A detailed technical paper

with four accompanying videos, showing

how an “Internet of Things” sensor

can be designed, built & programmed.

|  |  |
| --- | --- |
|  |  |
|  |  |

August 2017

Mike McKean mike.mckean@msm-systems.com 07539 341 216

DISCLAIMER; I believe that what I have written here is correct. However, I will not accept responsibility for any errors. Always use the manufacturers specifications. I have included url’s to the manufacturers documents.

Contents

[Introduction 3](#_Toc490295061)

[Maximising RoI 3](#_Toc490295062)

[Supporting Videos 3](#_Toc490295063)

[The 1980’s and 1990’s 3](#_Toc490295064)

[What’s changed in 30 years 3](#_Toc490295065)

[Development Issues 3](#_Toc490295066)

[Reading the Manual 6](#_Toc490295067)

[Development Tools 6](#_Toc490295068)

[Security 7](#_Toc490295069)

[The Microchip RN4870 I.C. 8](#_Toc490295070)

[Programming pins 14](#_Toc490295071)

[Square Wave Generator 19](#_Toc490295072)

[Building your own RN4870 Boards 20](#_Toc490295073)

[Project - Streaming Data 22](#_Toc490295074)

[Multiplexed Bi-Directional Mixed Data Streaming 24](#_Toc490295075)

[Custom Services & Characteristics 25](#_Toc490295076)

[Project - Remote Monitoring 27](#_Toc490295077)

[Useful Technical Information 32](#_Toc490295078)

[My Background in IoT 36](#_Toc490295079)

[What am I bad at? 36](#_Toc490295080)

[My IoT Company 36](#_Toc490295081)

[Other Publications 37](#_Toc490295082)

# Introduction

This paper will explain how to design, test, and build an Internet of Things (IoT) sensor, using a Microchip RN4870. It includes circuit diagrams, pin-outs, explanations of the technologies, programming scripts and configurations. I also include the same for some external tools I made, in the form of external real-time change monitoring, clocking and logic circuits.

# Maximising RoI

Maximum benefits will be realized by using IoT technology in situations where something is; old, distant or difficult to access. Older systems were not designed or built with sensors or Bluetooth in mind. Some were designed to be tamper proof. Others will not have any easy to access to inputs/outputs, at least not of the type an IoT sensor could connect with. But these are probably the systems that will provide the most benefits in terms of efficiencies, and RoI.

# Supporting Videos

I included four short videos, which are screen recordings of technical demonstrations. They are aimed at helping to explain some technology, or to demonstrate it in use. Their links are included in “**red**” in the relevant parts of this paper.

# The 1980’s and 1990’s

|  |  |
| --- | --- |
| I began working on this type of technology in the mid 1980’s. At the end of this paper you can review a list with url links, to some of my other recent technical publications, which include a range of associated compute technologies. |  |

# What’s changed in 30 years

The internet of things is largely built on the same technology. The differences are;

* Things are smaller, and packaged, in a modularised way. Do we understand what’s inside?
* People want everything quicker. That includes customers, and business executives. There is pressure on developers and project managers to deliver more functionality, quicker.
* Hackers are prolific. In some communities they are public heroes. They can make a lot of money. I am ex-military, secure communications. I understand security.
* GDPR (etc). When does a logic 1, received at a remote sensor become private or personal information? I don’t cover this off in this paper. But I understand the issues.
* I have made my own real-time vulnerability tool. It can alert to vulnerabilities in IoT sensors, both during development, and in production.

If you find any mistakes, or you have any advice, or a better explanation for anything, please email me; mike.mckean@msm-systems.com Thanks, Mike.

# Development Issues

Let me begin by stating that I like the Microchip range of products, and the RN4870. But, like any technology there can be issues. So, please do not consider the following list of issues, as me being negative with regard to Microchip. In fact, it is the opposite. I am very supportive of them. If you have a technical role, you will know that we all benefit if we honestly share experiences, and that technologies are never perfect.

**Variables not working?**

I found a way around this. I tried using the $VAR1 commands as per the User Manual Page 50, para 3.1.4. I tried several different options but I was unsuccessful. I delivered the same result, but used different commands.

**GPIO pins don’t stay programmed?**

This is relatively common. The Microchip RN4870 has a default setting on pins of Logic 1. If you program a GPIO pin to be a logic 0, and then read it, the pin reverts to a 1? i.e. if you are setting up a sensor, during a development project, you cannot simply program GPIO pins, to a set voltage, and then expect them to remain at that voltage. If you are in the early stages of development, use a breadboard, and tie pins high or low.

**Digital Voltmeter**

With specific regard to the previous point. Consider using a digital voltmeter. Then you can “see” voltage levels on pins, accurately. When you use a digital voltmeter, you don’t read the pin, so you don’t change it. That could be an alternative way of performing a test?

**GATT Characteristics change?**

As you change from “not connected” to “connected”, the available characteristics change. So, beware of this when you are checking to see if what you have done works, or not. E.g. you may think that the characteristic you have programmed is not working, but that’s because it’s only active when in the connected state.

**GATT Characteristics are “there” but don’t work?**

This refers to “Notification” and “Indication” on “Characteristics”. You have to enable them. You have to do that from the client, after connecting, and you need to use the “CI” command during the connection. You can do that via a script, after connecting.

Note. You don’t enable the characteristic handle itself. You enable the “characteristic enable” handle. E.g. if the “read handle” is 1234, then you will most likely see that right below it, is the “read enable handle”, and it is numbered 1235.

**Not all commands work, all of the time?**

Some commands only work, when you are in the right mode. Or connected. Or they are for scripting only. If you try entering some commands and you receive an error prompt, just check what mode you are in, and find out if those commands should be enabled.

**Cannot connect in command mode !**

* **Low Power Mode**

Some changes will disable your return to command mode? For instance, if you enable low power mode, that changes the clock from 16MBits to 32Khz, which in turn disables the UART connection. So, if you want to enable low power mode, you need to use Pin 26 (P3\_3) to switch the high speed clock back on, and hence the UART interface. I include a diagram and explanation of how this works, see the note at the bottom of page 20.

* **Command Prompt Timing**

Similar to above, except that there needs to be a 100ms delay between the three x $$$ inputs. If you don’t insert the delays, then the command is not recognized. I have not experienced this so far.

**Common earth**

When measuring voltage levels use a common earth. Otherwise you are likely to see readings that don’t make sense.

**External components**

A circuit that is using “logic 1’s and 0’s” should work, but because you add components that change voltage levels, or frequency characteristics, then voltage levels may change.

**Fan Out**

Fan-out will be an issue when trying to use low power I.C’s to drive external, regular components or logic I.C’s. To avoid this, use a buffer type circuit. I used hex inverters for driving more complex external logic circuits.

Note. For simple, individual signals, I found using a 300R (ohm) resistor enabled a 1.6V LED to operate, directly from a GPIO pin connector.

**Ground Plane**

If you build your own boards, make sure to design your PCB’s, taking into account Ground Plane factors. Remember without a correctly placed/sized ground plane, your antenna characteristics are severely impaired.

**Soldering**

Soldering, be careful. Dry Joint (soldering problem), while the wire may be physically connected, there will be a poor electrical connection.

**Component selection**

Some components have different characteristics at different frequencies. The most obvious being capacitors. They are almost short circuit at high frequencies, and have a much higher reactance at low frequencies.

**Upgradability**

Design it with a connector? Design it so that someone else can come along in XX years (or days) time, and add an external module. None of us know what’s coming. If you include some form of daughter board connector, at least you are trying to plan, for the future. Are you considering doing upgrades over the air? What about security?

**Documentation**

A really important part of the project. You need to document everything. Think about version control. Who is authorised to make changes.

**Maintenance**

You need to consider this. Are the components you are using likely to be around in XX years? Are they about to become End of Life? How easy is it to replace the board? Does the system use a battery? How easy is it to replace? Weather proofing? If it’s too easy to open the device, then the hacker, or vandal will open it. It’s a tough circle to square? Are you going to offer to support this ongoing? Then how do you control day to day issues?

**The “Two Key Process”. The catastrophe, waiting to happen**

Security, and mistakes. People think they have a “two-key” system, which ensures the safety of their IoT environment. I ask for them to explain what that is. Then they say, they always make sure a second control signal is input, from another source, before we operate the sluice gate (or other control). I ask, how is that done?

Then they explain that they use an AND Gate, and both inputs have to be a logic 1, before the remote gate/switch etc is operated.

So, the sluice gate control, to flood the valley is 100% dependant on a 20 pence AND gate? Did you know that in most Logic I.C’s, if a pin becomes open circuit, the pin “floats high”. What does that mean? It means that if someone pulled a cable, or stole it, or a connection went open circuit, that an AND Gate, and almost every logic I.C out there, sees that as a Logic 1, input !

So, careful about what you design, or you could flood a valley, just because someone pulled a cable out!

Notes

* The RN4870 pins and all the I.C’s I am using all float high with no external input. Check your vendors documentation for the default voltage/logic setting.
* I saw something similar with a storage cluster. The only thing that had not been tested was unplugging a particular cable? i.e. you need to design things, so they are constructed to always be in a failure condition. You don’t design them so everything works, until someone does something you had not thought of?

