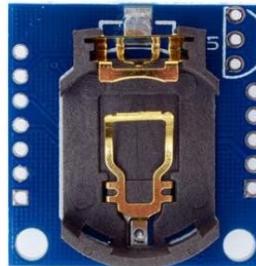


Features:

- A real time clock that counts seconds, minutes, hours, the day of the month, the month, the day of the week, the year Includes a leap year and a leap year correction and is valid until the year 2100.
- 56 bytes of general RAM that can be backed up by an external battery with an unlimited amount of write operations disability.

I. 2 • C serial interface

- Square wave output signal with programmable frequency.
- Automatic voltage detection that switches from the main voltage to the backup battery or vice versa.
- Small current consumption - nA 500 when connecting a backup battery and there is no external supply voltage. .85 degrees • Industrial temperature range from -40 degrees to
- Can be obtained in a package of 8 plastic DIP or SO pins.





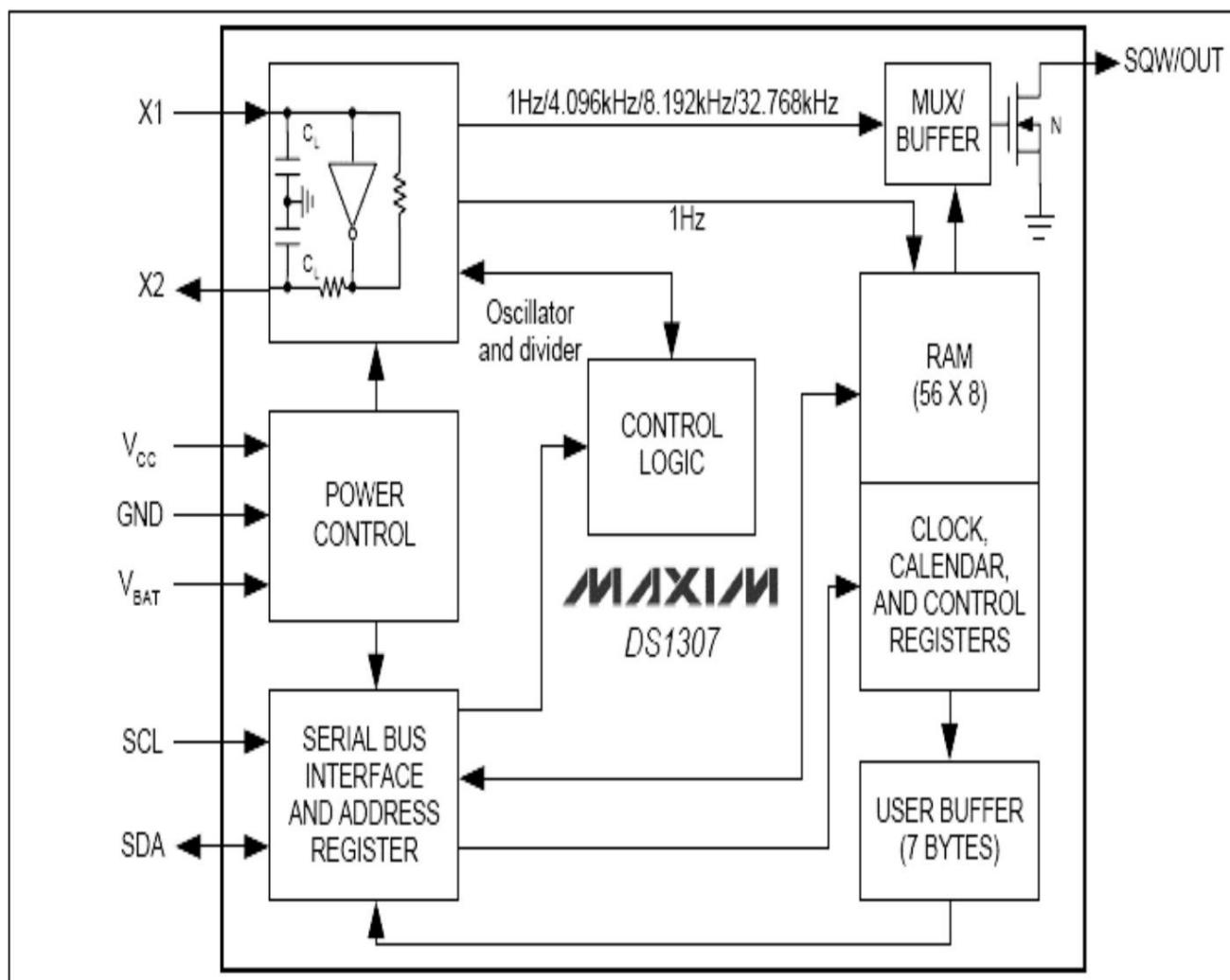
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Logic 1 Input	V_{IH}		2.2		$V_{CC} + 0.3$	V
Logic 0 Input	V_{IL}		-0.3		+0.8	V
V_{BAT} Battery Voltage	V_{BAT}		2.0	3	3.5	V

Maximum nominal values:

Recommended voltages:

- From the table you can see that the power supply voltage is between V 4.5 and V 5.5.
- Input voltage of logic 1 is from V 2.2 to V 0.3+ VCC voltage.
- Logical '0' voltage at the input is from V 0.3- to a maximum of V 0.8 (logic 0 maximum).
- The backup battery voltage is from V 2 to V 3.5.

Logical scheme:



The memory division -the time and date:

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE			
00h	CH	10 Seconds			Seconds			Seconds		00-59			
01h	0	10 Minutes			Minutes			Minutes		00-59			
02h	0	12	10	Hour	10	Hours			Hours	1-12 +AM/PM 00-23			
		24	PM/ AM										
03h	0	0	0	0	0	DAY		Day		01-07			
04h	0	0	10 Date		Date			Date		01-31			
05h	0	0	0	10	Month			Month		01-12			
06h	10 Year				Year			Year		00-99			
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—			
08h-3Fh								RAM 56 x 8		00h-FFh			

0 = Always reads back as 0.

All the data in the clock is in a rowed BCD code which means that every bits are used as one decimal digit. For example, 30 minutes in binary.) the maximum decimal number in the 4 digits 0011 0000 will be indicated in the address 1 in H30, meaning it will be 1001, which is a number greater than this cannot appear.

Address 0 contains the seconds. Bit 7 called CH- Halt Clock - is a bit controlled by the programmer. 0 bit makes the clock work. If you put in this bit the oscillator circuit stops working. When turning on the electricity, he looked Can wake up at a random situation. The bit must be initialized to 0. Stopping the clock is worthwhile if you want to extend its life of the external backup battery. Stopping the clock means that the electronic circuits in the component are not activated and thus significantly reduce the energy consumption from the battery.

Bits 0 to 3 are the units digit of the seconds. Bits 4 to 6 are of tens of seconds.

·digit Address 1 contains the minutes . Here again, bits to 3 are the seconds digit and bits to 6 are the tens Bit 7 is always in .0

Address 2 contains the hours. It is possible to work with a 12 or 24 hour working mode. Bits 0 to 3 determine the manner. If 12 hours 1 in this bit (or work) The unit digit of the hours. Bit 6 determines if you work in a (24 hour 0 bit) in a 12-hour manner, so bit 5 says whether AM (before noon) or PM (afternoon) when in bit says PM (afternoon). If you work 24 hours then bit 5 together with bit 4 are the tens digit of the hours. For example, if you write the binary number 00100011 to address 2, it means that you work 24 hours a day.

6 (bit at 0) and the time is 11 (11 pm).



.for Saturday **Address 3 contains the day of the week**. In bits 0 to 2, the number can appear between 1 and Sunday and 7. The other bits are 0.

Address 4 contains the day of the month(a number between 1 and 31). Bits 0 to 3 are the units in the month and bits 4 and 5 are the tens digit of the month. The 2 bits and 7 will be at 0

.4 to the tens digit **Address 5 contains the month** 1 (number between 12 and 12). Bits 0 to 3 are the units digit. The remaining bits are 0.

Address 6 contains the year (a number between 99 indicating the years 2000 to 2099). Bits 0 to 6 are the digit .7 the tens digit The units and 4 bits until the

, day of the week change is done at midnight. Relevant data must be entered. Sunday of the week should be noted as 1 Monday about 2 and so on. Entering illogical data will cause illogical results.

Current consumption:

To see in the first row the current consumption from the battery when the component is working and you do not output a square wave in leg 9 of The component (will be explained later). The typical current is 300 nanoamperes. If a square wave is output - second row in the table - The current increases to 480 nanoamps.

When the oscillator circuit is stopped (the electronic circuits are not activated) - the current decreases - third row in the table - to 10 nanoamps

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{BAT} Current (OSC ON); SQW/OUT OFF	I_{BAT1}			300	500	nA
V_{BAT} Current (OSC ON); SQW/OUT ON (32kHz)	I_{BAT2}			480	800	nA
V_{BAT} Data-Retention Current (Oscillator Off)	I_{BATDR}			10	100	nA

Control register:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

At address 7 is the control register. The programmer can determine whether to output a square wave with the component's

OUT/SQW 7 leg or determine a desired fixed logic level with this leg. If the programmer wants to output a square wave, he has the option of choosing one of 4 possible frequencies. If the programmer does not want to output a square wave he

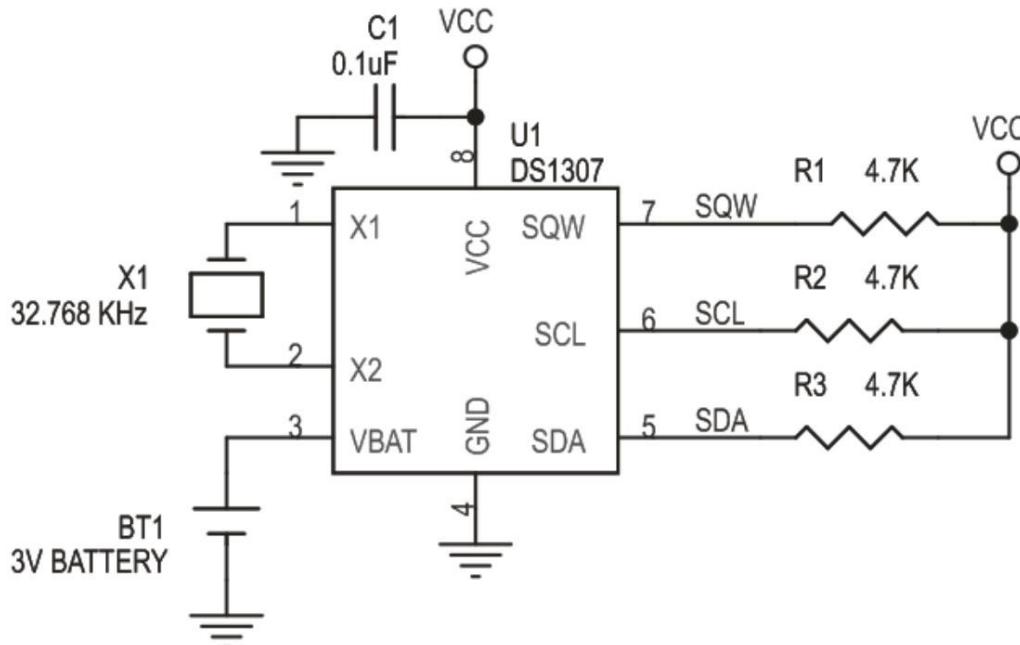
can determine whether this output leg (leg7) will be 0 or 1

With the help of bit4 SQWE - Enable Wave Square - we determine whether to allow the output of a square wave (put a 1 in the bit) or not (put a 0 in the bit). If we put a 1 in bit 4 (enable the output of a square wave) then the bits 0 and 1 RS ORS determine the output frequency on this leg Select Rate- RS - rate selection .

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

If we put the value 0 in bit4 (a square wave output is not allowed) then bit 7 of the register called OUT - it is possible to determine whether leg 7 of the component will be 0 or 1. If we put 0 in bit 7 then leg 7 will be 0 and if bit 7 we put 1 then leg 7 will be 1

From address 80 3FH55 (decimal) there are RAM addresses for general use according to the user's wishes.

Typical operating circuit:

The component connects as a **SLAVE** to a processor CPU - which uses a **MASTER** with the help of 2 lines of C2I communication. foot

SDA of serial data Data Serial and the SCL leg the serial clock Clock Serial, which synchronizes the input of the data

The serial from the processor to the component or from it to the processor.

2 "UpPull" resistors must be connected from these two legs to the Vcc voltage. The reason for this is that inside

The component has FET transistors in an Open Drain connection (open source). The reason for this connection is that on 2 communication lines

These C2Is, you can connect at the same time additional number of components such as EEPROM, FLASHADC, etc. and you don't want to
the communication lines that the resistors inside the component will affect load - you

Another external resistor is also seen at the output leg of the component - OUT/SQWE. Here too the port of the component is Open

Drain and an external resistor must be connected.

The external PullUp resistor that will be connected is a resistor in the order of 2 kilo ohms to ten kilo ohms.

The component communicates with the processor in C2I communication. A number of different components can be connected on a C2I communication band.

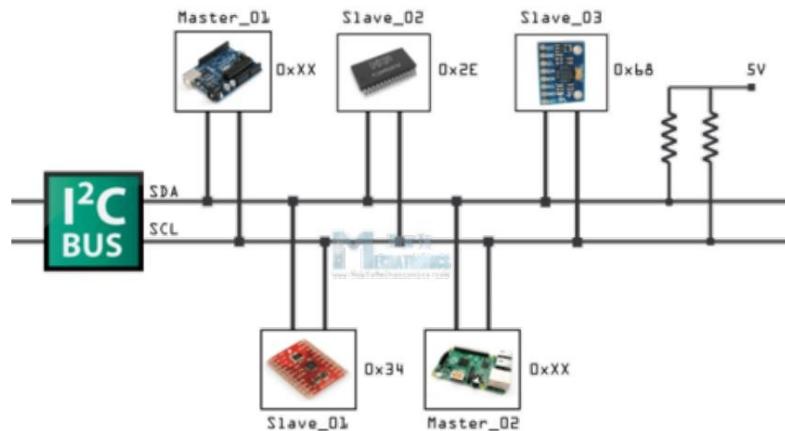
I²C Protocol.24

The C2I protocol is a two-line BUS communication protocol for **slow to medium rate**, as a result

As a result, the physical size of the components communicating through it can also be reduced - the number of their triggers remains small
And so is their cost.

More importantly, it is also possible to chain a large number of components to the same transmitter (memories, converters, real-time clocks, etc.).