# Reading the Manual

I am a firm believer in reading the manual. I have seen so many people “giving up”, when the answer is there, in the manual. See; http://www.microchip.com/wwwproducts/en/RN4870

People who write manuals are not perfect, and mistakes/typos happen. If the manual says one thing, but you see another, then your circuit may be right, but the manual could be wrong. This is a bigger issue with newer, recently released product, where it’s almost certain to have typos or mistakes. In such a situation, try reviewing any previous I.C’s manuals. You may get lucky and find the problem yourself. The manual also explains how to avoid some potentially, annoying pitfalls, so it is well worth reading.

Notes

* With regard to the Microchip documentation, I found it very good. However, don’t rely just on the RN4870 User Manual. Here’s one example; you will find references in there to pins that don’t exist? That’s because the technical author, wrote another manual, for a similar product. The other manual is for the RN4870 PICTail. It uses the exact same RN4870 I.C, but is a development board, and has more pins. That’s where the additional pin references come from. So, with regard to the RN4870, you should read the most up to date RN4870 user manual, alongside the RN4870 PICTail user manual.
* I found several people, complaining about losing access to the RN4870. The manual explains several reasons why this will happen. If you read the manual, it explains why this will happen, and how to regain access. Also see my notes at the bottom of page 21, in this paper, which explain how to avoid this issue.

# Development Tools

During the design, and build of any system, I need to see what is occurring. I need to know what state the various pins are in. That will help me to know if the programming I am implementing is correct or not. I also used this technology for a security toolset that I have built. It helps me to secure devices, when others add external circuitry, and/or new updates, or applications. See following paragraphs for more detail.

**Real Time Change Monitoring**

**Video** https://www.msm-systems.com/internet-of-things/

I have made a tool (see above url) which allows me to monitor the state of the device, as I change each individual line of code. In order, to “see” these occurrences, in real time, I set up a series of logic controlled LED’s. They connect to various pins on both RN4870’s and allow me to see, any changes to important functionality, as I change settings on the RN4870’s.

In this case, my monitoring tool monitors two status pins. I use Boolean algebra to return results which allow me to see which of four states the RN4870 is in. These four states are important to allowing me to “see” what is happening with the RN4870, as I build the circuitry and program it.

**Real Time Change Monitoring Schematic**



**Real Time Change Monitoring Circuit Diagram**



Rea**l-Time Change Monitoring Status Pins**

|  |  |  |
| --- | --- | --- |
| **Status 1** | **Status 2** | **State** |
| High | High | Power On |
| High | Low | Standby state |
| Low | Low | Connection established |
| Low | High | Data session open (Transparent UART) |

# Security

**IDE’s**

The more abstraction layers you put in between you and the I.C. then the less you know. The more layers of someone else’s; code, tools, firmware, patches you use, then the less you know, and the more vulnerable you are. If you want something quick, then use whatever IDE environment you are most comfortable with. If you want something secure, then work at logic I.C level, and/or machine code level. The balance is somewhere in between.

I suggest that during a large project, you insert regular security checkpoints (no pun intended). Don’t wait until you have a polished solution, and then arrange an ethical hack.

**Ethical Hack**

I used to supply a managed service to one of the largest banks in the world. I had two engineers. One of my contacts explained he was about to spend $30,000 on an ethical hack. He let one of my engineers take a look first. He hacked it in a few minutes. I think he found a weakness through the help options. The point is, don’t wait until a few weeks before it’s due to go into production, before doing your only ethical hack. It’s way too late then, and way too much work, to un-engineer everything. Do, your security checks at regular intervals. Then if you have a problem, you can flag it up early, and who is going to blame you for finding a security issue early? How much money and embarrassment did you just save the business?

**Security Monitoring Tool**

As you have guessed (from above) I have made my own tools which can evaluate the security of I.C’s, including FPGA’s, and ASIC’s. I use this type of technology as part of my “own development tools”. My security solution includes the same type of technology as used in “Real Time Change Monitoring”. I can build this for any type of circuit.

**Design Stage Vulnerabilities**

Once I understand the I.C. architecture, and what happens in various states, I can identify vulnerabilities. Once I have done that, I find out what happens in terms of “pin voltage changes”. I can then design a simple logic circuit which will trigger an output for any given set of conditions.

**Post Production Vulnerabilities**

This can include changes made through adding new application functionality, changes in the connectivity, networking, hardware, etc. Any changes at all which impact the security of the device will be identified.

**Low Latency Vulnerabilities**

Any short duration vulnerabilities, can be latched, and an alert sent, or remain until acknowledged, or saved to an audit report. There could be options for the security vulnerability to remain identified, stored, an alert sent, and for the security monitoring to continue in real time.

**Unlisted Commands**

Remember that this technology is a matrix of addressable pins. Once you address them you can write/read 1’s or 0’s to/from cells. Here’s what could happen, and I know because I have done it;

* If you don’t see a command that you want to use, then work out what the addresses of the pins would be.
* Calculate the hex value you want to read/write. Then construct the command, and try it?
* I found one that worked.

**Bluetooth Radio**

Installing encryption just before the antenna, means that everything before it, is vulnerable? Radio systems are constructed from layers of frequencies. E.g. If the public addressable frequency of Bluetooth is approximately 2.4GHz, then that is not the only frequency being used in a Bluetooth radio system.

Radio systems start with lower frequencies, and modulate information into them. Then depending on the radio system, they then multiple the frequency upwards.

What you would need to do, is identify the lower “intermediate frequencies” called “IF’s”. You should then massively attenuate and screen any such external frequencies.

I have not done any research on the Bluetooth radio stack, so I don’t know what issues there could be. (Secure radio systems was my trade in the military.)

# The Microchip RN4870 I.C.

**Short explanation of the RN4870.**

The RN4870 is a peripheral interface controller. It has a small CPU, with a limited set of commands, and a small amount of NVM, that can store programming.

It has four types of connectors to the “outside world”;

* Wire/Pin Connected;
  + Analogue. These are input only. i.e. they are connected to external devices, and they receive an analogue input. The voltage range is measured in millivolts.
  + Digital. These can be input or output. Generally, Logic 1 = 3.3V, and Logic 0 = 0v.
  + UART. This provides local access via a serial connector. I used a tera term, terminal emulator for the UART/USB connection. Is simple, robust and reliable. (and free).
* Radio;
  + Bluetooth radio.

**What pins are on the RN4870?**

With regard to the Microchip documentation, I found it very good. However, don’t rely just on the RN4870 User Manual. You will find references in there to pins that don’t exist? That’s because the technical author wrote another manual. That was for a development board, which also used the RN4870. It uses the exact same RN4870 I.C, but because it is a development board, it has additional pins for testing etc. That’s where the additional pin references come from. So, you should read the most up to date RN4870 user manual, alongside the RN4870 PICTail user manual.

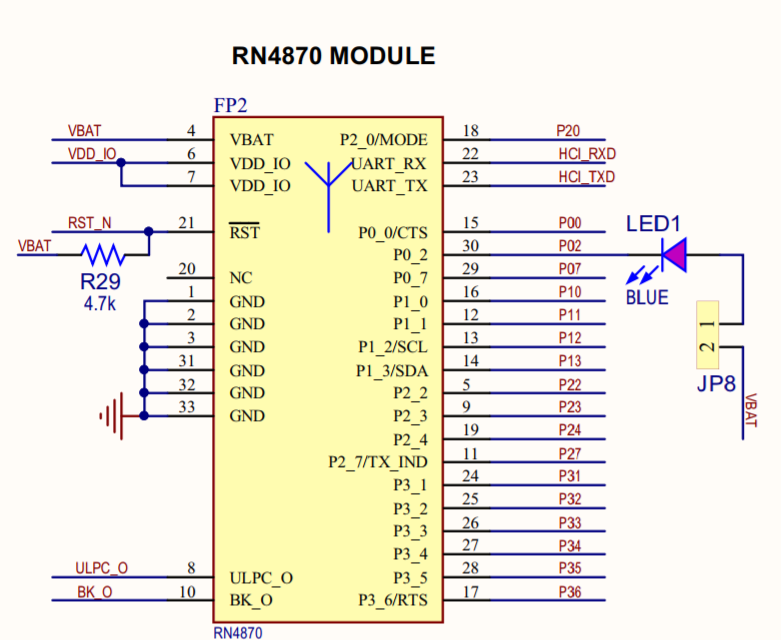
* RN4870 manual http://www.microchip.com/wwwproducts/en/RN4870
* RN4870 PICTail manual http://ww1.microchip.com/downloads/en/DeviceDoc/50002547A.pdf

**Pin Numbering Scheme**

The Microchip documents use (typically) three identifiers for each pin, and as above the pin numbering sometimes, is misleading. Please see diagram from the RN4870 PICTail manual. Here are some examples;

|  |  |  |  |
| --- | --- | --- | --- |
| I.C. Pin | Label  Inside I.C.  Schematic | Label on  Circuit  From I.C. Pin | Label on  External  Board  Connector |
| 5 | P2\_2 | P22 | P22 |
| 19 | P2\_4 | P24 | P24 |

See diagram below from the RN4870 PICTail user manual.