The component that manages the communication process (the processor) is called **MASTER** and the components that connect to it are called **SLAVES**. In addition, the **MASTER** controls access to the bus and creates the **START** and **STOP** states.



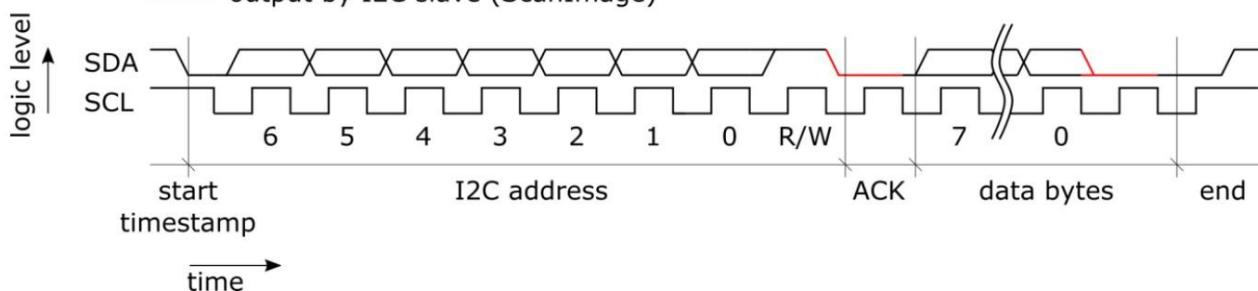
The mode of operation in this communication is called **Duplex-Half** because it is not possible to write and read at the same time. The bus lines include two

This job is to synchronize the **Serial clock line** - (Clock Signal (SCL)) which is one-way and powered by the **MASTER**

The information transferred between the two components.

- **Serial data line** - (Serial Data (SDA)) which is a device whose function is to transfer the information between two components.

— output by I²C master
— output by I²C slave (ScanImage)



It can be seen that several components can be connected to the 2 lines SDA (the data line) and SCL (the clock line). Each component has its own unique address. For component 1307DS the address is X1101000 (H0D or H1D).

In the figure you see 2 components that connect on the lines. In the lower part of the figure you can see the internal structure of a component and see that the component is connected using a buffer (depicted by the triangle) that receives data from the line. Above the buffer there is a transistor connected



that can Collector Open (Field Effect Transistor FET - in an open collector connection) (Draughtopen, write to a given line).

An external resistor between 2kilo ohms and 10kilo ohms must be connected to the transistor. The values are chosen so that one hand the resistors will not be too small so that a large current does not flow through the line through the component (when the component outputs 0) and on the other hand that the resistor should not be too large because it determines the charging and discharging time in transitions between 0 and 1 and vice versa and a resistor that is too large will limit the rate of communication.

Rules and definitions in I2C communication:

- Transfer can only start when the SDA line is not busy - BUSY NOT.
- While transferring data, the data line must remain stable when the clock line is high. Change in the given line when line is interpreted as control signals. The clock will be interpreted high.

Not Busy Bus - no busy bus - both the data line and the clock line are high.

• TRANSFER DATA START - Start data transfer

- A change in the state of the data line from high to low when the clock is high is defined as the START state.

• TRANSFER DATA STOP - stop data transfer

- Change in the data line state from low to high when the clock is high is defined as the STOP state.

• VALID DATA - data validity

- The state of the data line represents the validity of a data when after the START state - the data line is stable for the maximum time of signal of the clock. The data on the line must change only during the low state of the clock signal. There is one clock pulse for each bit of given.

Each data transfer starts with a START state and ends with a STOP state. The amount of bytes transferred between START to STOP is not limited and is determined by the MASTER component. The information transmitted byte by byte And each receiver acknowledges receiving the byte with the ninth bit of ACKNOWLEDGE.

I There is a standard for 100KHz and there is a standard for 400KHz. The DS1307 component works at a rate of 2² in the settings of C of . only KHz 100

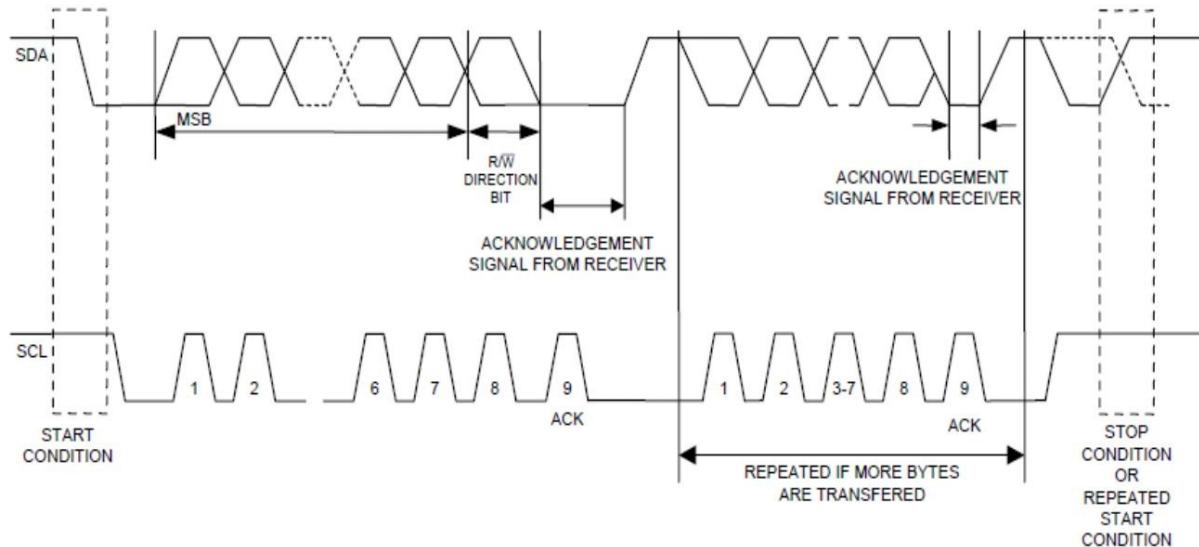
• ACKNOWLEDGE - approval

- Each receiving component must, at the end of receiving a byte, which was transferred to it, create an ACKNOWLEDGE bit. The MASTER component generates another clock pulse associated with this bit.

The ACKNOWLEDGE creator component must lower the serial data line - SDA to 0 during the clock pulse, i.e.

The data line will be steady low while the clock line is high. The MASTER component signals to the SLAVE that it is finished the communication by **not generating** the ACKNOWLEDGE bit when it received the last byte from the SLAVE.

In such a case, the SLAVE must leave the given line high to allow the MASTER to create a STOP state.

Scheme for the transfer of serial data:

The SCL line (the bottom line in the drawing) is always created by the MASTER .

The START state happens when the SCL line is high and then the MASTER lowers the data line to 0. Then the MASTER creates clock pulses and then it sends 8 bits on the given line - SDA. 7 bits are the address of the component and the bit (0 - reading, 1 - writing). The 8th bit says whether he wants to write to the component or read from it.

After that, the MASTER creates another pulse where the SLAVE has to return ACKNOWLEDGE afterwards. There is no need for an additional START and the bytes present one after the other when the receiving side gives an ACKNOWLEDGE. In bit number 9, the STOP (or repeated START) state is depicted on the right side of Fig. 6

to 1. The repeated START state is drawn with a dashed line and it is created when the clock line is at 1 and then the data line has a transition from 0 to 1. You see that while the clock line is at 1, the data line drops to 0

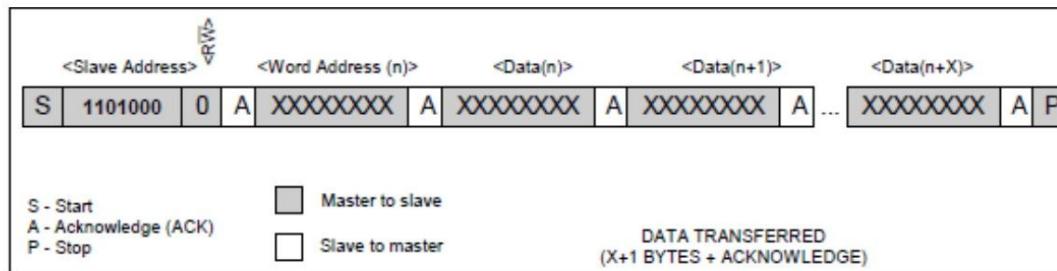


Data transmission in I2C communication:

Two data transfer options exist in I2C communication line:

A. The MASTER transmits and the SLAVE receives - write mode - WriteMode

In this case the first byte transmitted by the MASTER is the address of the SLAVE (in the case of a component 1307DS). The address is X1101000 - H0D in the case of writing to the component or H1D reading from the component. After this will be followed by several bytes of data. The SLAVE returns ACKNOWLEDGE at the end of each byte of data it received. The data is transmitted with the MSB bit first.

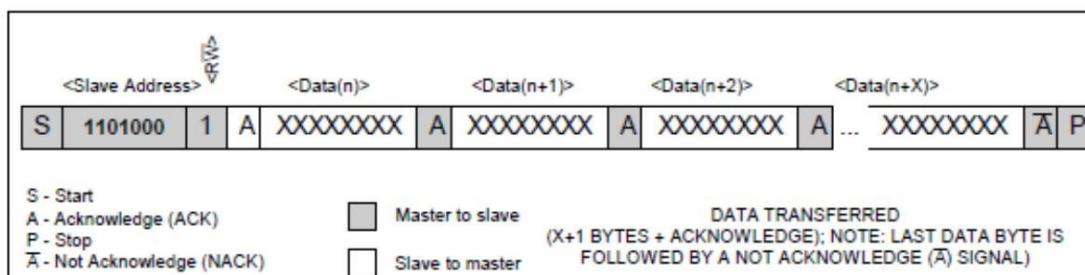


The MASTER creates a START state (marked with S). Then it sends bits of the component address - H0D. In the case of the 1307DS component - and the 8th bit is 0 indicating that it is the writer and the SLAVE is the receiver. On the SLAVE Answer with ACKNOWLEDGMENT (marked with A). Then the MASTER sends another byte that loads the pointer (register). Each data written to the addresses inside the component. The following data is written to this address and the address pointer automatically increases by 1. At the address the address pointer and the address pointer advances by 1 after each byte received by the SLAVE is sent acknowledgment received - ACKNOWLEDGE. The MASTER terminates the communication using the STOP mode (appears On the right side with the letter P).

B. A byte is transmitted from the SLAVE to the MASTER - read mode - ReadMode

In this case the first address sent is by the MASTER sending the address of the SLAVE which returns the

The ACKNOWLEDGE bit. From here the SLAVE sends a number of data bytes. The MASTER returns bit ACKNOWLEDGE after each reception of a data byte except for the last byte which does not return ACKNOWLEDGE Or you can say that it returns ACKNOWLEDGE. Not



This situation also always starts in the situation where the MASTER transmits to the SLAVE but there he wants to read from him. The first byte that the MASTER transmits is received by the SLAVE as described in the previous paragraph, i.e. the MASTER

Electronics Major - School of Engineering, Technion, May 2019

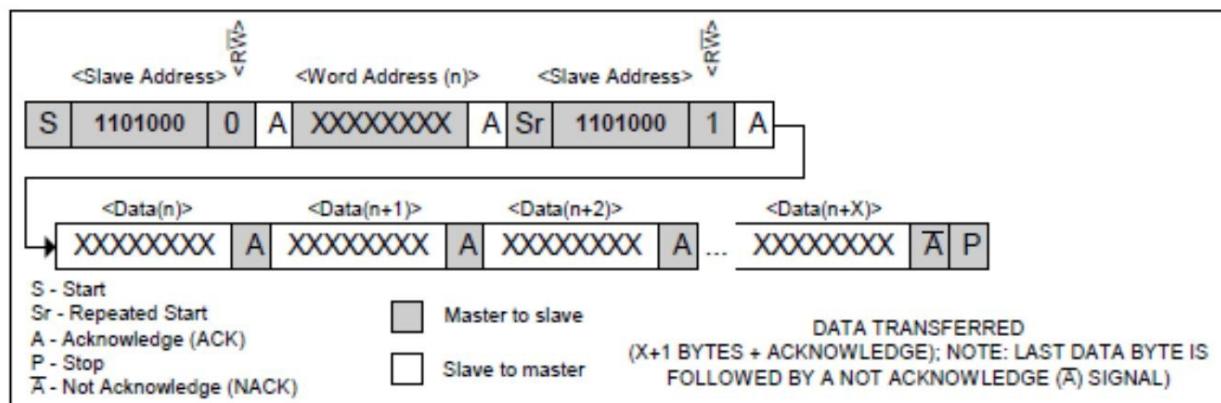
Creating a START state will send the bits of the address 1011000 but the eighth bit will be 1 where it says that he wants to read. From here the SLAVE transmits the data and the MASTER replies with an ACKNOWLEDGE bit. It does not respond with the 9th bit and does not send when the MASTER wants the end of the communication in the last byte, ACKNOWLEDGE (marked in figure b A) and also sends a tenth STOP bit.

The data transmitted from the SLAVE starts from the last address where the address pointer is located. Any data that

- automatically SLAVE sends it advances the address pointer to the next address

In two stanzas to the component in the first stanza he says. Usually the communication process will be as follows: the MASTER will write to the component it addresses for writing. In the second stanza he will indicate the desired address. Immediately after, STOP will be sent (or START returns) and then will perform a new communication where it will refer to the component to be read from the address it sent to it in the operation of the writing.

Write and read operation from the address:



The MASTER sent 2 bytes to the SLAVE. Immediately after that created repeated START mode (marked by Sr), send home added where there is the component address 10110001 (the eighth bit is for reading) and from this moment the SLAVE transmits and MASTER responds with ACKNOWLEDGMENT. At the end of the communication, the MASTER does not give an ACKNOWLEDGMENT (marked with A) and then gives STOP mode.