**RN4870 Pin Out**

The table below, lists out the pins, as per the RN4870 official specification. The difference in my version is that I removed a column which listed out the RN4871 product, which is a variation on the RN4870. If you find different references, e.g. from the PICTail manual, always cross-reference them back here, or to the official RN4870 user manual. I also included some additional remarks, in some areas.

**RN4870 Pin Out Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RN4870** | **Name** | **Type** | **Description** | **Comments** |
| 1 | GND | Power | Ground reference |  |
| 2 | GND | Power | Ground reference |  |
| 3 | GND | Power | Ground reference |  |
| 4 | VBAT | Power | Positive supply input. Range: 1.9V~3.6V |  |
| 5 | P2\_2 | D I/O | GPIO, PWM1,  Default: Input; pulled-high |  |
| 6 | VDD\_IO | Power | VDD; power input  Same input pin as VBAT  Can be connected to the VBAT pin |  |
| 7 | VDD\_IO | Power | VDD; power input  Same input pin as VBAT  Can be connected to the VBAT pin |  |
| 8 | ULPC\_O | Power | 1.2V ULPC LDO output Used for diagnostic purposes  *Do not connect* to any pin or device  For measurement, connect a bypass 1 µF capacitor to ground |  |
| 9 | P2\_3 | D I/O | GPIO, PWM2,  Default: Input; pulled-high |  |
| 10 | BK\_O | Power | 1.55V Buck power supply output for diagnostic purpose  *Do not connect* |  |
| 11 | P2\_7 | D I/O  A I/p | GPIO; default: Input; pulled-high AD14  Configured as the TX\_IND pin by default |  |
| 12 | P1\_1 | D I/O  A I/p | GPIO; default: Input; pulled-high AD9  Configured as the BLEDK\_STATUS1\_IND pin by default |  |
| 13 | P1\_2 | D I/O  A I/p | GPIO; default: Input; pulled-high AD10  Seems to be available? |  |
| 14 | P1\_3 | D I/O | GPIO; default: Input; pulled-high AD11 |  |
| 15 | P0\_0 | D I/O | GPIO; default: Input; pulled-high AD0  Configured as the UART\_CTS pin by default |  |
| 16 | P1\_0 | D I/O | GPIO; default: Input; pulled-high AD8  Configured as the BLEDK\_STATUS2\_IND pin by default |  |
| 17 | P3\_6 | D I/O | GPIO; default: Input; pulled-high PWM0  Configured as the UART\_RTS pin by default |  |
| 18 | P2\_0 | D I/p | System configuration input; 1: Application mode  0: Test mode, for Flash update and EEPROM settings  Default: Input; pulled-high |  |
| 19 | P2\_4 | D I/O | GPIO; default: Input; pulled-high |  |
| 20 | NC | — | No Connection |  |
| 21 | RST\_N | D I/p | Module Reset; active-low; Internally pulled-high |  |
| 22 | UART\_RX | D I/p | UART Data input |  |
| 23 | UART\_TX | D O/p | UART Data output |  |
| 24 | P3\_1 | D I/O | GPIO; default: Input; pulled-high Configured as RSSI\_IND pin by default.  Use this indication pin to indicate the quality of the link based on the RSSI level. If the RSSI level is lower than the specified threshold value, then the RSSI indication pin goes low. Set the threshold for the RSSI link quality in EEPROM. |  |
| 25 | P3\_2 | D I/p | GPIO; default: Input; pulled-high  Configured as the LINK\_DROP pin by default.  Use this pin to force the module to drop the current BLE link with a peer device. Pulling the Link Drop pin low forces the connection to close. The pin needs to be pulled low for at least 10 ms. |  |
| 26 | P3\_3 | D I/p | Use this pin to enable communication with the UART when the module is in Low- Power mode. When *not* in Low-Power mode, the module runs on a 16 MHz clock. If Low-Power mode is enabled on the module by using command SO,1, the module runs on a 32 kHz clock thus reducing power consumption. However, in Low-Power mode, the host MCU *cannot* communicate with the module via the UART since the UART is *not* operational. If the user intends to provide data or commands via UART in the Low-Power mode, then the UART RX INDICATION pin must be pulled low and the user needs to wait for at least five milliseconds before sending the data. Pulling the UART RX INDICATION pin low allows the module to operate the 16 MHz clock and to enable UART. |  |
| 27 | P3\_4 | D I/p | GPIO; default: Input; pulled-high  Configured as the PAIRING\_KEY pin by default |  |
| 28 | P3\_5 | D I/O  A I/p | GPIO; default: Input; pulled-high  LED1; provides indication whether the module is ON/OFF |  |
| 29 | P0\_7 | D I/O | GPIO; default: Input; pulled-high  Configured to the LOW\_BATTERY\_INDICATOR pin by default. Pin output goes low when the VDD is below a specified level. To set the threshold level, change the EEPROM settings. |  |
| 30 | P0\_2 | D I/O | AD2  LED0: Provides indication whether the module is in ON/OFF mode |  |
| 31 | GND | Power | Ground Reference |  |
| 32 | GND | Power | Ground Reference |  |
| 33 | GND | Power | Ground Reference |  |

**What pins are available to use?**

I need to identify which Pin(s) I can use. In the table below, pins identified as being already in use are greyed out. I have decided to use pins 5 (P2\_2) and 19 (2\_4).

Note. Some pins are noted as being solely, for digital input/output. Others have a dual, configurable role, as also being available for use as analogue inputs. I decided to opt for the straight forward, dedicated digital use pins. If I was to exploit these I.C’s more, I will invest time in testing out the dual role, analogue/digital pins.

**Table of Available Pins**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RN4870** | **Name** | **Type** | **Description** | **Comments** |
| 1 | GND | Power | Ground reference |  |
| 2 | GND | Power | Ground reference |  |
| 3 | GND | Power | Ground reference |  |
| 4 | VBAT | Power | Positive supply input. Range: 1.9V~3.6V |  |
| 5 | P2\_2 | D I/O | GPIO, PWM1,  Default: Input; pulled-high |  |
| 6 | VDD\_IO | Power | VDD; power input  Same input pin as VBAT  Can be connected to the VBAT pin |  |
| 7 | VDD\_IO | Power | VDD; power input  Same input pin as VBAT  Can be connected to the VBAT pin |  |
| 8 | ULPC\_O | Power | 1.2V ULPC LDO output Used for diagnostic purposes  *Do not connect* to any pin or device  For measurement, connect a bypass 1 µF capacitor to ground |  |
| 9 | P2\_3 | D I/O | GPIO, PWM2,  Default: Input; pulled-high |  |
| 10 | BK\_O | Power | 1.55V Buck power supply output for diagnostic purpose  *Do not connect* |  |
| 11 | P2\_7 | D I/O  A I/p | GPIO; default: Input; pulled-high AD14  Configured as the TX\_IND pin by default |  |
| 12 | P1\_1 | D I/O  A I/p | GPIO; default: Input; pulled-high AD9  Configured as the BLEDK\_STATUS1\_IND pin by default |  |
| 13 | P1\_2 | D I/O  A I/p | GPIO; default: Input; pulled-high AD10 |  |
| 14 | P1\_3 | D I/O | GPIO; default: Input; pulled-high AD11 |  |
| 15 | P0\_0 | D I/O | GPIO; default: Input; pulled-high AD0  Configured as the UART\_CTS pin by default |  |
| 16 | P1\_0 | D I/O | GPIO; default: Input; pulled-high AD8  Configured as the BLEDK\_STATUS2\_IND pin by default |  |
| 17 | P3\_6 | D I/O | GPIO; default: Input; pulled-high PWM0  Configured as the UART\_RTS pin by default |  |
| 18 | P2\_0 | D I/p | System configuration input; 1: Application mode  0: Test mode, for Flash update and EEPROM settings  Default: Input; pulled-high |  |
| 19 | P2\_4 | D I/O | GPIO; default: Input; pulled-high |  |
| 20 | NC | — | No Connection |  |
| 21 | RST\_N | D I/p | Module Reset; active-low; Internally pulled-high |  |
| 22 | UART\_RX | D I/p | UART Data input |  |
|  |  |  |  |  |
| 23 | UART\_TX | D O/p | UART Data output |  |
| 24 | P3\_1 | D I/O | GPIO; default: Input; pulled-high Configured as RSSI\_IND pin by default.  Use this indication pin to indicate the quality of the link based on the RSSI level. If the RSSI level is lower than the specified threshold value, then the RSSI indication pin goes low. Set the threshold for the RSSI link quality in EEPROM. |  |
| 25 | P3\_2 | D I/p | GPIO; default: Input; pulled-high  Configured as the LINK\_DROP pin by default.  Use this pin to force the module to drop the current BLE link with a peer device. Pulling the Link Drop pin low forces the connection to close. The pin needs to be pulled low for at least 10 ms. |  |
| 26 | P3\_3 | D I/p | Use this pin to enable communication with the UART when the module is in Low- Power mode. When *not* in Low-Power mode, the module runs on a 16 MHz clock. If Low-Power mode is enabled on the module by using command SO,1, the module runs on a 32 kHz clock thus reducing power consumption. However, in Low-Power mode, the host MCU *cannot* communicate with the module via the UART since the UART is *not* operational. If the user intends to provide data or commands via UART in the Low-Power mode, then the UART RX INDICATION pin must be pulled low and the user needs to wait for at least five milliseconds before sending the data. Pulling the UART RX INDICATION pin low allows the module to operate the 16 MHz clock and to enable UART. |  |
| 27 | P3\_4 | D I/p | GPIO; default: Input; pulled-high  Configured as the PAIRING\_KEY pin by default |  |
| 28 | P3\_5 | D I/O  A I/p | GPIO; default: Input; pulled-high  LED1; provides indication whether the module is ON/OFF |  |
| 29 | P0\_7 | D I/O | GPIO; default: Input; pulled-high  Configured to the LOW\_BATTERY\_INDICATOR pin by default. Pin output goes low when the VDD is below a specified level. To set the threshold level, change the EEPROM settings. |  |
| 30 | P0\_2 | D I/O | AD2  LED0: Provides indication whether the module is in ON/OFF mode |  |
| 31 | GND | Power | Ground Reference |  |
| 32 | GND | Power | Ground Reference |  |
| 33 | GND | Power | Ground Reference |  |

# Programming pins

**Analogue pins.**

These are “input only” pins, and operate in the millivolt range.

Notes

* The documentation suggests the RN48670 has analogue I/O, see Page 29 of the User Manual, where it states; “*Read and write analog data*”. However, it shows an analogue to digital converter, but no digital to analogue converter. The documentation on pages 62 through to 64, list all the commands. While analogue read commands are shown, none are given for analogue output.
* Calibration of Analogue Inputs. I have not yet investigated this properly. But a quick review of the circuitry, using a standard digital voltmeter, showed that the supplied voltage in, was slightly less than that output on various GPIO pins. Therefore, the RN4870 has some form of voltage pumping, and regulation. When I checked the analogue read out, using the command; @,4, the Hex equated to 3.4. The level I read was circa 3.2V. The analogue voltage in, is measured in millivolts. I would suggest that a few inputs are made, at e.g. 0V through to VBAT. We should check for linearity. That’s important as you could be using this to measure large, external voltages.

**Analogue Channels**

|  |  |  |
| --- | --- | --- |
| **Analog Port Parameter** | **RN4870 Analog Port** | **Default Function** |
| 0 | P1\_0 | Status 2 |
| 1 | P1\_1 | Status 1 |
| 2 | P1\_2 | None |
| 3 | P1\_3 | None |
| 4 | Battery sensor | Battery Sensor |
| 5 | Temperature sensor | Temperature sensor |

The analogue channels are simple.

P1\_1 would be identified as 1 and P1\_0 would be identified as 0

**Analogue Read Command …… @,<0 to 5>**

e.g. @,5 would read in a four digit hex number which represented the analogue temperature of the RN4870 I.C. So it would return something such as; 1234

**Analogue Write Command**

There is not a command listed to write to an analogue channel/port. If you refer to the summary of commands, in the RN4870 User Manual, starting on page 62 through to page 64, you will see that none is listed to write to an analogue port. See: http://www.microchip.com/wwwproducts/en/RN4870 Further, if you review the functions of the analogue ports you can also see they are all inputs. This is also confirmed in the user manual, where all these pins are shown as Analogue Inputs. This means you cannot take a digital hex value, and output that on an analogue port as a discrete voltage. If you had such a need, you would use an external Digital to Analogue converter, I.C.

**Digital pins**

I am going into a bit of detail to explain how the individual pins are accessed. E.g. see P2\_2, highlighted in yellow.

**Digital Pin I/O Matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bitmap** | **RN4870 PICTail**  **Pins** | **RN4870**  **Pins** | **Function** |
| 01 | P2\_2 | 5 | PWM |
| 02 | P2\_4 | 19 | D I/O |
| 04 | P3\_5 | 28 | Digital I/O or Analogue input, power on/off LED |
| 08 | P1\_2 | 13 |  |
| 10 | P1\_3 | 14 |  |

**Bitmap representation of digital pins**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |  |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04 |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 08 |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 10 |

P2\_2 would be identified in hex as 01

P1\_3 would be identified in hex as 10

**Digital Write Command …… |O**

You can set the output level on a digital pin, or even multiple digital pins, using one command.

Command: |O,07,05 // Set digital I/O output on P2\_2, P2\_4 and P3\_5.

// Set P2\_2 and P3\_5 high and P2\_4 low.

07 refers to the bitmap of the digital I/O ports affected. If the hex value is 07, or 00000111 then the following pins are affected; 22 + 21 + 20 = 4 + 2 +1 = 7

**Table explanation of how logic is set on each output pin**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04 |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  | 1 | 0 | 1 | 05 |

The output value is hex 05, which is 00000101. So the following values are output on each pin;

P2\_2 = logic 1

P2\_4 = Logic 0

P3\_5 = Logic 1

**Digital Read Command …. |I**

Command: |I,06 // Read digital I/O P2\_4 and P3\_5. As an example, if return value is 04,

// then P2\_4 is low and P3\_5 is high

**Table explaining how the read command addresses the pins**

The 06 part of the command can only be made up of P2\_4 and P3\_5, because 06 can only be made up of 22 + 21 = 4 + 2 = 06. So those two pins will be accessed, and their data read.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04 |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  | 1 | 0 | 0 | 04 |

To explain the above differently, if both P2\_4 and P3\_5 contained logic 1’s, the output would be;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04 |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  | 1 |  | 1 | 05 |

**Configuring the function for Pin 19**

In this case Pin 19 (P24) was already available. If it was not, then I could simply have entered SW,04,00 That would have set Pin 19 (P24 , Pin Index 04) to Function 00, which is none (or no pre-set function). i.e. available for me to use.

Note. I can change most pins from a default allocation to “none”, so that I can then use that pin for something else. You usually need to be careful that anything you stop reporting on, is not needed elsewhere, or later.

**Table of Pin Default Functions and their Configurable Functions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Pin Index** | **Pin** | **Default Function** |  | **Function**  **Index** | **Function Description** |
| 00 | P07 | Low Battery Indication |  | 00 | None |
| 01 | P10 | Status 2 |  | 01 | Low Battery Indication |
| 02 | P11 | Status 1 |  | 02 | RSSI Indication |
| 03 | P22 | None |  | 03 | Link Drop |
| 04 | P24 | None |  | 04 | UART RX Indication |
| 05 | P31 | RSSI Indication |  | 05 | Pairing |
| 06 | P32 | Link Drop |  | 06 | RF Active Indication |
| 07 | P33 | UART Rx Indication |  | 07 | Status 1 |
| 08 | P34 | Pairing |  | 08 | Status 2 |
| 09 | P35 | None |  | 09 | Pin Trigger 1 |
| 0A | P12 | None |  | 0A | Pin Trigger 2 |
| 0B | P13 | None |  | 0B | Pin Trigger 3 |
|  |  |  |  | 0C | UART Mode Switc. Rising edge for UART Transparent mode; falling edge Command mode. |

**Read Pin 19; |I,02**

Let’s read pin 19 (P2\_4). P2\_4 is addressed using bitmap 00000010 or 02. See Table below.

So the command is; |I,02 // Read digital input at P2\_4. The response will be either 00 or 02

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  |  | 00 or 02 |  |  |

**Read Pin 5; |I,01**

Let’s read pin 19 (P2\_2). P2\_2 is addressed using bitmap 00000001, or 01. See Table below.

So the command is; |I,01 // Read digital input at P2\_1, The response will be either 00 or 01

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  |  |  | 00 or 01 |  |

**Write to Pin 19; |O,02,xx**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  |  | 00 or 02 |  |  |

|0,02,02 Set o/p on P2\_4 to 1

|0,02,00 Set o/p on P2\_4 to 0

**Write to Pin 5; |O,01,xx**

Set O/P for pin 5, P2\_2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pin | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | Hex |
| P2\_2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| P2\_4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |
| P3\_5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| P1\_2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| P1\_3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| Outputs |  |  |  |  |  |  |  | 00 or 01 |  |

|0,01,01 Set o/p on P2\_2 to 1

|0,01,00 Set o/p on P2\_2 to 0

**Initial testing of Pin 19 , P2\_4**

Remember that after a pin is read, unless it is electronically tied to a voltage level, it returns to its default level. The RN4870 like many I.C’s uses a default voltage level of logic 1. So once the pin has been read, it reverts back to logic 1. I wanted to make sure that I could address pin 19, and that my digital voltmeter would read the voltages I expected.

**Table detailing how I tested readings on pin 19**

|  |  |  |
| --- | --- | --- |
| Item | Action | Comments |
|  | Check logic value on Pin 19 on RN4870, number 326 by using |I,02 | Output is 02, i.e. logic 1. This is the default level. |
|  | Change logic level on 326, by using IO,02,00 | Check value using voltmeter. It has changed to a logic 0 |
|  | Read logic value on Pin 19 on RN4870, number 326 by using |I,02 | Output is 00  Note. As soon as the pin is read, the digital voltmeter shows that the pin has returned to its default level of 1. |
|  | Read logic value on Pin 19 on RN4870, number 326 by using |I,02 | Output is 02, i.e. logic 1  Because the pin has been read, it changes back to its default level of logic 1, or 02  Here’s what has happened in the above. The RN4870 has default states. It will set a condition on a pin, at the state you command. Once you read that state, the RN4870 changes the logic level back to its default state. |

# Square Wave Generator

These RN4870’s are new I.C’s to me, so I want to step through my programming slowly, and see what is happening. I could step through the program manually, one machine cycle at a time. I do that sometimes. In this case, I also built a small clock circuit that I can change from 3 seconds a pulse to 30K pulses per second.