.25 touch screen "3.95 TFT - LCD

Name	Parameter
Display Color	RGB 65K color
SKU	MAR3953
Screen Size	3.95 inches
Type	TFT
Driver IC	ST7796
Resolution	480*320 (Pixels)
Interface Module	8-bit parallel transmission, interface reading from an SD card
Active Area	60.53x88.38(mm)
Module PCBSize	61.54x105.69 (mm)
back light	6 chip HighLight white LEDs
Operating Temperature	-10~60°
Storage Temperature	-20~70°
Operating Voltage	5V/3.3V
Power Consumption	TBD
Product Weight	About 45(g)

Details of the clamps:

Pin Name	Pin Description	Pin Name	Pin Description
1 5V	Positive powersupply	2 5V	Positive power supply
3 DB8	8th bit of data bus	4 DB9	9th bit of databus
5 DB10	10th bit of data bus	6 DB11	11th bit of databus
7 DB12	12th bit of data bus	8 DB13	13th bit of databus
9 DB14	14th bit of data bus	10 DB15	15th bit of databus
11 DB7/NC7th bit of data bus(does not need to be connected when using 8-bit mode)		12 DB6/NC 6th bit of databus (does not need to be connected when using 8-bit mode)	
13 DB5/NC5th bit of data bus(does not need to be connected when using 8-bit mode)		14 DB4/NC 4th bit of databus (does not need to be connected when using 8-bit mode)	
15 DB3/NCthird bit of data bus(does not need to be connected when using 8-bit mode)		16 DB2/NC 2nd bit of databus (does not need to be connected when using 8-bit mode)	
17 DB1/NC1st bit of data bus(does not need to be connected when using 8-bit mode)		18 DB0/NC 0 bit of databus (does not need to be connected when using 8-bit mode)	
19 LCD_RS	LCD register / data select signal Low level: register, high level command	20 LCD_WR	LCD write control signal
21 LCD_CS	LCD screen select control signal, low level enable	22 LCD_RST	LCD reset control signal, low reset
23 NC	Undefined, reserved	24 LCD_RD	LCD read control signal
25 TP IRQ	Touch screen interrupt control signal, low level when touch is detected	26 NC	Undefined, reserved
27 NC	Undefined, reserved	28 NC	Undefined, reserved
29 SD_CS	SD card select control signal, low level enable	30 NC	Undefined, reserved
31 MISO	SPI bus input signal	32 MOSI	Touch screen chip select control signal, low level enable
33 EX_CLK SPI	bus clock signal	34 TP_CS	SD card select control signal, low level enable
35 GND	Power ground	36 GND	Power ground

It is important to note that all communication in front of the screen is within the 'closed code' library reserved for the project manager as well.

The manufacturer of the development board, and all rights are reserved to them regarding the implementation of the interface as well as a variety of graphic drawing functions on it and define their color. The screen that actually builds the various polygons in the final user interface

Write cycle of the internal controller on the screen:

The WRX signal is driven from high to low then pulled back to high during the write cycle. The host processor provides information while the display module captures the information from the host processor on the rising edge of the WRX. Figure 1 below shows the write cycle of the DBI Type B interface.

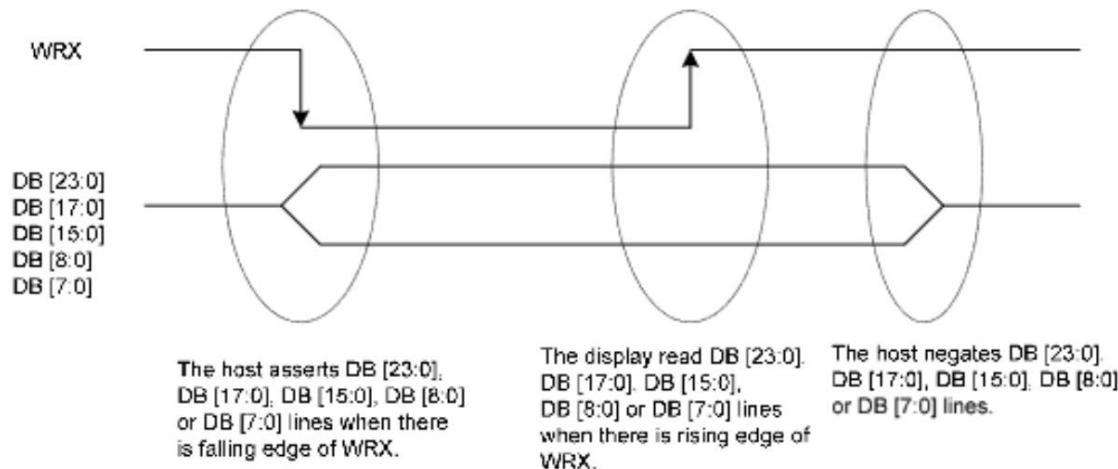


Figure 1: DBI Type B Write Cycle

Note: WRX is an unsynchronized signal that can be terminated when not being used.

When the D/CX signal is driven to low level, the input data on the interface is interpreted as command information.

The D/CX signal can also be pulled to high level when the data is RAM data or command parameter.

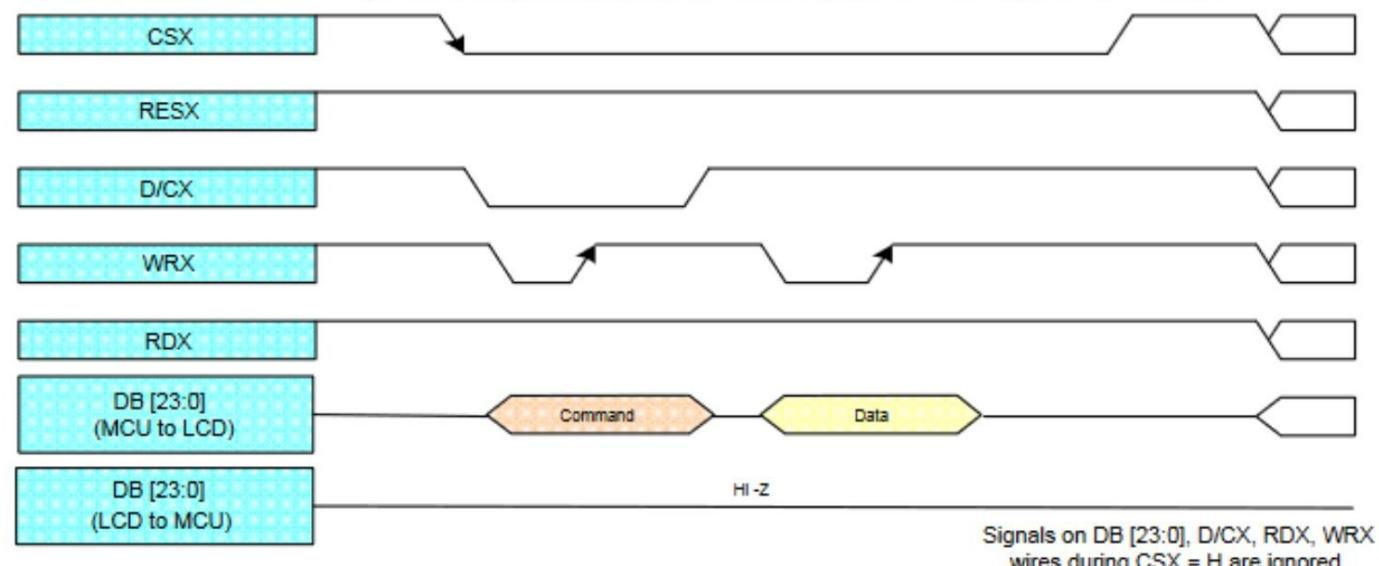


Figure 2: DBI Type B Write Cycle Sequence

The RDX signal is driven from high to low and then pulled back to high during the read cycle. The display module provides information to the host processor while the host processor reads the display module information on the rising edge of the RDX signal. Figure 3 below shows the read cycle of the DBI Type B interface.

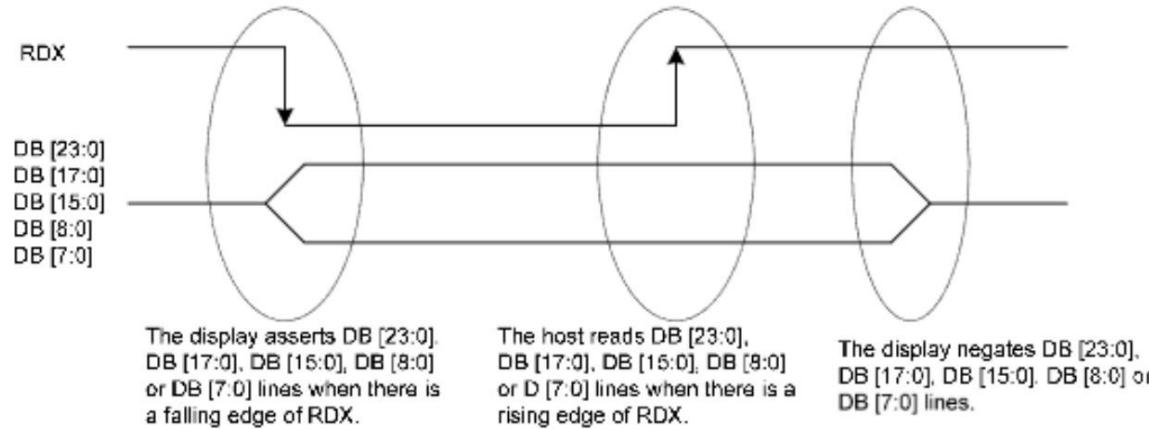


Figure 3: DBI Type B Read Cycle

Note: RDX is an unsynchronized signal that can be terminated when not being used.

When the D/CX signal is driven to the low level, the input data on the interface is interpreted as internal status or parameter data. The D/CX signal can also be pulled to a high level when the data on the interface is RAM data or a command parameter data.

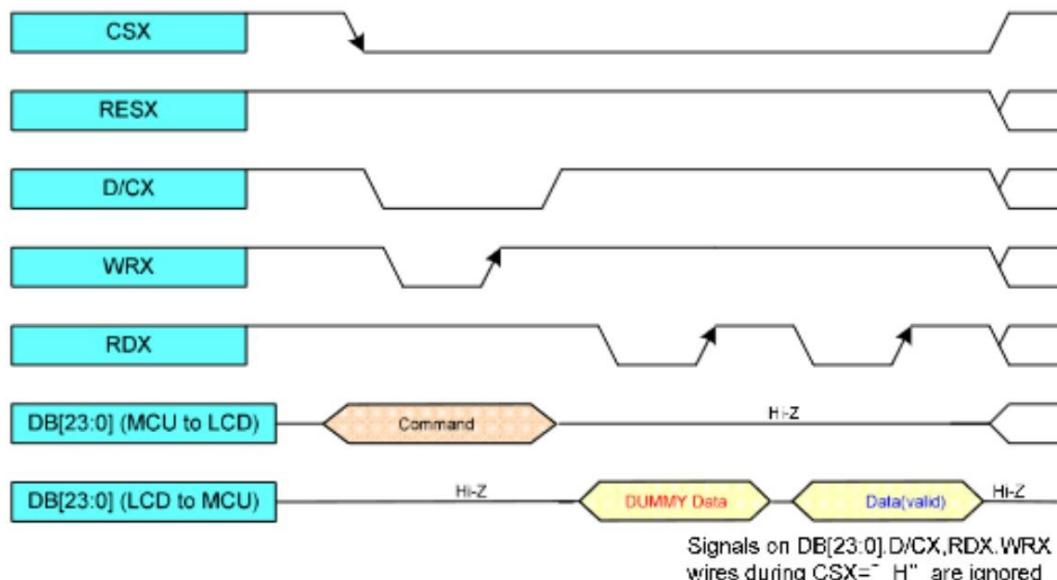


Figure 4: DBI Type B Read Cycle Sequence

Note: Read Data is only valid when the D/CX input is pulled high. If the D/CX signal is driven to low during the read cycle then the display information outputs will be High-Z.



26. The means of development- microcontroller

A microcontroller is a programmable component that contains in one case hardware units that allow it to be a control system.

To control and control hardware devices, where the flexibility of the interface and the speed of communication is the 'name of the game'. This

Unlike computer systems such as the personal computer (PC), central computers of organizations/businesses or cloud servers - and has a fast processor. All intended for data storage and processing - which are memory intensive

, memories, oscillators (clocks), ports for connecting devices

The main components of a microcontroller are an input/output

processor, serial and parallel communication interfaces, an array of counters and timers, an interrupt system and interrupt control, converters

D/A and A/D, therefore it is common to think of a microcontroller as a compromise between the two worlds presented in the opening system

Data control and processing.

These characteristics of the controller make it a very popular platform for didactic needs for educational institutions.

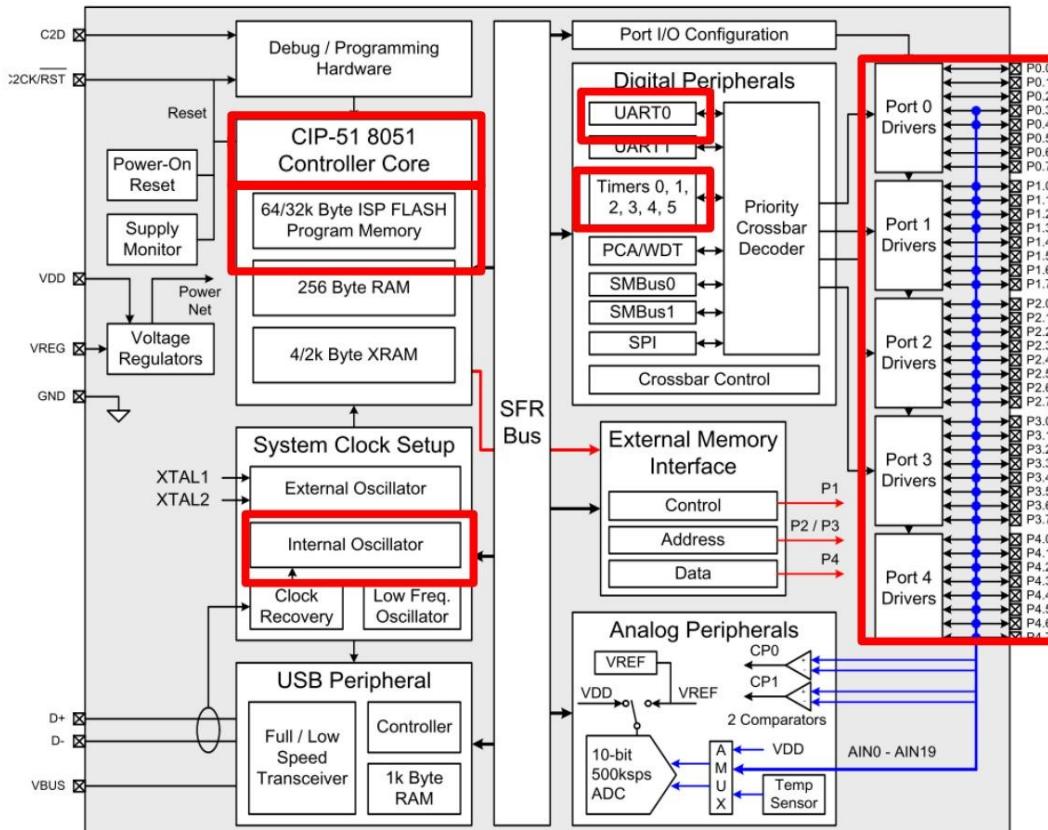
A new system and students, a cheap and common option for small businesses that wish to develop in an experimental/initial way and more importantly - the wide variety of hardware options available in the market allows to 'tailor a controller to the customer's needs' - reasons

of budget, development environment, threshold requirements, etc.