I built a small square wave generator. It outputs a logic 1, then 0, approximately every three seconds. That allows me to input a Logic 1, or 0, at a GPIO pin, and see if what I have done works. Remember, that you may set a GPIO port at logic 1, then read it, and you think that what you just did worked. But, (oops) the default reading of a GPIO pin is logic 1. So, have you actually just read in the default voltage level? i.e. what you just think you did, never in fact changed the port to a logic 1. It was already at logic 1, as that’s its default level?

So, to avoid this, I connect a voltmeter, or oscilloscope to the pin I am testing. I then use a square wave input, at such a slow frequency, so I can make my changes, then check that the pin changes as expected.

**Circuit Diagram for Square Waver Generator**



# 

# Building your own RN4870 Boards

If you decide to build your own PCB, or Veroboard, here’s a circuit diagram you could use. For clarity, I left out the GPIO pins. But you would bring them all out to an external edge connector.



You will note the coloured UART connectors. Those are for a UART/USB cable from http://www.ftdichip.com

You will also need a terminal emulator. I used Tera Term. It’s open source, well known, and I have found it robust, and easy to use.

Notes

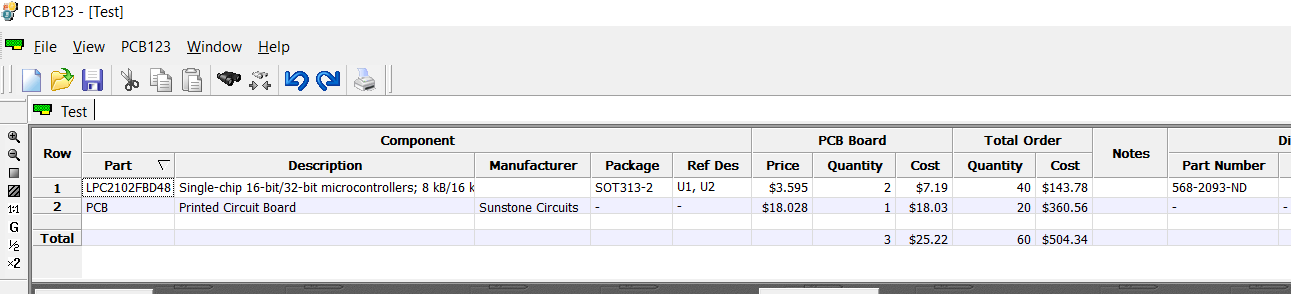
* The reason for the resistors on the left (220K, 100K, and 10K) was that I needed a voltage divider as my power source is 5V. I needed to reduce it to 3.3V.
* The 47µF is a reservoir capacitor, and the smaller 0.047 µF is to remove any spikes from the power input.
* I included a manual reset switch, because in some operating modes the RN4870 UART command access is disabled? E.g. in low power mode the RN4870 switches off the high speed 26MHz clock, and runs on the low power, slower 32Khz clock. When that happens the UART command mode is disabled. So, you cannot access the RN4870 via a computer, or tablet etc. You then need a way to manually reset that. You could also do that with any type of signal, that was converted to a logic 0.
* The above low power option is how you could run an RN4870, for very long periods on a coin type battery cell. Then, if something triggered an event, you could use many different types of local logic I.C’s, or circuit, to wake up the RN4870, and have it autonomously run several scripts to take action. Obviously at that point the UART connectivity is enabled. You could then stream data/video, or remotely access it.

**Who/Where to get your PCB made?**

There are too many options to mention. Some include quality, and electronic verification of a design. Others include components at no charge (e.g. resistors, capacitors). There are some firms who cater for the small-scale manufacture. One of the important things to find out about is the “Gerber” files. If the PCB is manufactured online, or someone else owns the software, make sure you have access to the Gerber files. Those are the industry standard files used to hold the information about your PCB.

Things that affect price are; quantity, lead time, and the number of layers on the board.

I’ll include a PCB package that I have used. It is an application called PCB123. They provide the software free, and as you build/design the board their application documents the Build of Materials (BOM) and pricing. See below;



# Project - Streaming Data

The purpose of this project is to demonstrate the transparent UART data function of the RN4870. Although RN4870 is set up as a client, and RN4870 326 as a server, my video will show that data can be transmitted, in either direction, in real time.

**Video** https://vimeo.com/229124625

**Schematic**



This demonstration used a USB/UART cable to my laptop. You could connect the serial UART data connection and integrate that with your application.

**Programming**

I did the following, and I have included comments about what I did, and why.

**Device Supported Features**

I have two RN4870’s. One is number 324, and the other is number 326. I have them set up, so that on connection, the data stream feature is open, and the link secured.

See table on next page.

**Bitmap Table of Features to be Supported**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** |
| **Enable Flow Control** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No prompt** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Fast Mode** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No beacon scan** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No connect scan** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No duplicate scan result filter** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Passive scan** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **UART Transparent, no ack** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Reboot after disconnect** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Run script at power on** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Streaming** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Configuration | 0 | | | | 1 | | | | 6 | | | | 0 | | | |

I decided on these features for the following reasons;

* UART Transparent, no acknowledgement. I just want the receiving RN4870 to receive data without any handshaking.
* Run Script at power On. I always set that up. In this case, I am running a script at “connection” which automates security and bonding, and client status, from 324 to 326. I explain that script further down.
* Streaming. I need the streaming feature enabled for this project.

The command to enable those three features is; SR,0160

Note. For a separate project, I implemented hardware flow control, but for this project, for showing a concept that is not needed. However, if you need hardware control, do the following;

* In your terminal emulator, enable flow control
* The command for the bitmap of features, would change from SR,0160, to SR,8160.
* If you are using the PICTail development board, you then need to insert two jumpers, on pin connector J3. See the PICTail manual page 16, para 2.2.1 sub para 15, which explains that the RTS and CTS connectors need to be closed using “jumper connectors”.

**Configuring the RN4870’s**

I then need to configure the GAP Services that the RN4870 will support. I decided to configure them both for; Enable GATT services, Device Information Service and Transparent UART.

**Table of Commands to program the RN4870’s**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **324** | **326** |
|  | To change the default name | S-,MSM324 | S-,MSM326 |
|  | Enable GATT services, Device Information Service and Transparent UART | SS,C0 | SS,C0 |
|  | Runs script(s) after power on and starts streaming service, and UART without acknowledgement | SR,0160 | SR,0160 |
|  | Configure 2\_2 as GPIO, for future use | SW,03,00 | SW,03,00 |
|  | Configure 2\_4 as GPIO, for future use | SW,04,00 | SW,04,00 |

**Script in 324**

The RN4870 includes a small CPU, with limited scripting capability. Having said it’s small, it’s still about 20 times faster than the CPU’s, I started using, many years ago. It does give me some limited, local, autonomous compute. I can also use local logic circuits to significantly enhance what it can do. I used the following script to automate a secure connection, and enable transparent UART service.

**Table of Script for RN4870, 324**

|  |  |  |
| --- | --- | --- |
| Item | Command | Comments |
|  | @CONN | Start this script when RN4870, 324, connects with another RN4870. |
|  | SM,1,0002 | Set up timer 1, so it runs for 2 time periods. Each time period is 0.624 seconds. This timer will run for approximately 1.2 seconds. |
|  | @TMR1 | At the end of the 1.2 seconds execute instruction CI.  Notes.  1.I found that I needed to wait a short time, before the two RN4870’s had completed handshaking before I invoked instruction CI.  2. The handshaking that is taking place includes the RN4870’s verifying the bonding details for each other, and then they securely bond the connection. That’s what requires a slight time delay. |
|  | CI | The “C” part of this instruction sets RN4870, 324 as the client. The “I” part enables transparent UART service. |

# Multiplexed Bi-Directional Mixed Data Streaming

**The Expectation**

This has a happy ending, but read on (please). Remember I was working on this technology 30 years ago. I expected that it would have moved on in that time. For this project, I invested quite a lot of effort, which included working with Microchip’s technical staff. I thought I could construct a command sequence that would enable real-time, bi-directional transmission/receipt of mixed data types, through the Bluetooth connection. After two interactive support sessions, Microchip advised me, that this is not a standard feature.

**The Solution**

I have done it. Here is a short description of my solution;

* Bi-directional
* Real-time
* The limitations are;
  + I use polling or interrupts to control the direction of data flow.
  + Buffering would be needed to prevent data loss.
  + Bandwidth limitations, of Bluetooth.

I can multiplex, different types of inputs/outputs, at either end. You could have a real-time mixture of video, audio, data etc, and the correct output/input would be automatically routed to the appropriate device, or application. Bandwidth and switching latency would restrict the quality of any such capability.

Please contact me if you want a demonstration of this; mike.mckean@msm-systems.com 07539 341 216

# Custom Services & Characteristics

In the next project (on page 27), I will need to use a Custom Service, along with associated Custom Characteristics. If you know what those are, then skip this section, and go straight to page 29. If you are a little unsure, the video below, and this section may help.