The controller we worked with throughout the school year is **80F8051C** realized on a **development board** that was designed according to an order, personal of the project leader and includes several hardware elements such as LEDs, buttons, potentiometer, S 7' display and a matrix for additional hardware wiring.

This board was used by us and performed all the labs and exercises in preparation for the certification tests. The controller is based on an 8051 processor of Intel but **very different from the similar**, and usually only the names and terminology are varieties in the name. **in the following chapters**

mythological was chosen to delve only into the hardware most relevant to the project, the methods of operation, and how it was defined in our implementation, as which we marked in the above logical scheme.



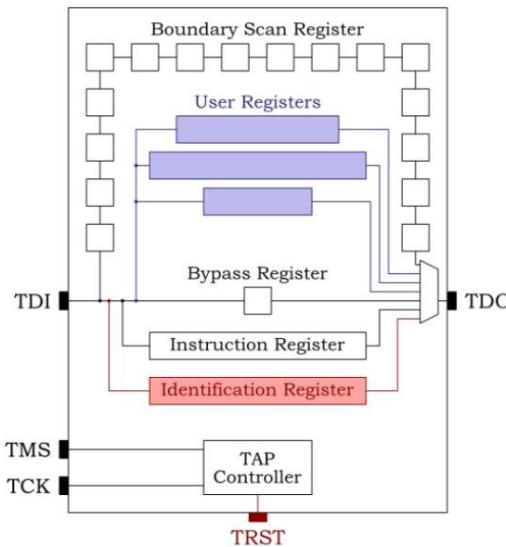
.27 JTAGand real-time development



The connection between a PC and our development board is made using a JTAG connector connected to the USB interface on the computer, and a unique 10-pin Xilinx interface to the board.

Inside the JTAG converter is a relatively small silicon chip whose purpose is to allow "real-time" access to data in the controller's memory and ports. And of course, with it we burn our program on the ROM every time a change is made to the code and the expansion of the functionality.

Only through this component, we were able to DEBUG in realtime, and in fact to make minor changes to the code while developing - at the end of the new software - to run a BreakPoint tool that allows you to monitor the execution of the program line by line, and watch in realtime the contents of registers and ports.



The use of JTAG was developed in the late 1980s as a method for testing large pieces of silicon, which were growing in size at an increasing rate and contained many, many thin BGA components. There was a need for comprehensive inspection of every detail and access to them despite the density of the silicon.

And in fact, a standard of the International Association of Engineers (IEEE) stated that silicon manufacturers will implement JTAG-supporting hardware in their products, which means peripheral hardware for each of the triggers on the silicon, which is actually an array of sliding pads that can be written to and called the "Scan Boundary".

The ability to individually access the entire chip and manipulate it is the one that provides the ability to debug, test, change the content in realtime - therefore in conclusion - **JTAG supporting hardware is needed and at the same time an external software and hardware interface to perform the testability.**

μVision® 5

IDE Keil μVision work environment

The work environment is a well-known and common environment in the world of programming embedded processor systems. Its name is μVision from the Keil company. The company is a veteran in the field, founded in 1982 in Germany, and as early as 1985 was converted into a software house for development adapted to silicon manufacturers, and even implemented the first compiler for the C language for an 8051 microcontroller. In 2005, it was acquired by the ARM corporation, which today leads the development of the world's most advanced processors of the same series. The names which are found in every conceivable device from cell phones to tablets.

μVision combines project management, a real-time environment, source code editing and debugging in a single, easy-to-use interface, as it supports multiple screens/controls that can be deployed anywhere on the screen. A very important ability lies in the file menu through which you can manage the software components - directories, source files, configuration files, and pages

and documentation.

An example of the runtime management window which expands the capabilities of the debugging tool - Debugger and allows you to locate bugs and filter errors, similar to other traditional features, but also use breakpoints and real-time viewing windows of the variables and their content in memory or alternatively the register values of the input-output devices.



The main screen which contains the code editor - includes all the standard features of a modern code editor, allows syntax coloring, text indentation, and optimized for the C language.

```

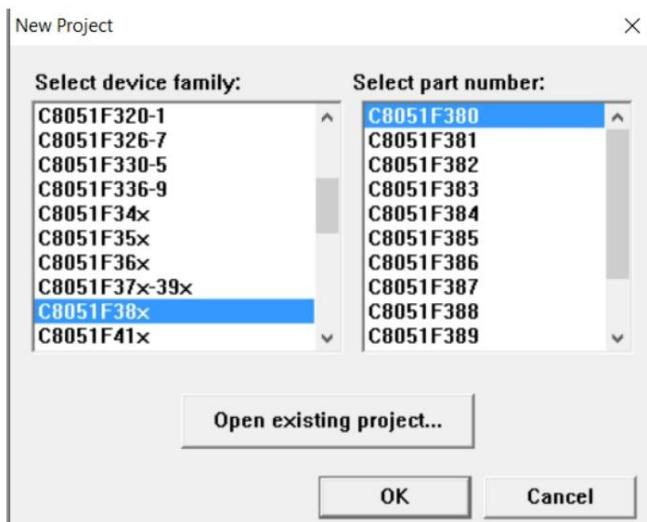
Blinky.c*
65 L
66 int main (void) {
67     int32_t max_num = LED_GetCount() - 1;
68     int32_t num = 0;
69     int32_t dir = 1;
70
71     HAL_Init(); /* Initialize the HAL Library */
72
73     SystemClock_Config(); /* Configure the System Clock */
74
75     LED_Initialize(); /* LED Initialization */
76     Buttons_Initialize(); /* Buttons Initialization */
77
78     while (1) {
79         LED_On();
80         int32_t LED_On(uint32_t num); /* Wait 500ms */
81         while (Buttons_GetState() & (1 << 0)); /* Wait while holding USER button */
82         LED_Off(num); /* Turn specified LED off */
83         osDelay(500); /* Wait 500ms */
84         while (Buttons_GetState() & (1 << 0)); /* Wait while holding USER button */
85
86         num += dir; /* Change LED number */
87         if (dir == 1 && num == max_num) { warning: using the result of an assignment as a condition without parentheses }
88     }
89     else if (num == 0) {
90         dir = 1; /* Change direction to up */
91     }
92     LED...
93
94

```



29 Wizard Configuration Tool

The configuration tool from Silicon Labs is a graphical interface (GUI) that allows to conveniently and quickly configure the microcontroller we are working with (there are a variety of versions embedded in the tool) and to save a huge amount of time planning the registers that control timers, clock frequency, communication protocols, enabling ports, etc.



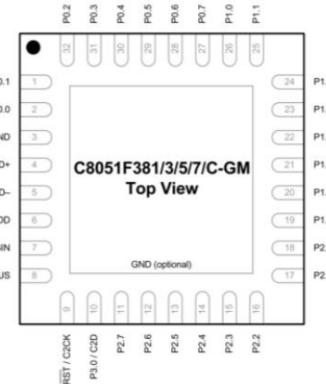
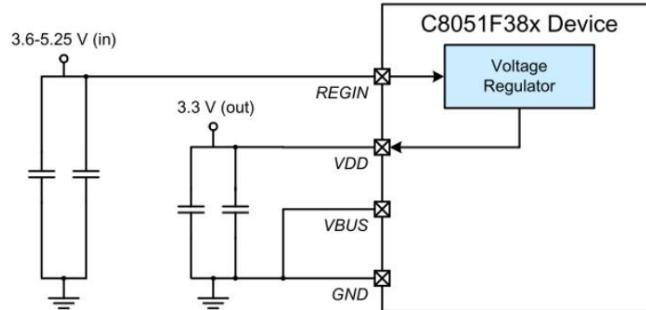
These settings are made only once when starting work on each project, and a C code file is obtained which is an accurate interpretation of the register values that represent the settings we made. This file is added to the list of configuration files in the work environment and is an integral part of it.

In order to deeply understand the definitions that we will present below, one must delve deeper into the first chapters of the book that illustrate the features of the controller with which the development was carried out, and the special registers that store the settings for operating the hardware.



30 electrical properties of the controller

The processor in the system receives an operating voltage of V3.3 and therefore largely dictates currents and voltages in the entire circuit.



. The development board In order to provide this voltage a higher input voltage must be obtained, which is divided and supplies the components.

The conversion will be carried out using a regulator that the manufacturer marked as a 'black box' in the video. Several possible voltage sources :

- A JTAG converter connected to the USB interface on the computer, which provides V5.
- Input from a transformer connected to the wall that supplies 9V, pay attention to the + line in the middle.
- USB TYPE-A input on the board used for both development and operation.

The total current supply (maximum) of the board is mAmp500 in the voltage supply clamps on the board. This limitation

The difficulty in realizing the project because the total current consumption by all the hardware components is over 500 mAmp

And so we had to connect an external voltage of 9 and also use a voltage stabilizer for the network controller.

Parameter	Conditions	Min	Typ	Max	Units
Maximum Total Current through V _{DD} or GND		—	—	500	mA
Maximum Output Current sunk by RST or any Port Pin		—	—	100	mA

The maximum current that the trigger is able to provide is mAmp25 and the peak current that the entire trigger can "tolerate" is mAmp100 .

V_{DD} = 2.7 to 3.6 V, -40 to +85 °C unless otherwise specified

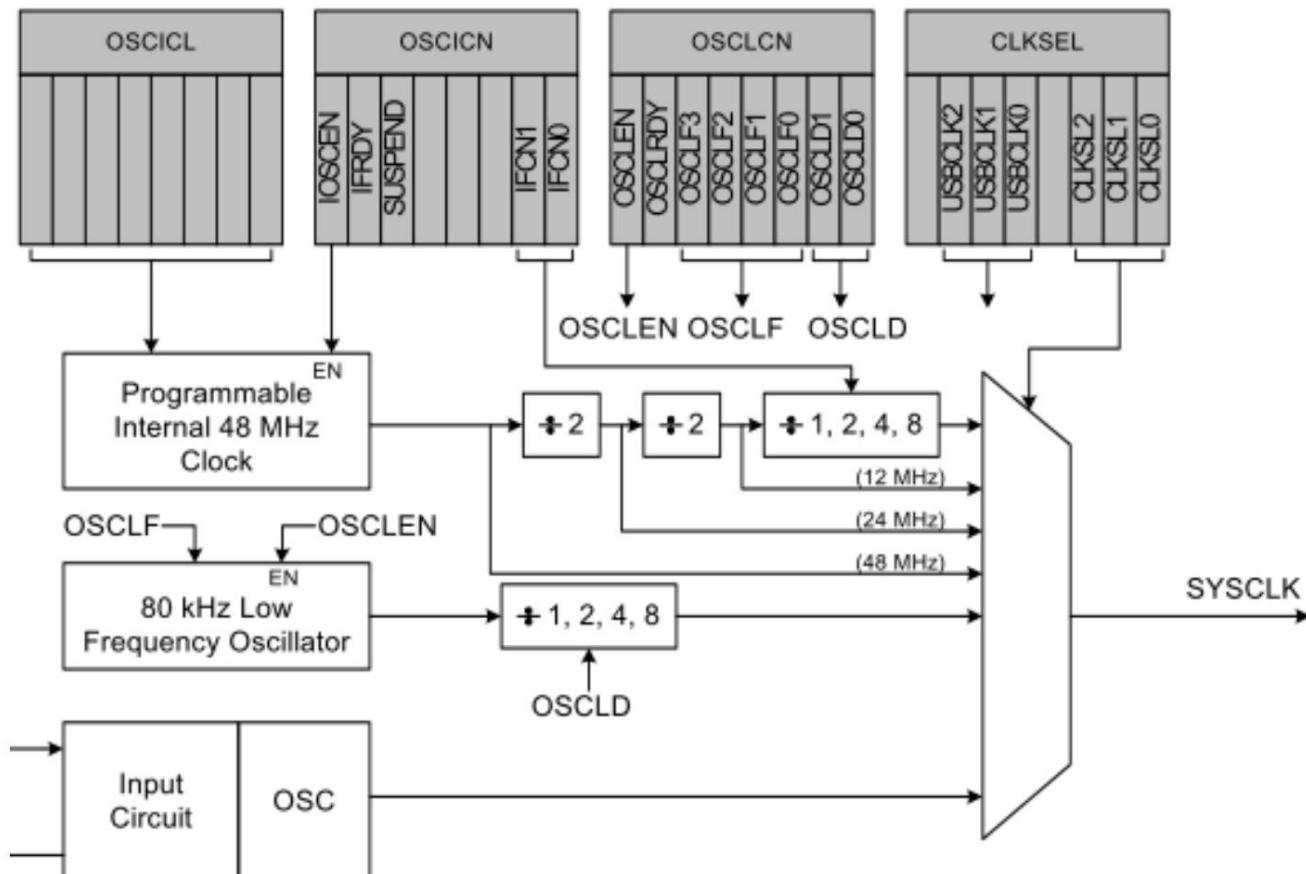
Parameters	Conditions	Min	Typ	Max	Units
Output High Voltage	I _{OH} = -3 mA, Port I/O push-pull	V _{DD} - 0.7			V
	I _{OH} = -10 µA, Port I/O push-pull	V _{DD} - 0.1			
	I _{OH} = -10 mA, Port I/O push-pull	V _{DD} - 0.8			
Output Low Voltage	I _{OL} = 8.5 mA			0.6	V
	I _{OL} = 10 µA			0.1	
	I _{OL} = 25 mA		1.0		
Input High Voltage		2.0			V
Input Low Voltage				0.8	V
Input Leakage Current	Weak Pull-up Off			±1	
	Weak Pull-up On, V _{IN} = 0 V		15	50	µA

.31 clock/oscillator

In the controller we have, there are **2 programmable oscillators** - one high frequency and the other for low frequency. Their programms underway Using OSCICL and OSCICN registers. The maximum speed of the internal clock is **Mhz 48** and that which we also defined.

As with any design - one must always make sure that the clockspeed is not a bottleneck for the realized communication speed, and indeed the network controller (which will be expanded upon later) communicates at a speed of **115,200BPS**. That is, the frequency of the signal (the same because it is a balanced square signal (it is Hz 57,600) and therefore it is understood that the processor will have enough time to process the program without any difficulty, divided into 2 and also have **serial communication**.

The architecture of the internal clock is as follows, and its definition is affected by the registers at the top. There is also an ability to connect an external oscillator, omitted from the drawing.



Setting the registers in the C language in the configuration of the controller:

```

void Oscillator_Init()
{
    FLSCL      = 0x90; //set as system main clock
    CLKSEL     = 0x03; //enable crossbar system clock output
    OSCICN    = 0xC3; //enable internal High Frequency oscillator- 48MHz
}
    
```

Electronics Major - School of Engineering, Technion, May 2019



We will explain the determination of the values, according to the different modes to which the watch can be set - as defined by the manufacturer!

SFR Definition 19.1. CLKSEL: Clock Select

Bit	7	6	5	4	3	2	1	0
Name	USBCLK[2:0]				OUTCLK	CLKSL[2:0]		
Type	R	R/W			R/W	R/W		
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA9; SFR Page = All Pages

Bit	Name	Function
7	Unused	Read = 0b; Write = don't care
6:4	USBCLK[2:0]	USB Clock Source Select Bits. 000: USBCLK derived from the Internal High-Frequency Oscillator. 001: USBCLK derived from the Internal High-Frequency Oscillator / 8. 010: USBCLK derived from the External Oscillator. 011: USBCLK derived from the External Oscillator/2. 100: USBCLK derived from the External Oscillator/3. 101: USBCLK derived from the External Oscillator/4. 110: USBCLK derived from the Internal Low-Frequency Oscillator. 111: Reserved.
3	OUTCLK	Crossbar Clock Out Select. If the SYSCLK signal is enabled on the Crossbar, this bit selects between outputting SYSCLK and SYSCLK synchronized with the Port I/O pins. 0: Enabling the Crossbar SYSCLK signal outputs SYSCLK. 1: Enabling the Crossbar SYSCLK signal outputs SYSCLK synchronized with the Port I/O.
2:0	CLKSL[2:0]	System Clock Source Select Bits. 000: SYSCLK derived from the Internal High-Frequency Oscillator / 4 and scaled per the IFCN bits in register OSCICN. 001: SYSCLK derived from the External Oscillator circuit. 010: SYSCLK derived from the Internal High-Frequency Oscillator / 2. 011: SYSCLK derived from the Internal High-Frequency Oscillator. 100: SYSCLK derived from the Internal Low-Frequency Oscillator and scaled per the OSCLD bits in register OSCLCN. 101-111: Reserved.

Clock selection register:

the desired value

0

000

0

011

SFR Definition 19.3. OSCICN: Internal H-F Oscillator Control

Bit	7	6	5	4	3	2	1	0
Name	IOSCEN	IFRDY	SUSPEND				IFCN[1:0]	
Type	R/W	R	R/W	R	R	R	R/W	
Reset	1	1	0	0	0	0	0	0

SFR Address = 0xB2; SFR Page = All Pages

Bit	Name	Function
7	IOSCEN	Internal H-F Oscillator Enable Bit. 0: Internal H-F Oscillator Disabled. 1: Internal H-F Oscillator Enabled.
6	IFRDY	Internal H-F Oscillator Frequency Ready Flag. 0: Internal H-F Oscillator is not running at programmed frequency. 1: Internal H-F Oscillator is running at programmed frequency.
5	SUSPEND	Internal Oscillator Suspend Enable Bit. Setting this bit to logic 1 places the internal oscillator in SUSPEND mode. The internal oscillator resumes operation when one of the SUSPEND mode awakening events occurs.
4:2	Unused	Read = 00b; Write = don't care
1:0	IFCN[1:0]	Internal H-F Oscillator Frequency Divider Control Bits. The Internal H-F Oscillator is divided by the IFCN bit setting after a divide-by-4 stage. 00: SYSCLK can be derived from Internal H-F Oscillator divided by 8 (1.5 MHz). 01: SYSCLK can be derived from Internal H-F Oscillator divided by 4 (3 MHz). 10: SYSCLK can be derived from Internal H-F Oscillator divided by 2 (6 MHz). 11: SYSCLK can be derived from Internal H-F Oscillator divided by 1 (12 MHz).

Frequency setting register:

the desired value

1

1

0

000

11

.32 memory

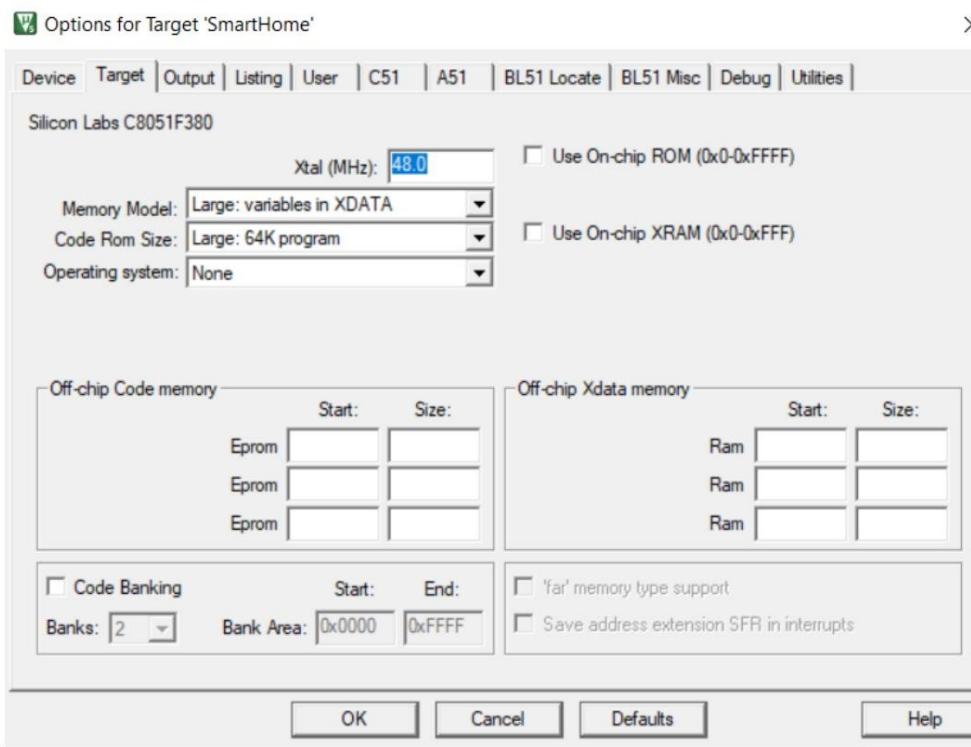
The ROM memory size in our controller is 64KB and our program is stored on it. Also onboard 4 KB of memory External RAM where all data is stored as arrays, variables. The total size of the program at the end of development amounts to about 30 KB and the size of the data about 2 additional KB, which means that in total we used half of the memory space allocated in the system.

The calculation of the total size of the program is possible at the end of the compilation after LINK has been performed for all the files, that is, they have been converted to machine language, and stored in registers. The prompt can be obtained from the work environment:

```
Build Output
linking...
*** WARNING L13: RECURSIVE CALL TO SEGMENT
SEGMENT: ?PR?_SERVER_ISR?WIFI
CALLER: ?PR?REACT?WIFI
Program Size: data=15.2 xdata=1740 code=27034
".\Objects\start" - 0 Error(s), 1 Warning(s).
Build Time Elapsed: 00:00:02
```

We note that our project does not include extensive use of memory mainly because of its nature - a controller that allows real-time communication with input-output devices in the house (sensors and lighting) and therefore no external memory expansion was required.

We will emphasize that we were required to define the size of the memory within the framework of the work environment in which we worked, which will be expanded on in a chapter dedicated to the subject.



33 Ports for connecting devices

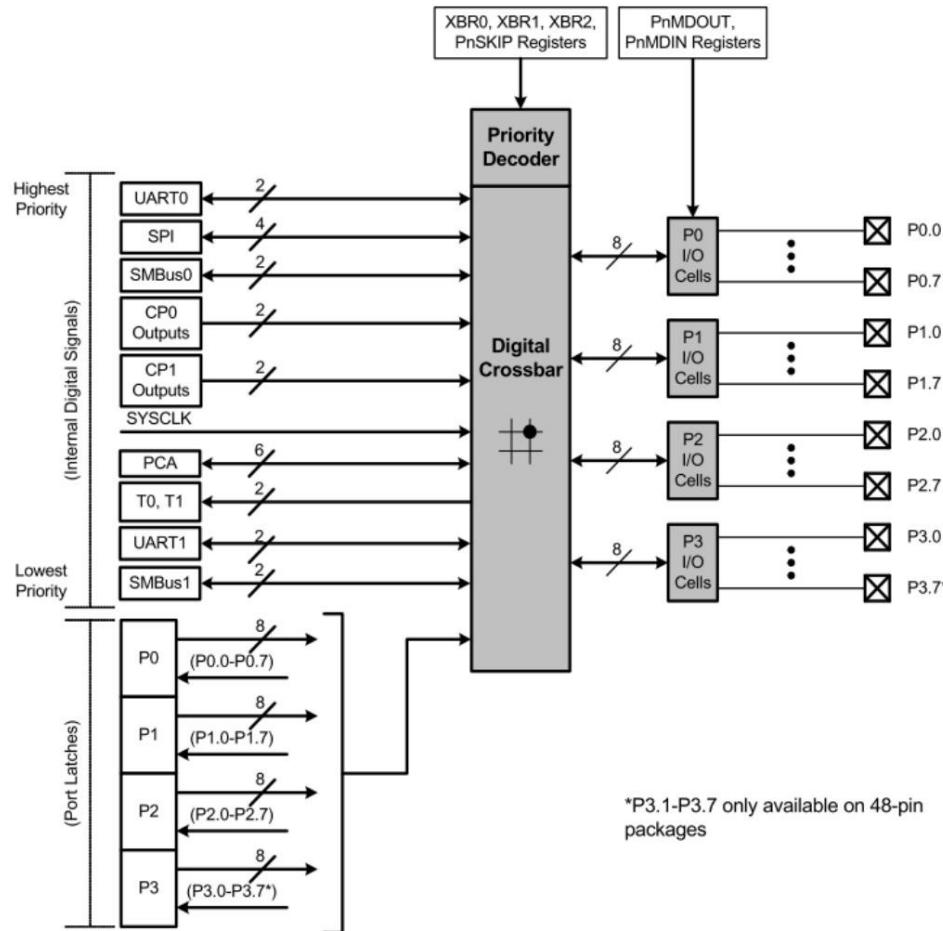
To control a wide and programmable port interface of 5 ports in total numbered 4 - 0.8. Each port contains the chips which

Linked to a 'lock' register which enables the holding of the information intended for reading/writing them.

The 40 minutes 'endure' a logic level of V₅ as 'input' and vice versa. Let's recall that all five ports (which make it possible to push current of up to 25mA if they were defined as output').

The control of the direction of the ports through the **CrossBar** interface that controls partitions through a read option

and writing to them, similar to the operation of the SPI interface. These settings are made using the registers at the top of the diagram:



Setting the registers in the C language the configuration file of the controller:

```
void Port_IO_Init()
{
    P1MDOUT = 0xFF; //port 1 enable for LEDs
    P2MDOUT = 0xB6; //port 2 enable for DATA wires of sensors
    //next are both for UART pins Rx & Tx enable:
    XBR0 = 0x01;
    XBR1 = 0x40;
}
```

Each of the triggers is wired to allow several additional modes under the control of these:

- Pushing low/high current is possible by enabling/releasing the WEAK-PULLUP line which is wired to the switch to leave '0' and allow the base of the gate, and thus the trigger accustomed. A logic '0' level on the line will cause the CMOSVDD OR gate to pull some of the current into it and leave a low current to the trigger itself.

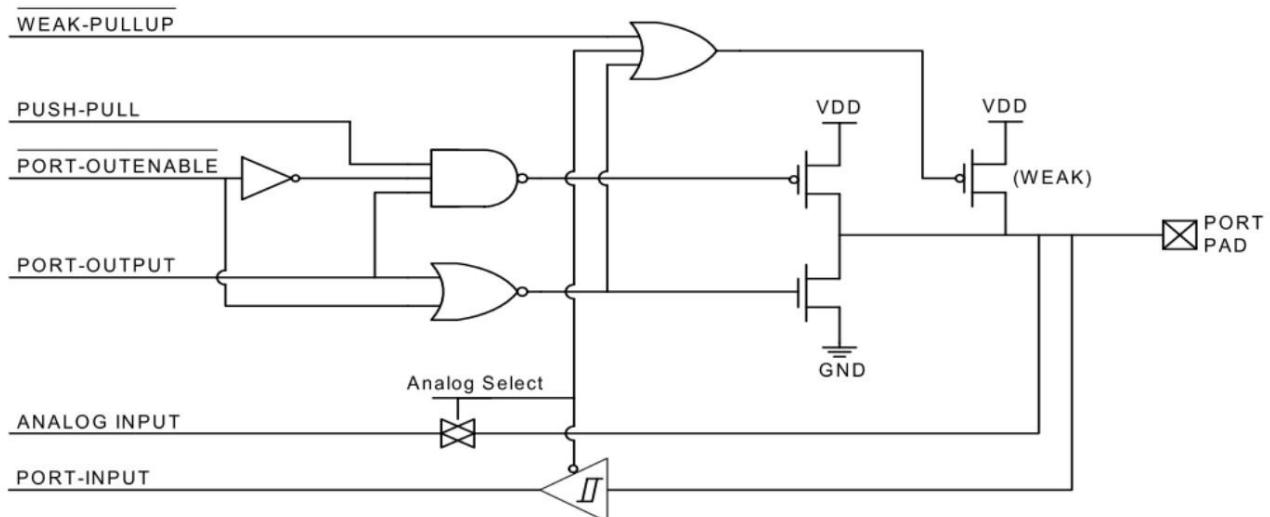
- Determining 'input-output' is possible using the PUSH-PULL switch, OUTENABLE PORT and PORT OUTPUT, which controls the NOT CMOS gate.

necessarily exclude a logical 1 and cause a 'logic 0' on the PUSH-PULL line will cause the NAND gate to keep '0' level the information received in the trigger unchanged (without influence from VDD), meaning the port is defined as 'input'. necessarily output a logical 0 and thus cause a logical level '1' on the PUSH-PULL line will cause the NAND gate to the voltage obtained from VDD to react to the trigger - that, the entire port respectively will show xF0 during initialization

The board, and this value will change of course, immediately after the game with the values obtained on the following lines: necessarily exclude a logical 1 and thus a logical 0 on the OUTPUT PORT line will cause the NAND gate to necessarily output a level 0, so the information that will be written in the end will be The upper gate is not relevant. At the same time the NOR gate is really '0' similarly change the OUTPUT PORT line to '1'. It will indeed dictate or the way out o The 'OUTENABLE PORT' line simply interferes with the previous process and 'disrupts' it or 'enables' it

Thus dictating when to write the information to the latch, therefore this line is used by the processor.