**Video** https://vimeo.com/229142382

Before, I explain what a custom service, and its associated characteristics are, let’s understand what a “normal” service/characteristic is?

If you go and buy a Bluetooth heart rate monitoring device, it will adhere to an agreed set of specifications. They are published at; https://www.bluetooth.com/specifications/gatt That means that anyone who develops anything in that area, must adhere to the published specification.

As a manufacturer, I would know how to build, and program my sensors. I would integrate that to an application. This makes support, and development simpler.

Here’s what that really means in some detail. I will explain this using a custom (or private) service, along with its associated characteristics.

Let’s consider a non-standard solution. Let’s say you want to design your own Bluetooth application, for your own use, or company use. Then you can write your own deliverables. You decide that you want to design a sensor which would monitor the temperature of a beehive, and then send you an alert if it fluctuated outside certain parameters.

You would write a custom specification. It will contain parameters. In Bluetooth terminology, those parameters are called characteristics. If the maximum allowed temperature was 400 C and the minimum was 00 C, then they could be two characteristics. You might also decide to send your alert using a digital logic level of 0 (remembering about default levels being a 1). That alert will be sent on GPIO pin 4. So, you now have four characteristics;

* Minimum temperature 00 C.
* Maximum temperature 400 C.
* Alert is a logic 0.
* Alert raised on GPIO pin 4. Note. That would limit the types of sensors you could use. But let’s include it.

But you cannot name this “Billy’s Beehive Specification for Bluetooth”. You need a name that is unique.

Here’s a summary of what you do;

* You visit this site; http://www.guidgenerator.com/online-guid-generator.aspx
  + You create five 128 bit GUID’s.
  + One is to be the custom service.
  + The other four will be the custom characteristics.
* For example here’s what you would have;
  + Billy’s beehive service, is now known as; 6AE5D097F9F942B7BE3B54DFC5DD5099
  + Minimum temp characteristic as; 3A6DD8D1BBF747D49D1747B23B777C0A
  + Maximum temp characteristic as; A8AF896A926D487598D578FB1FF8B435
  + Alert pin 4 as; F7834CD5312F45F9854A57D837137068
  + Alert logic = 0, is; 6AE5D097F9F942B7BE3B54DFC5DD5099
* As you can see those are long identifiers. However, once you create them in your programming, then the RN4870 will automatically shorten them to 16 bit ID’s. These are known as “handles”. So that could be;
  + Billy’s beehive service, is now known as; 1234
  + Minimum temp characteristic as; 22A4
  + Maximum temp characteristic as; AF33
  + Alert pin 4 as; 4A34
  + Alert = 0V; as 3E12
* Remember those are your internal handles.
* In terms of public access, anyone thinking of trying to connect to your custom service, would “see” a 128 bit GUID of “6AE5D097F9F942B7BE3B54DFC5DD5099” that they did not recognize, nor would they know what it did. A hacker may still want to connect. That’s a separate issue.

# Project - Remote Monitoring

If you are unsure of Custom Services and their associated Characteristics please go back to page 25.

**Video** https://vimeo.com/229129675

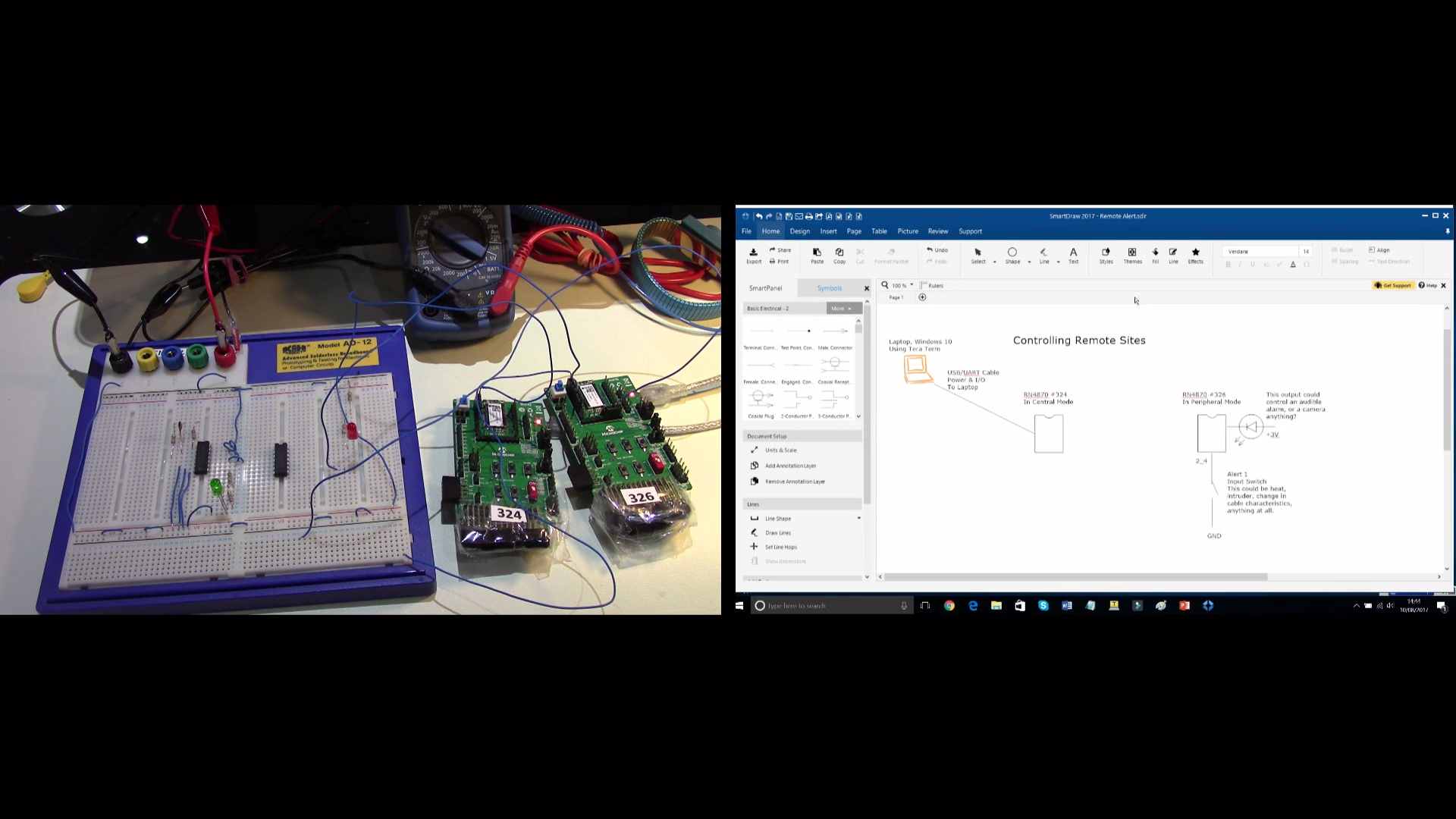
I set up two RN4870’s. The remote one (326) is acting as a remote site device. It can receive and output signals, and data. You can have human agents located with access to the central unit (324). I show how alerts can be received, identified, and you can take manual measures, such as switching on lights or cameras. Or you could integrate those identifiable data messages, with a back-end application. Or do both?

**Schematic**



**Screenshot of Demonstration**

**Video** https://vimeo.com/229129675



**Device Configuration**

I have two RN4870’s. One is number 324, and the other is number 326. I have them set up, so that on connection, the data stream feature is open, and the link secured. Please see below;

**Table 1, Features to be Supported**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** | **23** | **22** | **21** | **20** |
| **Enable Flow Control** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No prompt** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Fast Mode** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No beacon scan** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No connect scan** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **No duplicate scan result filter** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Passive scan** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **UART Transparent, no ack** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Reboot after disconnect** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Run script at power on** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Streaming** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Configuration | 0 | | | | 1 | | | | 6 | | | | 0 | | | |

**Table 2, I have configured the two RN4870’s as follows**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **324** | **326** |
|  | To change the default name | S-,MSM324 | S-,MSM326 |
|  | Enable GATT services, Device Information Service and Transparent UART | SS,C0 | SS,C0 |
|  | Runs script(s) after power on and starts streaming service, and UART without acknowledgement | SR,0160 | SR,0160 |
|  | Configure 2\_2 | SW,03,00 (GPIO) | SW,03,00 (GPIO) |
|  | Configure 2\_4 | SW,03,00 (GPIO) | SW,04,09 (Pin Trigger 1)  In this case when a threshold is met, the connection to Pin 2\_4, will go low. That will trigger the scripting event labelled as “@PIO1L”. Please see the scripting section. |

**Selection of Characteristics**

**Table 3, Notify on read part of characteristic is**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature** | **Comments** | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |  |
| Indicate | ack svr to client | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| Notify | No ack | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |
| Write |  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| Write no response |  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| Read |  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |
| Notify on read |  | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 12 |

**Table 4, Notify on write part of characteristic is**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature** | **Comments** | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |  |
| Indicate | ack svr to client | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| Notify | No ack | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |
| Write |  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
| Write no response |  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| Read |  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  |
| Notify on write |  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 14 |

**Creating the Custom Service & Characteristics**

PS,<hex16/hex128>

Command PS sets the UUID of the public or the private service. This command must be called before command PC. The effect of command PS can be verified after a valid PC command is added and after power cycle. Command PS expects one parameter that is either a 16-bit UUID for public service or a 128-bit UUID for private service.