WP	PP	POE	PO	T1	T2	T3	PORT-e mode
H						off	WEAK-PULLUP
		L		L off on			Output -L
L	L			H on off off			Output -H PullUp
	H	L		H on off			Output -H Push





We will explain the determination of the values, according to the different situations in which the shippers can be set: as defined by the manufacturer

SFR Definition 20.1. XBR0: Port I/O Crossbar Register 0

Bit	7	6	5	4	3	2	1	0
Name	CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE1; SFR Page = All Pages

Bit	Name	Function
7	CP1AE	Comparator1 Asynchronous Output Enable. 0: Asynchronous CP1A unavailable at Port pin. 1: Asynchronous CP1A routed to Port pin.
6	CP1E	Comparator1 Output Enable. 0: CP1 unavailable at Port pin. 1: CP1 routed to Port pin.
5	CP0AE	Comparator0 Asynchronous Output Enable. 0: Asynchronous CP0A unavailable at Port pin. 1: Asynchronous CP0A routed to Port pin.
4	CP0E	Comparator0 Output Enable. 0: CP0 unavailable at Port pin. 1: CP0 routed to Port pin.
3	SYSCKE	SYSLK Output Enable. 0: SYSLK unavailable at Port pin. 1: SYSLK output routed to Port pin.
2	SMB0E	SMBus I/O Enable. 0: SMBus I/O unavailable at Port pins. 1: SMBus I/O routed to Port pins.
1	SPI0E	SPI I/O Enable. 0: SPI I/O unavailable at Port pins. 1: SPI I/O routed to Port pins. Note that the SPI can be assigned either 3 or 4 GPIO pins.
0	URT0E	UART I/O Output Enable. 0: UART I/O unavailable at Port pin. 1: UART TX0, RX0 routed to Port pins P0.4 and P0.5.

SFR Definition 20.2. XBR1: Port I/O Crossbar Register 1

Bit	7	6	5	4	3	2	1	0
Name	WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME[2:0]		
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE2; SFR Page = All Pages

Bit	Name	Function
7	WEAKPUD	Port I/O Weak Pullup Disable. 0: Weak Pullups enabled (except for Ports whose I/O are configured for analog mode). 1: Weak Pullups disabled.
6	XBARE	Crossbar Enable. 0: Crossbar disabled. 1: Crossbar enabled.
5	T1E	T1 Enable. 0: T1 unavailable at Port pin. 1: T1 routed to Port pin.
4	T0E	T0 Enable. 0: T0 unavailable at Port pin. 1: T0 routed to Port pin.
3	ECIE	PCA0 External Counter Input Enable. 0: ECI unavailable at Port pin. 1: ECI routed to Port pin.
2:0	PCA0ME[2:0]	PCA Module I/O Enable Bits. 000: All PCA I/O unavailable at Port pins. 001: CEX0 routed to Port pin. 010: CEX0, CEX1 routed to Port pins. 011: CEX0, CEX1, CEX2 routed to Port pins. 100: CEX0, CEX1, CEX2, CEX3 routed to Port pins. 101: CEX0, CEX1, CEX2, CEX3 routed to Port pins. 11x: Reserved.

Register 0XBR

the desired value

0

0

0

0

0

0

1

1XBR register

the desired value

0

1

0

0

0

000

**SFR Definition 20.10. P1MDOUT: Port 1 Output Mode**

Bit	7	6	5	4	3	2	1	0
Name	P1MDOUT[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA5; SFR Page = All Pages

Bit	Name	Function
7:0	P1MDOUT[7:0]	Output Configuration Bits for P1.7–P1.0 (respectively). These bits are ignored if the corresponding bit in register P1MDIN is logic 0. 0: Corresponding P1.n Output is open-drain. 1: Corresponding P1.n Output is push-pull.

SFR Definition 20.14. P2MDOUT: Port 2 Output Mode

Bit	7	6	5	4	3	2	1	0
Name	P2MDOUT[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA6; SFR Page = All Pages

Bit	Name	Function
7:0	P2MDOUT[7:0]	Output Configuration Bits for P2.7–P2.0 (respectively). These bits are ignored if the corresponding bit in register P2MDIN is logic 0. 0: Corresponding P2.n Output is open-drain. 1: Corresponding P2.n Output is push-pull.

Register enable port 1

The entire portis usedto connect
to 8 built-in LEDson a board
The development simulates 8bulbs
in the house lighting

the desired value

1111,1111

Register Enable Port 2

Quilmuses informationfor
climate control sensors and location
control as functionsof the 'smart
home' and communication
indication via arRGB light

the desired value

1011,0110

Summary ofthe allocationof triggers in the project:

```
*****  
SBIT DECLARATION  
*****  
//LEDs are connected to each wire to represent room/area light  
sbit Cameras = P1^2;  
sbit Alarm = P1^3;  
sbit MainLights = P1^4;  
sbit PetFeeder = P1^5;  
sbit FrontGate = P1^6;  
sbit Heater = P1^7;  
//RGB LEDs are connected to each wire to represent AirCon_ON status  
sbit AirCon_OFF = P2^0; //OFF = RED  
sbit AirCon_WAIT = P2^1; //Waiting for Clock = BLUE  
sbit AirCon_ON = P2^2; //ON = GREEN  
sbit SDA= P2^3; //I2C DATA  
sbit Garage_OFF = P2^4; //OFF = RED  
sbit Garage_WAIT = P2^5; //Waiting for Clock = BLUE  
sbit Garage_ON = P2^6; //ON = RED  
sbit SCL= P2^7; //I2C CLK  
//Wires for date stream of each sensor  
sbit Trigger_Car= P0^0;  
sbit Echo_Car= P0^1;  
sbit Trigger_Pet= P0^2;  
sbit Echo_Pet= P0^3;  
sbit TX= P0^4;  
sbit RX= P0^5;  
sbit DHT_DATA= P0^6;
```

We note that port 3 is usedfor communicationwith an LCD touch screen in a closed code package frame, and its configuration is done separately.



.34 timers/counters

A timer is a counter that counts pulses that reach it at a constant rate, so it actually measures time. The distinctions it makes are made according to the role for which it is required. If you connect to the input of the counter a signal derived from the CPU clock (that is, it arrives at a uniform rate), the component functions as a TIMER. If you connect the counter input to an external source of signals, which produces pulses not necessarily at a uniform rate, the component will function as an event counter (COUNTER).

In the controller there are 6 different counters for the use of the developer. 2 of them are completely identical to the counters known from the 8051 processor and 4 more are multi-use timers. In our implementation, timer #1 is used for serial communication and timer #2 for measuring time for an ultrasonic sensor.

Timer 0 and Timer 1 Modes:	Timer 2, 3, 4, and 5 Modes:
13-bit counter/timer	16-bit timer with auto-reload
16-bit counter/timer	
8-bit counter/timer with auto-reload	Two 8-bit timers with auto-reload
Two 8-bit counter/timers (Timer 0 only)	

In addition, as in many systems, the controller contains a WatchDog clock which is an important tool that allows a window of time to reset the system, if there is a requirement for this. The WatchDog functions in such a way that it expects an interrupt from a counter that counts down a fixed amount of time that can be set.

If the count ends, the system is reset and the counter is reloaded. If you want to cancel this option, you can do so - and that's what we chose to do, although it is important to note that it is acceptable to use the function to disable WatchDog in final products because in reality even a system designed to live 'always' requires an update maintenance and initialization routine, and it is possible to plan so that they are not felt by the end user.

Setting the registers in the C language in the configuration file of the controller:

```

void PCA_Init()
{
    //Watchdog disable for proper continuity of system
    PCA0MD    &= ~0x40;
    PCA0MD    = 0x00;
}

void Timer_Init()
{
    //Timers 1,2,5 enable
    TCON      = 0x40; //50
    //Timer 1 mod 8bit auto-reload for UART0 auto BAUD rate handling
    TMOD      = 0x20;
    //sysclock as configured in Oscillator_Init
    CKCON    = 0x08; //0C
    //115200 BAUD equivalent value to reload in hex (at 48MHz)
    TH1      = 0x30;
    //Timer 2 mod 16bit timer- for ultrasonic sensors
    TMR2CN   = 0x04;
}

```



SFR Definition 26.1. CKCON: Clock Control

Bit	7	6	5	4	3	2	1	0
Name	T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA[1:0]	
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x8E; SFR Page = All Pages

Bit	Name	Function
7	T3MH	Timer 3 High Byte Clock Select. Selects the clock supplied to the Timer 3 high byte (split 8-bit timer mode only). 0: Timer 3 high byte uses the clock defined by the T3XCLK bit in TMR3CN. 1: Timer 3 high byte uses the system clock.
6	T3ML	Timer 3 Low Byte Clock Select. Selects the clock supplied to Timer 3. Selects the clock supplied to the lower 8-bit timer in split 8-bit timer mode. 0: Timer 3 low byte uses the clock defined by the T3XCLK bit in TMR3CN. 1: Timer 3 low byte uses the system clock.
5	T2MH	Timer 2 High Byte Clock Select. Selects the clock supplied to the Timer 2 high byte (split 8-bit timer mode only). 0: Timer 2 high byte uses the clock defined by the T2XCLK bit in TMR2CN. 1: Timer 2 high byte uses the system clock.
4	T2ML	Timer 2 Low Byte Clock Select. Selects the clock supplied to Timer 2. If Timer 2 is configured in split 8-bit timer mode, this bit selects the clock supplied to the lower 8-bit timer. 0: Timer 2 low byte uses the clock defined by the T2XCLK bit in TMR2CN. 1: Timer 2 low byte uses the system clock.
3	T1	Timer 1 Clock Select. Selects the clock source supplied to Timer 1. Ignored when C/T1 is set to 1. 0: Timer 1 uses the clock defined by the prescale bits SCA[1:0]. 1: Timer 1 uses the system clock.
2	T0	Timer 0 Clock Select. Selects the clock source supplied to Timer 0. Ignored when C/T0 is set to 1. 0: Counter/Timer 0 uses the clock defined by the prescale bits SCA[1:0]. 1: Counter/Timer 0 uses the system clock.
1:0	SCA[1:0]	Timer 0/1 Prescale Bits. These bits control the Timer 0/1 Clock Prescaler: 00: System clock divided by 12 01: System clock divided by 4 10: System clock divided by 48 11: External clock divided by 8 (synchronized with the system clock)

Register definition and choice counters

the value the desired

0

0

0

0

1

0

00



SFR Definition 26.4. TMOD: Timer Mode

Bit	7	6	5	4	3	2	1	0
Name	GATE1	C/T1	T1M[1:0]		GATE0	C/T0	T0M[1:0]	
Type	R/W	R/W	R/W		R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x89; SFR Page = All Pages

Bit	Name	Function
7	GATE1	Timer 1 Gate Control. 0: Timer 1 enabled when TR1 = 1 irrespective of INT1 logic level. 1: Timer 1 enabled only when TR1 = 1 AND INT1 is active as defined by bit IN1PL in register IT01CF (see SFR Definition 16.7).
6	C/T1	Counter/Timer 1 Select. 0: Timer: Timer 1 incremented by clock defined by T1M bit in register CKCON. 1: Counter: Timer 1 incremented by high-to-low transitions on external pin (T1).
5:4	T1M[1:0]	Timer 1 Mode Select. These bits select the Timer 1 operation mode. 00: Mode 0, 13-bit Counter/Timer 01: Mode 1, 16-bit Counter/Timer 10: Mode 2, 8-bit Counter/Timer with Auto-Reload 11: Mode 3, Timer 1 Inactive
3	GATE0	Timer 0 Gate Control. 0: Timer 0 enabled when TR0 = 1 irrespective of INT0 logic level. 1: Timer 0 enabled only when TR0 = 1 AND INT0 is active as defined by bit IN0PL in register IT01CF (see SFR Definition 16.7).
2	C/T0	Counter/Timer 0 Select. 0: Timer: Timer 0 incremented by clock defined by T0M bit in register CKCON. 1: Counter: Timer 0 incremented by high-to-low transitions on external pin (T0).
1:0	T0M[1:0]	Timer 0 Mode Select. These bits select the Timer 0 operation mode. 00: Mode 0, 13-bit Counter/Timer 01: Mode 1, 16-bit Counter/Timer 10: Mode 2, 8-bit Counter/Timer with Auto-Reload 11: Mode 3, Two 8-bit Counter/Timers

Register definition

my situation work

the value the desired

0

0

10

0

0

00



SFR Definition 26.3. TCON: Timer Control

Bit	7	6	5	4	3	2	1	0
Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x88; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function	Register definition
7	TF1	Timer 1 Overflow Flag. Set to 1 by hardware when Timer 1 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 1 interrupt service routine.	0
6	TR1	Timer 1 Run Control. Timer 1 is enabled by setting this bit to 1.	1
5	TF0	Timer 0 Overflow Flag. Set to 1 by hardware when Timer 0 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 0 interrupt service routine.	0
4	TR0	Timer 0 Run Control. Timer 0 is enabled by setting this bit to 1.	0
3	IE1	External Interrupt 1. This flag is set by hardware when an edge/level of type defined by IT1 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 1 service routine in edge-triggered mode.	0
2	IT1	Interrupt 1 Type Select. This bit selects whether the configured INT1 interrupt will be edge or level sensitive. INT1 is configured active low or high by the IN1PL bit in the IT01CF register (see SFR Definition 16.7). 0: INT1 is level triggered. 1: INT1 is edge triggered.	0
1	IE0	External Interrupt 0. This flag is set by hardware when an edge/level of type defined by IT1 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 0 service routine in edge-triggered mode.	0
0	IT0	Interrupt 0 Type Select. This bit selects whether the configured INT0 interrupt will be edge or level sensitive. INT0 is configured active low or high by the IN0PL bit in register IT01CF (see SFR Definition 16.7). 0: INT0 is level triggered. 1: INT0 is edge triggered.	0

Register definition
Rulings and level
Logical

the value
the desired

**SFR Definition 26.9. TMR2CN: Timer 2 Control**

Bit	7	6	5	4	3	2	1	0
Name	TF2H	TF2L	TF2LEN	TF2CEN	T2SPLIT	TR2	T2CSS	T2XCLK
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xC8; SFR Page = 0; Bit-Addressable

Bit	Name	Function	the desired value
7	TF2H	Timer 2 High Byte Overflow Flag. Set by hardware when the Timer 2 high byte overflows from 0xFF to 0x00. In 16 bit mode, this will occur when Timer 2 overflows from 0xFFFF to 0x0000. When the Timer 2 interrupt is enabled, setting this bit causes the CPU to vector to the Timer 2 interrupt service routine. This bit is not automatically cleared by hardware.	0
6	TF2L	Timer 2 Low Byte Overflow Flag. Set by hardware when the Timer 2 low byte overflows from 0xFF to 0x00. TF2L will be set when the low byte overflows regardless of the Timer 2 mode. This bit is not automatically cleared by hardware.	0
5	TF2LEN	Timer 2 Low Byte Interrupt Enable. When set to 1, this bit enables Timer 2 Low Byte interrupts. If Timer 2 interrupts are also enabled, an interrupt will be generated when the low byte of Timer 2 overflows.	0
4	TF2CEN	Timer 2 Low-Frequency Oscillator Capture Enable. When set to 1, this bit enables Timer 2 Low-Frequency Oscillator Capture Mode. If TF2CEN is set and Timer 2 interrupts are enabled, an interrupt will be generated on a falling edge of the low-frequency oscillator output, and the current 16-bit timer value in TMR2H:TMR2L will be copied to TMR2RLH:TMR2RL.	0
3	T2SPLIT	Timer 2 Split Mode Enable. When this bit is set, Timer 2 operates as two 8-bit timers with auto-reload.	0
2	TR2	Timer 2 Run Control. Timer 2 is enabled by setting this bit to 1. In 8-bit mode, this bit enables/disables TMR2H only; TMR2L is always enabled in split mode.	1
1	T2CSS	Timer 2 Capture Source Select. This bit selects the source of a capture event when bit T2CE is set to 1. 0: Capture source is USB SOF event. 1: Capture source is falling edge of Low-Frequency Oscillator.	0
0	T2XCLK	Timer 2 External Clock Select. This bit selects the external clock source for Timer 2. However, the Timer 2 Clock Select bits (T2MH and T2ML in register CKCON) may still be used to select between the external clock and the system clock for either timer. 0: Timer 2 clock is the system clock divided by 12. 1: Timer 2 clock is the external clock divided by 8 (synchronized with SYSCLK).	0

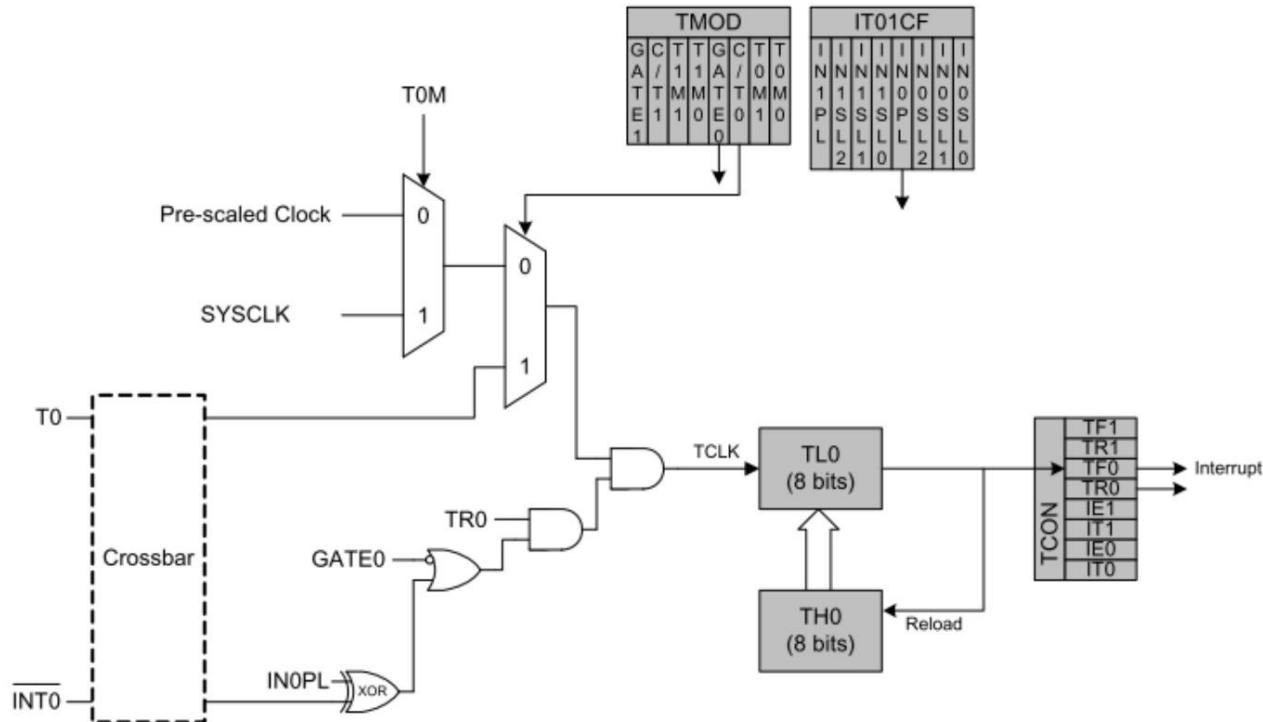
Register setting

Timer 2) definition

Also similar to (-5



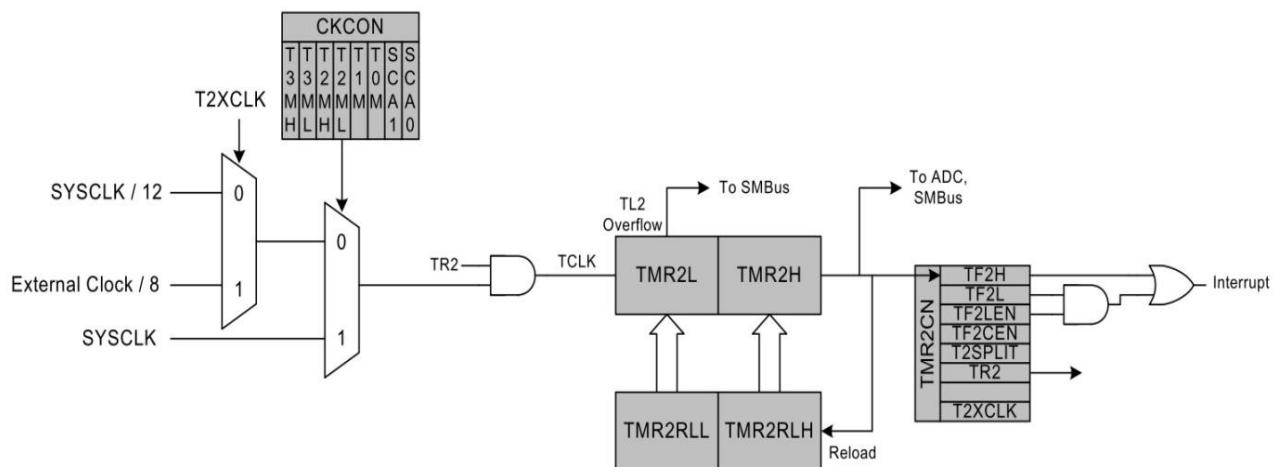
Below is a logic diagram of 1TIMER configured for BIT8 working mode and automatic loading in order to measure time for UART serial communication.



Below is a logic diagram of 52TIMER configured for BIT16 working mode

2TIMER was set to measure time for the distance sensor, 5TIMER was set to measure response delay

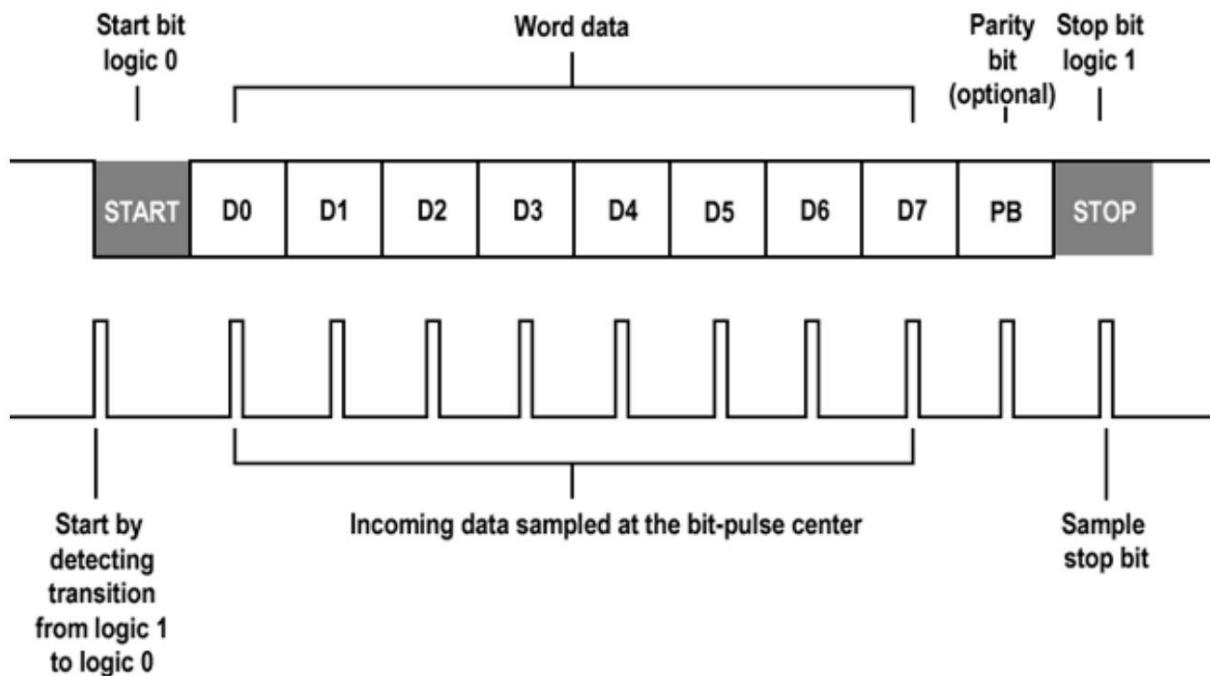
From a client in the system who requests to send a command via the cell phone to the component.



.35 UART communication

Transmitter Receiver Asynchronous Universal is **a synchronous communication protocol** meaning **without existence of a common clock signal between the two** communicating end units. In this way **the microcontroller can communicate with the devices and more** external such as computer, modem, bluetooth.

The information in the communication is transmitted within the framework of **rules for the composition of an information house and rules for sending/receiving at a specified rate.**



The UART is a **programmable component**, because different devices transmit different speeds, and also in bit composition changed (within the rules of the protocol). All information received and sent should be written to a register that will hold this data until reading/writing. This register is called SBUF and in our controller 0SBUF because there are several UART interfaces.

The SBUF register actually contains two separate registers located at the same address. When **writing information** to SBUF there . When **reading information** from the SBUF this information (received) arrives. This is to a register known as DATA TX transmission information, comes first in serial communication (from a register known as DATA RX).

In order to activate the UART the provider needs a squarewave (CLOCK) according to the required communication rate. A each clock pulse the UART performs a certain operation, therefore **the clock frequency must be higher than the transmission and reception frequency** so that the UART will be able to perform several operations between one bit to another. A clock frequency 16 times the communication frequency is accepted, indeed in the chapter of frequency The watch showed that met this requirement.

At this frequency it is actually divided (gal). , The signal frequency to the UART is obtained by dividing the CPU clock frequency squared. The UART clock frequency is according to the required communication rate BAUD and is determined according to the following formula:

$$\text{UARTBaudRate} = \frac{1}{2} \times \text{T1_Overflow_Rate}$$



To createthis frequency, use~~TIMER~~ whose output is connectedo the UARTas aclock generator. Kotsev programmers the Load the **timewith an initial value**, which will cause~~to~~generate the ,Time to work in RELOAD AUTO mode UAR~~to~~ock frequency. We will place the given and desiredvalues in the equation (from left right).

$$\begin{array}{c}
 = \frac{1}{f} = \frac{1}{\frac{1}{f_0} \cdot \frac{1}{N}} = \frac{1}{f_0} \cdot N \\
 = \frac{1}{f_0} = \frac{1}{\frac{1}{f_0} \cdot \frac{1}{N}} = \frac{1}{f_0} \cdot N \\
 = \frac{1}{f_0} = \frac{1}{\frac{1}{f_0} \cdot \frac{1}{N}} = \frac{1}{f_0} \cdot N \\
 = \frac{1}{f_0} = \frac{1}{\frac{1}{f_0} \cdot \frac{1}{N}} = \frac{1}{f_0} \cdot N
 \end{array}$$

(decimal base) = 48

= 30H(HEX base)

12 - Converthe VALUE RELOADvalue The timer inputreceives the clocksignal ofthe CPU in thedivider. tobase 16and then to base2 and perform 2's complemetoperations onthe resulting binarynumber.

Convert the number obtainedn the parables to 2 - base 16 and then to decimal base, and this is the numberthat needsto be loaded to register 1TH 1(using a timer) in order to receive~~the~~ required communicationrate (Baud Rate) (register 1Treceives the The value0.

After initializingthe timer and the UART,the UARTstarts transmittingand receiving.During transmission,the UART transmits the The information bits, eacfor 16 clockpulses. During reception, as soon~~s~~ theUART detects that thdine drops to0"-", it is Waits 8 clock cycles andperforms another sample for verification. Thercount 16more clock cycles and collecthe the bitof information.