Example: PS,010203040506070809000A0B0C0D0E0F

// Define a private service with

//UUID 0x010203040506070809000A0B0C0D0E0F

The above is 8 x 16 bits = 128 bits, e.g. 0 to F in each bit.

Notes

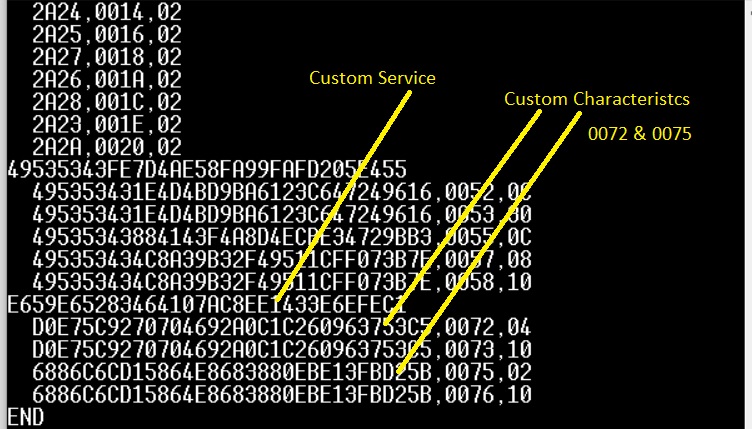
1. To create a private service, generate a 128 bit private UUID using; <http://www.guidgenerator.com/online-guid-generator.aspx>
2. A private service must be created before any characteristics are created.
3. I created this private service from above URL; PS,E659E65283464107AC8EE1433E6EFEC1
4. You also use the same url to generate individual 128 bit ID’s for each characteristic.

**Table 5, the Custom Characteristic and Services on RN4870, 326**

|  |  |  |
| --- | --- | --- |
| Item | Command | Comments |
|  | PZ | Clear any existing services |
|  | PS,E659E65283464107AC8EE1433E6EFEC1 | Custom Service |
|  | PC,D0E75C9270704692A0C1C260963753C5,14,04 | Write Characteristic, (no ack) with notify |
|  | PC,6886C6CD15864E8683880EBE13FBD25B,12,02 | Read Characteristic, with notify |
|  | R,1 | Reboot, to save configuration |

**Screenshot 2. List of Services, on 324**

Note. You will not “see” any characteristics, until you connect to another sensor.



**Handles**

|  |  |  |
| --- | --- | --- |
| Item | Command | Comments |
|  | 0072 Write handle  0073 Notify on Write enable/disable  0075 Read Handle  0076 Notify on Read enable/disable |  |

Note. You need to create the custom services, and characteristics before you write the scripts. That is because you need to know what the handles are to be used in the scripts. You will not know what the handles are, until you create the custom services, and characteristics.

**Script in 324 (client)**

Remember; To enable notification or indication to changes on handle 0072, you enable “the enable handle”, for 0072, which is 0073. Similarly, for 0075, you enable 0076.

|  |  |  |
| --- | --- | --- |
| Item | Command | Comments |
|  | @CONN |  |
|  | SM,1,0002 |  |
|  | @TMR1 |  |
|  | CI |  |
|  | SM,2,0100 |  |
|  | @TMR2 |  |
|  | CHW,0073,0100 | Enable notification on write characteristic 0072 |
|  | CHW,007**6**,0100 | Enable notification on read characteristic 007**5** |

**Script in 326 (server)**

|  |  |  |
| --- | --- | --- |
| Item | Command | Comments |
|  | @PW\_ON |  |
|  | |O,01,%0072 | Any commands using CHW,0072,xx, e.g. CHW,0072,00 or CHW,0072,01, will output on 2\_2, on 326, a logic 0, or 1. |
|  | @PIO1L | Do following (next) action, when Pin Trigger 1, goes low. Note Pin 2\_4, is connected to the alert switch, which goes low, when closed. |
|  | SHW,0075,ABCD | This sends a message to 324. The message characters are limited to 4 Hex characters, and can only contain 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E That character string could simply notify a human agent that something had occurred, and he could refer to a list of incidents. Or it could be integrated with an application, and other automated tasks could be invoked. This still offers 65,536 individual notifications, per node. I could increase this to billions, if needed. |

# Useful Technical Information

The following is a collection of information that I thought you may find useful.

**Embedded Scripting Events Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input Parameter** | **Event**  **Label** |  | **Event** |  |
| 00 | @PW\_ON |  | Power On |  |
| 01 | @TMR1 |  | Timer 1 Expired |  |
| 02 | @TMR2 |  | Timer 2 Expired |  |
| 03 | @TMR3 |  | Timer 3 Expired |  |
| 04 | @CONN |  | Connected |  |
| 05 | @DISCON |  | Disconnected |  |
| 06 | @PIO1H |  | Trigger Pin 1 Rising Edge |  |
| 07 | @PIO1L |  | Trigger Pin 1 Falling Edge |  |
| 08 | @PIO2H |  | Trigger Pin 2 Rising Edge |  |
| 09 | @PIO2L |  | Trigger Pin 2 Falling Edge |  |
| 0A | @PIO3H |  | Trigger Pin 3 Rising Edge |  |
| 0B | @PIO3L |  | Trigger Pin 3 Falling Edge |  |

**Timer Commands Explained**

Examples:

SM,1,000E // Start the timer 1 to expire in about 9 seconds, e.g. each unit is 640 ms, and in Hex

//E = 15, so delay = 15 x 640ms = approx. 9 seconds

SM,1,0000 // Stop timer 1 immediately

Let’s look at these timers in detail. Please refer to table below, if needed. The first parameter is the timer identifier, specifying one of the three available timers. The second parameter is expiration time. If the second parameter is zero, then the timer specified in the first parameter is cancelled. Unit value for timer 1 is 640 ms, while for timers 2 and 3 are 10 ms. This is the only Set command that does not save parameter in NVM and becomes effective immediately

SM,3,0100 // start timer 3 to expire in about 2.5 seconds , e.g. each unit is 10ms, so delay is

//256 x 10ms = 2.5 seconds.

Table for Timers example above, using 0000 0001 0000 0000

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 215 | 214 | 213 | 212 | 211 | 210 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PWM Command Explained**

[,<1-2>,<0-3>,<hex16>,<hex16>

Command [ supports Pulse-With Modulation (PWM) function on RN4870. It expects up to four parameters.

Note. The command “[“ includes a comma e.g. “[,” not just “[“ else the RN4870 will be waiting for content, and a closing square bracket ] and trying to treat any text as data for a parameter.

The first parameter is the PWM channel to be used in this command. Two PWM channels are supported. Channel 1 is on pin P22 and channel 2 is on pin P23. If pin P22 has been assigned to a system function, such command is ignored and RN4870 returns error message.

The second parameter is used to enable/disable PWM and clock source selection. Refer to Table 2-12 for details.

PWM Operation Selection

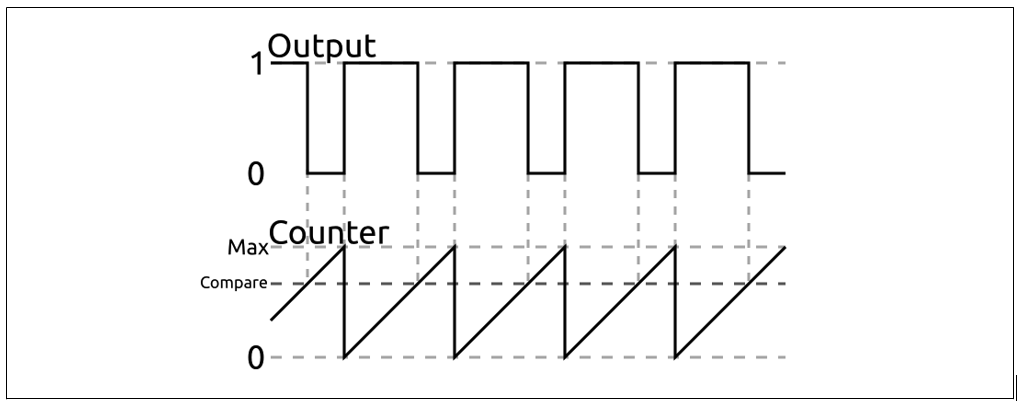
|  |  |  |
| --- | --- | --- |
| **Value** | **Description** | **Time Unit** |
| 0 | Disable PWM. Third and fourth parameters ignored | — |
| 1 | Enable PWM with 32 kHz clock | 31.25 µs |
| 2 | Enable PWM with 1024 kHz clock | 977 ns |
| 3 | Enable PWM with 16 MHz clock | 62.5 ns |

The third and fourth parameters are 16-bit hex values, defining maximum and compare values, respectively.

RN4870 follows standard PWM operations. The clock source decides the unit time used in maximum and compare values. Maximum value multiplying time unit is the PWM period; compare value multiplying time unit is the PWM width which is output high within the period. The basic concept of PWM operations is shown in Figure 2-1.

Note. You need to refer to the diagram below, to understand the above, and read this at the same time. When the level being read is greater than the compare reference level, then a logic 0 is output. When the level is less than the reference (or compared) level then a Logic 1 is output. PWM output voltage levels change based on how frequently they are sampled. So, if a level being compared to the reference level was always greater than it, then the output would always be a 0. If the level you are comparing is larger 80% of the time, then the output is a series of short duration Logic 1 pulses. In contrast, if the level you are comparing is smaller 80% of the time, then the output would be a series of much longer duration pulses.

Basic Concept of PWM Operation



Example:

[,1,3,00A0,0050 // Use PWM on P22, use 16 MHz clock. max is 10 ms, compare is 5 ms

So above example is;

[, This is a PWM command

1, Use Channel 1, which is configured by default on pin 22

3, Use 16MHz clock. Each time unit is 62.5ns

00A0 This is the maximum value time in ms

0050 This is the time when the value is compared, in ms

**Beacons, and Advertisements**

A Bluetooth device transmits a message at pre-set intervals. This can be switched off. The message is made up of information, that is contained in an advertisement. Advertisement this is the message that is transmitted. The message can be used by applications within Bluetooth devices. E.g. if you had an application that helped you to find your way around a store, then this device could be advertising its location, using a message termed an advertisement. That message will include details about its location. Your application then knows it is close to that beacon.

**Table of ASCII Programming Commands**

|  |  |
| --- | --- |
| ASCII Cmd | Description |
| S- | Set serialized device name |
| S$ | Set configuration detect character |
| S% | Set pre and post delimiter of status string |
| S: | Modify any configurations in Eflash |
| SA | Set Pairing mode |
| SB | Set UART baud rate |
| SC | Set beacon features |
| SDA | Set appearance in GAP service |
| SDF | Set firmware version in Device Info service |
| SDH | Set hardware revision in Device Info service |
| SDM | Set model string in Device Info service |
| SDN | Set manufacturer name in Device Info service |
| SDR | Set software revision in Device Info service |
| SDS | Set serial number in Device Info service |
| SF,1 | Factory Reset |
| SGA | Set RF power in advertisement |
| SGC | Set RF power in connected state |
| SM | Start timer |
| SN | Set device name |
| SO | Set power saving mode |
| SP | Set fix pin for pin code display authentication |
| SR | Set feature |
| SS | Enable default services |
| ST | Set connection parameters for central |
| SW | Assign GPIO functions |
|  | Get Commands |
| G: | Read any settings in configuration eFlash |
| GK | Get current connection status |
| GNR | Get remote device name |
| G<char> | Get the stored settings for a corresponding set command. |
|  | Action Commands |
| + | Echo |
| $$$ | Get into Command mode |
| --- | Get into Data mode |
| ! | Enter/exit Remote Command mode |
| @ | Read analog port |
| |I | Read digital port |
| |O | Set digital port |
| [ | PWM control |
| & | Static private address assignment |
| &C | Clear random address and use MAC address |
| &R | Create and use a resolvable random address |
| A | Start advertisement |
| B | Start bonding process |
| C | Connect to peer device as central |
| D | Display RN4870 critical information |
| F | Start scanning as central |
| I | Start UART Transparent with RN4020 and RN4677/4678 |
| IA | Set advertisement content immediately |
| IB | Set beacon content immediately |
| IS | Set scan response content immediately |
| JA | Add device into white list |
| JB | Add all bonded device into white list |
| JC | Clear white list |
| JD | Display all devices in white list |
| K,1 | Disconnect |
| M | Read RSSI value of connected device |
| NA | Set advertisement content permanently |
| NB | Set beacon content permanently |
| NS | Set scan response content permanently |
| O | Shut down device |
| R,1 | Reset |
| T | Change connection parameters instantly |
| U | Unbond device(s) |
| V | Display firmware version |
| X | Stop scan |
| Y | Stop advertisement |
| Z | Stop connection process |
|  | List Commands |
| LB | List all bonded device |
| LC | List all remote services as client |
| LS | List all local services as server |
| LW | List current script |
|  | Service Definition |
| PC | Define characteristic |
| PS | Define service UUID |
| PZ | Clear all service definition |
|  | Characteristic Access |
| CHR | Read remote characteristic value as client |
| CHW | Write remote characteristic value as client |
| CI | Discover remote services/characteristics as client |
| SHR | Read local characteristic value as server |
| SHW | Write local characteristic value as server |
|  | Script Control |
| WC | Clear current script |
| WP | Pause script execution |
| WR | Run script |
| WW | Write script |

# My Background in IoT

|  |  |
| --- | --- |
| I began working on this type of technology in the mid 1980’s. I then changed career, and returned to it recently. At the end of this paper you can review a list with url links, to some of my other recent technical publications, which include a range of associated compute technologies. |  |

When I thought about writing this paper, I thought things would have moved on a lot, and I might struggle to catch up. Surprisingly, I caught up fast. I soon realized that I was not so rusty after all.

I began my career in the Royal Corps of Signals. My official technical trade was; “Radio Relay Technician”. I repaired large radio systems in the frequency range from 225 MHz through to 960Mhz. I worked down at component level.

If you are ex Royal Signals you will understand this next part. I passed the Foreman of Signals exams, the week-long assessment, and was allocated an 18 month course at Blandford. Instead, I decided not to take the course, as it would effectively commit me to the Army for the rest of my working life.

The Army, then gave me a role in my last few years which allowed me to build “network of things” devices. I was also given time off to study at various colleges. I earned; OND, HNC, and much later a BSc. I was also sent on computer courses with firms such as Motorola. I built things, using TTL, CMOS I.C’s, and controlled them with 8 Bit Motorola 68XX series CPU’s. I programmed them using Boolean Logic, Assembler, or a combination of the two. These things were connected securely, using radio, or wires.

For components, and CPU’s, I used Farnell, and RS components catalogues. I did my prototyping on Veroboard, and DIL packaged I.C’s. I then made the PCB’s using Indian Ink Drawings, UV Light Box, and Acid Tank for etching the boards, and a small model/hobby drill for drilling out the PCB holes.

After the military, I worked in technical, and later sales roles for several telecommunications and computer companies. *I spent way too much time in sales type roles.* ☹

# What am I bad at?

I don’t like high level languages. Having said that I earned an A Grade in my BSc, which focused on C++. If you want me to do that, I’d need time to get up to speed on that.

# My IoT Company

In March 2016, I started my own company. I used my computing sales experience, to set it up also as a traditional computer reseller.

To begin with, I sold the normal things I was accustomed to. I won a 1,000 node Grid Computing Solution for a global bank, and a remote access middleware solution for the NHS. But, that’s not where my heart is. My interest is in the internet of things.

Obviously, I cannot do everything. I see myself as a sales engineer, who partners to deliver a complete solution.

For computing pre-sales, and installation I partner with Avnet. You cannot buy from them. They are a $26 Bn IT Distributor who have 30,000 staff globally, including vendor accredited staff on infrastructure, such as; HPE, IBM, SAP, CISCO, NetApp etc. If you work with me, I will introduce you to the engineers at Avnet, and you will work with them, as you would do with a traditional computer reseller. The difference being, that Avnet has 30,000 staff, and presence in every developed country in the world.

For IoT, industrial requirements, or Logic I.C’s and CPU’s etc, I still use Farnell, and RS components. The main change is that instead of using a paper based catalogue, I use their web based shops.

For any computer related security requirements, I will still use AVNET. If they cannot deliver what is needed, I use a specialist security distributor, called e92plus. I do feel my secure military communications background helps me in this area, but I need AVNET or e92plus to assist in terms of staff to deliver vendor accredited design and installation.

If you are an IT or Project Manager, you may be asked by your Business to deliver an Internet of Things project. My company is set up as an accredited reseller for every technology you will need. This means that if you engage with me, we can work together, to design a project delivery plan, with nothing missing, i.e. from sensor, to network, to servers, storage, through to databases, and security.

# Other Publications

* A very technical paper on NAND Flash technology.

See https://www.linkedin.com/pulse/nand-flash-technical-paper-mike-mckean

* There are several others on my Linkedin, but this IoT and the above NAND Flash paper are the most detailed.

My Linkedin https://uk.linkedin.com/in/mike-mckean-991a7743

**Internet of Things, Videos supporting this paper;**

* Custom Services & Characteristics https://vimeo.com/229142382
* Controlling a Remote Site https://vimeo.com/229129675
* Bi-Directional Streaming Data https://vimeo.com/229124625
* Real Time Development Tool https://vimeo.com/225991207

**Other Videos for IoT Turnkey Projects;**

* Infrastructure, Servers etc https://www.msm-systems.com/infrastructure/
* Legacy Server, IoT Upgrades https://www.msm-systems.com/upgrade-existing/

**Other Publications. These are much shorter;**

* Brief Technical Description of what an API is;

https://www.linkedin.com/post/edit/brief-technical-description-what-api-mike-mckean

* Ransomware through the Counterfeit Firmware, back-door;

https://www.linkedin.com/pulse/ransomware-topical-what-boring-old-firmware-mike-mckean

* Firmware;

https://www.linkedin.com/pulse/firmware-technical-description-mike-mckean

* Snapshots (Storage);

https://www.linkedin.com/pulse/short-explanation-snapshots-mike-mckean

* Cost cutting in IT. Don’t do it;

https://www.linkedin.com/pulse/dont-cut-costs-mike-mckean