Below is thedefinition ofthe register value that defines the UART, aswell asthe main function thainitializes the restule The components - WatchDog, countersports and clock havebeen explainedso far.

```

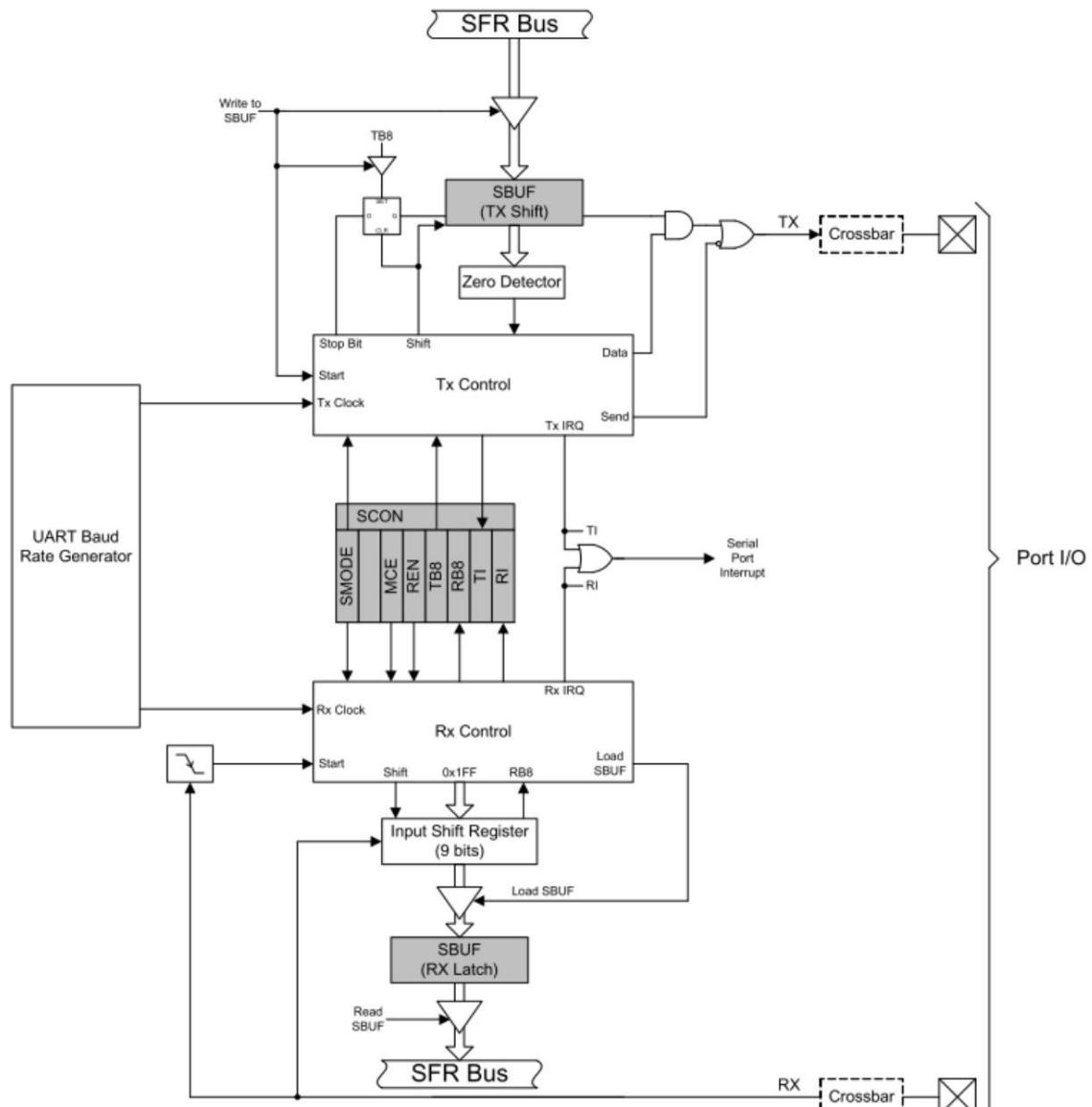
void UART_Init()
{
    SCON0      = 0x30;
}

void Init_Device(void)
{
    //Here all above functions are called
    PCA_Init();
    Timer_Init();
    UART_Init();
    Port_IO_Init();
    Oscillator_Init();
}

```

Below is a logic diagram of the UART Olauncher/receiver in the controller:

Target Baud Rate (bps)	Actual Baud Rate (bps)	Baud Rate Error	Oscillator Divide Factor	Timer Clock Source	SCA1-SCA0 (pre-scale select*)	T1M	Timer 1 Reload Value (hex)
230400	230769	0.16%	208	SYSCLK	XX	1	0x98
115200	115385	0.16%	416	SYSCLK	XX	1	0x30
57600	57692	0.16%	832	SYSCLK / 4	01	0	0x90
28800	28846	0.16%	1664	SYSCLK / 4	01	0	0x30
14400	14388	0.08%	3336	SYSCLK / 12	00	0	0x75
9600	9615	0.16%	4992	SYSCLK / 12	00	0	0x30
2400	2404	0.16%	19968	SYSCLK / 48	10	0	0x30

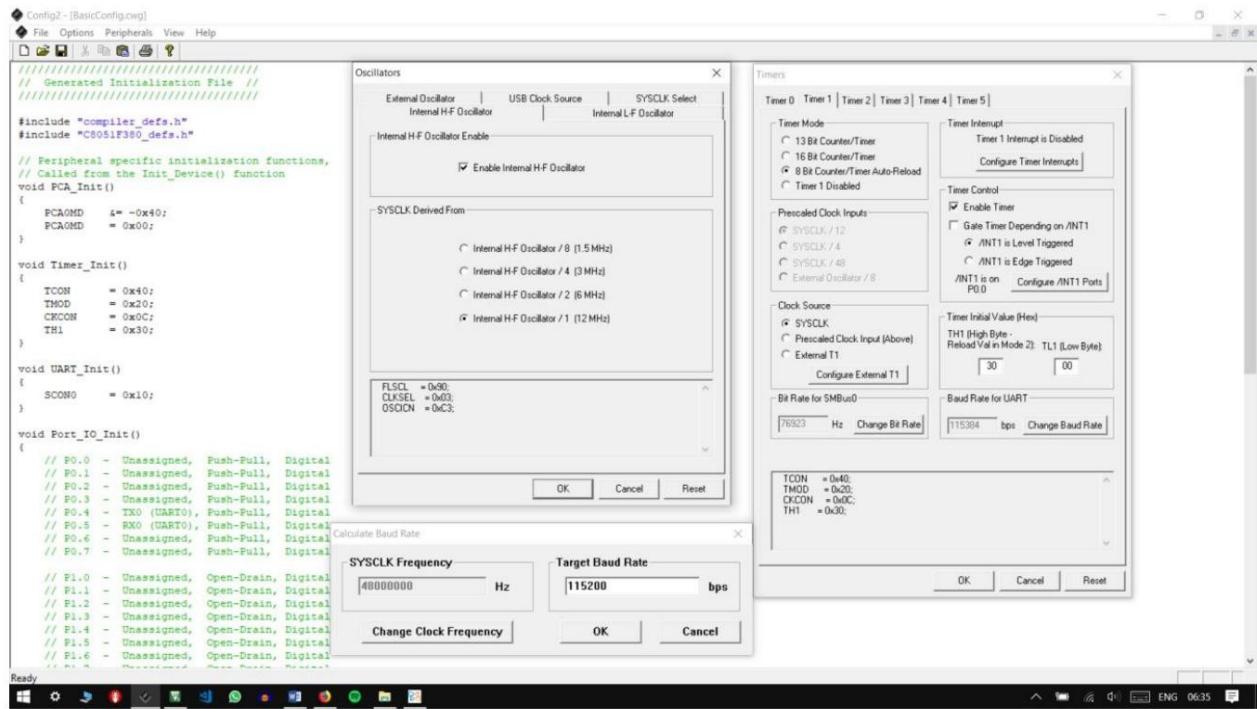




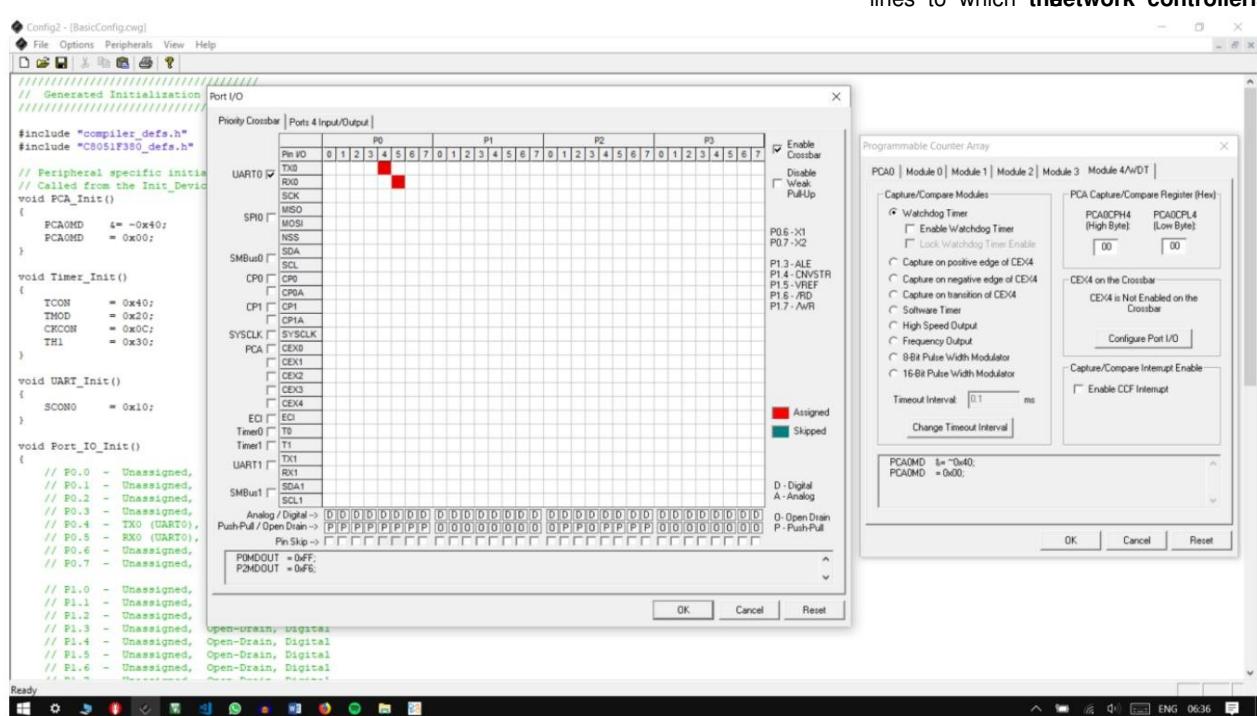
Electronics Major - School of Engineering, Technion, May 2019

Below is a screenshot that illustrates the settings we actually made in our project regarding the desired internal clock frequency.

activation of 1Timer for the purpose of allocating time for transmission within the **UART protocol** - which is derived from the bit transmission rate we chose 115,200.; BAUD



Below is a screenshot that illustrates settings we made in visual port matrix which defines which of the ports will be accessible for use for connecting input-output devices and Tx lines are marked red, which are the data lines to which the network controller is connected.





.36 String functions deepening

As you can understand, throughout the development a very important aspect of the functionality that we wanted to offer developed

The 'smart home' controller - the ability **request information** from the sensors in the home and display it within the application on the cell phone (**DataOnDemand**). Even without this ability, the information received from the sensor has to be **processed so that it is valid**

And not necessarily quantitative, **A short, matter-of-fact meaning**

Therefore - the code we developed makes a lot of use of **strings and functions that manipulate** them, in order to copy, connect, cut, convert and reset - all the modifiers that are these strings, so that they can be displayed

Or alternatively in **theSmartPhone application** as part of the WiFi communication , on the **graphics screen**

As a result we were required to work with a popular **code library for handling strings**, which usually comes with an environment rule development, and in this case one suitable for **the C language**:> **h.string <**

The library makes extensive use of **'pointers'** - a programming tool designed to refer to and control the location of variables in memory, in order to allow functions to access them thus saving unnecessary **operations** such as copying variables and passing them as parameters that take up additional space in memory .

:miscellaneous Below is a small part of the functions we used, the full library contains 22 functions

Function and description	function
void * memset (void * str, int size_t n) Copies the character to the first n characters of the string str .	5
char * strcat (char* dest, const char src) Concatenates (concatenates) the string src with the end of the string dest.	6
char * strncat (char * dest,const char * src, size_t n) (concatenates) the string src with the place in the string dest.	7
char * strchr (const char * str, int) searches for the first occurrence of the character in place in the string str and returns it (its character value)	8
int strcmp(const char * str1, const char * str2) Compares the string 1 with 2 str and returns yes/no if there is an equality of form 1/0.	9
int strncmp (const char * str1, const char * str2, size_t n) Compares first n bytes in strings 1 and 2 str and returns yes/no if equality exists in the form 1/0.	10
char * strcpy (char* dest, const char src) Copies the string src into the string dest. char* strncpy (char * dest, const char * src, size_t n) Copies up to n characters from the string src into the string dest.	12
Size_t strlen(str char * str) Calculates and returns the length of the string str up to the null character and not including the terminating character.	16
char * strstr (const char * haystack, const char * needle) Searches for and returns the first instance of the string needle within the string haystack if found, returns null .	20



.37 Bibliography

address	site/organization	essence of chapters
https://www.silabs.com/documents/public/data-sheets/C8051F38x.pdf	SiliconLabs	3-9 micro-cattle
https://en.wikipedia.org/wiki/Internet_protocol_suite https://en.wikipedia.org/wiki/Port_(computer_networking) https://en.wikipedia.org/wiki/Transmission_Control_Protocol https://en.wikipedia.org/wiki/Internet_Protocol https://en.wikipedia.org/wiki/User_Datagram_Protocol http://stackoverflow.com/questions/176264/what-is-the-difference-between-a-uri-a-url-and-a-urn	Wikipedia forums	10-14 protocol communication Internet
https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html https://en.wikipedia.org/wiki/List_of_HTTP_status_codes		
https://nurdspace.nl/ESP8266	NURDspace Wiki	15-21 - ESP8266 hardware and features
https://cdn.sparkfun.com/datasheets/Wireless/WiFi/Command%20Doc.pdf	SparkFun	22-26 communication design and definitions
https://www.electronicwings.com/8051/dht11-sensor-interfacing-with-8051 https://www.electronicwings.com/8051/ultrasonic-module-hc-sr04-interfacing-with-8051 http://www.lcdwiki.com/4.0inch_Arduino_Display-Mega2560	28-32 ElectronicWings	hardware his censors
http://www2.keil.com/mdk5/uvision/	Keil	33 environment work
https://github.com/esp8266/esp8266-wiki/wiki https://github.com/ExploreEmbedded/Code-Libraries/tree/master/CodeLibraries/8051/UART https://www.esp8266.com/ https://www.instructables.com/id/Using-the-ESP8266-module/	GitHub , (ESP8266) Instructables tutorials, ESP8266 Community Forum	- Libraries code and examples